Spring 3-2019

A Malware Analysis and Artifact Capture Tool

Dallas Wright
Dakota State University

A MALWARE ANALYSIS AND ARTIFACT CAPTURE TOOL

A dissertation submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Cyber Operations

March 2019

By
Dallas Wright

Dissertation Committee:

Dr. Wayne Pauli
Dr. Josh Stroschein
Dr. Jun Liu

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Philosophy degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: Dallas Wright

Dissertation Title: A Malware Analysis and Artifact Capture Tool

Dissertation Chair/Co-Chair: Wayne Pauli Date: 3/26/19

Dissertation Chair/Co-Chair: _____ Date: _____

Committee member: JUN LIU jun liu Date: 3/26/19

Committee member: Josh Stroschein Date: 3/26/19

Committee member: _____ Date: _____

Committee member: _____ Date: _____

Original to Office of Graduate Studies and Research
Acid-free copies with written reports to library

Abstract

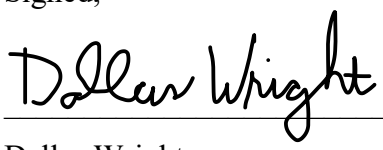
Malware authors attempt to obfuscate and hide their execution objectives in their program's static and dynamic states. This paper provides a novel approach to aid analysis by introducing a malware analysis tool which is quick to set up and use with respect to other existing tools. The tool allows for the intercepting and capturing of malware artifacts while providing dynamic control of process flow. Capturing malware artifacts allows an analyst to more quickly and comprehensively understand malware behavior and obfuscation techniques and doing so interactively allows multiple code paths to be explored. The faster that malware can be analyzed the quicker the systems and data compromised by it can be determined and its infection stopped. This research proposes an instantiation of an interactive malware analysis and artifact capture tool.

Declaration

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,



Dallas Wright

Table of Contents

List of Tables

List of Figures

CHAPTER ONE

Introduction

Malware is malicious software intentionally designed to harm data, computers and other devices or people. There are numerous classifications of malware including ransomware, trojans, rootkits, keyloggers, backdoors and others (Uppal, Mehra, & Verma, 2014; Wagner et al., 2015). Malware may be used to gather intelligence, alter data or even hold the files on a device or computer for ransom. It has been used to compromise individual financial records (Lawler, 2017), financial markets (Lynch, 2017), voting (Times, 2017) and utility infrastructure (Greenberg, 2017). The threat of malware infections has brought about the need for large investments of capital by utility companies, corporations and government entities at all levels (Amin, 2016).

Malware analysis is the study of the method of execution of malware and its effects on host systems. There are two general types of malware analysis, static and dynamic. Static analysis is the study of malware, without execution, of the program in a binary state. This analysis usually requires the disassembly and/or de-obfuscation of the malware's source or binary code. Dynamic malware analysis is the study of malware while it is executing. This typically involves executing the malware in a virtual test environment and studying its behavior (Willems, Holz, & Freiling, 2007). Malware authors employ code obfuscation to deter both dynamic and static analysis (Christodorescu & Jha, 2004). This includes obfuscation of code by functionally encrypting source code, numeric representations of characters, non-base 10 numbers, unused variables, misleading variable names, unneeded instructions and modifying program flow to confuse, and encryption.

During the execution of malware numerous artifacts are often created. The artifacts can be useful in the analysis and detection of the malware. The artifacts are often short lived existing only for brief periods of time making the task of identifying and capturing such artifacts difficult (Christodorescu & Jha, 2004). Interactive debuggers can be used to stop the execution of the malware at the moment they exist, but this is a manual process. It can be quite difficult to identify the precise moment when an artifact is created and in a non-obfuscated state.

There are tools that attempt an automatic analysis of malware including the capture of their artifacts, but these tools can be costly to set up and use, may not trigger certain portions of the

code and have limited interaction with the malware analyst. To address these issues, this work proposes a framework which could be used to automatically detect and capture malware artifacts, provide for customization, and log malware activity as well as provide a level of interactivity for the control of program flow and analysis focus. The artifact produced by this research will be a Malware Analysis and Artifact Capture Tool (MACT).

Problem Relevance

A malware intrusion can be costly both in terms of financial and reputational cost for an organization (Amin, 2016). By identifying malware quickly, it is possible to prevent infection before it occurs and remove it more rapidly after it does. Additionally, by determining the impact of malware after an infection a company can have a more direct and focused reaction to secure data that may have been compromised by the malware. This research focuses on the analysis of malware both for establishing its effect but also to aid in incident response after an infection occurs.

Several automatic malware analysis systems and services exist including Cuckoo Sandbox, ProcMon, Malwr, ThreatExpert and others. While these tools should be utilized as they do provide useful information, they lack a high level of malware analyst interactivity, customization during dynamic analysis, limited path traversal and have shortcomings that are exploited my malware authors (Mehra & Pandey, 2015). A framework that allows the analyst to control the execution path of the sample, customize the data logged during interaction with the OS and pause execution would address significant limitations of these existing tools.

Interactive dynamic debuggers, disassemblers and decompilers are available that can be used to pause execution, examine memory, set breakpoints, examine instructions and code. These tools include IDA, WinDbg, OllyDbg, Intel Debugger and others. These are effective tools, but they are general reverse engineering tools not tailored specifically toward malware analysis (Yakdan, Dechand, Gerhards-Padilla, & Smith, 2016). They are designed to look at binaries and trace process execution. These tools have functionality and customization that can be utilized to perform some functions required during malware analysis. Unfortunately, some of this customization and associated setup must be performed each time the tool is used, disassembly can be thwarted or produce incorrect results, and is not specialized to malware analysis. Additionally,

the data these tools return must be interpreted, filtered, and possibly be reanalyzed with other tools or custom programs.

A tool designed for malware analysts which would require minimal setup, limited low level review of disassembled code, analyst interactivity, and the ability to capture malware artifacts could resolve many of the issues of the current typically used tools. A malware specific software tool would allow for more efficient, effective, and timely malware analysis. These improvements could reduce the cost, both financially and reputationally, that malware incurs.

Problem Statement

There are certain phases, techniques and execution properties that malware analysts systematically and repeatedly employ that could be better performed with a tool more directly suited to the needs of a malware analyst. Research shows there is a gap in the availability of tools designed for the specific purpose of malware analysis. There is a need for a malware specific tool that would increase efficiency by limiting configuration time with fewer and more malware specific options and defaults, provide a more functionally direct interface, create more relevant reporting, and capture artifacts produced by the malware. Having a tool specifically designed to dynamically control execution flow, monitor system API's, monitor memory, log system events and capture artifacts would allow a malware analyst to more efficiently and accurately examine malware.

Purpose Statement

This study addresses the limited availability of malware analysis specific software applications. Using design science research, the purpose of this exploratory instantiation will be to design an application and then to test this application by comparing it to traditional malware analysis tools. The study will be a quantitative exploration of the analysis of malware in which malware activity and malware analysis specific artifacts will be collected from the analysis of various malware samples.

First there will be a general design that maps to the solution objectives and will consist of an application overview, architecture diagrams and subsystem descriptions. To further establish the requirements in detail, technical specifications will be identified, coding techniques established, and major functional requirements defined.

To demonstrate the artifact the research will illustrate its use while performing malware analysis. This will be done by following the tasks laid out in the testing plan. Additionally, a step by step walk through will be documented for one of the test cases and for any additional steps in other test cases needed to demonstrate each aspect of its functionality. Observation and measurement will also be part of the test plan. Measurement will take the form of quantifiable results of MACT's effectiveness in comparison to the outputs of existing tools in the form of a count of unique malware artifacts collected.

Primary Research Questions

This research project endeavored to answer the following research questions:

1. What information is pertinent to performing malware analysis?

   To design a better tool, the information required by a malware analyst must first be identified. This information is composed of malware artifacts. To answer this question, it will be necessary to explore the current literature and examine the outputs of the tools typically used to perform malware analysis.

2. What malware analysis specific feature sets will be useful to a malware analyst?

   Identify the feature sets that provide for the capture of the required information identified in question 1. This research will look at the features provided by current tools including those that assist with path traversal, saving of artifacts and activity logging. Once the feature sets are defined they can be included in the application design of MACT.

3. What are the common objectives and goals for malware analysis?

   Determine what primary malware behaviors and effects are typically targeted during analysis. Classify and define how the objectives are reached and identified.

4. Can the proposed automated malware analysis tool instantiation enable the analyst to quickly achieve the common objectives?

   It will be determined, after MACT is instantiated and evaluated, if it allows the analyst to meet some or all of the requirements of an analyst as defined by answering question 3.

Hypotheses

The hypotheses for this study is:

A software tool can be designed and created to capture and present malware analysis relevant data that can facilitate more efficient and effective malware analysis and provide additional information for analysis over what existing tools provide.

The null hypothesis will be defined as:

A software tool cannot be designed and created to capture and present malware analysis relevant data to facilitate more efficient and effective malware analysis and provide additional information for analysis over what existing tools provide.

Theoretical Framework

The theoretical framework to be used for this research will be that of a single case mechanism study (Wieringa, 2014). Each sample in the malware sample set will define a unique case for the purposes of this study. This framework can be used to determine the effectiveness of MACT at performing malware analysis.

Each sample in the malware set will be analyzed using MACT. Each of these analyses will be considered a case study. This framework will be used to compare the results produced by MACT to a predefined set of malware analysis objectives. Success or failure of each analysis objective for each sample will be determined and recorded. A tabular representation of the results for each sample will be created to aid evaluation.

.

Research Design

A set of malware samples will be established as the cases to be used for comparison. Samples will be chosen that meet the following criteria:

- can take multiple process paths
- allocates memory
- stages code in memory
- modifies Windows registry
- spawns new processes
- creates files
- modifies files
- deletes files
- communicates with external hosts

Specific Windows System APIs that are used to perform the above functionality will be identified. MACT will then be instrumented to intercept these calls. API calls whose functionality can contribute to the creation of malware analysis artifacts will be modified to log and save these artifacts for each sample.

MACT will pause and redirect program flow providing the opportunity for the interactive inspection of the system state, system resources such as memory, API parameters, and API return values. During the paused process state the analyst may substitute API return values and memory artifacts. The malware analyst will interact with the tool and the process being analyzed through a command line interface. Data will be collected by the capture of logs and files containing information identified by the analyst or tool as relevant to the analysis.

Standard malware analysis objectives related to indicators of attack and compromise will be classified and defined. The sample set of malware will be processed through MACT to determine if MACT can meet the defined standard objectives. API calls, API parameters, and artifacts will be captured and used to meet the analysis objectives. The API calls and API parameters will be documented chronologically and saved within a database log. The artifacts will include code and data from memory, data transmitted and received from external hosts, system registry modifications, created files, and modified files. A grid will be created that will identify

the standard malware analysis objectives MACT was able to reach, if any, for each malware sample.

Introduction Summary

Malware infections can be financially and reputationally expensive events. The role of the malware analyst is to identify the infection, determine the effects on the system, and to propose the remediation. To perform their role malware analysts rely on software based tools to aid them. There are many commonly used malware analysis tools. This research proposes that these tools have weaknesses that inhibit their usefulness. These weaknesses include long set up time, difficult configurations, lack of interactivity, lack of path coverage, and some have a non-specific malware analysis design.

This research presents a new tool that attempts to overcome some of the weaknesses of the current tools. A Malware Analysis and Artifact Capture Tool (MACT) is proposed. MACT will attempt to provide feature sets and a level of interactivity that will allow for the capture of specific data elements and artifacts that directly aids malware analysis. A Design Science Research study will be then conducted. MACT will be evaluated using a comparative multi-case study framework. The effectiveness of MACT will be determined by quantitatively comparing the specific common malware analysis artifacts resulting from the output of the traditional tools and MACT.

Chapter 2 will examine the current research regarding the employed methodologies of malware analysis, the classifications of tools used to support the analysis, and an evaluation of their individual strengths and weaknesses. Chapter 3 will present the application design for a new proposed tool that addresses the defined weaknesses, defines the research samples chosen, and provides a methodology for evaluating its effectiveness. In Chapter 4 the results of the research will be enumerated by employing the evaluation methodology defined in Chapter 3, specifically a comparison on the types and numbers of artifacts captured by each respective tool including the tool instantiated for this research. Finally, in Chapter 5, conclusions are drawn as to the comparative performance of the new tool with respect to currently used tools.

CHAPTER TWO

Literary Review

The activity of malware analysis can be aided by various software applications. This literary review looks at several commonly used dynamic malware analysis tools. These tools exhibit various characteristics in the form of feature sets and outputs. Some of the characteristics of the various applications overlap with some of the other tools while some provide unique information. Literature is reviewed to provide insight into the comparative and overall use, weaknesses, strengths and shortcomings of these applications.

The scope of this review will be limited to research papers that define, demonstrate, contrast, attempt to bypass, or propose variants of currently used malware analysis applications. The documentation for the software applications, when available, will be used to identify feature sets and outputs. The research papers will be used to provide clarity regarding these feature sets, how they are employed and how they aid in malware analysis.

The first section of the literary review introduces, defines and examines the activity of malware analysis. The second section examines research to identify the various categories of tools. The third section examines the commonly referenced software applications for dynamic malware analysis. The fourth section defines API hooking and how it is used by sandboxes. Finally, the results of the literary review are summarized.

Review of Literature

Defining Malware and Malware Analysis

As defined in Chapter 1, malware is malicious software intentionally designed to harm data, computers and other devices or people. Also, from Chapter 1, malware analysis is the study of the method of execution of malware and its effects on host systems. Malware is typically classified within a family based on its functional characteristics such as ransomware, spyware, trojans, rootkits, keyloggers, backdoors and others (Uppal et al., 2014; Wagner et al., 2015). Classification of malware aids in developing effective detection techniques and malware removal

tools. The precursor to classification is malware analysis (Xie, Lu, Su, Wang, & Li, 2013). To initially classify a sample of new malware it's functionality must be analyzed.

One method antivirus scanners utilize in an attempt to detect malware is syntactic signature detection. This method is a form of static analysis that is fast and returns few false positives. A weakness associated with syntactic signature detection is its inability to effectively detect new malware or obfuscated variants of existing malware (Patanaik, Barbhuiya, & Nandi, 2012). Obfuscation is a technique by which a program's functionality can remain intact while making it more difficult to detect and/or understand. Variants of malware are often modified to appear and execute in a different manner while still performing the same nefarious functions. Malware writers attempt to obfuscate their code to defeat analysis and detection by antivirus scanners (Lee, Jeong, & Lee, 2010).

There are different methodologies employed by malware authors to avoid detection. A polymorphic virus transforms itself dynamically to try to evade detection (Patanaik et al., 2012). Metamorphic viruses attempt to evade detection techniques by changing their code (Christodorescu & Jha, 2004). Oligomorphic viruses use mutated decryptors to change its form and bypass detection (Uppal et al., 2014). Obfuscations can be employed to hide binary code, prevent accurate disassembly, and mask flow. There are various coding techniques used to implement obfuscation including code unpacking, code overwriting, ambiguous code and data, obfuscated calls and returns, call-stack tampering, calling convention violations, and no-op code (Roundy & Miller, 2013). The effect of obfuscation does not limit itself to antivirus scanner detection of malware it also complicates malware analysis.

The two types of malware analysis are static and dynamic (Egele, Scholte, Kirda, & Kruegel, 2012; Gadhiya & Bhavsar, 2013). Static analysis is the study of malware without executing it. Dynamic analysis involves examining the malware as it executes and its effect on the system on which it is running. A malware analyst attempts to understand the effects of malware on a host system. To do this, the analyst must learn how the malware functions which can be done using static or dynamic methods (Uppal et al., 2014). Sometimes analysts must bypass anti-detection and anti-analysis techniques employed by the malware author (Bulazel, #252, & Yener, 2017; Dai, Pang, Zhao, & Ma, 2008). Software tools exist that are used by malware analysts to aid them in the study malware samples. These tools capture execution artifacts that are indicative

of the impact the malware is having on the system under which it is executing (Wagner et al., 2015).

An Examination of Tool Categories

To determine the categories of dynamic malware analysis tools that are commonly referenced in current research, several studies and research papers are examined. The tools studied fall into the following categories:

- Sandboxes
- Disassemblers
- Debuggers
- Utilities

The tools typically fall into two more general classifications, manual interactive tools and automated tools (Egele et al., 2012; Ömer & Samet, 2017; Xie et al., 2013). Sandboxes, such as Cuckoo, Anubis, CWSandbox, and Joe Sandbox fall into the dynamic classification while debuggers and disassemblers such as IDA Pro, WinDbg, OllyDbg, and Immunity Debugger fall under manual interactive tools (Kunwar & Sharma, 2016; Wagner et al., 2015).

Sandboxes are virtual environments used to isolate the effects of executing programs from the host environment on which they run. Sandboxes are used to help ensure any negative effects caused by programs being executed do not adversely affect the host system by rendering it unstable or unusable (Mehra & Pandey, 2015; Willems et al., 2007). These virtual environments are often used to aid malware analysts by allowing them to execute malware as well as observe and document its effects on the virtual host system (Prayudi & Riadi, 2015). Some examples of software tools providing this functionality include VirtualBox (VirtualBox, 2018), VMWare Workstation Pro (Vmware, 2018) and Cuckoo (Cuckoo, 2018).

Sandboxes are commonly used tools for performing malware analysis and are therefore targeted for evasion by many malware creators (Marpaung, Sain, & Lee, 2012). Sandbox evasion techniques include requesting human interaction such as dialog boxes, sleeping, masking processes, limited code traversal, and modifying behavior after detecting the sandbox or execution in a virtual environment (Dinaburg, Royal, Sharif, & Lee, 2008; Singh & Bu, 2013). Because of sandbox specific evasion techniques employed by malware authors, dynamic analysis using sandboxes is not as reliable of a technique as it once was (Mehra & Pandey, 2015).

Disassemblers are applications that can be used to convert binary code into low level assembly code. There are two types of disassemblers. The first type are static disassemblers which disassembles a binary without execution. The second type is a dynamic disassembler that disassembles the code during execution (Linn & Debray, 2003). Many debuggers disassemble code dynamically to present the analyst with the detailed instruction being executed allowing them to step through the execution of a program (Jämthagen, Lantz, & Hell, 2013).

Malware authors use various techniques in attempts to thwart disassembly of their binary code (Botacin et al., 2018). These techniques introduce instructions to manipulate the disassembly process and create disassembly that does not match the original assembly (Xu, Sesma, Freeman, & Li, 2006). The primary method employed to inhibit disassembly is to confuse the disassembler regarding the start location or end location of an instruction. It is possible to perform this disassembly avoidance in such a way that only isolated sections of the code are affected. The assembler can function correctly before and after these sections (Linn & Debray, 2003). One form of this method of obfuscation is to insert junk bytes that are not executed but rather inserted to cause the disassembler to present these junk bytes as code (Linn & Debray, 2003). Another way to confuse a disassembler is by altering the flow of a program by countering assumptions regarding the instruction following a branch, call, or returns (Krishnamoorthy, Debray, & Fligg, 2009; Linn & Debray, 2003). Additionally, some methods of disassembly have difficulty distinguishing data and code when both exist in the binary, (Linn & Debray, 2003).

Debuggers can be used in conjunction with disassemblers to provide a reference while stepping through the execution of a program. There are numerous debugger evasion techniques malware authors can employ including checking execution time between instructions (Gao, Lu, & Luo, 2014), examining operation system flags, registry values, string searching in memory for references to known debuggers, or looking for processes associated with debuggers (Smith, Mills, Bryant, Peterson, & Grimaila, 2014).

Utilities are tools that are focused on a limited, often individual, aspect of the malware analysis process. Wireshark focuses on network related activity, Regshot captures registry images, PEiD for detecting code packing, PEview on the structure of a given PE file, Process Explorer monitors the currently executing processes, and BinText can be used to search for specific character screens (Prayudi & Riadi, 2015). These tools are quite useful to a malware analyst and

are often used prior to and in conjunction with the more comprehensive tools defined previously. They are not intended to provide a comprehensive solution for malware analysis.

Specific Tools

In this section, the literature is studied to identify specific tools of the various categories previously defined. Kunwar and Sharma look at several of the available dynamic malware analysis tools including IDA Pro, FileMon, RegMon, Process Monitor, Process Explorer, ApateDNS, Netcat, Wireshark, InetSim, OllyDbg, Norman Sandbox, GFI Sandbox, Anubis, Malwr, Joe Sandbox, CWSandbox, ThreatExpert, BitBlaze, Cuckoo, IObit Cloud, ViCheck, and Comodo MA (Kunwar & Sharma, 2016). No judgements regarding effectiveness, comparisons, or detailed introspections of the tools are made.

Xie, Lu, et al. look at developing a malware analysis specific tool that achieves its goals by providing detailed reporting tailored to malware analysis. The authors state that the use of reverse engineering tools for malware analysis, such as IDA Pro, is very time consuming. It is also stated that the analysis results from online analysis tools such as Anubis, CWSandbox, and Panaroma do not provide enough detailed information for a malware analyst to perform an adequate evaluation of a sample (Xie et al., 2013).

Wagner, Fisher, et al. attempt to create a malware analysis aid by visualizing analysis data extracted using various tools. The authors state that Anubis provides an overview of system and network activity. Cuckoo is described as providing system interaction artifacts, provides for customization, and allows for third party tool integration. CWSandbox and Joe Sandbox, like Cuckoo, are described as providing system event malware artifacts. FireEye MAS is paired with a hardware device and is focused on issue alerts based on detected malware activity and produces an execution trace file. Process Monitor can be used to monitor process and thread activity. APIMon is mentioned as a reverse engineering tool that presents data regarding API calls but makes no determination regarding the effects of the calls on systems. IDA Pro is grouped with and classified as a generic disassembler and debugger capable of providing detailed low-level information (Wagner et al., 2015).

While documenting an overview of malware analysis, Prayudi and Riadi list of dynamic tools are Virtualbox, Anubis, Comodo Instant Malware Analysis, Process Monitor, Process

Explorer, ApateDNS, Wireshark, OllyDbg and Regshot (Prayudi & Riadi, 2015). Gadhiya and Bhavsar, for their study of malware analysis tools and techniques list Anubis, CWSandbox, Norman Sandbox, and JoeBox as dynamic analysis tools (Gadhiya & Bhavsar, 2013). Pandey and Mehtre perform a comparison of existing tools and techniques for malware detection using Regshot, Process explorer, Process Monitor, Immunity Debugger, OllyDbg, Anubis, Virus Total and Comodo (Pandey & Mehtre, 2014). Ömer and Samet, in their study to evaluate dynamic malaware analysis tools, document Process Explorer, Process Monitor, Regshot, ApateDNS, Netcat, INetSim, Wireshark, OllyDbg, Burp Suite, Cuckoo, Norman Sandbox, DroidBox, VirusTotal, Jotti's Malware Scan, VirSCAN, Votrio, Malwr, ThreatExpert, Joe Sandbox, and CWSandbox (Ömer & Samet, 2017).

While there are some tools that have more staying power than others, the pool of available tools fluctuates. Some previously open source tools are commercialized, modified, repackaged in part or full as another tool, or abandoned entirely. For the purposes of this research, tools will be chosen for each tool type defined with an emphasis on tools identified in the literature that are still in use.

API Hooking

Operating system Application Program Interfaces (API) are functions employed to abstract and perform common low-level system functions. These APIs include functions that create files, allocate memory, start processes, perform network communication, and update the system registry. API hooking is a method by which function calls can be intercepted during execution and then allowing different or additional functionality to be employed.

It is claimed that a program can be identified as malware by looking at the API calls a process makes along with the parameters used and the frequency and order in which they are called (Shaid & Maarof, 2015). The research of Ye, Wang, et al. used API call analysis to demonstrate a 92% detection rate and a faster detection time with fewer false positives (Ye, Wang, Li, & Ye, 2007). Further, it is claimed by many researchers that by using API analysis it is also possible to identify the family of malware to which the program belongs (Nair, Jain, Golecha, Gaur, & Laxmi, 2010). Lee, Jeong, et al. have proposed creating a Control Flow Graph (CFG) from the API call sequence to create a semantic signature representation of a binary (Lee et al., 2010). The authors

claim that using this semantic signature instead of the typical syntactic signature results in more effective detection of malware.

Another proposed use of API hooking is to create a log of all API events including the APIs called and their parameters. A check point for the system can be taken periodically that records the state of the system. The research theorizes that a system recovery could be done by restoring to a given check point. Additionally, starting with a check point and applying the log one event at a time it would be possible to step through malware execution and study how has affected the system (Oliveira et al., 2006).

There are existing automated dynamic malware analysis tools that utilize API hooking to perform their analysis. Sandboxes are commonly used tools and are therefore targeted for evasion by many malware creators (Marpaung et al., 2012). Sandbox evasion techniques and limitations include requesting human interaction such as dialog boxes, sleeping, masking processes, limited code traversal, and modifying behavior after detecting the sandbox or execution in a virtual environment (Dinaburg et al., 2008; Singh & Bu, 2013). Because of sandbox specific evasion techniques and limitations exploited by malware authors, dynamic analysis using sandboxes is not as reliable of a technique as it once was (Mehra & Pandey, 2015).

Literary Review Summary

Malware analysts attempt to determine what malware does and how it does it. Analysts use both static and dynamic methods to study malware (Prayudi & Riadi, 2015). Malware authors use software obfuscation techniques to make it more difficult for analysts to detect and to determine what their code is doing (Smith et al., 2014). This is because malware authors recognize that the sooner their code is understood the quicker that protections and other remediations can be employed to negate the effects of their malware.

Analysts use software tools to aid them in their study of malware samples. Interactive dynamic debuggers, disassemblers and decompilers are available that can be used to pause execution, examine memory, set breakpoints, examine instructions and code. These tools include IDA, WinDbg, OllyDbg, Intel Debugger and others. These are effective tools, but they are general reverse engineering tools not tailored specifically toward malware analysis (Mehra & Pandey, 2015; Yakdan et al., 2016). They are designed to look at binaries and trace process execution.

These tools have functionality and customization that can be utilized to perform some functions required during malware analysis. Unfortunately, some of this customization and associated setup must be performed each time the tool is used and is not specialized to malware analysis. Also, disassembly can be flawed and intentionally impeded by the malware writer using specific coding techniques (Jämthagen et al., 2013). Additionally, the data these tools return must be interpreted, filtered and possibly be reanalyzed with other tools.

While malware analysis is a form of reverse engineering there are certain phases, techniques and execution properties that malware analysts systematically and repeatedly employ that could be better served with a tool more directly suited to their needs. Having a tool specifically designed to dynamically control execution flow, monitor system API's, monitor memory, log system events and capture artifacts would allow a malware analyst to more efficiently and accurately examine malware. It would increase efficiency by limiting configuration time with fewer and more malware specific options and defaults, provide a more functionally direct interface and more relevant reporting.

CHAPTER THREE

Application Design and Specifications

Chapter 3 defines in detail the requirements, development and implementation of the Malware Analysis and Artifact Capture Tool (MACT). Software tools typically used to perform malware analysis can be difficult to set up, lack a high level of malware analyst interactivity, customization during dynamic analysis, limited path traversal, or are general purpose reverse engineering tools and are not tailored specifically for malware analysis (Mehra & Pandey, 2015; Yakdan et al., 2016). MACT is an instantiation of a malware analysis specific tool by which a malware analyst can interact with binaries, intercept relevant APIs, capture artifacts, modify the functionality performed by APIs to gather and log execution information and control the execution flow of the process using by overriding API functionality, parameters, and return values.

Research Approach

This research will follow the Design Science Methodology. Peffers, Tuunanen, et. al approach to Design Science Research includes six activities. These activities are problem identification and motivation, defining the objectives for the solution, design and development, demonstration of the artifact, observation and measurement of how well the artifact supports the solution to a problem, and communication of the research (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007).

The deductive technique will be used in conducting the research. The theory will be introduced, a literature review conducted, the artifact will be proposed, research variables will be defined, results from the artifact test will be analyzed and a conclusion will be drawn as to the effectiveness of the tool (Creswell, 2013).

Adhering to the previously defined six activities the research will begin with problem identification and motivation. The problem being identified is the need for a novel, malware analysis specific tool that allows for the capture of malware artifacts and interactive flow control. The motivation is the lack of available tools to perform the tasks required and the difficulty of malware analysis without such a tool.

The next activity will be defining the objectives for the solution. The objectives are identified as part of the design. The objectives for the solution are establishing rules that identify specific calls of an API as calls of interest, interception of relevant APIs, for each API of interest a method of information capture must be defined and coded, detailed logging of events and tool actions and user interactivity.

Design and development will be outlined in specific terms. First there will be an overall general design that maps to the solution objectives and will consist of a system functional component diagram, architecture diagram and subsystem descriptions. To further define the requirements in detail the functional specifications will be defined with a textual description of each. The functional specifications will define the objectives of each function, how they will operate, and under what circumstances they will be performed.

To demonstrate the artifact the paper will illustrate its use while performing malware analysis. This will be done while performing the tasks laid out in the testing plan. A step by step walk through will be documented for one of the test cases and for any additional steps in other test cases needed to demonstrate each aspect of its functionality.

Observation and measurement will also be part of the test plan. The use of the tool will be observed and documented. Measurement will take the form of quantifiable results of the tools effectiveness.

For communication a conclusion and summary of results will be created. Resultant dependent variables will be measured against known results to draw conclusions regarding the tools effectiveness. Future work for the artifact, improvements that can be made and the contribution made will be reviewed and discussed.

Assumptions and Limitations

For the purposes of this research and to limit the scope the MACT instantiation will be limited to 32-bit architecture which includes the version of Windows used and the malware tested. Although there will be pre-written intercept functions the success of the tool depends on the ability of the malware analyst to situationally, by sample, to create intercept functions that provide appropriate functionality in their attempt to perform an analysis. This tool, apart from some

generic routines, merely provides the framework for intercepting Windows API's. Its success depends on analysts to create and share robust intercept functions.

The tool does not claim to be universally purposed for all types of malware. Malware can be written to bypass tools by recognizing that such tools are being used or defensively by utilizing particular coding techniques. For example, if inline hooking is implemented by MACT, malware can bypass the hook by skipping 4 bytes ahead with a jump instead of using a call. This malware analysis tool, like many other tools such as IDA pro and debuggers in general, provides another means that can be used for analysis and hence another hurdle with additional steps that must be taken by malware writers to bypass. A bypass by the malware analyst, for example, would necessitate additional code and testing by the malware author thereby increasing the code complexity and cost of the malware.

MACT does not provide any of the protections of infection that a sandbox or virtual machine provides for an analyst's system. It is best to use the tool from within a protected framework. It is recommended to run MACT from within a Microsoft Windows virtual machine. This could further limit its functionality depending on the implementation of the virtual machine as the malware could be coded to recognize it is running in such an environment.

Malware Samples

In order to conduct the research a sample set of malware must be established that will be evaluated by MACT. The research samples used must meet the following criteria:

- Affect Windows 7 32-bit operating system.
- Represent variants of different malware families.
  - Backdoor – allows unauthorized access to a system.
  - Potentially Unwanted Application (PUA) – may not be malicious but considered unsuitable.
  - Ransomware - encrypts files or locks the computer so that the user cannot access their data until a ransom is paid.
  - Trojans – appears to be a normal file or program but provides access to a computer.
  - Virus - copies itself and spreads to other computers.

  - Worm – Exploits operating system vulnerabilities to spread over computer networks.
- Some or all the samples need to create memory artifacts.
- Some or all the samples need to modify and/or create files.
- Some or all the samples need to call Windows system APIs.
- Some or all the samples need to modify the windows registry.
- Preferably the samples will meet many of these criteria simultaneously.

VirusShare.com is a website that hosts a repository of malware samples for researchers (VirusShare, 2018). Using the malware share website, samples fitting the above categories, as classi can be located and downloaded. This sample set will be saved to the test platform. For the purposes of this research these malware samples will be identified and referred to by their MD5 hash. Samples tested are listed in Appendix F Malware Samples.

Iterative Development

To validate MACT, it will be necessary to verify its results against expected API response. To ensure MACT is intercepting and processing APIs correctly a test program will be written. The test program does not have to be actual malware but merely call the APIs of interest defined in Table 13 and trigger interception. The test program will provide a known set of results that can be compared to actual results created by the tool. Additionally, benign calls will be included in the test program as well as deploying different code obfuscation and encryption techniques. The test program will also be used to test the effectiveness of the interactive functionality. Logs will be generated to verify all APIs of interest were intercepted properly.

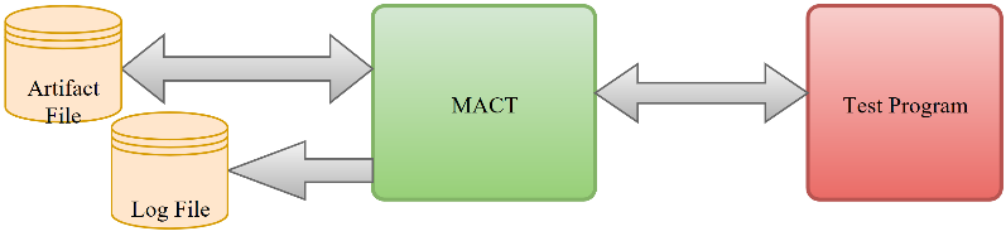


Figure 1 Using MACT Test Program

The next phase will require artifact examination and post processing of the logs generated from the execution of the test program. APIs of interest that are not referenced in the log and test analysis elements not identified or deemed to be providing inaccurate results will be identified. After reviewing the error report, the test program will be modified to reconcile the errors. This process will be continued iteratively until the are no more errors found.

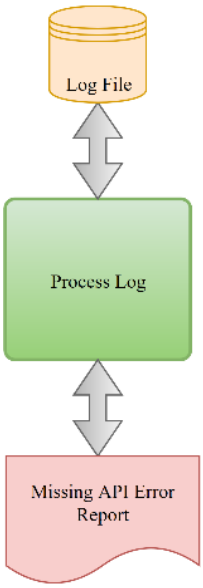


Figure 2 Generate Report from Test Process Log File

Data Analysis

To validate the effectiveness of MACT the principle objectives of malware analysis will be identified and defined. MACT will then be used to analyze malware and evaluated based on the numbers of objectives met for each sample. The key objectives of malware analysis used for evaluation will be defined as:

- Method of persistence.
- Files added or modified.
- Registry elements added or modified.
- Type of information collected.

- Communications.

To validate the effectiveness of MACT the principle objectives of malware analysis will be identified and defined. MACT will then be used to analyze malware and evaluated based on the numbers of objectives met for each sample. The key objectives of malware analysis used for evaluation will be defined as:

- Method of persistence.
- Files added or modified.
- Registry elements added or modified.
- Type of information collected.
- Communications.

Malware Persistence

One of the primary objectives of most malware is to ensure that it can spread and infect other hosts. Malware propagation refers to the spreading of malware from one host to another. This can occur by exploiting vulnerabilities in host based operating systems or applications, infecting content on one host that is eventually copied to another host and using social engineering to manipulate trusted users to bypass proper security protocols enabling the malware to gain access.

Malware persistence occurs when malware acts to ensure that it continues to execute after the initial infection. For example, a keylogger would want to ensure that it is executing every time a computer is booted up. There are several methods of persistence that may be employed by malware. One of these methods involves modifying the Windows registry (Kono, Phomkeona, & Okamura, 2018). The Windows registry is a database used to store system and application information and settings. Modifying the registry can ensure the malware is executed when certain conditions arise such as startup, service failure, at certain intervals, when logging in, and other at other system states. Additionally, data and even code can be stored in registry entries. Another common method of persistence is by intercepting the DLL search path. Programs can search a list of default directories when looking for a DLL. When the first DLL with a given name is found it

is that DLL that is loaded. Placing a malicious DLL higher in the search order can cause it to be loaded.

Files Added or Modified

Malware often creates and modifies files on the host system. These files are created for various reasons including containing transmitted programs, encryption of host data, or to establish a repository of comprised data to be transmitted later. The malware may also make copies of itself in different locations and with different file names for the purposes of persistence and propagation. These new file names may align with registry modifications made by the malware.

If examined the files can provide information to a malware analyst that may indicate what data was compromised, how to remove the malware from the host, and stop the spread of the malware. For example, the files may contain program code, internet addresses, executables for further examination, captured account information, captured keystrokes, passwords, etc. There may also be clues in these files that will aid in identifying the origin of the malware.

Registry Elements Added or Modified

The Windows registry is an integral component of the Windows operating system. It is a database used to store information related to installed programs, program options, operating system settings, software licenses, startup programs, and much more. Malware may examine registry elements to identify if antivirus is running on a system and which antivirus product it is. It could also identify and utilize any local software like remote access tools such as team viewer. Malware can add registry elements to track its installation, execute programs, and store code.

Examining what registry elements are added or modified could help the malware analyst identify a method of persistence and a means to thwart it. For example, malware may modify the registry to ensure it executes on start up. Identifying and removing this registry entry would be useful in removing malware from a system. If code is found it may be examined to provide more insight into the functioning of the malware. If malware is modifying a system's registry it is most likely for a nefarious purpose and inspection of that registry change is warranted.

Type of Information Collected

A principal purpose of malware analysis is the determination of what data is or could be compromised. Determining the best remediation method for a given virus often requires the analyst to know specifically what or who's data was compromised. Data is often collected, either quickly or over time. A keylogger can compromise what is entered into form fields including user id's and passwords. More advanced malware may target specific types of files containing confidential information.

Once it is determined what specifically has been compromised those affected could be contacted to take appropriate actions to protect themselves. For example, if it is determined that personal financial account information was compromised then blocks could be put on those accounts. If user id's and passwords were being logged, then the individuals affected could be alerted to look for malicious activity on those accounts and to change their passwords.

Communication

Malware sometimes communicates with remote hosts. The communication can be for receiving executables, instructions regarding what actions to perform, encryption/decryption keys, or in the case of ransomware payment information. Communication can also be used to transmit data back to the malware command and control. The IP of the remote host may be hard coded in the malware, resolved via DNS during execution or calculated using a domain generation algorithm. To obfuscate its communication, the malware may transmit data in small bursts to make its traffic more difficult to detect.

Identifying the address of any remote hosts can be useful in mitigating the malware. Blocking a discovered malware host address may prevent the malware from downloading additional components, interrupt its command and control, and prevent it from executing further stages.

Conclusion

It is proposed that the best method of evaluation would be to determine whether MACT could collect adequate information for a malware analyst to meet the analysis objectives of detecting methods of persistence, propagation, files modified, registry modifications, information collected, and method of transferring the data. If MACT can identify the information required to satisfy the objectives it can be considered a viable solution for a malware analyst to quickly determine the effects of malware and develop a remediation plan.

Application Overview

This research contributes to the detection and analysis of malware by presenting a novel malware analysis focused tool that provides both interactivity during dynamic analysis and customizable functionality for automated logging, more comprehensive path execution and artifact capture.

To capture malware artifacts, a Malware Artifact Analysis and Artifact Capture Tool (MACT) instantiation is proposed. The general design objectives for the tool are 1) rules that identify specific calls of an API as calls of interest, 2) interception of relevant APIs, 3) for each API of interest a method of information capture must be defined and coded 4) detailed logging of events and tool actions and 5) user interactivity.

Using API hooking the tool can intercept Windows API calls. A user interface will be used to specify APIs that the user wishes to hook. The code for the hooked API will be selected from a repository of functions. Each intercepted function must be coded, based on the functional purpose of the specific API, to provide information useful by a malware analyst either to capture an artifact, assist in its capture or record information relating to the execution of malware. These functions can be previously defined defaults installed with the tool or user created.

When performing the injected behavior, injected code will be programmed to differentiate normal execution behavior with that of malware. For example, allocation of memory on the heap with *VirtualAlloc* may be acceptable and hence its execution not monitored or logged if it doesn't attempt to make memory executable. These rules will need to be established and coded on a function by function basis.

Once the Windows APIs of interest are chosen and their associated functions defined the tool can inject the hooks into the malware process. During execution, the injected code can perform the logging, capturing of artifacts and monitoring. The captured malware artifacts will provide information useful to a malware analyst. By examining these artifacts, insight can be gained as to the methods and goals of malware. The tool can be fine-tuned for a specific sample of malware to capture additional and more relevant data.

The ability to identify specific artifacts can be used to simplify the analysis of malware by using previously analyzed artifact results. By using these more atomic malware artifacts it will be easier to identify encrypted, oligomorphic, metamorphic, polymorphic and obfuscated versions of the malware. This can lead to faster analysis and source attribution.

Each injected function will be responsible for performing relevant logging. This includes standard logging items such as API name, parameters, event messages and API specific events such as the launching of a monitor for memory related APIs. Logging of events can be used to piece together the method of attack. It can also provide information that can guide the analyst to focus on different types of calls or process paths and to modify the hooked function to gather more relevant information.

Monitoring may need to be established based on specific API calls. The monitoring can capture what is written at specific memory locations and then serialize it. It can also be used to track memory, files, registry settings, and other processes. The monitoring can be performed by the spawning of threads from injected functions. For example, if the malware allocates memory and makes it readable, writable and executable it would be prudent to monitor writes to that memory location as it is possible that an artifact is being created and will be executed from that location.

MACT can also provide the capability to pause process execution when a Windows API is called. This pause can allow for the analysis of other processes, memory, files, system registry and external communication with the MACT tool or other tools.

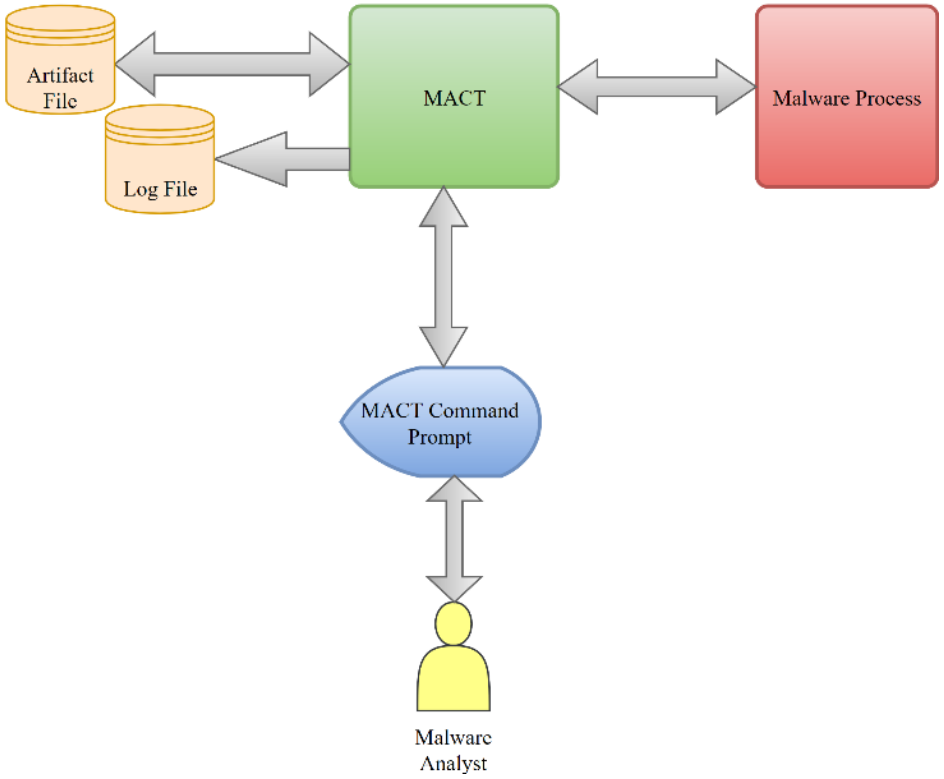


Figure 3 MACT System Flowchart

Windows API

Microsoft Windows APIs allow user programs to perform common functions in a uniform way as well as to access system functionality through the kernel (Shaid & Maarof, 2015). They abstract the low-level functionality from an application. These include creating files, reading files, allocating memory, writing to memory, accessing the network, accessing the internet, reading and writing registries.

Legitimate applications and malware both use APIs to perform their functions. Some APIs may be used to provide common and uniform services such as a print or file selection dialog. Other APIs communicate from userland to the kernel. Those APIs that allow userland to kernel communication are often used by malware to elevate privileges or perform other types of functionality available only at the kernel level.

An application's functionality determines the types, parameters and sequences of API calls. There have been various studies that claim these API characteristics can be used to classify

applications as malware and identify which class of malware. They can even be used in place of binary signature detection to identify malware and to improve upon the success of binary signature detection to identify previously unregistered polymorphic malware (Nair et al., 2010).

Malware often uses specific APIs to form the framework upon which it builds and obfuscates its attacks. Some of these API functions include VirtualAlloc to allocate memory, CreateProcess to create a new process, AdjustTokenPrivileges to modify access privileges etc. While these API functions have perfectly legitimate uses in benign software, a malware analyst may look for these calls in a program as they are often used by malware to perform its functions.

API Hooking

Hooking is the act of redirecting the program flow of a process by jumping to an address and executing injected code. Hooking is typically achieved by one of three methods, these are the Import Address Table (IAT) hook, debugger hook, and inline hook (Shaid & Maarof, 2015). The Import Address Table hook modifies the Import Address Table in the header of the PE file. Inside this table are pointers to the API code. The pointers are modified to point to a stub that logs the API calls and returns to the actual API. The debugger hook occurs when a breakpoint is set from within a debugger. In this case the entry point is overwritten with an INT 3 that causes the CPU to throw a debug exception. Inline hooking modifies the entry point of an API with code that jumps to a different function called the *detour* function. Changing the entry point modifies the original code so it is important to save the code overwritten so it can be executed later (Kumar & Goyal).

Starting with Windows XP SP2, Microsoft introduced a method by which they could perform hot patching. To implement hot patching functionality Microsoft added five bytes of data in the preamble of each API function that they could use to redirect the flow of processing and execute different or additional code. This provided an easy mechanism for inline hooking. The dummy code inserted by Microsoft was MOV EDI, EDI. By replacing the dummy instruction with a jump to another address the flow of the process can be altered (Mariani, 2011). The five bytes provide enough space for short jumps or long jumps to another code segment.

```
                        ; FUNCTION CHUNK AT 739B8F6C SIZE 0000002D BYTES
8B FF                   mov     edi, edi
55                      push    ebp
8B EC                   mov     ebp, esp
81 EC 24 04 00 00       sub     esp, 424h
A1 EC AF BC 73          mov     eax, ___security_cookie
33 C5                   xor     eax, ebp
89 45 FC                mov     [ebp+var_4], eax
8B 45 08                mov     eax, [ebp+lpFile]
53                      push    ebx
8B 5D 0C                mov     ebx, [ebp+lpDirectory]
```

Figure 4 Patch code in the function preamble.

Using API Calls to Capture Malware Artifacts

As mentioned previously, artifacts created by malware may only exist for a short time during the execution of a process before they are modified, deleted or hidden. Capturing these artifacts during dynamic analysis can be difficult and time consuming. Halting execution at the exact instruction during which the artifacts are complete and whole is challenging.

API hooking allows for altering the flow of a process and the execution of analyst code to perform other or additional functionality. After the analyst's code is executed the flow can be returned to the original function. Inline hooking can provide this functionality dynamically while not requiring the original program to be modified.

The user code jumped to can log API functions that are indicative of malware. These include allocation of memory, creation of threads, manipulation of files, modification of the Windows registry and adjustment of system privileges. With this information, the user code can further examine what is occurring and log the relevant information to be examined later. The user code may also be able to capture artifacts based on addresses used in memory allocation, process creation or thread creation. These artfiacts may contain deobfuscated script code such as HTML or unpacked programs the malware is preparing for execution.

Additionally, the user code could be used to check the system state by looking at changes in the Windows registry, memory, processes, etc. by comparing them to a previous system state (Han, Hao, Cui, Wang, & Sang, 2016) (Mosli, Li, Yuan, & Pan, 2016). These changes can be serialized, processes tracked, and malicious changes uncovered thus giving insight into what the malware is doing and how it is accomplishing its goals. These state checks could provide a method to detect malicious activity that is not apparent solely because of specific API calls. For example, a process may write over its own code and jump to that code without ever initiating a Windows API call during the process of rewriting. This may be detectable using state comparisons in a subsequent API call.

API Calls of Interest

There are numerous APIs malware could use to perform their tasks. For the purposes of this research, the categories of APIs for interception will be defined as those that use used to perform memory management, windows registry modifications, process and thread manipulation, file functions, external communication, and DLL control. The APIs to be intercepted by MACT are defined in Appendix A APIs of Interest and include some of the more common APIs used by malware (Griffin, 2013). The APIs to be intercepted by MACT can be adjusted to hook more APIs as deemed necessary. The list used may be tailored and modified during malware analysis to prevent the unnecessary interception of some calls or to add additional APIs to intercept.

Capture of Information and Malware Artifacts

After intercepting the relevant APIs, the detour function can be coded to serialize the effects before returning to the trampoline to complete execution. The elements serialized may vary depending on the API being intercepted. Each API intercepted will have to be evaluated to code the appropriate detour function actions required to serialize any relevant information required for analysis.

VirtualAlloc, for example, will require setting up a function to monitor the memory address at which the allocation occurs and intercept data when it is written to. The malware may only use this memory location temporarily to stage or execute code before it is released. The capture of the

artifact at this address can be incremental as it is written to as well as when other API functions, such as CreateThread act upon it. When these events occur a representation of the memory at the address can be written to disk for reference later.

AdjustTokenPrivleges can be serialized by recording the parameters used during the call as well as the return value. The privilege token acted upon, the privileges requested and the privileges at the time of the call will be useful in the analysis of the malware. The return values will indicate whether the privilege change was successful.

Logging of Events

Each API call being monitored will be chronologically logged. The log will take the form of a relational database to facilitate reporting and analysis of the results. The name of the API call, timing of call, address called from, return address, parameters, process thread and return values must be included. Additional information will be logged depending on the type of API call.

This log will be useful to the analyst in monitoring the execution flow of the process and identifying points of interest. Once points of interest are identified the API interception list can be modified to provide a more refined look at the processing.

User Interactivity

User interactivity can be facilitated in two ways. The creation of user defined API hook targets can be used to pause execution, create user specific logging or other functionality. This user defined functionality can provide information to the analyst dynamically in real time.
Additionally, API breakpoints can be set through the tools GUI interface. The tool can then generate the break point by creating or modifying an API hook function to perform the required break.

The tool's user interactivity will provide the analyst with the ability to monitor and make decisions regarding the analysis during the dynamic execution of the sample. Different execution paths can be followed during additional analysis sessions or by altering the detour functions. This will help to ensure better and more focused execution path coverage than other automated analysis tools currently provide.

Technical Specifications

Environment

This instantiation of the tool will be limited to a Windows executable and therefore will be designed to intercept Windows API. Development will be focused on 32-bit applications. It will be developed using Microsoft Visual Studio which is an Integrated Development Environment (IDE) developed by Microsoft. Visual Studio allows for program development as a project and facilitates the creation, source management, compilation, interactive testing, and debugging of an application.

Language

C++ will be used as the primary language. Certain aspects of the implementation will be best served using short instream assembly code segments. C++ is a language used across multiple platforms. There are a wide variety of pre-built APIs and development tools available for C++. C++ is a structured object-oriented programming language which allows applications to be broken down into callable functions. The modularization of a structured language allows a program to be logically divided into smaller, easier to understand components which simplifies development, maintenance and customization.

Windows APIs

System research can depend on the ability to extend the functionality of the operating system or user applications. Often the source code is not available to be modified to do the functional extension. Microsoft Detours Express is a Windows API used to perform inline hooking on Win32 APIs to intercept process flow and add user defined code. Detours also provides the functionality to modify a binary's DLL import table, to attach payloads to existing binaries, and do DLL injection (Hunt & Brubacher, 1999). Typical reasons to hook API functions are to add functionality, modify returned results, or insert code for debugging. This product is used by nearly

all product development teams at Microsoft and is also licensed extensively within the application development industry (Research).

One way the Detours API facilitates inline hooking is by replacing the five bytes added to the preamble for hot patching purposes to perform an unconditional jump to a target address. The replaced code is copied to another function called the "trampoline". The trampoline is composed of the instructions removed from the API and an unconditional jump to the rest of the API. These modifications are done during execution. The target function is modified in memory, but the original subroutine is not modified. The detour process then returns to the original API or it can call the trampoline function, which invokes the API. When the API completes, it returns control to the detour function. The detour function can then resume processing and then return to the original API (Galen Hunt, 1999).

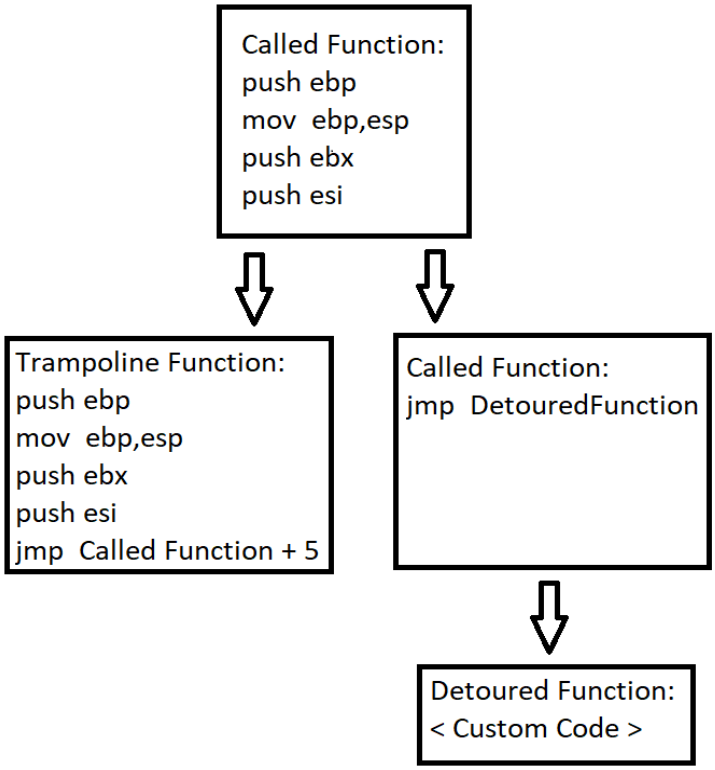


Figure 5 API Interception

Another hooking method supported by the Detours API is DLL injection. DLL injection occurs when a DLL is loaded that is not normally loaded by a process. Detours provides the

DetourCreateProcessWithDllsA method to perform the DLL insertion process as shown in Figure 6.

```
BOOL WINAPI DetourCreateProcessWithDllsA(_In_opt_ LPCSTR lpApplicationName,
                                         _Inout_opt_ LPSTR lpCommandLine,
                                         _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,
                                         _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
                                         _In_ BOOL bInheritHandles,
                                         _In_ DWORD dwCreationFlags,
                                         _In_opt_ LPVOID lpEnvironment,
                                         _In_opt_ LPCSTR lpCurrentDirectory,
                                         _In_ LPSTARTUPINFOA lpStartupInfo,
                                         _Out_ LPPROCESS_INFORMATION lpProcessInformation,
                                         _In_ DWORD nDlls,
                                         _In_reads_(nDlls) LPCSTR *rlpDlls,
                                         _In_opt_ PDETOUR_CREATE_PROCESS_ROUTINEA pfCreateProcessA);
```

Figure 6 detours.h definition of DetourCreateProcessWIthDllsA

This method in turn calls DetourCopyPayloadToProcess, shown in Figure 7, which performs the insertion into the process.

```
BOOL WINAPI DetourCopyPayloadToProcess(_In_ HANDLE hProcess,
                                       _In_ REFGUID rguid,
                                       _In_reads_bytes_(cbData) PVOID pvData,
                                       _In_ DWORD cbData)
{
    DWORD cbTotal = (sizeof(IMAGE_DOS_HEADER) +
                     sizeof(IMAGE_NT_HEADERS) +
                     sizeof(IMAGE_SECTION_HEADER) +
                     sizeof(DETOUR_SECTION_HEADER) +
                     sizeof(DETOUR_SECTION_RECORD) +
                     cbData);

    PBYTE pbBase = (PBYTE)VirtualAllocEx(hProcess, NULL, cbTotal,
                                         MEM_COMMIT, PAGE_READWRITE);
    if (pbBase == NULL) {
        DETOUR_TRACE(("VirtualAllocEx(%d) failed: %d\n", cbTotal, GetLastError()));
        return FALSE;
    }

    PBYTE pbTarget = pbBase;
    IMAGE_DOS_HEADER idh;
    IMAGE_NT_HEADERS inh;
    IMAGE_SECTION_HEADER ish;
    DETOUR_SECTION_HEADER dsh;
    DETOUR_SECTION_RECORD dsr;
    SIZE_T cbWrote = 0;

    ZeroMemory(&idh, sizeof(idh));
    idh.e_magic = IMAGE_DOS_SIGNATURE;
```

Figure 7 Detours – DetourCopyPayloadToProcess Part 1

Major Functional Requirements

For the design of the application the major functional requirements must be clearly defined. Each major functional area is placed in the MACT process flow diagram below. Following are expanded detailed description of the functionality to be implemented.

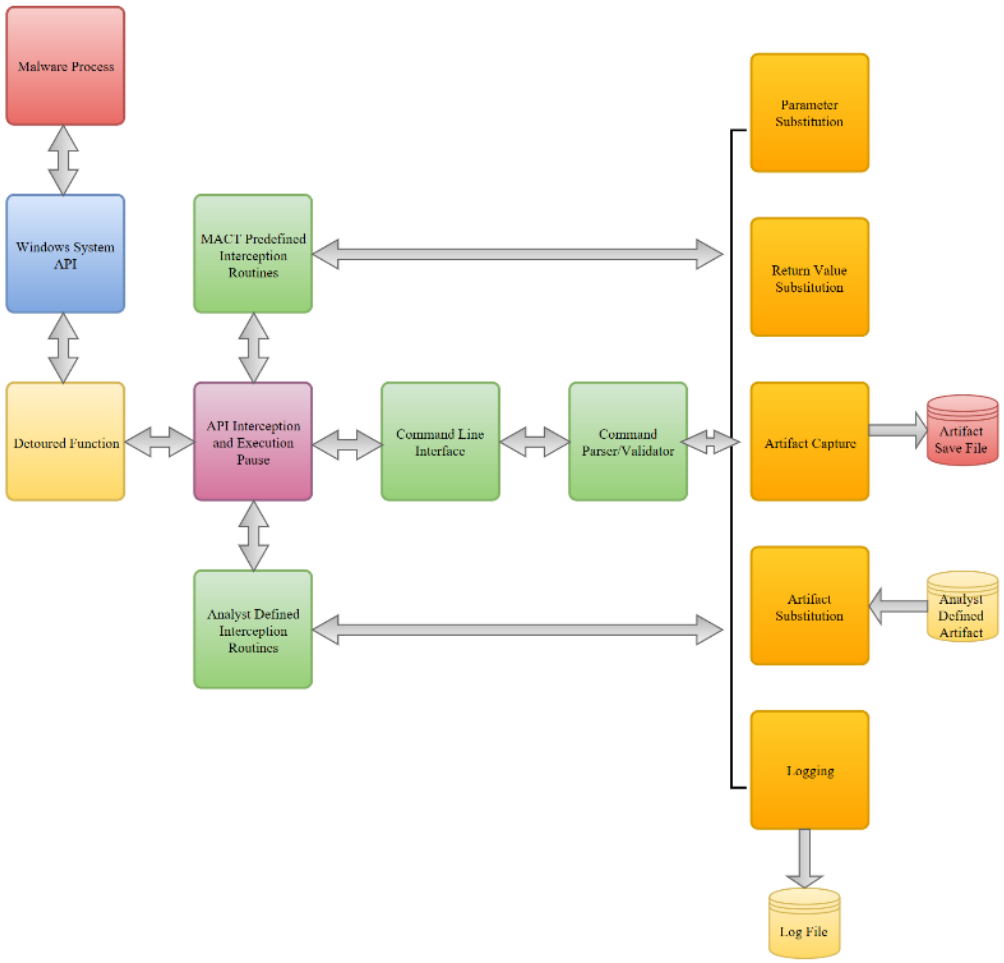


Figure 8 MACT Process Flow

Command Line Interface

This section describes the function requirements necessary to implement MACT's command line interface. The purpose of the command line interface is to provide the ability to

interact with MACT. This command line will be in a separate command window from the executing malware. Commands can be sent to MACT and messages can be sent back to the window for display.

The analyst will open a Microsoft Windows command prompt. From this command prompt MACT may be executed with a parameter indicating the program to be analyzed. The program to be analyzed will then begin executing and a MACT command line will be available in the Windows command prompt window.

User commands will be composed of two primary components, a verb and a noun, followed by the specific parameters required by the verb/noun combination. The verb will define the action and the noun will define the element to be acted upon. The list of noun and verb's along with their combinations, parameters, and descriptions are listed in **Appendix G MACT Commands**.

Command Parser/Validator

The functional requirements for the Command Parser/Validator component of MACT specifies the requirement of validating the malware analysts command line inputs. The command line inputs will be parsed into tokens and validated. Based on the validation, error messages will be displayed or valid commands will be executed.

Appendix G identifies what will be considered valid commands. Once parsed, the validation will conclude whether a valid command was given. If not, an error will be generated and displayed to the MACT user. Error messages can be found in Appendix B Error Messages.

If a valid command is identified, then the parameters are parsed for validity. After API interception but before user input, each intercepted function will create an acceptable parameter list that will be checked by the validator. If it is determined that invalid parameters were given in a command, an error message will be displayed along with the syntax for the acceptable parameter list.

The valid parameter list for an API function will consist of an array of structures equal with the number of elements equal to the number of parameters for the function. Each element will provide a textual description of the parameter and identify the type of parameter and its length in bytes. In pseudo-code the structure will be defined as follows:

```
struct FunctionParameters {
        String FunctionParamterText;
        String FunctionParameterType;
        int     FunctionParameterLength;
        }
```

Figure 9 FunctionParameters Structure

Likewise, the valid return value list for an API function will also consist of an array of structures with the number of elements equal to the number of return values from the function.

```
struct FunctionReturnValues {
        String FunctionReturnValueText;
        String FunctionReturnValueType;
        int     FunctionReturnValueLength;
}
```

Figure 10 FunctionReturnValues Structure

API Interception

There are various techniques used to implement API interception. This section describes the methods and requirements for API interception required by MACT. For the purposes of this research it will be limited to 32-bit API calls. When intercepted, control is transferred to a MACT tool method which can then perform custom functionality and then having the option of transferring control to the originally called Windows API.

MACT will intercept the API calls for the functions of interest specified in Table 13. This will be instrumented by modifying the Import Address Table (IAT). The Microsoft Detours library will be used to assist with this functionality.

Pause Execution

There is a requirement to pause execution of the application being analyzed. This pause will occur on API interception. Once a function is intercepted the interaction with the malware analyst can begin. This pause is where the majority of malware analyst interactivity and logging occurs. Once the interaction is complete the analyst will have the option to transfer control to the original target function or issue its own substituted return parameters.

Parameter Substitution

Parameter substitution will be implemented in MACT to control the flow of the program or otherwise control the processing of the malware. Once the function is intercepted and processing paused, the user can enter substitute parameters using a command with the syntax of "S P <parameter list>". The parser/verifier component of MACT will validate the parameters. If the given parameter list is invalid for the function as specified in the function parameter structure previously defined, then an appropriate error message is displayed.

Return Value Substitution

The objectives of return value substitution are like that of parameter substitution. Both can be used to control the flow and processing of the malware with the additional goal of bypassing analysis detection. When the malware is paused after API interception the analyst can enter a command and provide alternate return values. These alternate return values will be returned to the malware.

Once the function is intercepted and processing paused, the user can enter substitute return values using a command with the syntax of "S R <parameter list>". The parser/verifier component of MACT will validate the return values. If the given return value list is invalid for the function as specified in the function return structure previously defined, then an appropriate error message is displayed.

Structures and Methods to Facilitate Monitoring Memory Artifacts

To keep track and monitor for memory artifacts being created and for memory artifact substitution it is necessary to create data structures to track them. The structures will be represented by an array of objects containing memory related information that will aid in the capture of the memory artifacts. In pseudo-code the structure will be defined as follows:

```
struct MemoryBlock {
   LPVOID       MemoryAddress;
   SIZE_T       MemorySize;
   CHAR         MemoryType;
   std::string  MemoryStatus;
   DWORD        MemoryProtect;
}
```

Figure 11 MemoryBlock Structure

To establish a framework to allow for the capture of memory artifacts, information regarding memory activity will need to be acquired during execution of the malware. This information will be acquired by employing three different methods for identifying memory activity. These methods are:

1. automated by MACT
2. automated by malware analyst
3. manually by malware analyst during malware execution

As a component of MACT the memory management API functions will be intercepted and instrumented to automatically build this structure based on the memory management API functions identified in Appendix A APIs of Interest. A MACT method will be provided to allow for the building of the structures and arrays. The parameters for this function, *AddMemoryInfo*, will be *MemroyMonitorMethod*, *MemoryAddress*, and *MemoryLength*. The *MemoryMonitorMethod* to identify MACT's automated method of monitoring will have the value of "M" for MACT.

The malware analyst can choose to intercept different APIs or code for different or additional rules to create memory tracking objects. The code will need to include a call to the *AddMemoryInfo* function with the appropriate parameters. The *MemoryMonitorMethod* to

identify the malware analyst defined automated method of monitoring will have the value of "A" for analyst.

Finally, the structure may be created interactively from the MACT command prompt during execution. This can be done with the command "M O <address> <length>. The *MemoryMonitorMethod* to identify the malware analyst defined automated method of monitoring will have the value of "C" for command line.

Artifact Capture

For the purposes of this research the captured artifacts will be written to C:/MACT/ARTIFACTS/<address>-<sequence number>. The sequence number will be used to ensure a unique file name for the given execution and will be assigned numerically beginning with '001'. In future versions of MACT the malware analyst will be allowed to define the location at they wish to store the artifacts. A MACT method, *SaveArtifact* will be created with the parameters *ArtifactAddress* and *ArtifactLength*.

The same three methods used to initiate memory monitoring as described in the previous section will be used to initiate artifact capture. Restating the methods:

1. automated by MACT
2. automated by malware analyst
3. manually by malware analyst during malware execution

Automated artifact capture by MACT will be directed by the types and order of API calls using information contained in the previously defined *MemoryBlock* objects. Using the memory management API functions from Table 13. First, those API functions that imply the artifact is ready to copy, write or execute will be identified. Next, MACT will be coded to intercept those functions and write the artifact to a file using the *SaveArtifact* method.

MACT will allow the malware analyst to modify and automate intercepted functions to implement artifact capture using code for different or additional rules. The malware analyst will be able to call the *SaveArtifact* method and specify the address and length. This functionality can also be used to serialize any memory they may want to analyze later.

Artifacts may be created interactively from the MACT command prompt during execution. This can be done with the command "M A <address> <length>. This will cause MACT to initiate

the *SaveArtifact* method with the <address> and <length> parameters entered on the command line.

Artifact Substitution

The execution of certain malware artifacts may alter program flow or damage the executing environment in such a way that performing the analysis is impaired. To aid the analyst in controlling and preventing such a situation the ability to substitute artifacts will be useful. With this functionality the malware analyst can substitute more benign, or other analysis advantageous code in place of the malware created artifact. Artifact substitution will be initiated in one of the following ways:

1. automated by malware analyst
2. manually by malware analyst during malware execution

MACT will provide a function, *SubstituteArtifact*, to automate the substitution. This function will require three parameters *FileName*, *Address*, and *Length*. This function will read the file specified, for the length specified, and write it to the address given. The original contents of memory at the location specified will be written to C:/MACT/ARTIFACTS/<address>-<sequence number>.PRIORSUB.

The malware analyst will be able to call *SubstituteArtifact* from their customized code which can be placed in any intercepted API of their choosing. Using this method the analyst can automate specific artifact substitution during execution of the sample.

If the malware analyst wishes to substitute artifacts during process execution they can indicate this through MACT command prompt. This can be done with the command "M S <filename> <address> <length>. This will cause MACT to call the *SubstituteArtifact* method with the <filename>, <address>, and <length> parameters entered on the command line.

Activity Logging

To document and aid the analysis it will be necessary to create a log of all relevant malware, malware analyst, and MACT activity. Logs will be written to a process specific uniquely named file. Selected log items will be displayed to the console during execution. It will be important not

to overload the malware analyst with information that would not aid in the interactive analysis but keep that information as it may be useful to statically analyze after process execution. Some logging will be system generated by the MACT tool and other logging will be controlled from within the detoured function. Callable logging routines for use by the malware analyst will need to be provided by MACT.

The items to be logged will include:

- Detoured API function name and parameters.
- Detoured API function return values.
- Event origin. Either MACT, analyst automated, or MACT command line.
- Any MACT command parsed into noun, verb, and dommand options.
- Thread.
- Any file created.
- Errors.

All log records will include the date and time. The log file will be written to C:/MACT/LOGS/<sequence number>. This file will be an SQLite database consisting of the following fields:

Log File Definition		
Field Name	**Type**	**Description**
CALLSEQ	Integer	Sequential number indicating the order of calls.
CALLRAW	Text	The raw API call and return values as transmitted by the client to the server.
APINAME	Text	Name of the API called.
APIP01	Text	API first parameter.
APIP02	Text	API second parameter.
APIP03	Text	API third parameter.
APIP04	Text	API fourth parameter.
APIP05	Text	API fifth parameter.
APIP06	Text	API sixth parameter.
APIP07	Text	API seventh parameter.
APIP08	Text	API eighth parameter.
APIP09	Text	API ninth parameter.
APIP10	Text	API tenth parameter.
APIP11	Text	API eleventh parameter.
APIP12	Text	API twelfth parameter.
APIRET	Text	API function return value.
APIOV	Text	API override type.
APIOVRET	Text	API overridden function return value.

Table 1 Log File Definition

Application Design Summary

This chapter provided the application design for the Malware Analysis and Artifact Capture Tool (MACT). The design covered included the research approach used, assumptions and limitations, application overview, graphical and detailed textual technical specifications, coding techniques, and the major functional components. The next chapter will describe the case study results of the implementation of the MACT tool.

CHAPTER FOUR

MACT Evaluation

As described in Chapter 3, the final phase of testing was to validate the effectiveness of the Malware Analysis and Artifact Capture Tool (MACT). The evaluation involved executing MACT against various malware samples to determine if it could identify five of the objectives of malware analysis. These objectives are defined as:

1. Method of persistence.
2. Files added or modified.
3. Registry elements added or modified.
4. Type of information collected.
5. Method of information transference.

APIs captured are stored in the logs. There are two logs that are used mainly for execution tracing and debugging. All information for a sample is saved under a uniquely named sample run directory under the main MACT folder. For the server the execution log is *serverlog.txt* and for MACT it is *log.txt*. These files are used primarily for debugging the server and the client. *Serverlog.txt* may also contain informational messages sent by MACT which do not fit the normal structure of *MACT.db*.

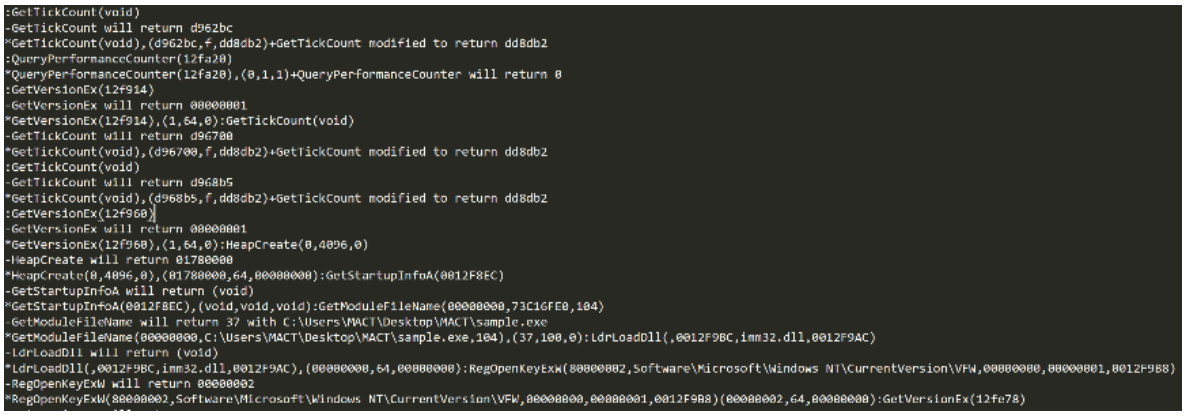


Figure 12 log.txt

```
:GetTickCount(void)
>GetTickCount too many ticks.
*LdrLoadDll(,01DCFDF4,C:\Users\MACT\Desktop\MACT\\vsystem.dll,01DCFDE4),(C0000135,64,00000000)-GetTickCount will return daec26
-LoadLibrary will return (void)
*GetTickCount(void),(daec26,f,dd8db2)*LoadLibrary(C:\Users\MACT\Desktop\MACT\\vsystem.dll),(00000000,64,00000000)+GetTickCount modified to return dd8db2
:Sleep(3e8)
-Sleep will return void.
*Sleep(3e8),(void,void,void):LoadLibrary(C:\Users\MACT\Desktop\MACT\\vsystem.dll)
:LdrLoadDll(,01DCFDF4,C:\Users\MACT\Desktop\MACT\\vsystem.dll,01DCFDE4)
-LdrLoadDll will return (void)
*LdrLoadDll(,01DCFDF4,C:\Users\MACT\Desktop\MACT\\vsystem.dll,01DCFDE4),(C0000135,64,00000000)-LoadLibrary will return (void)
*LoadLibrary(C:\Users\MACT\Desktop\MACT\\vsystem.dll),(00000000,64,00000000):Sleep(3e8)
-Sleep will return void.
```

Figure 13 serverlog.txt

Persistence indicates the method by which the malware survives on the host operating system after rebooting or during attempts to remove it. To determine the method of persistence used by a malware sample the API calls and captured artifacts are examined. For example, API calls creating new copies of the original malware binary are identified by examining the database of API calls collected by MACT. Additionally, the registry entries created or modified could indicate how the malware re-executes itself when specific events occur.

Files created, deleted, or closed are also identified and saved by MACT. Files can be used to store compromised data, consist of malware unpacked from the original sample or be copies of the original sample. Files are saved in the sample's MACT "Files" artifact folder for examination. Files are checked for being an executable binary by using the Microsoft C++ *GetBinaryA* function. If a file is identified as an executable it is noted in the serverlog.txt file under the main MACT directory for the sample being analyzed.

Figure 14 Files sub folders.

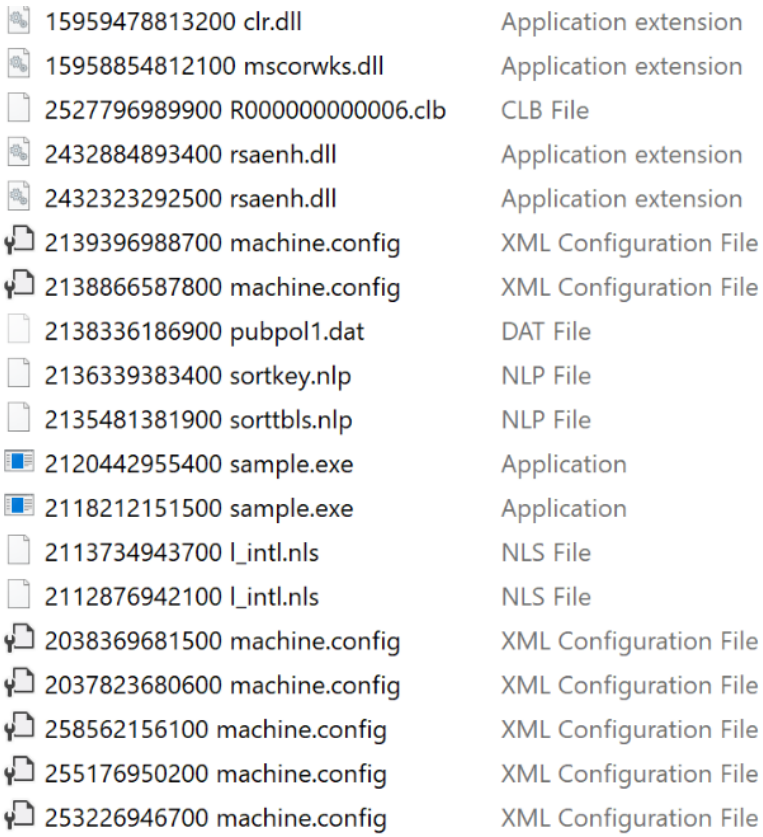


Figure 15 Example of file artifacts.

Registry modifications can be used to execute malware when certain conditions arise, including on boot. The file *reg.txt* contains messages with detail regarding the intercepted malware registry open, read, and write attempts. The API call used for the inquiry or update is also displayed. Brackets at the beginning and end of the strings returned from the registry are added by MACT to indicate the beginning and end of the string.

```
>RegOpenKeyEx Key = [Software\Microsoft\WAB\WAB4\Wab File Name]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegQueryValueEx Key = []
>RegQueryValueEx Key = [TcpAutotuning]
>RegGetValueA SubKey = [SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegGetValueA Value  = [ProxySettingsPerUser]
>RegGetValueA SubKey = [SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegGetValueA Value  = [EnableLegacyAutoProxyFeatures]
>RegQueryValueEx Key = [BadProxyExpiresTime]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegQueryValueEx Key = [AutoProxyDetectType]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegQueryValueEx Key = [DisableBranchCache]
>RegQueryValueEx Key = [UseFirstAvailable]
>RegQueryValueEx Key = [CombineFalseStartData]
>RegQueryValueEx Key = [DisableFalseStartBlocklist]
>RegQueryValueEx Key = [EnforceP3PValidity]
>RegQueryValueEx Key = [DuoProtocols]
>RegQueryValueEx Key = [EnableSpdyDebugAsserts]
>RegOpenKeyEx Key = [System\Setup]
>RegQueryValueEx Key = [SystemSetupInProgress]
>RegSetValueExW Key = [ProxyEnable]
>RegSetValueExW Key = [SavedLegacySettings]
>RegOpenKeyEx Key = [Software\Microsoft\Internet Account Manager\Accounts]
```

Figure 16 reg.txt example.

Memory allocations may exist only for short periods of time but can contain important information for the analyst such as executable binaries, DLLs, strings used for comparison, and data to be transmitted. Memory artifacts are saved by the memory monitors when memory changes, on free, periodically, or manually via the command interface. Memory artifacts are saved under the samples "Mem" directory. The function that initiated the saving of the artifact is included in its name as well as it's virtual address.

591000 VirtualAlloc 2123828161400.bin	BIN File
593000 VirtualAlloc 212249758300.bin	BIN File
593000 VirtualAlloc 2122377358800.bin	BIN File
593000 VirtualAlloc 2123500560800.bin	BIN File
600000 VirtualAlloc 2122970159900.bin	BIN File
650000 VirtualAlloc 2123984161700.bin	BIN File
650000 VirtualAlloc 2123999761700.bin	BIN File
650000 VirtualFree 212379360100.bin	BIN File
650000 VirtualFree 2123843761400.bin	BIN File

Figure 17 Examples of memory artifacts.

Communication is identified by intercepting various functions that may be used to connect to the internet and transmit data. This information includes the IPs connected to and possible data

sent and received. This data is stored in the *comm.txt* log file under the main MACT directory for the sample. The file *comms.txt* contains messages relating to intercepted malware communication attempts. This includes some of the API calls for IP resolution, internet connection, and various transmission protocols. The API call and the IP or address are displayed. Data transmitted by the sent command is also written to this log.

```
>inet_addr 192.168.56.2
>connect IP: 4.241.24.72 Port: 4612

>inet_addr mx4.org.aalto.fi
>gethostbyname: mx4.org.aalto.fi

>connect IP: 130.233.0.135 Port: 6400

>inet_addr edmweb.com
>gethostbyname: edmweb.com

>connect IP: 207.148.248.143 Port: 6400

>connect IP: 207.59.54.11 Port: 4612

>inet_addr mx1.org.aalto.fi
>gethostbyname: mx1.org.aalto.fi

>connect IP: 130.233.0.132 Port: 6400

>inet_addr mx.edmweb.com
>gethostbyname: mx.edmweb.com

>connect IP: 207.148.248.143 Port: 6400

>inet_addr mx2.org.aalto.fi
>gethostbyname: mx2.org.aalto.fi
```

Figure 18 comms.txt example.

Finally, *MACT.db* is the SQLite database defined as in Table 1 Log File Definition. This file is the primary source of information related to API activity. It is the primary source for analyzing the activity of the malware.

APINAME	APIP01	APIP02	APIP03	APIP04	APIP05
Filter	Filter	Filter	Filter	Filter	Filter
GetProcAddress	77470000	ImmSetCompositionStringW			
GetProcAddress	77470000	ImmSetCandidateWindow			
RegOpenKeyExW	80000002	Software\Microsoft\Windows NT\CurrentVersion\VFW	00000000	00000001	0012F9B8
GetVersionEx	12fe78				
GetModuleHandle	NULL				
HeapCreate	0	4096	0		
GetModuleHandle	kernel32.dll				
GetProcAddress	76360000	InitializeCriticalSectionAndSpinCount			
GetModuleHandle	kernel32.dll				
GetProcAddress	76360000	FlsAlloc			
GetProcAddress	76360000	FlsGetValue			
GetProcAddress	76360000	FlsSetValue			
GetProcAddress	76360000	FlsFree			
GetStartupInfoA	0012FE30				
GetModuleFileName	00000000	C:\Users\MACT\Desktop\MACT\sample.exe	104		
GetModuleHandle	KERNEL32				
GetProcAddress	76360000	IsProcessorFeaturePresent			
GetTickCount	void				
QueryPerformanceCounter	12fe50				
GetVersionEx	12fdc4				

Figure 19 MACT.db

MACT Example Use

When testing MACT, two command line windows are opened. In one, the server executes. The server recieves the data from MACT, creates a data base of API calls, creates the server log, and provides a command line interface through which the user can interact with the malware via the injected DLL *mact32.dll*. Figure 20 shows the server starting up an attaching to the malware. The server window displays API activity formed by data sent from MACT. This includes the name of the API call, parameters of the API call, the return value, and informational messages.

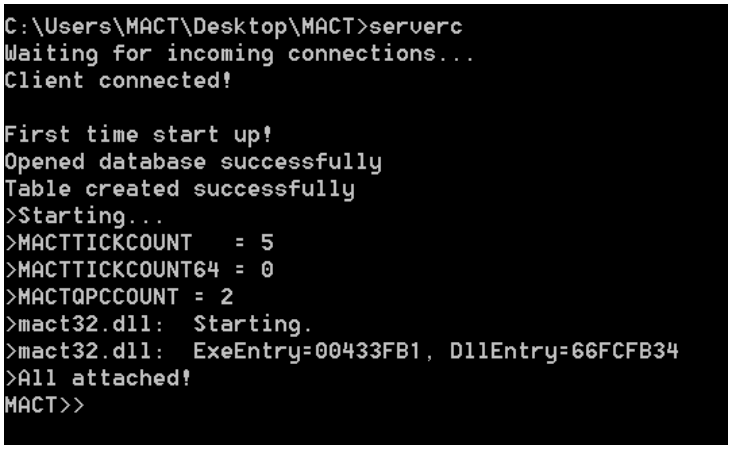


Figure 20 mact32.dll startup on injection.

The second window is the one through which *mact32.dll* is injected in to the malware process using the command *withdll.exe -d:mact32.dll <malware binary file name>*. *Withdll.exe* is a program that is provided with the Detours API that injects the specifed DLL in to the specified process. An injected DLL would be difficult for malware to identify. For example, a DLL may be checked for relative offsets of functions but that may prove difficult as offsets can vary based on the version of the Windows API being used.

The commands available through the server are displayable via the "D C" command from the server MACT prompt as shown in Figure 21.

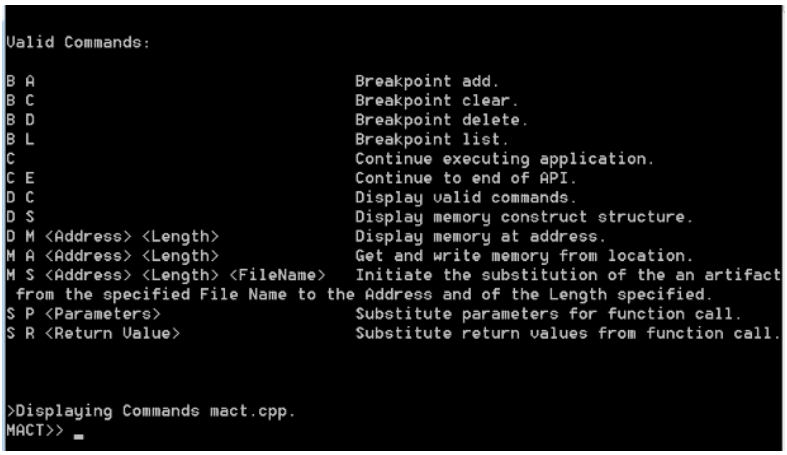


Figure 21 Available MACT commands.

Breakpoints can be set, cleared and listed as demonstrated in Figure 22. When an API function name matching the breakpoint is encountered execution halts before the ececution of the "true" function. This pause allows for the substituion of return values or execution of other MACT commands.

```
MACT>> b a createprocess
Breakpoint Add
>Breakpoint added for CREATEPROCESS
MACT>> b a winexec
Breakpoint Add
>Breakpoint added for WINEXEC
MACT>> b l
Breakpoint List
>Breakpoints:
>CREATEPROCESS
>WINEXEC
MACT>> b d createprocess
Breakpoint Delete
>Breakpoint CREATEPROCESS deleted.
MACT>> b l
Breakpoint List
>Breakpoints:
>WINEXEC
MACT>> b c
Breakpoint Clear
>Breakpoints cleared.
MACT>> b l
Breakpoint List
>Breakpoints:
>No breakpoints defined.
MACT>>
```

Figure 22 Breakpoint command examples.

In Figure 23 a breakpoint is set on GetProcAddress and a MACT command is issued to continue execution until the end of a breakpoint is encountered. API execution messages are displayed until the breakpoint is reached at which point execution of the malware is paused. Once paused, the command prompt is available to the analyst.

```
MACT>> b a getprocaddress
Breakpoint Add
>Breakpoint added for GETPROCADDRESS
MACT>> c e
Continue to end..
:GetTickCount(void)
-GetTickCount will return dfa603
+GetTickCount modified to return dd8db2
:QueryPerformanceCounter(12fa20)
+QueryPerformanceCounter will return 0
+QueryPerformanceCounter modified to return 12fa20
:GetVersionEx(12f914)
-GetVersionEx will return 00000001
:GetTickCount(void)
-GetTickCount will return dfaa47
+GetTickCount modified to return dd8db2
:GetTickCount(void)
-GetTickCount will return dfabfb
+GetTickCount modified to return dd8db2
:GetVersionEx(12f960)
-GetVersionEx will return 00000001
:HeapCreate(0,4096,0)
-HeapCreate will return 01630000
:GetProcAddress(76360000,FlsAlloc)
MACT>> c
-GetProcAddress will return 763AF339
:GetProcAddress(76360000,FlsGetValue)
MACT>> _
```

Figure 23 Breakpoint on GetProcAddress

When the breakpoint is triggered execution of the “true” API function has not occurred. Stepping through with the “c” MACT command will show the return value. This example also demonstrates displaying the contents of memory using the “d m” command as show in Figure 24.

```
-HeapCreate will return 01630000
:GetProcAddress(76360000,FlsAlloc)
MACT>> c
-GetProcAddress will return 763AF339
:GetProcAddress(76360000,FlsGetValue)
MACT>> d m 0x763af339 40
763af339: 8b ff 55 8b ec 5d eb 05 90 90 90 90 90 ff 25 c0   ..U..]........%.
763af349: 1d 36 76 90 90 90 90 90 8b ff 55 8b ec 83 ec 10   .6v.......U.....
763af359: 53 33 db 38 1d ec 02 fe 7f 75 73 64 a1 18 00 00   S3.8.....usd....
763af369: 00 8b 40 30 f6 40 03 02 75 64 8d 45 f8 50 e8 f3   ..@0.@..ud.E.P..
MACT>> _
```

Figure 24 Stepping through execution and "d m" command.

Using the “m a” command it is possible to capture an artifact while interactively executing the malware. In Figure 25 a memory artifact is captured by executing the “m a”

command followed by the address and length. Memory artifacts are also captured automatically as previously described. While this functionality is similar to what is typically already provided via a debugger, it is still sometimes necessary to examine memory locations to make decisions regarding interactive analysis steps. For example, addresses returned as API parameters or return values may need to be explored or saved during the analysis and provide insight into relevant breakpoints to set. Figure 26 shows an example of a manually saved memory artifact.

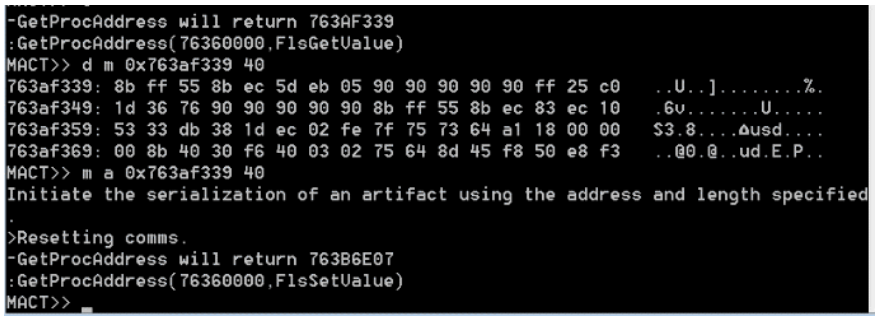


Figure 25 Interactively capture memory artifacts.

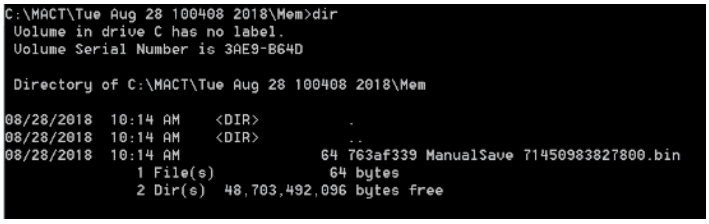


Figure 26 Directory listing of interactively captured memory artifact.

Preliminary Functional Testing

After the preliminary testing of MACT it was determined that the original list of functions to be intercepted needed changes and additions. One of the issues discovered using the original list of functions was that some functions called other supporting functions that were also intercepted causing a lot of unnecessary and non-malware related function call logging. Another issue was the usefulness of the information provided by some of the intercepted functions was determined to not provide insight into the malware. Also, during testing new functions emerged as being necessary for the analysis. These modifications were necessary to allow the minimum

functionality required for analyzing malware and capturing artifacts. **Appendix C Modified APIs Intercept Functions** contains the final list of intercepted APIs used for this research.

Testing Process

Testing MACT involved using it to analyze various malware samples and examining the information and artifacts produced in an attempt to meet the analysis objectives previously defined. The general testing steps are:

1. Start a Windows 7 32-bit virtual machine (VM).
2. Copy the server program, *mact32.dll*, and the malware sample to the virtual machine (VM).
3. Activate VPN.
4. Execute the server.
5. Execute the sample while injecting *mact32.dll* into the malware.
6. Interact with the malware via the MACT server to execute the malware sample.
7. Collect logs and artifacts.
8. Evaluate the information from the logs and artifacts.

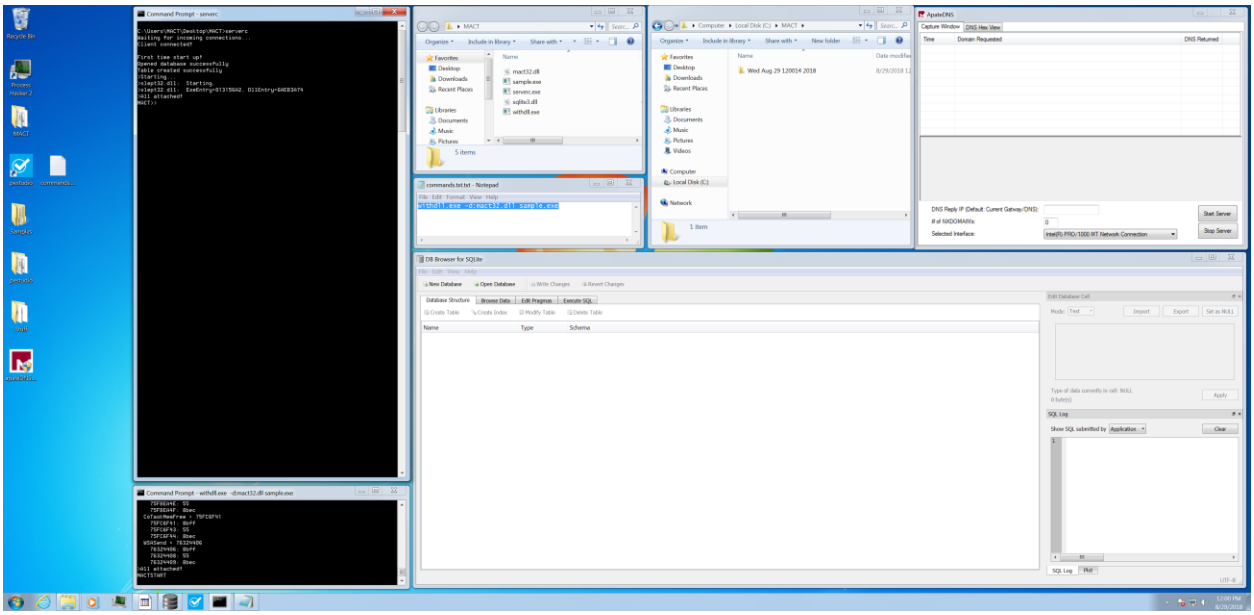


Figure 27 MACT test VM environment.

Applications available on the VM are:

1) MACT
2) PE Studio – Examines a PE file and identifies what it considers suspect elements. This was used to extract MD5 hash and verify the sample is a 32-bit windows application.
3) ApateDNS – Response to DNS queries locally allowing the response to be customized.
4) Process Hacker – Process Viewer. This was used to monitor and kill processes.
5) strings – a utility that searches a file or directory for strings of 3 or more characters

How MACT is used to perform an analysis is up to the individual malware analyst. For the purposes of this study, the malware was executed initially with *mact32.dll* injected, no breakpoints, and with communications intercepted. This is the least invasive way to run the malware as its execution effects should not extend beyond the VM. This execution produced information and artifacts to be used to identify data and breakpoints that may provide more useful information or control the flow of processing.

Evaluation of the database, registry entries, communication, memory, and file artifacts of the initial run is performed. Signs of anti-analysis techniques such as *GetTickCount*, are noted. Any anti-analysis techniques will have to be bypassed to get a fuller execution of the malware sample. Messages from the server log relating to executables being created in memory allocations and their associated artifacts are indicative, at this point, of unpacked executable files that may be executed in their own process or saved as a file. Created file artifacts can also be examined for executable files or data that may indicate the purpose of the malware. Registry entries may indicate the malware's method of persistence or other malicious activity such as turning off operating system or third-party protections.

The next part of the test, if necessary, is to reset the VM to a starting snapshot. For this phase any anti-analysis techniques are bypassed during the malware execution and more information is gathered. More anti-analysis techniques may be discovered which may lead to subsequent executions of the malware to bypass. The types of anti-analysis discovered in this research included timing, process spawning, process termination, driver termination and service disabling.

Once all discovered anti-analysis attempts are bypassed, if any were found, the next focus is on the purpose and communication methods of the malware. To determine the purpose the

analyst can examine the file and memory artifacts looking for the data being examined, the type, location, and content of the files accessed and created, as well as the specific APIs, types, and order of API calls may provide insight into the malware's intent. Sometimes thousands of memory artifacts are captured. To further aid in analyzing the contents of the memory artifacts the Windows Sysinternals *strings.exe* (Russinovich, 2018) utility was used.

Bypassing Anti-analysis Techniques with MACT

This section will look at various anti-analysis techniques detected in the samples studied for this research. The techniques mentioned are not intended to be all inclusive. Additionally, it is possible that some anti-analysis techniques implemented by the malware samples tested were not detected.

Bypassing Timing Techniques

Anti-analysis timing techniques are sometimes used by malware to try to determine if it is being executed in a sandbox, debugger, or other analysis tool. For example, when using a debugger to step through the execution of a program and pausing for analyst interaction the delay between the timing checks would be much greater than expected during a normal execution. If the malware suspects, based on the elapsed time between instructions, that it is being analyzed it can then take actions to delete files, registry items, and other elements in attempt to hide its intent. It can also terminate execution thereby stopping any further dynamic analysis. While using MACT, the act of intercepting the call, logging, communicating with the server, and custom code even without running in an interactive mode was sometimes enough to affect the timing to a degree which would register with the anti-analysis technique.

There were three types of timing API's detected during testing. These were *GetTickCount*, *GetTickCount64*, and *QueryPerformanceCounter*. *GetTickCount* returns the number of milliseconds since system startup with a maximum of 49.7 days (Microsoft, 2018b). *GetTickCount64* returns the number of milliseconds since system startup with no maximum (Microsoft, 2018a). *QueryPerformanceCounter* functions similarly to a stop watch and returns the current value with a resolution of 1 microsecond (Microsoft, 2018d). Timing check API calls

exceeded all other types of anti-analysis techniques detected for the samples analyzed in this research.

Using MACT interactively it is possible to override the return parameters with specific values and bypass the timing checks. With some samples executing over 1,000 timing checks it was not practical to depend on the interactive override. Due to the large number of timing checks and the number of the tested samples using this technique MACT was functionally modified to provide a more automated solution. To implement this new solution MACT was changed to check for a *gettickcount.txt* file in the directory it was executed from. If this file was found and there were entries, they were used in sequential order to replace the return values from *GetTickCount*. If the number of *GetTickCount* API calls exceeded the number of entries in the file, the last entry from the file was returned for every subsequent call and a message sent from MACT to the server to be displayed in *serverlog.txt* indicating the number of API calls expected had been exceeded. The same methodology was used for *GetTickCoun64* with *gettickcount64.txt* and *QueryPerformanceCounter* with *opccount.txt*. This process works if malware uses any of these API calls for timing anti-analysis.

To gather the data for the *.txt* files a separate, limited functionality, version of MACT, *mactgt32.dll,* was created. *Mactgt32.dll* was copied to the VM and injected into the malware in place of *mact32.dll*. This DLL intercepted only *GetTickCount*, *GettickCount64* and *QueryPerformanceCounter*. To limit the numbers of instructions executed the intercepted function only called the non-detoured versions of the functions and logged the return values to *systemlog.txt*. It performed no other custom processing as that would increase the probability of triggering the detection. The return values were then extracted from the log and inserted into the files of the appropriate name for the specific API function called.

After the *.txt* files were created *mact32.dll* was then injected into the malware sample with the return values being substituted. For the samples studied in the research, this method appeared to be successful at bypassing the timing checks to avoid analysis detection. Execution continued past the timing check API calls. Further code was executed and as a result more information and artifacts were collected allowing further insight into the functioning of the malware.

Terminating Anti-virus Monitoring Processes

Some of the malware samples used *Process32FirstW* and *Process32NextW* and then compared those processes to a list of known drivers for anti-virus software. After finding such a driver they would use *TerminateProcess* to end the process to prevent it from detecting the malware. This anti-analysis technique could be overcome by using MACT to change the return value from the string compares used to identify the process to indicate the string was not a match for a given driver. Because there was no anti-virus running in the VM used for testing this anti-analysis technique was not an issue.

Bypassing Sandbox Detection

Several API functions allow for detecting a debugging state such as *IsDebuggerPresent* and *CheckRemoteDebuggerPresent*. Additionally, examining registry entries, file names, drivers, and other indicators may provide a means to identify installed malware analysis sandboxes and other anti-analysis tools. Since MACT is not a known malware analysis tool it is unlikely to be identified using these methods.

Another approach to detect debugging would be for the malware to identify it is running in sandbox or other virtual environment. There was no evidence of the VM being detected in the data collected. However, since some of the samples did stop executing it cannot be ruled out that some of the malware was able to successfully recognize the VM. If so, there was no work around unused by MACT for the samples examined.

Creating New Processes

ShellExecuteExA and *ShellExuecuteExW* use the structure *ShellExecuteInfo* which contains relating to the process being created. When overriding *ShellExecuteEx** MACT can be used to modify the elements of the structure. For example, *lpParameters* can be modified to include the command statements necessary to inject MACT on execution.

```
typedef struct _SHELLEXECUTEINFOA {
  DWORD     cbSize;
  ULONG     fMask;
  HWND      hwnd;
  LPCSTR    lpVerb;
  LPCSTR    lpFile;
  LPCSTR    lpParameters;
  LPCSTR    lpDirectory;
  int       nShow;
  HINSTANCE hInstApp;
  void      *lpIDList;
  LPCSTR    lpClass;
  HKEY      hkeyClass;
  DWORD     dwHotKey;
  union {
    HANDLE hIcon;
    HANDLE hMonitor;
  } DUMMYUNIONNAME;
  HANDLE    hProcess;
} SHELLEXECUTEINFOA, *LPSHELLEXECUTEINFOA;
```

Figure 28 ShellExecuteInfoA structure.

As demonstrated below, as well as in the analysis for the sample 2F54B592E62E66FBFE7F7DB24F70F36A,

```
*GetForegroundWindow(),(000B074E,64,00000000)>>DEBUG Function: MyShellExecuteExW
>ShellExecuteExW lpVerb       = runas
>ShellExecuteExW lpFile       = C:\Windows\system32\wbem\wmic
>ShellExecuteExW lpParameters = process call create "cmd /c start C:\Users\MACT\Desktop\MACT\sample.exe"
>ShellExecuteExW lpDirectory  = NULL
:ShellExecuteExW(000F0000)
```

Figure 29 ShellExcuteInfoW example parameters.

when executing with *lpVerb wmic* to start the Windows Management Instrumentation Command it is possible to modify *lpParameter*s with *start withdll.exe -d:mact32.dll sample.exe.* By modifying *lpParameters* in this way it is possible to inject MACT into the new process. To monitor and interact with the API calls in the new process another instance of the server, listening on another port, must be started.

MACT Testing

To evaluate the malware the API call database, MACT log, server log, and captured artifacts will be examined. While examining the results non-malware related information and

artifacts generated by Windows or MACT itself were found. This "noise" can be caused by non-detoured Windows API functions calling detoured windows functions as well as Windows activity related to setting up the PE file for execution.

Some noise was created by the .NET assemblies. .NET uses just-in-time compilnation that converts compiled code into the executable binary. The .NET assembly is an extension of the PE file format. In this format, when the program is executed, the code section contains a small program which executes _CorExeMain which in turn invokes the virtual machine startup (Microsoft, 2018c).

The key point for testing .NET extended PE files is that the execution of _CorExeMain and its related activity of API calls, registry reading/writing, and DLL manipulation can create some noise when starting a .NET based malware sample. It can also be considered a form of obfuscation as tools that examine a base PE file will not be able to determine its properties in a static state. It isn't until the binary is loaded that its full properties can be determined.

MACT Malware Evaluation Case Studies

Sample 1 – PUA

MD5 361B481084C41D0DF4C4EB763F268043

Link

https://www.virustotal.com/#/file/a779f1bb69a4767aaafa6cfee551563ee19455f48c19b1a0d3f0bae4063b16d4/detection

Classification PUA

Overview Copies itself to conhost and set itself up to execute via registry modifications.

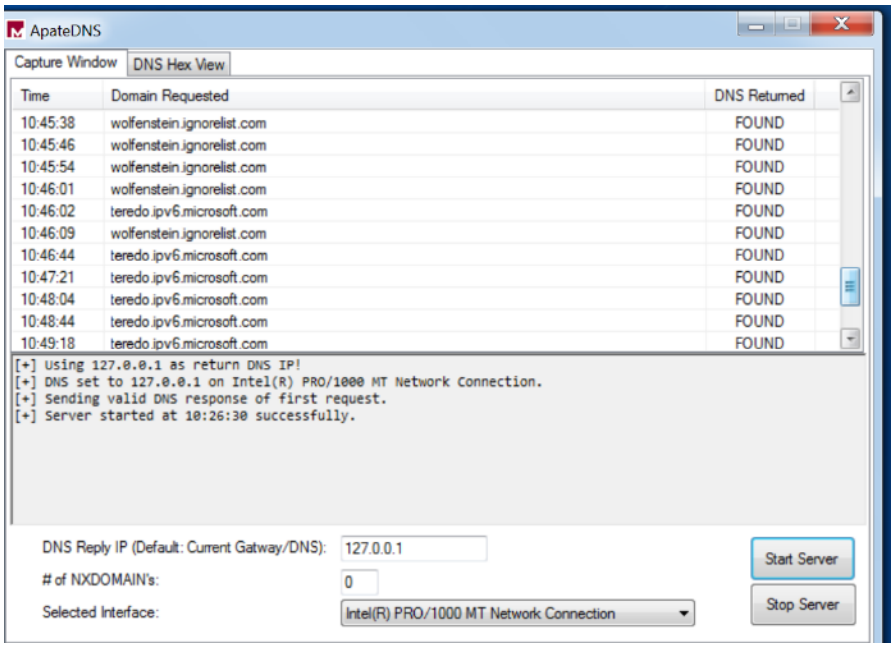
Results

Sample	Persistence	Files	Registry	Data	Communication
361B481084C41D0DF4C4EB763F268043	X	X	X		

Table 2 361B481084C41D0DF4C4EB763F268043 Results

Comments

Communication to Wolfenstein.ignorelist.com were not detected by MACT. Likely started by a spawned process or an API function that has not been intercepted.



Method of Persistence

Copies itself to the two files below:

C:\conhost\conhost.exe

C:\Users\MACT\AppData\Roaming\conhost\conhost.exe

Runs conhost via registry entries using RegSetValue API

Relevant Files Created, Modified, or Opened

.tlb files are indicative of WMI use. conhost.exe is a copy of the malware used for persistence.

C:\Windows\system32\stdole2.tlb

C:\Windows\system32\wbem\wbemdisp.TLB

C:\conhost\conhost.exe

Relevant Registry Elements Created, Modified, or Opened

The following registry entries appear to set up a copy of the malware, conhost.exe, for execution.

>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [policy.2.2.AForge.Video.DirectShow__61ea4348d43881b7]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegOpenKeyExW Key = [NI\7454be22\631aafc0]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [NI\20444198\25b15397]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [CLSID\{62BE5D10-60EB-11D0-BD3B-00A0C911CE86}\InprocServer32]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [Software\Microsoft\ActiveMovie\devenum]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]

>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [Software\Microsoft\ActiveMovie\devenum\{860BB310-5D01-11D0-BD3B-00A0C911CE86}]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegOpenKeyExW Key = [CLSID]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [{860BB310-5D01-11D0-BD3B-00A0C911CE86}\Instance]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegOpenKeyExW Key = [Control Panel\International]
>RegSetValueExW Key = [conhost]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegOpenKeyExW Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>RegSetValueExW Key = [conhost]

Type of Information Collected
Undetected.

Method of Communication

Undetected as only local network activity was documented. Partical communication artficats are listed below.

>inet_addr 127.0.0.1
>inet_addr 127.0.0.1
>inet_addr 0.0.0.0
>inet_addr 0.0.0.0
>inet_addr 0.0.0.0
>inet_addr 192.168.56.128
>inet_addr 255.255.255.0
>inet_addr 192.168.56.2

>inet_addr 192.168.56.254

Sample 2 - Trojan

MD5 5CB07299CEDD69F096B09358754831E0

Link

https://www.virustotal.com/#/file/9633246f366d63cbc70eb14b3c50d58de41ffce75ba7685d82c185ecfdda5686/detection

Classification Trojan

Comments RegSetValueEx was misspelled in log as RetSetValueEx. Fixed for future samples.

Overview

APINAME	APIP01	APIP02	APIP03	APIP04	APIP05	APIP06
Filter	Filter	Filter	Filter	Filter	Filter	Filter
GetVersionEx	12f914					
GetModuleHandle	User32.dll					
GetProcAddress	75DE0000	GetCursorInfo				
LdrLoadDll		0012FE48	User32.dll	0012FE38		
LoadLibrary	User32.dll					
GetProcAddress	75DE0000	GetLastInputInfo				
LdrLoadDll		0012FE48	kernel32.dll	0012FE38		
LoadLibrary	kernel32.dll					
GetProcAddress	76360000	GetConsoleWindow				
GetStartupInfoA	0012FF2C					
GetModuleHandle	NULL					
FindResourceExA	00000000		SETTINGS	00000000		
FindResourceA	00000000	SETTINGS				
LoadResource	00000000	0041A0C8				
OpenMutexA	00100000	0	Remcos_Mute...			
GetOpenKeyEx	80000001	Software\kiuyt-YWU9I4\	00000000	00020019	0012FBE8	
CreateMutex	00000000	00000001	kiuyt-YWU9I4			
LdrLoadDll		0012FB5C	Psapi.dll	0012FB4C		
LoadLibrary	Psapi.dll					
GetProcAddress	76440000	GetModuleFileNameExA				
LdrLoadDll		0012FB5C	Psapi.dll	0012FB4C		
LoadLibrary	Psapi.dll					
GetProcAddress	76440000	GetModuleFileNameExW				
LdrLoadDll		0012FB5C	kernel32.dll	0012FB4C		
LoadLibrary	kernel32.dll					
GetProcAddress	76360000	GlobalMemoryStatusEx				
GetModuleHandle	kernel32					
GetProcAddress	76360000	IsWow64Process				
GetModuleHandle	kernel32					
GetProcAddress	76360000	GetComputerNameExW				
GetModuleHandle	Shell32					
GetProcAddress	76820000	IsUserAnAdmin				
GetModuleHandle	kernel32					
GetProcAddress	76360000	SetProcessDEPPolicy				
GetOpenKeyEx	80000002	SOFTWARE\Microsoft\Windows NT\CurrentVersion	00000000	00020019	0012FBE8	
RegQueryValueEx	00000168	ProductName	00000000	00000000	0012F7D4	0012FBD4
IsUserAnAdmin	void					
RetSetValueEx	0000016C	exepath	00000000	00000003	01690C81	0000004C
RetSetValueEx	0000016C	lic	00000000	00000001	0039FEB9	00000020
CreateRemoteThre...	FFFFFFFF	00000000	00000000	00404619	00419830	00000000
CreateRemoteThre...	FFFFFFFF	00000000	00000000	004045F9	00419830	00000000

Figure 30 5CB07299CEDD69F096B09358754831E0 MACT.db API information.

- gets the handle of the executable
- loads the resource section named “SETTINGS”
- creates a mutex named kiuyt-YWU9I4, typically to ensure multiple copies are not running
- loads kernel32.dll and psapi.dll to get the proc addresses of a few functions to get system info
- attempts to use *SetDEPPolicy* to turn off DEP
- creates and sets some Windows registry values
- monitors the foreground windows for windows heading text and for field data
- saves data in logs.dat
- does DNS resolution for Prince.jumpingcrab.com
- transmits data in small bursts to the resolved IP, in this case 185.125.205.82 Port: 50695
- no apparent persistence method

Results

Sample	Persistence	Files	Registry	Data	Communication
5CB07299CEDD69F096B09358754831E0	X	X	X	X	X

Table 3 5CB07299CEDD69F096B09358754831E0 Results

Method of Persistence

Set itself up to execute via registry modifications.

Relevant Files Created or Modified

C:\Users\MACT\AppData\Roaming\kjuhyt\logs.dat

Relevant Registry Elements Created or Modified

>RegOpenKeyEx Key = [Software\kiuyt-YWU9I4\]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows NT\CurrentVersion]
>RegQueryValueEx Key = [ProductName]
>RegSetValueEx Key = [exepath]
>RegSetValueEx Key = [lic]

>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinSAT]

>RegQueryValueEx Key = [PrimaryAdapterString]

>RegOpenKeyEx Key = [Software\kiuyt-YWU9I4\]

>RegQueryValueEx Key = [name]

Setting a breakpoint on RegSetValueEx I can see that binary data is being stored to the registry. I viewed the data using the D M command and then saved it as an artifact using the M A command.

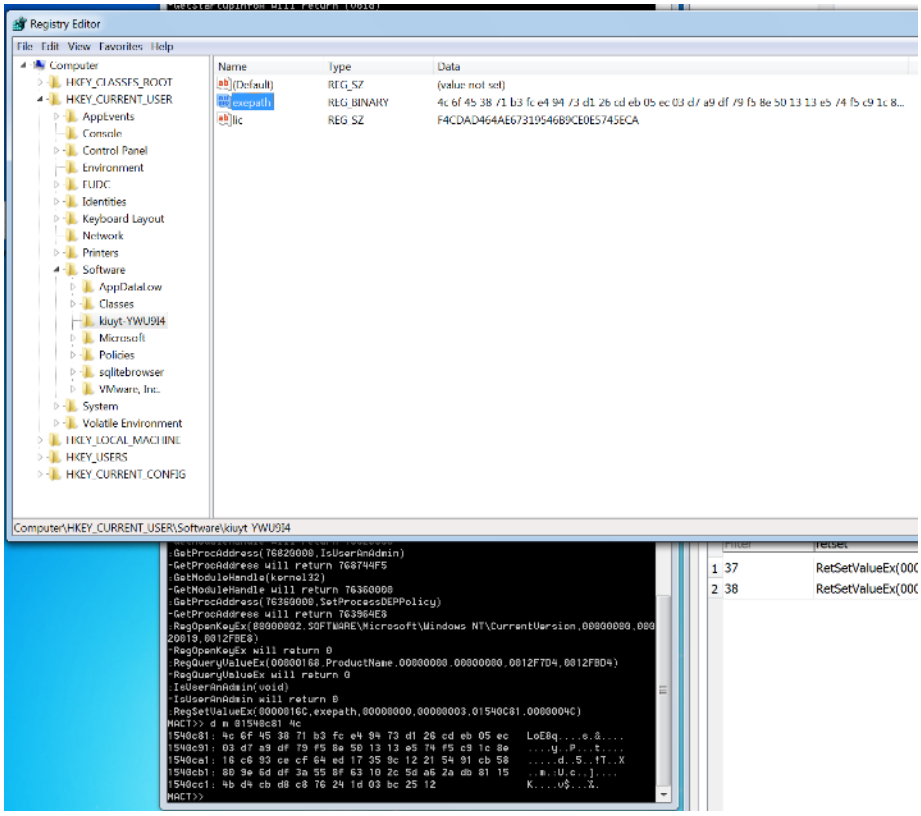


Figure 31 5CB07299CEDD69F096B09358754831E0 writing binary data to the registry.

Type of Information Collected

Captures form fields and window titles.

APINAME	APIP01	APIP02
Filter	Filter	Filter
GetWindowText	000B074E	Command Prompt - serverc
GetWindowText	000B074E	Command Prompt - serverc
GetWindowText	000B074E	Command Prompt - serverc
GetWindowText	000B074E	Command Prompt - serverc
GetWindowText	000B074E	Command Prompt - serverc
GetWindowText	000D03C8	MSN | Outlook
GetWindowText	000D03C8	MSN | Outlook
GetWindowText	000D03C8	MSN | Outlook
GetWindowText	000D03C8	MSN | Outlook
GetWindowText	000D03C8	MSN | Outlook
GetWindowText	000D03C8	Google - Internet Explorer
GetWindowText	000D03C8	Google - Internet Explorer
GetWindowText	000D03C8	Google - Internet Explorer
GetWindowText	000D03C8	Google - Internet Explorer
GetWindowText	000D03C8	Google - Internet Explorer
GetWindowText	000D03C8	dakota state university - Google Search - Internet Explorer
GetWindowText	000D03C8	dakota state university - Google Search - Internet Explorer
GetWindowText	000D03C8	dakota state university - Google Search - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	Dakota State University - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer
GetWindowText	000D03C8	mydsu.edu - Bing - Internet Explorer

Figure 32 5CB07299CEDD69F096B09358754831E0 capturing form data.

Method of Information Transference

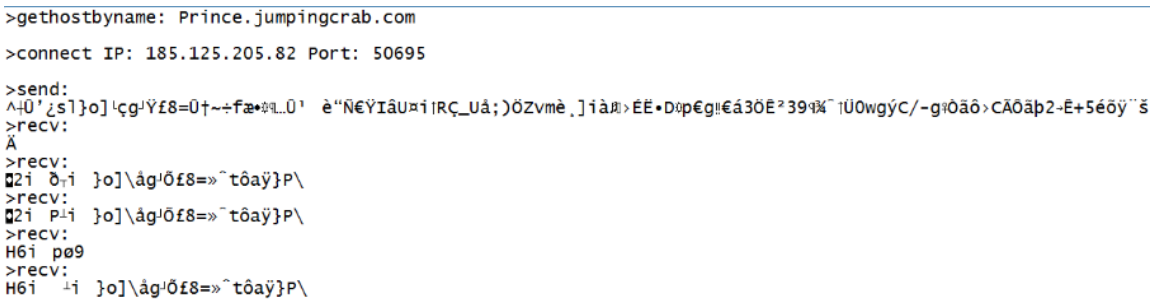


Figure 33 5CB07299CEDD69F096B09358754831E0 communication.

Sample 3 - PUA

MD5 F54D48B019436E28C1AF5BABF247748B

Link

https://www.virustotal.com/#/file/6d348cae79bbd4680f0ec0b59079828c64b5058d3226f604f880f79536be6f90/detection

Classification PUA

Comments Could not determine the type of data being looked at.

Overview

- A lot of the files and registry updates, as seen in prior samples, appear to be a result of the .net PE format.
- Attempts to gather some local network information.
- No memory allocations were captured.
- Appears to contact 191.101.22.139 which resolves to 139.22-101-191.adsl-dyn.isp.anmaxx.co.uk
- Seems to be cycleing through trying to communicate to 191.101.22.139. May not be the same host at the IP as the malware, at one time, expected to be communicating with. This may be preventing the malware from running further.

Results

Sample	Persistence	Files	Registry	Data	Communication
F54D48B019436E28C1AF5BABF247748B	X	X	X		X

Table 4 F54D48B019436E28C1AF5BABF247748B Results

Method of Persistence

The malware appears to open itself and copy to another file name. Cannot be sure by only looking at the API calls in the log. It seems to copy itself to C:\Folder Name\Monitor exe.exe although the file is empty in the captured artifact directory.

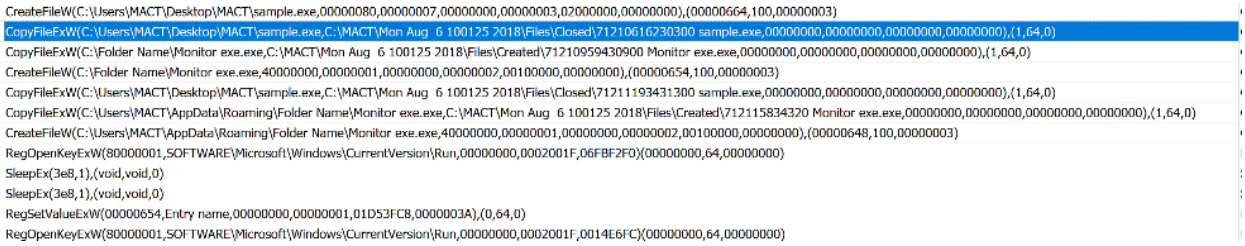


Figure 34 F54D48B019436E28C1AF5BABF247748B creating Monitor exe.exe

In an attempt to capture monitor exe.exe I set a break point on CreateFileW based on the log entry that shows which API created the file.

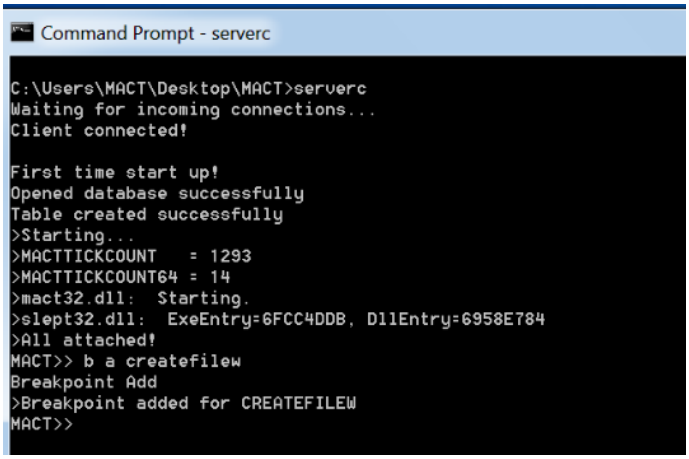


Figure 35 F54D48B019436E28C1AF5BABF247748B creating breakpoint on CreateFileW

```
:CopyFileExW(C:\Users\MACT\Desktop\MACT\sample.exe,C:\MACT\Fri Aug 17 130527 201
8\Files\Created\22345536203600 sample.exe,00000000,00000000,00000000,00000000)
-CopyFileExW will return 1
-CreateFileW will return 00000558
:CopyFileExW(C:\Users\MACT\Desktop\MACT\sample.exe,C:\MACT\Fri Aug 17 130527 201
8\Files\Closed\2234987311200 sample.exe,00000000,00000000,00000000,00000000)
-CopyFileExW will return 1
:SleepEx(0)
:CreateFileW(C:\Folder Name\Monitor exe.exe,40000000,00000001,00000000,00000002,
00100000,00000000)
>CreateFileW pszFilename = C:\MACT\Fri Aug 17 130527 2018\Files\Created\22350200
611800 Monitor exe.exe
MACT>>
```

Figure 36 F54D48B019436E28C1AF5BABF247748B Mact breaking on CreateFileW for Monitor exe.exe

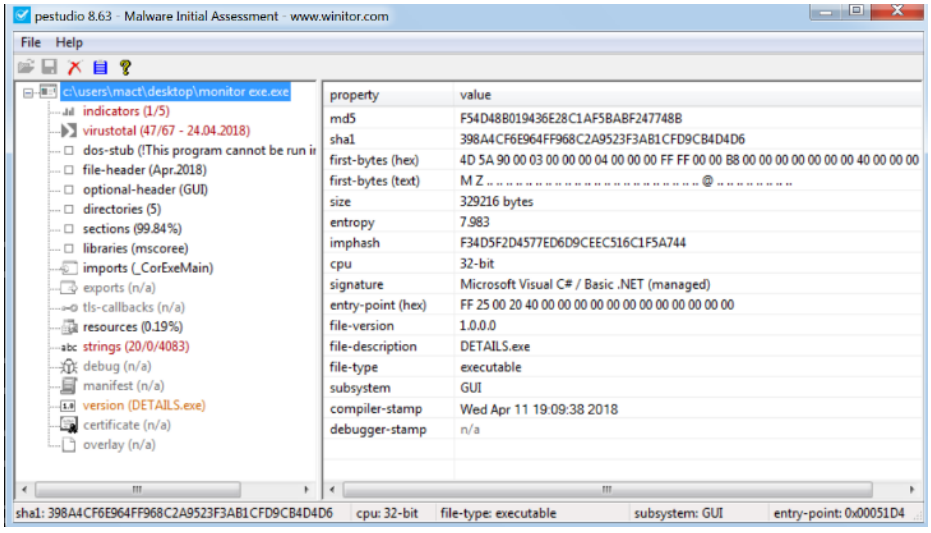


Figure 37 F54D48B019436E28C1AF5BABF247748B shows Monitor exe.exe has the same hash as the original sample.

Based on the above analysis it is confirmed that it persists by copying itself to Monitor exe.exe

Relevant Files Created, Modified, or Opened

Among the files created are copies of the malware, .tlb files which indicate WMI use

0169445681600 06-08-2018

02519816648800 06-08-2018

02538567881800 06-08-2018

02545494293900 06-08-2018

0537207771200 sample.exe

05382841730 sample.exe

0713381940100 sample.exe

01026666279600 wbemdisp.TLB

01549571246700 Monitor exe.exe

015508647600 Monitor exe.exe

0161114684600 Path.dat

0161379889300 Path.dat

01616434493900 Path.dat

0161910298600 Path.dat

01621769703300 Path.dat

016244841080 Path.dat

01625732110200 network.dat

01625732110200 system.dat

01627120512700 Path.dat

0224642705500 network.dat

0536131369300 sample.exe

0537207771200 sample.exe

05374970800 sample.exe

053881372700 sample.exe

Relevant Registry Elements Created, Modified, or Opened

[Software\Microsoft\.NETFramework\Security\Policy\Extensions\NamedPermissionSets]

>RegEnumKeyExW Subkey = []

>RegOpenKeyExW Key = [Internet]

>RegEnumKeyExW Subkey = []

>RegEnumKeyExW Subkey = [MediaPermission]

>RegEnumKeyExW Subkey = [WebBrowserPermission]

>RegEnumKeyExW Subkey = [Internet]

>RegOpenKeyExW Key = [LocalIntranet]

>RegEnumKeyExW Subkey = [WebBrowserPermission]

>RegEnumKeyExW Subkey = [MediaPermission]

>RegEnumKeyExW Subkey = [WebBrowserPermission]

>RegEnumKeyExW Subkey = [LocalIntranet]
>RegOpenKeyExW Key = [Software\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-3098272619-2655189810-4162345375-1000]
>RegOpenKeyExW Key = [Software\Microsoft\.NETFramework\v2.0.50727\Security\Policy]
1000\Installer\Assemblies\C:|Users|MACT|Desktop|MACT|sample.exe]
>RegOpenKeyExW Key =
[Software\Microsoft\Installer\Assemblies\C:|Users|MACT|Desktop|MACT|sample.exe]
>RegOpenKeyExW Key =
[SOFTWARE\Classes\Installer\Assemblies\C:|Users|MACT|Desktop|MACT|sample.exe]
>RegOpenKeyExW Key =
[SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Managed\S-1-5-21-3098272619-2655189810-4162345375-1000\Installer\Assemblies\Global]
>RegOpenKeyExW Key = [Software\Microsoft\Installer\Assemblies\Global]
>RegOpenKeyExW Key = [SOFTWARE\Classes\Installer\Assemblies\Global]
>RegOpenKeyExW Key = [NI\4e752a19\2541721]
>RegOpenKeyExW Key = [NI\5d608f43\663c74a9]
>RegOpenKeyExW Key = [policy.2.0.System__b77a5c561934e089]
>RegOpenKeyExW Key = [policy.2.0.System.Web__b03f5f7f11d50a3a]
>RegOpenKeyExW Key = [policy.2.0.System.Management__b03f5f7f11d50a3a]
>RegOpenKeyExW Key = [policy.2.0.System.Runtime.Remoting__b77a5c561934e089]
>RegSetValueExW Key = [PID]
>RegOpenKeyExW Key = [Software\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-3098272619-2655189810-4162345375-1000]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap]

>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [FEATURE_INITIALIZE_URLACTION_SHELLEXECUTE_TO_ALLOW_KB936610]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [FEATURE_ALLOW_REVERSE_SOLIDUS_IN_USERINFO_KB932562]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer]
>RegOpenKeyExW Key = [Microsoft\Internet Explorer\Security]
>RegOpenKeyExW Key = [Microsoft\Internet Explorer\Security]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [sample.exe]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [*]
>RegOpenKeyExW Key = [FEATURE_ZONES_DEFAULT_DRIVE_INTRANET_KB941000]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [sample.exe]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [*]

Type of Information Collected

Unknown. Anti-analysis seems to have shut down the analysis before this could be determined. Files captured do not appear to provide any definite insight.

Method of Communication

Appears to contact 191.101.22.139 which resolves to 139.22-101-191.adsl-dyn.isp.anmaxx.co.uk. Some samples artfacts are listed here.

>inet_addr 192.168.56.2
>inet_addr 192.168.56.254
>inet_addr 192.168.56.2
>inet_addr 192.168.56.2
>inet_addr 191.101.22.139
>inet_addr 191.101.22.139
>inet_addr 191.101.22.139

Sample 4 - Worm

MD5 830400121A557B0668AE9374CD847C8B

Link

https://www.virustotal.com/#/file/d09be55ca322e01976e795aba73e0da56a801d28089b3202d9cba0a80dce2dc6/detection

Classification Worm

Overview

- Uses FindFirstFile and FindNextFile to iterate through files looking for those of a specific type. It appears some of the files targeted are .txt, .asp, .htm, .html and those with no extension.

APINAME	APIP01	APIP02
find	Filter	Filter
FindNextFile	0019E410	0012F458
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$R560SPV\Mem*.*	0012EF68
FindNextFile	0019E610	0012EF68
FindNextFile	0019E610	0012EF68
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E410	0012F458
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$R5R6RX7*.*	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$R5R6RX7\Files*.*	0012EF68
FindNextFile	0019E610	0012EF68
FindNextFile	0019E610	0012EF68
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E410	0012F458
FindNextFile	0019E410	0012F458
FindNextFile	0019E410	0012F458
FindNextFile	0019E410	0012F458
FindNextFile	0019E410	0012F458
FindNextFile	0019E410	0012F458
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$REQNJ0M*.*	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindNextFile	0019E5D0	0012F1E0
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$REQNJ0M\Files*.*	0012EF68
FindNextFile	0019E610	0012EF68
FindNextFile	0019E610	0012EF68
FindFirstFile	C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$REQNJ0M\Files\Clos...	0012ECF0

Figure 38 830400121A557B0668AE9374CD847C8B MACT.db API calls.

- Downloads or creates .htm and .pl files which appear to be used for transmitting data via http.
- Makes numerous copies of itself.
- Scans emails, TO and FROM looking for email addresses possibly even specific email addresses.

APINAME	APIP01	APIP02
str	Filter	Filter
lstrcmpiA	john@adnxs.com	john@foxnews.com
lstrcmpiA	james@adnxs.com	john@foxnews.com
lstrcmpiA	sales@cc.hut.fi	john@foxnews.com
lstrcmpiA	james@broofa.com	john@foxnews.com
lstrcmpiA	xo@adnxs.com	john@foxnews.com
lstrcmpiA	john@edmweb.com	john@foxnews.com
lstrcmpiA	james@foxnews.com	john@foxnews.com
lstrcmpiA	john@broofa.com	john@foxnews.com
lstrcmpiA	sales@foxnews.com	john@foxnews.com
lstrcmpiA	john@foxnews.com	john@foxnews.com
lstrcmpiA	james@cc.hut.fi	james@cc.hut.fi
lstrcmpiA	james@cc.hut.fi	sales@adnxs.com
lstrcmpiA	sales@broofa.com	sales@adnxs.com
lstrcmpiA	sales@edmweb.com	sales@adnxs.com
lstrcmpiA	james@edmweb.com	sales@adnxs.com
lstrcmpiA	john@cc.hut.fi	sales@adnxs.com
lstrcmpiA	sales@adnxs.com	sales@adnxs.com
lstrcmpiA	james@cc.hut.fi	sales@broofa.com
lstrcmpiA	sales@broofa.com	sales@broofa.com
lstrcmpiA	james@cc.hut.fi	james@edmweb.com
lstrcmpiA	sales@broofa.com	james@edmweb.com
lstrcmpiA	sales@edmweb.com	james@edmweb.com
lstrcmpiA	james@edmweb.com	james@edmweb.com
lstrcmpiA	james@cc.hut.fi	sales@cc.hut.fi
lstrcmpiA	sales@broofa.com	sales@cc.hut.fi
lstrcmpiA	sales@edmweb.com	sales@cc.hut.fi
lstrcmpiA	james@edmweb.com	sales@cc.hut.fi

Figure 39830400121A557B0668AE9374CD847C8B searching for email addresses.

- Files with no extension and .pl seem to be programs and data used by the malware.
- Can download files, programs, sets you up a bot.

Results

Sample	Persistence	Files	Registry	Data	Communication
830400121A557B0668AE9374CD847C8B	X	X	X	X	X

Table 5 830400121A557B0668AE9374CD847C8B Results

Method of Persistence

Makes numerous copies of itself. Registry values set to the name of the malware binary.

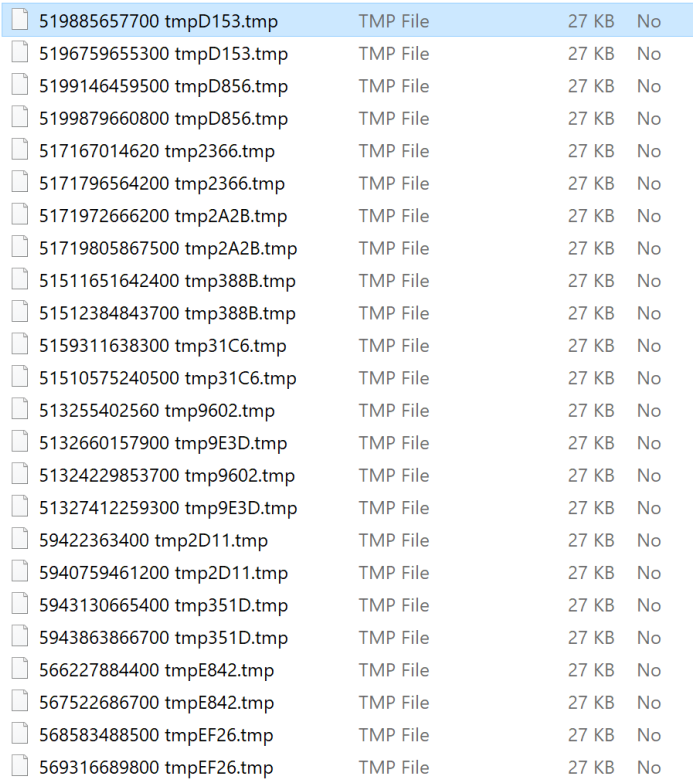


Figure 40 830400121A557B0668AE9374CD847C8B Persistence

Relevant Files Created, Modified, or Opened

The malware checks the recycle bin, creates several .html files, looks for files to compile programs, creates files in the temporary internet files directory, and makes copies of itself. Below are a few of the indicative file artifacts.

C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$I1YFEUL
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$I560SPV
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$I5R6RX7
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$IEQNJ0M
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$IS2DM5V
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$IVWHKU3
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$IZ9YU5T
C:\$Recycle.Bin\S-1-5-21-3098272619-2655189810-4162345375-1000\$IZVE4CT

C:\Program Files\Common Files\microsoft shared\Stationery\Bears.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Garden.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Green Bubbles.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Hand Prints.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Orange Circles.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Peacock.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Roses.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Shades of Blue.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Soft Blue.htm
C:\Program Files\Common Files\microsoft shared\Stationery\Stars.htm
C:\Program Files\Process Hacker 2\CHANGELOG.txt
C:\Program Files\Process Hacker 2\COPYRIGHT.txt
C:\Program Files\Process Hacker 2\LICENSE.txt
C:\Program Files\Process Hacker 2\README.txt
C:\Windows\SoftwareDistribution\Download\14d19c27b28cc3990260d7191f6e0ff6c7483623
C:\Windows\lsass.exe

Relevant Registry Elements Created, Modified, or Opened

Registry elements updated appear to mainly be associated with downloading files. Below is a partial listing of the captured registry artifacts.

>RegOpenKeyExW Key = [FEATURE_BUFFERBREAKING_818408]
>RegOpenKeyExW Key =
[FEATURE_SKIP_POST_RETRY_ON_INTERNETWRITEFILE_KB895954]
>RegOpenKeyExW Key =
[FEATURE_FIX_CHUNKED_PROXY_SCRIPT_DOWNLOAD_KB843289]
>RegOpenKeyExW Key = [FEATURE_USE_CNAME_FOR_SPN_KB911149]
>RegOpenKeyExW Key =
[FEATURE_PERMIT_CACHE_FOR_AUTHENTICATED_FTP_KB910274]
>RegOpenKeyExW Key =
[FEATURE_DISABLE_UNICODE_HANDLE_CLOSING_CALLBACK]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [sample.exe]

Type of Information Collected

Files of with an extension of .txt, .asp, .htm, .html, those with no extension and possibly others are targeted.

E-Mail addresses are targeted. Possibly spamming.

lstrcmpiA	adsales@foxnews.com	newsmanager@foxnews.com
lstrcmpiA	adsales@foxnews.com	robert@broofa.com
lstrcmpiA	adsales@foxnews.com	john@foxnews.com
lstrcmpiA	adsales@foxnews.com	sales@foxnews.com
lstrcmpiA	adsales@foxnews.com	john@broofa.com
lstrcmpiA	adsales@foxnews.com	james@foxnews.com
lstrcmpiA	adsales@foxnews.com	john@edmweb.com
lstrcmpiA	adsales@foxnews.com	james@broofa.com
lstrcmpiA	adsales@foxnews.com	sales@cc.hut.fi
lstrcmpiA	adsales@foxnews.com	james@adnxs.com
lstrcmpiA	adsales@foxnews.com	john@adnxs.com
lstrcmpiA	adsales@foxnews.com	sales@adnxs.com
lstrcmpiA	adsales@foxnews.com	john@cc.hut.fi
lstrcmpiA	adsales@foxnews.com	james@edmweb.com
lstrcmpiA	adsales@foxnews.com	sales@edmweb.com
lstrcmpiA	adsales@foxnews.com	sales@broofa.com
lstrcmpiA	adsales@foxnews.com	james@cc.hut.fi
lstrcmpiA	broofa.com	broofa.com
lstrcmpiA	cc.hut.fi	cc.hut.fi
lstrcmpiA	edmweb.com	edmweb.com
lstrcmpiA	foxnews.com	foxnews.com
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To
lstrcmpiA	from	From
lstrcmpiA	from	To

Figure 41 830400121A557B0668AE9374CD847C8B possible email activity.

Method of Communication

Creates and uses .htm files to send/receive data.

Figure 42830400121A557B0668AE9374CD847C8B downloaded .htm file.

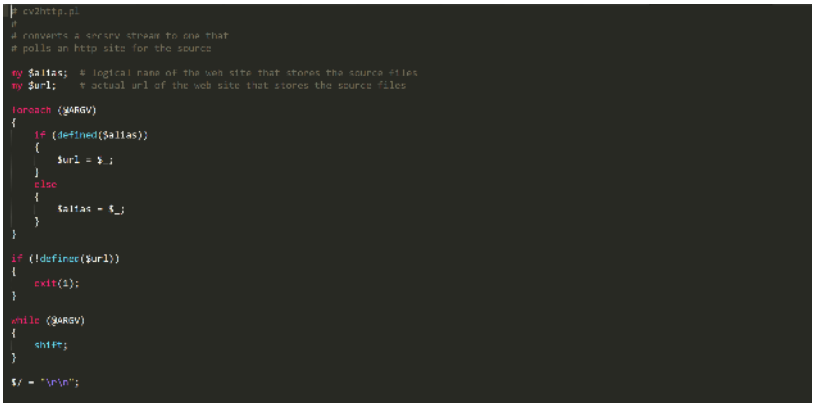


Figure 43 830400121A557B0668AE9374CD847C8B downloaded progam script.

Bootstrap[1].htm seems particularly interesting

```
function newTrackingClient(params) {
    var baseURL = params.baseURL;

    return {
        submitEvent: function submitEvent(event) {
            var body = {
                isJsClient: true
            };

            copyProperties(body, event, [
                "isJsClient", // allows adapter to handle JS and no-JS events uniformly
                "ae",
                "eventId",
                "ip",
                "referer",
                "postalCode",
                "debug"
            ]);

            var collection = event.type + "s";
            var url = (
                baseURL + "/campaigns/" +
                encodeURIComponent(event.serviceId) + "/" +
                encodeURIComponent(event.campaignId) + "/variations/" +
                encodeURIComponent(event.variation) + "/" +
                collection
                );
```

Figure 44 830400121A557B0668AE9374CD847C8B Bootstrap[1].htm

Partial listing of intercepted communication:

>bind IP: 0.0.0.0 Port: 4612

>accept

>connect IP: 15.252.62.106 Port: 4612

>connect IP: 165.247.171.187 Port: 4612

>connect IP: 160.10.42.175 Port: 4612

>connect IP: 172.182.139.16 Port: 4612

>connect IP: 16.101.235.73 Port: 4612

>connect IP: 15.8.159.59 Port: 4612

>connect IP: 15.228.161.69 Port: 4612

>connect IP: 16.115.193.45 Port: 4612

>connect IP: 68.33.57.248 Port: 4612

>connect IP: 192.168.100.70 Port: 4612

>connect IP: 10.120.49.55 Port: 4612

>connect IP: 67.108.192.18 Port: 4612

>connect IP: 141.240.121.8 Port: 4612

>connect IP: 68.251.64.233 Port: 4612

>connect IP: 10.16.104.62 Port: 4612

>connect IP: 15.7.212.82 Port: 4612
>connect IP: 164.101.72.91 Port: 4612
>connect IP: 166.77.228.111 Port: 4612
>connect IP: 216.86.207.198 Port: 4612
>connect IP: 69.149.0.92 Port: 4612
>inet_addr 192.168.56.2
>connect IP: 4.241.24.72 Port: 4612
>inet_addr mx4.org.aalto.fi
>gethostbyname: mx4.org.aalto.fi
>connect IP: 130.233.0.135 Port: 6400
>inet_addr edmweb.com
>gethostbyname: edmweb.com
>connect IP: 207.148.248.143 Port: 6400
>connect IP: 207.59.54.11 Port: 4612
>inet_addr mx1.org.aalto.fi

Sample 5 - Ransomware

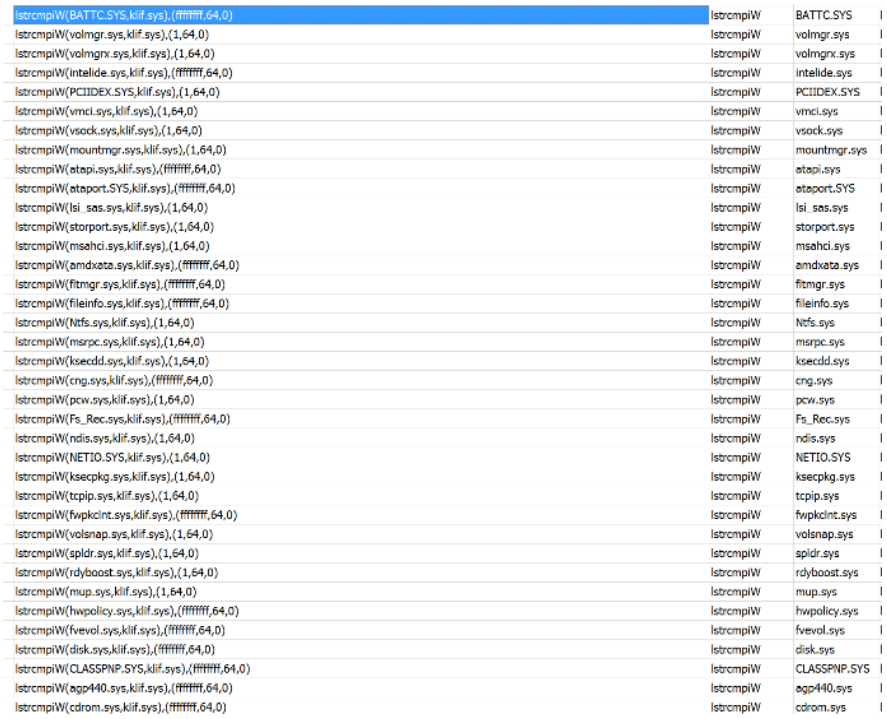
MD5 2F54B592E62E66FBFE7F7DB24F70F36A

Link
https://www.virustotal.com/#/file/a80573ce58cb7cb444d5daba578a529541b67ae407d72b826b57ba0c59ee9029/detection

Classification Ransomware

Comments Used parameter substitution for shellexecute structure parameters.

Overview
Compares drivers to antivirus drivers such as klif.sys and kl1.sys from Kaspersky, fsdfw.sys from F-Secure Internet Shield by F-Secure, and srtsp.sys from Symantic.



lstrcmpiW(BATTC.SYS,klif.sys),(ffffffff,64,0)	lstrcmpiW	BATTC.SYS
lstrcmpiW(volmgr.sys,klif.sys),(1,64,0)	lstrcmpiW	volmgr.sys
lstrcmpiW(volmgrx.sys,klif.sys),(1,64,0)	lstrcmpiW	volmgrx.sys
lstrcmpiW(intelide.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	intelide.sys
lstrcmpiW(PCIIDEX.SYS,klif.sys),(1,64,0)	lstrcmpiW	PCIIDEX.SYS
lstrcmpiW(vmci.sys,klif.sys),(1,64,0)	lstrcmpiW	vmci.sys
lstrcmpiW(vsock.sys,klif.sys),(1,64,0)	lstrcmpiW	vsock.sys
lstrcmpiW(mountmgr.sys,klif.sys),(1,64,0)	lstrcmpiW	mountmgr.sys
lstrcmpiW(atapi.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	atapi.sys
lstrcmpiW(ataport.SYS,klif.sys),(ffffffff,64,0)	lstrcmpiW	ataport.SYS
lstrcmpiW(lsi_sas.sys,klif.sys),(1,64,0)	lstrcmpiW	lsi_sas.sys
lstrcmpiW(storport.sys,klif.sys),(1,64,0)	lstrcmpiW	storport.sys
lstrcmpiW(msahci.sys,klif.sys),(1,64,0)	lstrcmpiW	msahci.sys
lstrcmpiW(amdxata.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	amdxata.sys
lstrcmpiW(fltmgr.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	fltmgr.sys
lstrcmpiW(fileinfo.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	fileinfo.sys
lstrcmpiW(Ntfs.sys,klif.sys),(1,64,0)	lstrcmpiW	Ntfs.sys
lstrcmpiW(msrpc.sys,klif.sys),(1,64,0)	lstrcmpiW	msrpc.sys
lstrcmpiW(ksecdd.sys,klif.sys),(1,64,0)	lstrcmpiW	ksecdd.sys
lstrcmpiW(cng.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	cng.sys
lstrcmpiW(pcw.sys,klif.sys),(1,64,0)	lstrcmpiW	pcw.sys
lstrcmpiW(Fs_Rec.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	Fs_Rec.sys
lstrcmpiW(ndis.sys,klif.sys),(1,64,0)	lstrcmpiW	ndis.sys
lstrcmpiW(NETIO.SYS,klif.sys),(1,64,0)	lstrcmpiW	NETIO.SYS
lstrcmpiW(ksecpkg.sys,klif.sys),(1,64,0)	lstrcmpiW	ksecpkg.sys
lstrcmpiW(tcpip.sys,klif.sys),(1,64,0)	lstrcmpiW	tcpip.sys
lstrcmpiW(fwpkclnt.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	fwpkclnt.sys
lstrcmpiW(volsnap.sys,klif.sys),(1,64,0)	lstrcmpiW	volsnap.sys
lstrcmpiW(spldr.sys,klif.sys),(1,64,0)	lstrcmpiW	spldr.sys
lstrcmpiW(rdyboost.sys,klif.sys),(1,64,0)	lstrcmpiW	rdyboost.sys
lstrcmpiW(mup.sys,klif.sys),(1,64,0)	lstrcmpiW	mup.sys
lstrcmpiW(hwpolicy.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	hwpolicy.sys
lstrcmpiW(fvevol.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	fvevol.sys
lstrcmpiW(disk.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	disk.sys
lstrcmpiW(CLASSPNP.SYS,klif.sys),(ffffffff,64,0)	lstrcmpiW	CLASSPNP.SYS
lstrcmpiW(agp440.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	agp440.sys
lstrcmpiW(cdrom.sys,klif.sys),(ffffffff,64,0)	lstrcmpiW	cdrom.sys

Figure 45 Sample looking for antivirus drivers to terminate.

Looks for and kills the following 40 tasks

Process32FirstW(0000016C,000F0000)(1,64,0)	Process32Firs...	0000016C	000F0000
lstrcmpiW(msftesql.exe,[System Process]),(1,64,0)	lstrcmpiW	msftesql.exe	[System Proc...
lstrcmpiW(sqlagent.exe,[System Process]),(1,64,0)	lstrcmpiW	sqlagent.exe	[System Proc...
lstrcmpiW(sqlbrowser.exe,[System Process]),(1,64,0)	lstrcmpiW	sqlbrowser.exe	[System Proc...
lstrcmpiW(sqlservr.exe,[System Process]),(1,64,0)	lstrcmpiW	sqlservr.exe	[System Proc...
lstrcmpiW(sqlwriter.exe,[System Process]),(1,64,0)	lstrcmpiW	sqlwriter.exe	[System Proc...
lstrcmpiW(oracle.exe,[System Process]),(1,64,0)	lstrcmpiW	oracle.exe	[System Proc...
lstrcmpiW(ocssd.exe,[System Process]),(1,64,0)	lstrcmpiW	ocssd.exe	[System Proc...
lstrcmpiW(dbsnmp.exe,[System Process]),(1,64,0)	lstrcmpiW	dbsnmp.exe	[System Proc...
lstrcmpiW(synctime.exe,[System Process]),(1,64,0)	lstrcmpiW	synctime.exe	[System Proc...
lstrcmpiW(mydesktopqos.exe,[System Process]),(1,64,0)	lstrcmpiW	mydesktopqos.exe	[System Proc...
lstrcmpiW(agntsvc.exeisqlplussvc.exe,[System Process]),(1,64,0)	lstrcmpiW	agntsvc.exeisqlplussvc.exe	[System Proc...
lstrcmpiW(xfssvccon.exe,[System Process]),(1,64,0)	lstrcmpiW	xfssvccon.exe	[System Proc...
lstrcmpiW(mydesktopservice.exe,[System Process]),(1,64,0)	lstrcmpiW	mydesktopservice.exe	[System Proc...
lstrcmpiW(ocautoupds.exe,[System Process]),(1,64,0)	lstrcmpiW	ocautoupds.exe	[System Proc...
lstrcmpiW(agntsvc.exeagntsvc.exe,[System Process]),(1,64,0)	lstrcmpiW	agntsvc.exeagntsvc.exe	[System Proc...
lstrcmpiW(agntsvc.exeencsvc.exe,[System Process]),(1,64,0)	lstrcmpiW	agntsvc.exeencsvc.exe	[System Proc...
lstrcmpiW(firefoxconfig.exe,[System Process]),(1,64,0)	lstrcmpiW	firefoxconfig.exe	[System Proc...
lstrcmpiW(tbirdconfig.exe,[System Process]),(1,64,0)	lstrcmpiW	tbirdconfig.exe	[System Proc...
lstrcmpiW(ocomm.exe,[System Process]),(1,64,0)	lstrcmpiW	ocomm.exe	[System Proc...
lstrcmpiW(mysqld.exe,[System Process]),(1,64,0)	lstrcmpiW	mysqld.exe	[System Proc...
lstrcmpiW(mysqld-nt.exe,[System Process]),(1,64,0)	lstrcmpiW	mysqld-nt.exe	[System Proc...
lstrcmpiW(mysqld-opt.exe,[System Process]),(1,64,0)	lstrcmpiW	mysqld-opt.exe	[System Proc...
lstrcmpiW(dbeng50.exe,[System Process]),(1,64,0)	lstrcmpiW	dbeng50.exe	[System Proc...
lstrcmpiW(sqbcoreservice.exe,[System Process]),(1,64,0)	lstrcmpiW	sqbcoreservice.exe	[System Proc...
lstrcmpiW(excel.exe,[System Process]),(1,64,0)	lstrcmpiW	excel.exe	[System Proc...
lstrcmpiW(infopath.exe,[System Process]),(1,64,0)	lstrcmpiW	infopath.exe	[System Proc...
lstrcmpiW(msaccess.exe,[System Process]),(1,64,0)	lstrcmpiW	msaccess.exe	[System Proc...
lstrcmpiW(mspub.exe,[System Process]),(1,64,0)	lstrcmpiW	mspub.exe	[System Proc...
lstrcmpiW(onenote.exe,[System Process]),(1,64,0)	lstrcmpiW	onenote.exe	[System Proc...
lstrcmpiW(outlook.exe,[System Process]),(1,64,0)	lstrcmpiW	outlook.exe	[System Proc...
lstrcmpiW(powerpnt.exe,[System Process]),(1,64,0)	lstrcmpiW	powerpnt.exe	[System Proc...
lstrcmpiW(steam.exe,[System Process]),(1,64,0)	lstrcmpiW	steam.exe	[System Proc...
lstrcmpiW(sqlservr.exe,[System Process]),(1,64,0)	lstrcmpiW	sqlservr.exe	[System Proc...
lstrcmpiW(thebat.exe,[System Process]),(1,64,0)	lstrcmpiW	thebat.exe	[System Proc...
lstrcmpiW(thebat64.exe,[System Process]),(1,64,0)	lstrcmpiW	thebat64.exe	[System Proc...
lstrcmpiW(thunderbird.exe,[System Process]),(1,64,0)	lstrcmpiW	thunderbird.exe	[System Proc...
lstrcmpiW(visio.exe,[System Process]),(1,64,0)	lstrcmpiW	visio.exe	[System Proc...
lstrcmpiW(winword.exe,[System Process]),(1,64,0)	lstrcmpiW	winword.exe	[System Proc...
lstrcmpiW(wordpad.exe,[System Process]),(1,64,0)	lstrcmpiW	wordpad.exe	[System Proc...

Figure 46 Sample terminates these tasks.

Launches WMIC

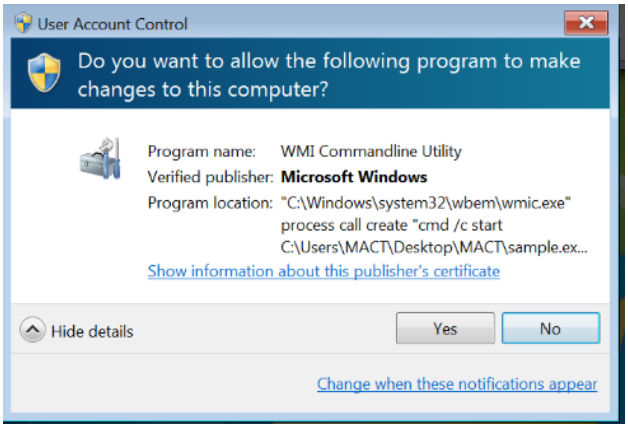


Figure 47 WMIC Launching With Parameters

```
*GetForegroundWindow(),(000B074E,64,00000000)>>DEBUG Function: MyShellExecuteExW
>ShellExecuteExW lpVerb       = runas
>ShellExecuteExW lpFile       = C:\Windows\system32\wbem\wmic
>ShellExecuteExW lpParameters = process call create "cmd /c start C:\Users\MACT\Desktop\MACT\sample.exe"
>ShellExecuteExW lpDirectory  = NULL
:ShellExecuteExW(000F0000)
```

Figure 48 serverlog.txt showing WMIC parameters.

Overrode WMDI command to inject mact32.dll and start stage 2. Started another server on a different port for stage 2 to communicate with.

lpParameters element of the SHELLEXECUTEINFOW structure was modified.

```
BOOL WINAPI MyShellExecuteExW(SHELLEXECUTEINFOW *a0)
{
    if(MACTDEBUG2)
        MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueShellExecuteExW(a0);

    typedef struct _MACTSHELLEXECUTEINFOW {
      DWORD     cbSize;
      ULONG     fMask;
      HWND      hwnd;
      LPCWSTR   lpVerb;
      LPCWSTR   lpFile;
      LPCWSTR   lpParameters;
      LPCWSTR   lpDirectory;
      int       nShow;
      HINSTANCE hInstApp;
      void      *lpIDList;
      LPCWSTR   lpClass;
      HKEY      hkeyClass;
      DWORD     dwHotKey;
      union {
        HANDLE hIcon;
        HANDLE hMonitor;
      } DUMMYUNIONNAME;
      HANDLE    hProcess;
    } MACTSHELLEXECUTEINFOW, *LPMACTSHELLEXECUTEINFOW;
```

Figure 49 ShellExecute showing SHELLEXECUTEINFOW structure.

```
>ShellExecuteExW lpFile       = [C:\Windows\system32\wbem\wmic]
>ShellExecuteExW lpParameters = [process call create "cmd /c start C:\Users\MACT\Desktop\MACT\sample.exe"]
>ShellExecuteExW lpDirectory  = [NULL]
>ShellExecuteExW lpParameters = [process call create "cmd /c start C:\Users\MACT\Desktop\MACT\withdll.exe -d:C:\Users\MACT\Desktop\MACT\mact32.dll C:\Users\MACT\Desktop\MACT\sample.exe"]
:ShellExecuteExW(00180000)
```

Figure 50 Parm substitution of parameters in SHELLEXECUTEINFOW structure as showin in serverlog.txt

Stage two executes and creates two new files buaqzz.exe and nvhbwt.exe. Not found on virustotal.

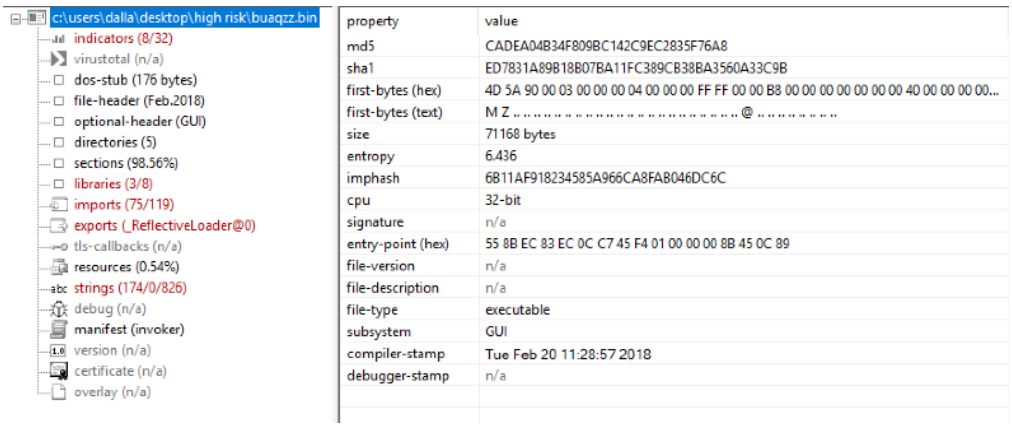


Figure 51 2F54B592E62E66FBFE7F7DB24F70F36A Phase 2 Payload

Appears to vary from the original only slightly with differences in positions 53-58.

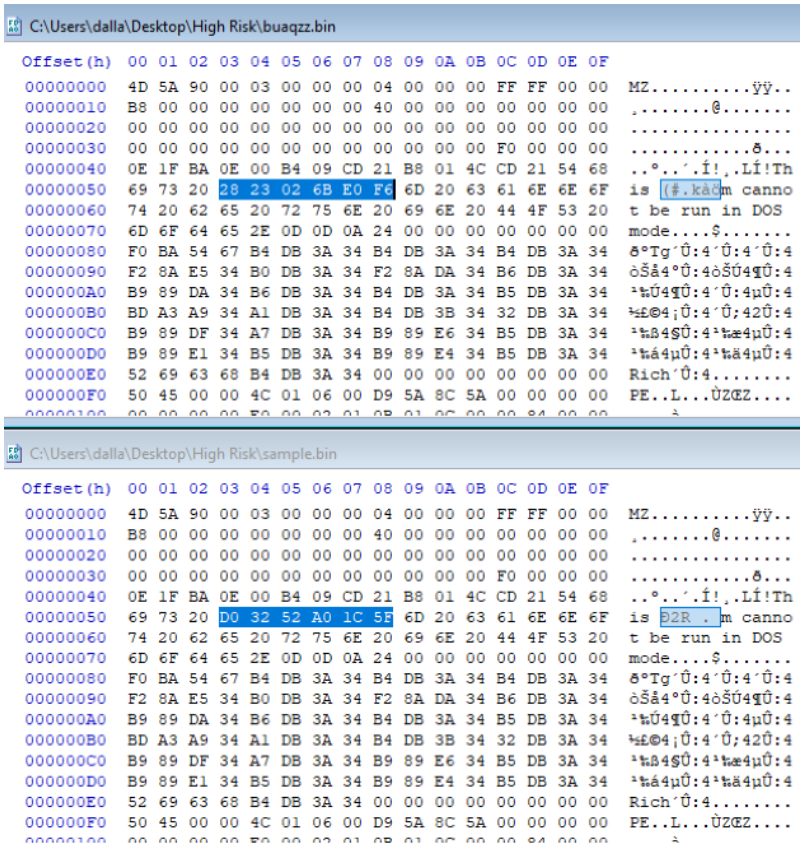


Figure 52 2F54B592E62E66FBFE7F7DB24F70F36A Differences between orignal sample and created binary.

Stage 2 then looks for the processes to kill again and stops. Cannot determine, with MACT, what the next stage is or if ani-analysis has ended the execution.

Results

Sample	Persistence	Files	Registry	Data	Communication
2F54B592E62E66FBFE7F7DB24F70F36A	X	X	X		

Table 6 2F54B592E62E66FBFE7F7DB24F70F36A Results

Method of Persistence

Sets itself up to run via registry modifications.

Files Created, Modified, or Opened

Makes sure WMIC is there.

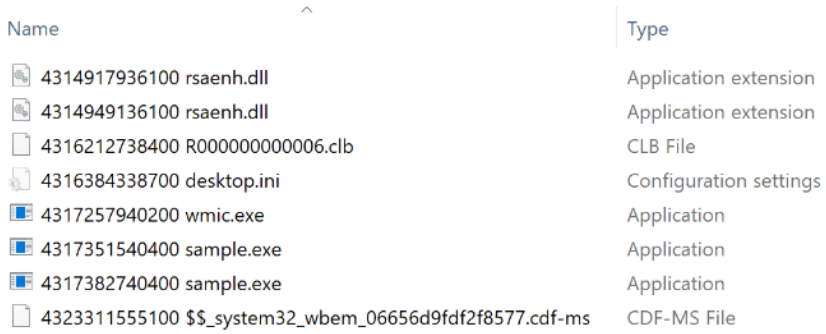


Figure 53 Sample 2F54B592E62E66FBFE7F7DB24F70F36A Phase 1 Files

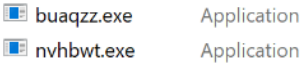


Figure 54 Sample 2F54B592E62E66FBFE7F7DB24F70F36A Phase 2 Files

Registry Elements Created, Modified, or Opened

Phase 1:

>RegOpenKeyExW Key = [SYSTEM\CurrentControlSet\services\Tcpip\Parameters]

>RegOpenKeyExW Key = [HARDWARE\DESCRIPTION\System\CentralProcessor\0]

>RegOpenKeyExW Key = [HARDWARE\DESCRIPTION\System\CentralProcessor\0]

>RegOpenKeyExW Key = [Software\Microsoft\Windows\CurrentVersion\Explorer\KindMap]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [.exe]
>RegGetValueW SubKey = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegGetValueW Value = [Security_HKLM_only]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main\FeatureControl]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main\FeatureControl]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main\FeatureControl]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main\FeatureControl]
>RegOpenKeyExW Key = [FEATURE_IGNORE_POLICIES_ZONEMAP_IF_ESC_ENABLED_KB918915]
>RegOpenKeyExW Key = [FEATURE_ZONES_CHECK_ZONEMAP_POLICY_KB941001]
>RegOpenKeyExW Key = [Software\Policies]
>RegOpenKeyExW Key = [Software\Policies]
>RegOpenKeyExW Key = [Software]
>RegOpenKeyExW Key = [Software]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [Software\Policies\Microsoft\Internet Explorer\Main]
>RegOpenKeyExW Key = [FEATURE_INITIALIZE_URLACTION_SHELLEXECUTE_TO_ALLOW_KB936610]

Phase 2:

>RegOpenKeyExW Key = [SYSTEM\CurrentControlSet\services\Tcpip\Parameters]

>RegOpenKeyExW Key = [HARDWARE\DESCRIPTION\System\CentralProcessor\0]

>RegOpenKeyExW Key = [HARDWARE\DESCRIPTION\System\CentralProcessor\0]

>RegSetValueExW Key = [itgsgjwlrsp]

Type of Information Collected

Unknown as Stage 2 stops without any extra info

Method of Communication

Unsure of the communication method.

Sample 6 - PUA

MD5 78B661723E5A4DE6446EE585A030E5C1

Link

https://www.virustotal.com/#/file/65217fdf8776433eb31a4572b0e094394916cfeb187167ae2a8eecf2e5eda6bb/detection

Classification PUA

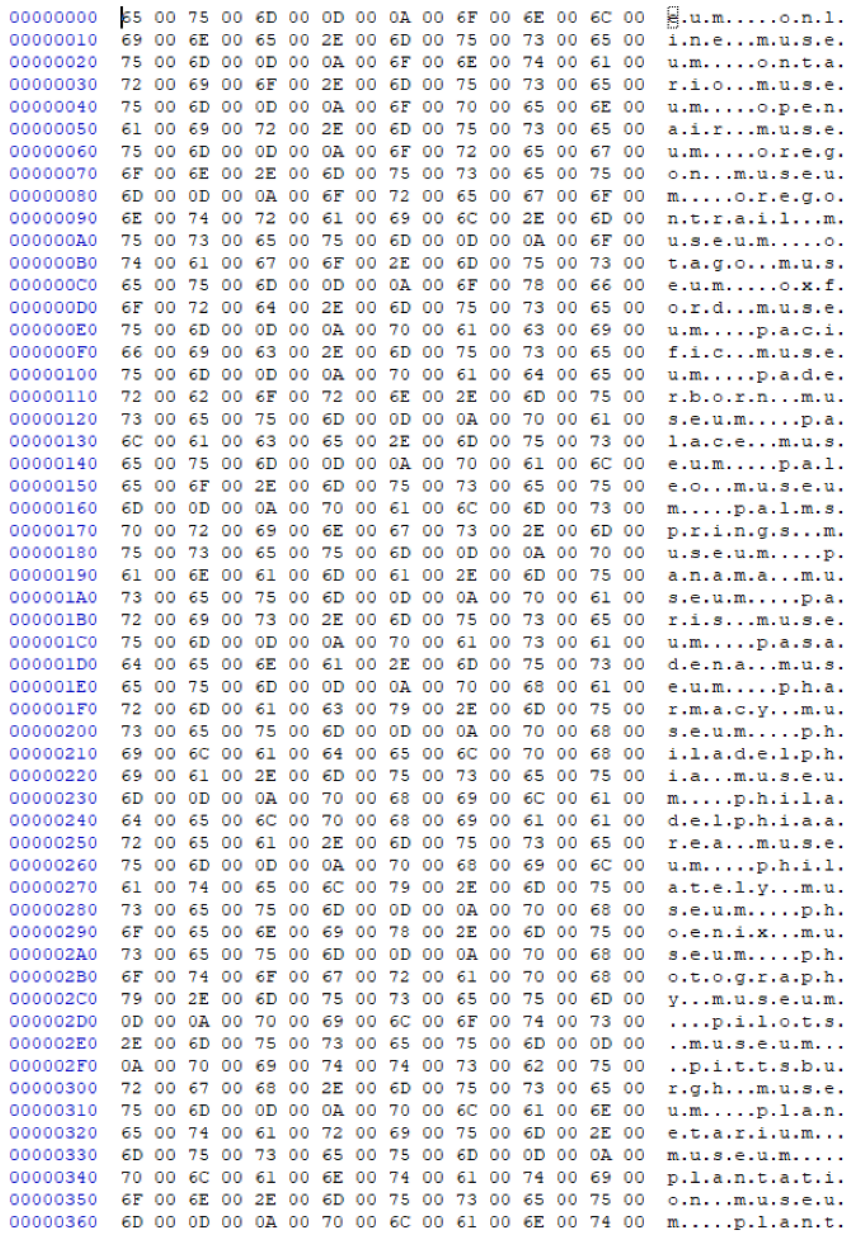
Overview

3,902 memory artifacts were created.

Examining the memory artifacts, the malware appears to have some specific DNS addresses from multiple nationalities as well as the ability to generate new ones.

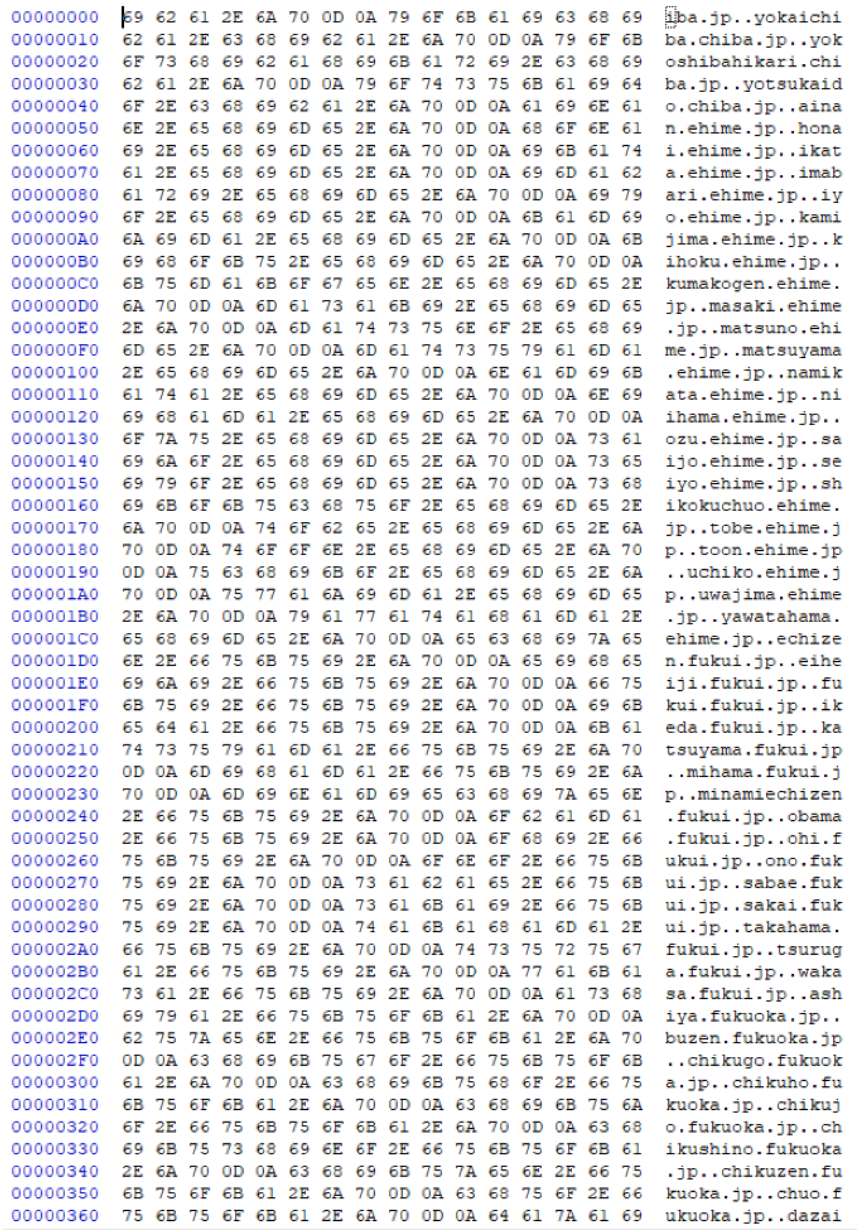
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000AB0   74 69 63 65 2E 66 72 0D 0A 6D 65 64 65 63 69 6E  tice.fr..medecin
00000AC0   2E 66 72 0D 0A 6E 6F 74 61 69 72 65 73 2E 66 72  .fr..notaires.fr
00000AD0   0D 0A 70 68 61 72 6D 61 63 69 65 6E 2E 66 72 0D  ..pharmacien.fr.
00000AE0   0A 70 6F 72 74 2E 66 72 0D 0A 76 65 74 65 72 69  .port.fr..veteri
00000AF0   6E 61 69 72 65 2E 66 72 0D 0A 67 61 0D 0A 67 62  naire.fr..ga..gb
00000B00   0D 0A 67 64 0D 0A 67 65 0D 0A 63 6F 6D 2E 67 65  ..gd..ge..com.ge
00000B10   0D 0A 65 64 75 2E 67 65 0D 0A 67 6F 76 2E 67 65  ..edu.ge..gov.ge
00000B20   0D 0A 6F 72 67 2E 67 65 0D 0A 6D 69 6C 2E 67 65  ..org.ge..mil.ge
00000B30   0D 0A 6E 65 74 2E 67 65 0D 0A 70 76 74 2E 67 65  ..net.ge..pvt.ge
00000B40   0D 0A 67 66 0D 0A 67 67 0D 0A 63 6F 2E 67 67 0D  ..gf..gg..co.gg.
00000B50   0A 6E 65 74 2E 67 67 0D 0A 6F 72 67 2E 67 67 0D  .net.gg..org.gg.
00000B60   0A 67 68 0D 0A 63 6F 6D 2E 67 68 0D 0A 65 64 75  .gh..com.gh..edu
00000B70   2E 67 68 0D 0A 67 6F 76 2E 67 68 0D 0A 6F 72 67  .gh..gov.gh..org
00000B80   2E 67 68 0D 0A 6D 69 6C 2E 67 68 0D 0A 67 69 0D  .gh..mil.gh..gi.
00000B90   0A 63 6F 6D 2E 67 69 0D 0A 6C 74 64 2E 67 69 0D  .com.gi..ltd.gi.
00000BA0   0A 67 6F 76 2E 67 69 0D 0A 6D 6F 64 2E 67 69 0D  .gov.gi..mod.gi.
00000BB0   0A 65 64 75 2E 67 69 0D 0A 6F 72 67 2E 67 69 0D  .edu.gi..org.gi.
00000BC0   0A 67 6C 0D 0A 67 6D 0D 0A 67 6E 0D 0A 61 63 2E  .gl..gm..gn..ac.
00000BD0   67 6E 0D 0A 63 6F 6D 2E 67 6E 0D 0A 65 64 75 2E  gn..com.gn..edu.
00000BE0   67 6E 0D 0A 67 6F 76 2E 67 6E 0D 0A 6F 72 67 2E  gn..gov.gn..org.
00000BF0   67 6E 0D 0A 6E 65 74 2E 67 6E 0D 0A 67 6F 76 0D  gn..net.gn..gov.
00000C00   0A 67 70 0D 0A 63 6F 6D 2E 67 70 0D 0A 6E 65 74  .gp..com.gp..net
00000C10   2E 67 70 0D 0A 6D 6F 62 69 2E 67 70 0D 0A 65 64  .gp..mobi.gp..ed
00000C20   75 2E 67 70 0D 0A 6F 72 67 2E 67 70 0D 0A 61 73  u.gp..org.gp..as
00000C30   73 6F 2E 67 70 0D 0A 67 71 0D 0A 67 72 0D 0A 63  so.gp..gq..gr..c
00000C40   6F 6D 2E 67 72 0D 0A 65 64 75 2E 67 72 0D 0A 6E  om.gr..edu.gr..n
00000C50   65 74 2E 67 72 0D 0A 6F 72 67 2E 67 72 0D 0A 67  et.gr..org.gr..g
00000C60   6F 76 2E 67 72 0D 0A 67 73 0D 0A 67 74 0D 0A 63  ov.gr..gs..gt..c
00000C70   6F 6D 2E 67 74 0D 0A 65 64 75 2E 67 74 0D 0A 67  om.gt..edu.gt..g
00000C80   6F 62 2E 67 74 0D 0A 69 6E 64 2E 67 74 0D 0A 6D  ob.gt..ind.gt..m
00000C90   69 6C 2E 67 74 0D 0A 6E 65 74 2E 67 74 0D 0A 6F  il.gt..net.gt..o
00000CA0   72 67 2E 67 74 0D 0A 2A 2E 67 75 0D 0A 67 77 0D  rg.gt..*.gu..gw.
00000CB0   0A 67 79 0D 0A 63 6F 2E 67 79 0D 0A 63 6F 6D 2E  .gy..co.gy..com.
00000CC0   67 79 0D 0A 6E 65 74 2E 67 79 0D 0A 68 6B 0D 0A  gy..net.gy..hk..
00000CD0   63 6F 6D 2E 68 6B 0D 0A 65 64 75 2E 68 6B 0D 0A  com.hk..edu.hk..
00000CE0   67 6F 76 2E 68 6B 0D 0A 69 64 76 2E 68 6B 0D 0A  gov.hk..idv.hk..
00000CF0   6E 65 74 2E 68 6B 0D 0A 6F 72 67 2E 68 6B 0D 0A  net.hk..org.hk..
00000D00   E5 85 AC E5 8F B8 2E 68 6B 0D 0A E6 95 99 E8 82  å…¬å¸.hk..æ•™è‚
00000D10   B2 2E 68 6B 0D 0A E6 95 8E E8 82 B2 2E 68 6B 0D  ².hk..æ•Žè‚².hk.
00000D20   0A E6 94 BF E5 BA 9C 2E 68 6B 0D 0A E5 80 8B E4  .æ"¿åºœ.hk..å€‹ä
```

Figure 55 78B661723E5A4DE6446EE585A030E5C1 Memory Artifact A



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   65 00 75 00 6D 00 0D 00 0A 00 6F 00 6E 00 6C 00  e.u.m.....o.n.l.
00000010   69 00 6E 00 65 00 2E 00 6D 00 75 00 73 00 65 00  i.n.e...m.u.s.e.
00000020   75 00 6D 00 0D 00 0A 00 6F 00 6E 00 74 00 61 00  u.m.....o.n.t.a.
00000030   72 00 69 00 6F 00 2E 00 6D 00 75 00 73 00 65 00  r.i.o...m.u.s.e.
00000040   75 00 6D 00 0D 00 0A 00 6F 00 70 00 65 00 6E 00  u.m.....o.p.e.n.
00000050   61 00 69 00 72 00 2E 00 6D 00 75 00 73 00 65 00  a.i.r...m.u.s.e.
00000060   75 00 6D 00 0D 00 0A 00 6F 00 72 00 65 00 67 00  u.m.....o.r.e.g.
00000070   6F 00 6E 00 2E 00 6D 00 75 00 73 00 65 00 75 00  o.n...m.u.s.e.u.
00000080   6D 00 0D 00 0A 00 6F 00 72 00 65 00 67 00 6F 00  m.....o.r.e.g.o.
00000090   6E 00 74 00 72 00 61 00 69 00 6C 00 2E 00 6D 00  n.t.r.a.i.l...m.
000000A0   75 00 73 00 65 00 75 00 6D 00 0D 00 0A 00 6F 00  u.s.e.u.m.....o.
000000B0   74 00 61 00 67 00 6F 00 2E 00 6D 00 75 00 73 00  t.a.g.o...m.u.s.
000000C0   65 00 75 00 6D 00 0D 00 0A 00 6F 00 78 00 66 00  e.u.m.....o.x.f.
000000D0   6F 00 72 00 64 00 2E 00 6D 00 75 00 73 00 65 00  o.r.d...m.u.s.e.
000000E0   75 00 6D 00 0D 00 0A 00 70 00 61 00 63 00 69 00  u.m.....p.a.c.i.
000000F0   66 00 69 00 63 00 2E 00 6D 00 75 00 73 00 65 00  f.i.c...m.u.s.e.
00000100   75 00 6D 00 0D 00 0A 00 70 00 61 00 64 00 65 00  u.m.....p.a.d.e.
00000110   72 00 62 00 6F 00 72 00 6E 00 2E 00 6D 00 75 00  r.b.o.r.n...m.u.
00000120   73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00 61 00  s.e.u.m.....p.a.
00000130   6C 00 61 00 63 00 65 00 2E 00 6D 00 75 00 73 00  l.a.c.e...m.u.s.
00000140   65 00 75 00 6D 00 0D 00 0A 00 70 00 61 00 6C 00  e.u.m.....p.a.l.
00000150   65 00 6F 00 2E 00 6D 00 75 00 73 00 65 00 75 00  e.o...m.u.s.e.u.
00000160   6D 00 0D 00 0A 00 70 00 61 00 6C 00 6D 00 73 00  m.....p.a.l.m.s.
00000170   70 00 72 00 69 00 6E 00 67 00 73 00 2E 00 6D 00  p.r.i.n.g.s...m.
00000180   75 00 73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00  u.s.e.u.m.....p.
00000190   61 00 6E 00 61 00 6D 00 61 00 2E 00 6D 00 75 00  a.n.a.m.a...m.u.
000001A0   73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00 61 00  s.e.u.m.....p.a.
000001B0   72 00 69 00 73 00 2E 00 6D 00 75 00 73 00 65 00  r.i.s...m.u.s.e.
000001C0   75 00 6D 00 0D 00 0A 00 70 00 61 00 73 00 61 00  u.m.....p.a.s.a.
000001D0   64 00 65 00 6E 00 61 00 2E 00 6D 00 75 00 73 00  d.e.n.a...m.u.s.
000001E0   65 00 75 00 6D 00 0D 00 0A 00 70 00 68 00 61 00  e.u.m.....p.h.a.
000001F0   72 00 6D 00 61 00 63 00 79 00 2E 00 6D 00 75 00  r.m.a.c.y...m.u.
00000200   73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00 68 00  s.e.u.m.....p.h.
00000210   69 00 6C 00 61 00 64 00 65 00 6C 00 70 00 68 00  i.l.a.d.e.l.p.h.
00000220   69 00 61 00 2E 00 6D 00 75 00 73 00 65 00 75 00  i.a...m.u.s.e.u.
00000230   6D 00 0D 00 0A 00 70 00 68 00 69 00 6C 00 61 00  m.....p.h.i.l.a.
00000240   64 00 65 00 6C 00 70 00 68 00 69 00 61 00 61 00  d.e.l.p.h.i.a.a.
00000250   72 00 65 00 61 00 2E 00 6D 00 75 00 73 00 65 00  r.e.a...m.u.s.e.
00000260   75 00 6D 00 0D 00 0A 00 70 00 68 00 69 00 6C 00  u.m.....p.h.i.l.
00000270   61 00 74 00 65 00 6C 00 79 00 2E 00 6D 00 75 00  a.t.e.l.y...m.u.
00000280   73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00 68 00  s.e.u.m.....p.h.
00000290   6F 00 65 00 6E 00 69 00 78 00 2E 00 6D 00 75 00  o.e.n.i.x...m.u.
000002A0   73 00 65 00 75 00 6D 00 0D 00 0A 00 70 00 68 00  s.e.u.m.....p.h.
000002B0   6F 00 74 00 6F 00 67 00 72 00 61 00 70 00 68 00  o.t.o.g.r.a.p.h.
000002C0   79 00 2E 00 6D 00 75 00 73 00 65 00 75 00 6D 00  y...m.u.s.e.u.m.
000002D0   0D 00 0A 00 70 00 69 00 6C 00 6F 00 74 00 73 00  ....p.i.l.o.t.s.
000002E0   2E 00 6D 00 75 00 73 00 65 00 75 00 6D 00 0D 00  ..m.u.s.e.u.m...
000002F0   0A 00 70 00 69 00 74 00 74 00 73 00 62 00 75 00  ..p.i.t.t.s.b.u.
00000300   72 00 67 00 68 00 2E 00 6D 00 75 00 73 00 65 00  r.g.h...m.u.s.e.
00000310   75 00 6D 00 0D 00 0A 00 70 00 6C 00 61 00 6E 00  u.m.....p.l.a.n.
00000320   65 00 74 00 61 00 72 00 69 00 75 00 6D 00 2E 00  e.t.a.r.i.u.m...
00000330   6D 00 75 00 73 00 65 00 75 00 6D 00 0D 00 0A 00  m.u.s.e.u.m.....
00000340   70 00 6C 00 61 00 6E 00 74 00 61 00 74 00 69 00  p.l.a.n.t.a.t.i.
00000350   6F 00 6E 00 2E 00 6D 00 75 00 73 00 65 00 75 00  o.n...m.u.s.e.u.
00000360   6D 00 0D 00 0A 00 70 00 6C 00 61 00 6E 00 74 00  m.....p.l.a.n.t.
```

Figure 56 78B661723E5A4DE6446EE585A030E5C1 Memory Artifact B



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   69 62 61 2E 6A 70 0D 0A 79 6F 6B 61 69 63 68 69  iba.jp..yokaichi
00000010   62 61 2E 63 68 69 62 61 2E 6A 70 0D 0A 79 6F 6B  ba.chiba.jp..yok
00000020   6F 73 68 69 62 61 68 69 6B 61 72 69 2E 63 68 69  oshibahikari.chi
00000030   62 61 2E 6A 70 0D 0A 79 6F 74 73 75 6B 61 69 64  ba.jp..yotsukaid
00000040   6F 2E 63 68 69 62 61 2E 6A 70 0D 0A 61 69 6E 61  o.chiba.jp..aina
00000050   6E 2E 65 68 69 6D 65 2E 6A 70 0D 0A 68 6F 6E 61  n.ehime.jp..hona
00000060   69 2E 65 68 69 6D 65 2E 6A 70 0D 0A 69 6B 61 74  i.ehime.jp..ikat
00000070   61 2E 65 68 69 6D 65 2E 6A 70 0D 0A 69 6D 61 62  a.ehime.jp..imab
00000080   61 72 69 2E 65 68 69 6D 65 2E 6A 70 0D 0A 69 79  ari.ehime.jp..iy
00000090   6F 2E 65 68 69 6D 65 2E 6A 70 0D 0A 6B 61 6D 69  o.ehime.jp..kami
000000A0   6A 69 6D 61 2E 65 68 69 6D 65 2E 6A 70 0D 0A 6B  jima.ehime.jp..k
000000B0   69 68 6F 6B 75 2E 65 68 69 6D 65 2E 6A 70 0D 0A  ihoku.ehime.jp..
000000C0   6B 75 6D 61 6B 6F 67 65 6E 2E 65 68 69 6D 65 2E  kumakogen.ehime.
000000D0   6A 70 0D 0A 6D 61 73 61 6B 69 2E 65 68 69 6D 65  jp..masaki.ehime
000000E0   2E 6A 70 0D 0A 6D 61 74 73 75 6E 6F 2E 65 68 69  .jp..matsuno.ehi
000000F0   6D 65 2E 6A 70 0D 0A 6D 61 74 73 75 79 61 6D 61  me.jp..matsuyama
00000100   2E 65 68 69 6D 65 2E 6A 70 0D 0A 6E 61 6D 69 6B  .ehime.jp..namik
00000110   61 74 61 2E 65 68 69 6D 65 2E 6A 70 0D 0A 6E 69  ata.ehime.jp..ni
00000120   69 68 61 6D 61 2E 65 68 69 6D 65 2E 6A 70 0D 0A  ihama.ehime.jp..
00000130   6F 7A 75 2E 65 68 69 6D 65 2E 6A 70 0D 0A 73 61  ozu.ehime.jp..sa
00000140   69 6A 6F 2E 65 68 69 6D 65 2E 6A 70 0D 0A 73 65  ijo.ehime.jp..se
00000150   69 79 6F 2E 65 68 69 6D 65 2E 6A 70 0D 0A 73 68  iyo.ehime.jp..sh
00000160   69 6B 6F 6B 75 63 68 75 6F 2E 65 68 69 6D 65 2E  ikokuchuo.ehime.
00000170   6A 70 0D 0A 74 6F 62 65 2E 65 68 69 6D 65 2E 6A  jp..tobe.ehime.j
00000180   70 0D 0A 74 6F 6F 6E 2E 65 68 69 6D 65 2E 6A 70  p..toon.ehime.jp
00000190   0D 0A 75 63 68 69 6B 6F 2E 65 68 69 6D 65 2E 6A  ..uchiko.ehime.j
000001A0   70 0D 0A 75 77 61 6A 69 6D 61 2E 65 68 69 6D 65  p..uwajima.ehime
000001B0   2E 6A 70 0D 0A 79 61 77 61 74 61 68 61 6D 61 2E  .jp..yawatahama.
000001C0   65 68 69 6D 65 2E 6A 70 0D 0A 65 63 68 69 7A 65  ehime.jp..echize
000001D0   6E 2E 66 75 6B 75 69 2E 6A 70 0D 0A 65 69 68 65  n.fukui.jp..eihe
000001E0   69 6A 69 2E 66 75 6B 75 69 2E 6A 70 0D 0A 66 75  iji.fukui.jp..fu
000001F0   6B 75 69 2E 66 75 6B 75 69 2E 6A 70 0D 0A 69 6B  kui.fukui.jp..ik
00000200   65 64 61 2E 66 75 6B 75 69 2E 6A 70 0D 0A 6B 61  eda.fukui.jp..ka
00000210   74 73 75 79 61 6D 61 2E 66 75 6B 75 69 2E 6A 70  tsuyama.fukui.jp
00000220   0D 0A 6D 69 68 61 6D 61 2E 66 75 6B 75 69 2E 6A  ..mihama.fukui.j
00000230   70 0D 0A 6D 69 6E 61 6D 69 65 63 68 69 7A 65 6E  p..minamiechizen
00000240   2E 66 75 6B 75 69 2E 6A 70 0D 0A 6F 62 61 6D 61  .fukui.jp..obama
00000250   2E 66 75 6B 75 69 2E 6A 70 0D 0A 6F 68 69 2E 66  .fukui.jp..ohi.f
00000260   75 6B 75 69 2E 6A 70 0D 0A 6F 6E 6F 2E 66 75 6B  ukui.jp..ono.fuk
00000270   75 69 2E 6A 70 0D 0A 73 61 62 61 65 2E 66 75 6B  ui.jp..sabae.fuk
00000280   75 69 2E 6A 70 0D 0A 73 61 6B 61 69 2E 66 75 6B  ui.jp..sakai.fuk
00000290   75 69 2E 6A 70 0D 0A 74 61 6B 61 68 61 6D 61 2E  ui.jp..takahama.
000002A0   66 75 6B 75 69 2E 6A 70 0D 0A 74 73 75 72 75 67  fukui.jp..tsurug
000002B0   61 2E 66 75 6B 75 69 2E 6A 70 0D 0A 77 61 6B 61  a.fukui.jp..waka
000002C0   73 61 2E 66 75 6B 75 69 2E 6A 70 0D 0A 61 73 68  sa.fukui.jp..ash
000002D0   69 79 61 2E 66 75 6B 75 6F 6B 61 2E 6A 70 0D 0A  iya.fukuoka.jp..
000002E0   62 75 7A 65 6E 2E 66 75 6B 75 6F 6B 61 2E 6A 70  buzen.fukuoka.jp
000002F0   0D 0A 63 68 69 6B 75 67 6F 2E 66 75 6B 75 6F 6B  ..chikugo.fukuok
00000300   61 2E 6A 70 0D 0A 63 68 69 6B 75 68 6F 2E 66 75  a.jp..chikuho.fu
00000310   6B 75 6F 6B 61 2E 6A 70 0D 0A 63 68 69 6B 75 6A  kuoka.jp..chikuj
00000320   6F 2E 66 75 6B 75 6F 6B 61 2E 6A 70 0D 0A 63 68  o.fukuoka.jp..ch
00000330   69 6B 75 73 68 69 6E 6F 2E 66 75 6B 75 6F 6B 61  ikushino.fukuoka
00000340   2E 6A 70 0D 0A 63 68 69 6B 75 7A 65 6E 2E 66 75  .jp..chikuzen.fu
00000350   6B 75 6F 6B 61 2E 6A 70 0D 0A 63 68 75 6F 2E 66  kuoka.jp..chuo.f
00000360   75 6B 75 6F 6B 61 2E 6A 70 0D 0A 64 61 7A 61 69  ukuoka.jp..dazai
```

Figure 57 78B661723E5A4DE6446EE585A030E5C1 Memory Artifact C

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  72 61 2E 6E 61 67 61 6E 6F 2E 6A 70 0D 0A 74 6F  ra.nagano.jp..to
00000010  6D 69 2E 6E 61 67 61 6E 6F 2E 6A 70 0D 0A 75 65  mi.nagano.jp..ue
00000020  64 61 2E 6E 61 67 61 6E 6F 2E 6A 70 0D 0A 77 61  da.nagano.jp..wa
00000030  64 61 2E 6E 61 67 61 6E 6F 2E 6A 70 0D 0A 79 61  da.nagano.jp..ya
00000040  6D 61 67 61 74 61 2E 6E 61 67 61 6E 6F 2E 6A 70  magata.nagano.jp
00000050  0D 0A 79 61 6D 61 6E 6F 75 63 68 69 2E 6E 61 67  ..yamanouchi.nag
00000060  61 6E 6F 2E 6A 70 0D 0A 79 61 73 61 6B 61 2E 6E  ano.jp..yasaka.n
00000070  61 67 61 6E 6F 2E 6A 70 0D 0A 79 61 73 75 6F 6B  agano.jp..yasuok
00000080  61 2E 6E 61 67 61 6E 6F 2E 6A 70 0D 0A 63 68 69  a.nagano.jp..chi
00000090  6A 69 77 61 2E 6E 61 67 61 73 61 6B 69 2E 6A 70  jiwa.nagasaki.jp
000000A0  0D 0A 66 75 74 73 75 2E 6E 61 67 61 73 61 6B 69  ..futsu.nagasaki
000000B0  2E 6A 70 0D 0A 67 6F 74 6F 2E 6E 61 67 61 73 61  .jp..goto.nagasa
000000C0  6B 69 2E 6A 70 0D 0A 68 61 73 61 6D 69 2E 6E 61  ki.jp..hasami.na
000000D0  67 61 73 61 6B 69 2E 6A 70 0D 0A 68 69 72 61 64  gasaki.jp..hirad
000000E0  6F 2E 6E 61 67 61 73 61 6B 69 2E 6A 70 0D 0A 69  o.nagasaki.jp..i
000000F0  6B 69 2E 6E 61 67 61 73 61 6B 69 2E 6A 70 0D 0A  ki.nagasaki.jp..
00000100  69 73 61 68 61 79 61 2E 6E 61 67 61 73 61 6B 69  isahaya.nagasaki
00000110  2E 6A 70 0D 0A 6B 61 77 61 74 61 6E 61 2E 6E 61  .jp..kawatana.na
00000120  67 61 73 61 6B 69 2E 6A 70 0D 0A 6B 75 63 68 69  gasaki.jp..kuchi
00000130  6E 6F 74 73 75 2E 6E 61 67 61 73 61 6B 69 2E 6A  notsu.nagasaki.j
00000140  70 0D 0A 6D 61 74 73 75 75 72 61 2E 6E 61 67 61  p..matsuura.naga
00000150  73 61 6B 69 2E 6A 70 0D 0A 6E 61 67 61 73 61 6B  saki.jp..nagasak
00000160  69 2E 6E 61 67 61 73 61 6B 69 2E 6A 70 0D 0A 6F  i.nagasaki.jp..o
00000170  62 61 6D 61 2E 6E 61 67 61 73 61 6B 69 2E 6A 70  bama.nagasaki.jp
00000180  0D 0A 6F 6D 75 72 61 2E 6E 61 67 61 73 61 6B 69  ..omura.nagasaki
00000190  2E 6A 70 0D 0A 6F 73 65 74 6F 2E 6E 61 67 61 73  .jp..oseto.nagas
000001A0  61 6B 69 2E 6A 70 0D 0A 73 61 69 6B 61 69 2E 6E  aki.jp..saikai.n
000001B0  61 67 61 73 61 6B 69 2E 6A 70 0D 0A 73 61 73 65  agasaki.jp..sase
000001C0  62 6F 2E 6E 61 67 61 73 61 6B 69 2E 6A 70 0D 0A  bo.nagasaki.jp..
000001D0  73 65 69 68 69 2E 6E 61 67 61 73 61 6B 69 2E 6A  seihi.nagasaki.j
000001E0  70 0D 0A 73 68 69 6D 61 62 61 72 61 2E 6E 61 67  p..shimabara.nag
000001F0  61 73 61 6B 69 2E 6A 70 0D 0A 73 68 69 6E 6B 61  asaki.jp..shinka
00000200  6D 69 67 6F 74 6F 2E 6E 61 67 61 73 61 6B 69 2E  migoto.nagasaki.
00000210  6A 70 0D 0A 74 6F 67 69 74 73 75 2E 6E 61 67 61  jp..togitsu.naga
00000220  73 61 6B 69 2E 6A 70 0D 0A 74 73 75 73 68 69 6D  saki.jp..tsushim
00000230  61 2E 6E 61 67 61 73 61 6B 69 2E 6A 70 0D 0A 75  a.nagasaki.jp..u
00000240  6E 7A 65 6E 2E 6E 61 67 61 73 61 6B 69 2E 6A 70  nzen.nagasaki.jp
00000250  0D 0A 61 6E 64 6F 2E 6E 61 72 61 2E 6A 70 0D 0A  ..ando.nara.jp..
00000260  67 6F 73 65 2E 6E 61 72 61 2E 6A 70 0D 0A 68 65  gose.nara.jp..he
00000270  67 75 72 69 2E 6E 61 72 61 2E 6A 70 0D 0A 68 69  guri.nara.jp..hi
00000280  67 61 73 68 69 79 6F 73 68 69 6E 6F 2E 6E 61 72  gashiyoshino.nar
00000290  61 2E 6A 70 0D 0A 69 6B 61 72 75 67 61 2E 6E 61  a.jp..ikaruga.na
000002A0  72 61 2E 6A 70 0D 0A 69 6B 6F 6D 61 2E 6E 61 72  ra.jp..ikoma.nar
000002B0  61 2E 6A 70 0D 0A 6B 61 6D 69 6B 69 74 61 79 61  a.jp..kamikitaya
000002C0  6D 61 2E 6E 61 72 61 2E 6A 70 0D 0A 6B 61 6E 6D  ma.nara.jp..kanm
000002D0  61 6B 69 2E 6E 61 72 61 2E 6A 70 0D 0A 6B 61 73  aki.nara.jp..kas
000002E0  68 69 62 61 2E 6E 61 72 61 2E 6A 70 0D 0A 6B 61  hiba.nara.jp..ka
000002F0  73 68 69 68 61 72 61 2E 6E 61 72 61 2E 6A 70 0D  shihara.nara.jp.
00000300  0A 6B 61 74 73 75 72 61 67 69 2E 6E 61 72 61 2E  .katsuragi.nara.
00000310  6A 70 0D 0A 6B 61 77 61 69 2E 6E 61 72 61 2E 6A  jp..kawai.nara.j
00000320  70 0D 0A 6B 61 77 61 6B 61 6D 69 2E 6E 61 72 61  p..kawakami.nara
00000330  2E 6A 70 0D 0A 6B 61 77 61 6E 69 73 68 69 2E 6E  .jp..kawanishi.n
00000340  61 72 61 2E 6A 70 0D 0A 6B 6F 72 79 6F 2E 6E 61  ara.jp..koryo.na
00000350  72 61 2E 6A 70 0D 0A 6B 75 72 6F 74 61 6B 69 2E  ra.jp..kurotaki.
00000360  6E 61 72 61 2E 6A 70 0D 0A 6D 69 74 73 75 65 2E  nara.jp..mitsue.
```

Figure 58 78B661723E5A4DE6446EE585A030E5C1 Memory Artifact D

Comments Appears to establish a bot to attack websites. Contains internet addresses as shown below.

Results

Sample	Persistence	Files	Registry	Data	Communication
78B661723E5A4DE6446EE585A030E5C1	X	X	X	X	X

Table 7 78B661723E5A4DE6446EE585A030E5C1 Results

Method of Persistence

Persistence via registry entries pointing to the program.

Files Created, Modified, or Opened

Figure 59 78B661723E5A4DE6446EE585A030E5C1 Files Created

Relevant Registry Elements Created, Modified, or Opened

Most registry activity is focused on the .NET binary and is not included here. The ones to focus on for this sample were the ones related to internet activity.

>RegOpenKeyExW Key = [Internet]
>RegEnumKeyExW Subkey = []
>RegEnumKeyExW Subkey = [MediaPermission]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [Internet]
>RegOpenKeyExW Key = [LocalIntranet]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [MediaPermission]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [LocalIntranet]

Type of Information Collected/Purpose

The purpose appears to be to attack either fixed DNS address, generated DNS address, or both.

Method of Communication

neyzz.ddns.net does not currently resolve to an IP address.
>bind IP: 0.0.0.0 Port: 0
>bind IP: 0.0.0.0 Port: 0
>bind IP: 0.0.0.0 Port: 0
>bind IP: 0.0.0.0 Port: 0
>getaddrinfo: Node:neyzz.ddns.net Service:(null)
>bind IP: 0.0.0.0 Port: 0
>bind IP: 0.0.0.0 Port: 0
>getaddrinfo: Node:neyzz.ddns.net Service:(null)
>bind IP: 0.0.0.0 Port: 0
>bind IP: 0.0.0.0 Port: 0

>getaddrinfo: Node:neyzz.ddns.net Service:(null)

>bind IP: 0.0.0.0 Port: 0

>bind IP: 0.0.0.0 Port: 0

>getaddrinfo: Node:neyzz.ddns.net Service:(null)

Sample 7 - Backdoor

MD5 7D436F8B7C72498CD5738812D9E0EC61

Link

https://www.virustotal.com/#/file/2a3bbd43e5874d6fdd4c366a02627532dc07a2cc44bed00becaa475f0dc751ba/detection

Classification Backdoor

Comments

- Uses GetTickCount as an initial anti-analysis method.
- Bypassed GetTickCount anti-analysis using MACT.
- Creates MBSSCR.exe – uploaded to virustotal, was not prevoiusly referenced there.
- Runs MBSSCR.exe which launches wscript which in turn executes the visual basic script VZZDQG.VBS. wscript.exe is a Windows service that allows you to execute VBScript files.
- Attempted to inject MBSSCR from the ShellExecuteExW routine but was not successful. Instead ran MBSSCR directly.
- MBSSCR.exe becomes locked in GetTickCount analysis.
- Analysis ends in looping GetTickCount, unknown what is being processed.

Overview

Creates C:\Users\MACT\AppData\Local\Temp\autD6F2.tmp and the executable C:\Users\MACT\AppData\Local\Temp/MBSSCR.exe, 7DD04FA1670F10C182D9B6B75BC2B71D, which has no match on virustotal. I subsequently uploaded it to VirusTotal.

Figure 60 7D436F8B7C72498CD5738812D9E0EC61 creates MBSSCR.exe

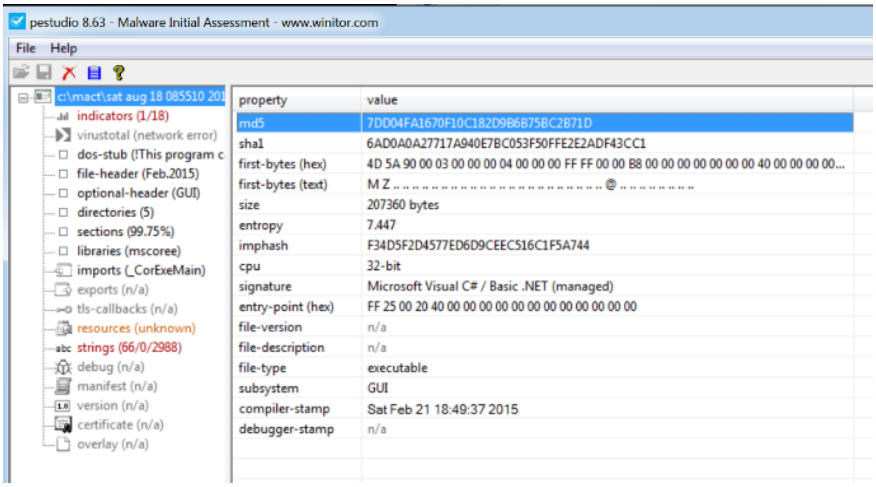


Figure 61 7D436F8B7C72498CD5738812D9E0EC61 created MBSSCR.exe in pestudio.

Results

Sample	Persistence	Files	Registry	Data	Communication
7D436F8B7C72498CD5738812D9E0EC61	X	X	X		X

Table 8 7D436F8B7C72498CD5738812D9E0EC61 Results

Method of Persistence

Visual basic script, VZZDQG.vbs attempts to maintain persistence by checking if the malware has a running process if not it runs the malware.

```
On error resume next
Dim strComputer,strProcess,fileset
strProcess = "sample.exe"
fileset = """C:\Users\MACT\Desktop\MACT\sample.exe"""
strComputer = "."
Dim objShell
Set objShell = CreateObject("WScript.Shell")
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
while 1
IF isProcessRunning(strComputer,strProcess) THEN
ELSE
objShell.Run fileset
END IF
Wend
FUNCTION isProcessRunning(BYVAL strComputer,BYVAL strProcessName)
DIM objWMIService, strWMIQuery
strWMIQuery = "Select * from Win32_Process where name like '" & strProcessName & "'"
SET objWMIService = GETOBJECT("winmgmts:" _
& "{impersonationLevel=impersonate}!\\" _
& strComputer & "\root\cimv2")
IF objWMIService.ExecQuery(strWMIQuery).Count > 0 THEN
isProcessRunning = TRUE
ELSE
isProcessRunning = FALSE
END IF
END FUNCTION
```

Figure 62 7D436F8B7C72498CD5738812D9E0EC61 created VZZDQG.vbs

Files Created, Modfied, or Opened

Phase 1:

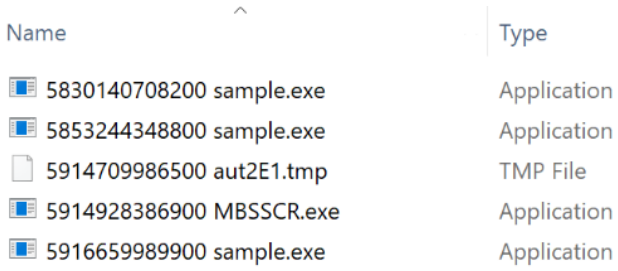


Figure 63 7D436F8B7C72498CD5738812D9E0EC61 phase 1 files created.

51033976720 desktop.ini	Configuration settings
51062876600 desktop.ini	Configuration settings
59582808630 desktop.ini	Configuration settings
585341548500 sample.exe	Application
594013831200 desktop.ini	Configuration settings
594271235700 desktop.ini	Configuration settings
595339253800 desktop.ini	Configuration settings
5100839267500 desktop.ini	Configuration settings
5829391906900 sample.exe	Application
5829922307800 sample.exe	Application
5846427136800 desktop.ini	Configuration settings
5849687542600 desktop.ini	Configuration settings
5852417547400 sample.exe	Application
5912276382200 sample.exe	Application
5912806783200 aut2E1.tmp	TMP File
5913368384200 aut2E1.tmp	TMP File
5914507186200 aut2E1.tmp	TMP File
5914709986500 MBSSCR.exe	Application
5916457189600 sample.exe	Application
5919280794500 R000000000006.clb	CLB File
5945270440200 desktop.ini	Configuration settings
5947329643800 desktop.ini	Configuration settings
5950449649300 desktop.ini	Configuration settings
5955691258500 desktop.ini	Configuration settings
51038560133800 MBSSCR.exe	Application
51044300943900 setupapi.app.log	Text Document
winwrar.exe	Application

Figure 64 7D436F8B7C72498CD5738812D9E0EC61 phase 1 files created continued.

Phase 2:

Name	Type
7161869480900 MBSSCR.exe	Application
7175866255500 machine.config	XML Configuration File
7176648565200 machine.config	XML Configuration File
7184765941500 rsaenh.dll	Application extension
71619801682900 MBSSCR.exe	Application
71745976234200 l_intl.nls	NLS File
71755429850800 MBSSCR.exe	Application
71915473591400 mscorlib.dll	Application extension
71917548395100 MBSSCR.exe	Application
71932602421500 machine.config	XML Configuration File
71933366822900 machine.config	XML Configuration File

Figure 65 7D436F8B7C72498CD5738812D9E0EC61 phase 2 files created.

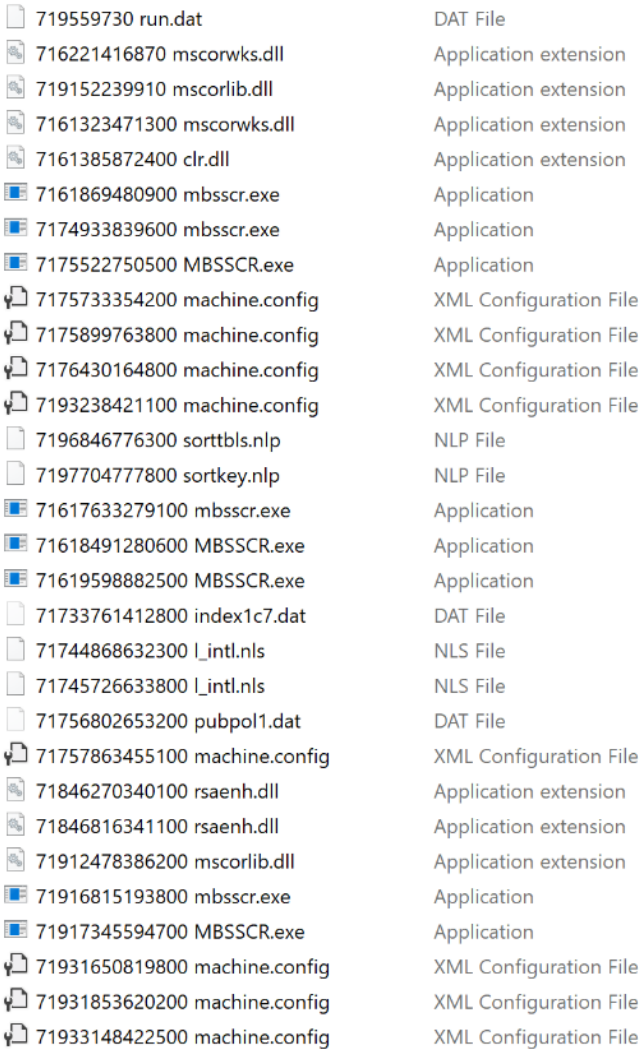


Figure 66 7D436F8B7C72498CD5738812D9E0EC61 phase 2 files created continued.

Relevant Registry Elements Created, Modified, or Opened

Phase 1:

>RegOpenKeyExW Key = [FEATURE_LOCALMACHINE_LOCKDOWN]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [sample.exe]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [*]

>RegOpenKeyExW Key = [FEATURE_ZONES_DEFAULT_DRIVE_INTRANET_KB941000]

>RegOpenKeyExW Key = [FEATURE_PROTOCOL_LOCKDOWN]

>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [sample.exe]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [*]

Phase 2:

>RegOpenKeyExW Key = [Internet]
>RegEnumKeyExW Subkey = []
>RegEnumKeyExW Subkey = [MediaPermission]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [Internet]
>RegOpenKeyExW Key = [LocalIntranet]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [MediaPermission]
>RegEnumKeyExW Subkey = [WebBrowserPermission]
>RegEnumKeyExW Subkey = [LocalIntranet]

Type of Information Collected/Purpose

Unable to determine. Seems to be hooking into windows dialog controls.

Method of Communication

The following IP was detected:
>inet_addr 191.101.22.32

Sample 8 - Trojan

MD5 0EC2EF48ECCC4E25FA35C59D3CB4A56E

Link

https://www.virustotal.com/#/file/3d0a7297e47072373459e095e7c3debfd206a833300b233a24897c910baa96d9/detection

Classification Trojan

Comments Cannot detect malware completion or reason for it to stop executing.

Overview

- Use MACT to bypass GetTickCount anti-analysis.
- Compares the executable against a list of file names, continues when it matches against one of the entries in the list. This appears to be another form of anti-analysis. Can set a breakpoint and override the lstrcmpiA to return TRUE or rename the sample.

lstrcmpiA	WPWIN7.EXE	sample.exe
lstrcmpiA	PRWIN70.EXE	sample.exe
lstrcmpiA	PS80.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	QFINDER.EXE	sample.exe
lstrcmpiA	PFIM80.EXE	sample.exe
lstrcmpiA	UA80.EXE	sample.exe
lstrcmpiA	PDXWIN32.EXE	sample.exe
lstrcmpiA	SITEBUILDER...	sample.exe
lstrcmpiA	HOTDOG4.EXE	sample.exe
lstrcmpiA	RNAAPP.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	SIZEMGR.EXE	sample.exe
lstrcmpiA	SMARTCTR.EXE	sample.exe
lstrcmpiA	WPWIN8.EXE	sample.exe
lstrcmpiA	PRWIN8.EXE	sample.exe
lstrcmpiA	UE32.EXE	sample.exe
lstrcmpiA	PP70.EXE	sample.exe
lstrcmpiA	PP80.EXE	sample.exe
lstrcmpiA	PS80.EXE	sample.exe
lstrcmpiA	ABCMM.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	CORELDRW.EXE	sample.exe
lstrcmpiA	FILLER51.EXE	sample.exe
lstrcmpiA	AUTORUN.EXE	sample.exe
lstrcmpiA	AUTORUN.EXE	sample.exe
lstrcmpiA	POWERPNT.EXE	sample.exe
lstrcmpiA	MSMONEY.EXE	sample.exe
lstrcmpiA	soffice.EXE	sample.exe
lstrcmpiA	WPWIN9.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	PRWIN9.EXE	sample.exe
lstrcmpiA	DAD9.EXE	sample.exe

Figure 67 0EC2EF48ECCC4E25FA35C59D3CB4A56E Malware Sample File Name Compare

- Creates C:\Users\MACT\AppData\Local\Temp\x.htmlfile in but never writes anything to it.

- Memory artifacts indicate using internet explorer for communications.

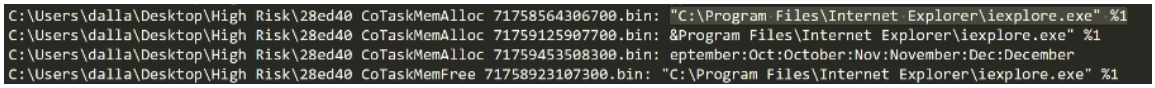


Figure 68 270EC2EF48ECCC4E25FA35C59D3CB4A56E Memory artifact referencing iexplore.exe

Results

Sample	Persistence	Files	Registry	Data	Communication
0EC2EF48ECCC4E25FA35C59D3CB4A56E					

Table 9 0EC2EF48ECCC4E25FA35C59D3CB4A56E Results

Method of Persistence

Unknown.

Files Created, Modified, or Opened

Likely incomplete list as the malware appears to stop executing.

R000000000006.clb

desktop.ini

x.html

Registry Elements Created, Modified, or Opened

Likely incomplete list as the malware appears to stop executing.

>RegSetValueExW Key = [Mutex]

>RegOpenKeyExW Key = [Software\Microsoft\Windows\CurrentVersion\Explorer\KindMap]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [.htm]

Type of Information Collected/Purpose

Unknown.

Method of Communication

Unknown. Memory artifact analysis indicates the malware may be executing html via iexplore.exe.

Sample 9 - Virus

MD5 8F7AB3C56E3FF4A6D23C57D0E07A4D1E

Link

https://www.virustotal.com/#/file/93a61700a8340ee9ec7d1781ef6bf6d5a556c65dde7f7f357cf0927c3356142d/detection

Classification Virus

Overview

From the starting process two more processes were started via WinExec.

WinExec	C:\Users\MACT\AppData\Local\Temp\LhIgoE.exe	5
WinExec	C:\Users\MACT\AppData\Local\Temp\BUSeJQv.exe	5

Figure 69 8F7AB3C56E3FF4A6D23C57D0E07A4D1E Initial sample creating 2 new processes.

Perfromed mact32.dll injection by changing the WinExec parm. The parameters were hardcoded so that they would not have to be manually entered via the MACT console. Setting a breakpoint on *WinExec* allowed for the capture of the file as MACT failes to capture the file artifacts.

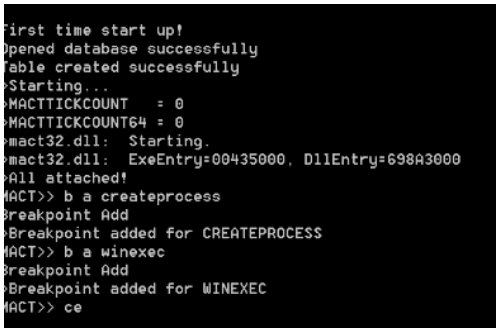


Figure 70 8F7AB3C56E3FF4A6D23C57D0E07A4D1E WinExec breakpoint.

```
if(strncmp(a0, "C:\\Users\\MACT\\AppData\\Local\\Temp\\LhIgoE.exe", sizeof(a0)) == 0) {
    a0 = "C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe -d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\Desktop\\MACT\\LhIgoE.exe";
    MACTPrint(">WinExec override to C:\\Users\\MACT\\AppData\\Local\\Temp\\LhIgoE.exe");
}

if(strncmp(a0, "C:\\Users\\MACT\\AppData\\Local\\Temp\\BUSeJQv.exe", sizeof(a0)) == 0) {
    a0 = "C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe -d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\Desktop\\MACT\\BUSeJQv.exe";
    MACTPrint(">WinExec override to C:\\Users\\MACT\\AppData\\Local\\Temp\\BUSeJQv.exe");
}
```

Figure 71 8F7AB3C56E3FF4A6D23C57D0E07A4D1E Override WinExec parm to inject mact32.dll.

C:\Users\MACT\AppData\Local\Temp\LhIgoE.exe has an MD5 of 4063BF31242DDAD360909610F44F04CB.
C:\Users\MACT\AppData\Local\Temp\BUSeJQv.exe has an MD5 of DAC549678B799C97B9BDC3FC5B8BAE6F.

After injecting capturing the output of LhIgoE.exe, there are two files of interest, 60e86271.bat and C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe.

CopyFileExW	C:\Users\MACT\AppData\Local\Temp\60e86271.bat	C:\MACT\Fri Aug 24 175101 2018\Files\Created\65127821660500 60e86271.bat
CreateFileW	C:\Users\MACT\AppData\Local\Temp\60e86271.bat	C0000000
CreateFileA	C:\Users\MACT\AppData\Local\Temp\60e86271.bat	C0000000
CreateFileW	C:\Users\MACT\AppData\Local\Temp\60e86271.bat	00000080

Figure 72 8F7AB3C56E3FF4A6D23C57D0E07A4D1E creates C:\Users\MACT\AppData\Local\Temp\60e86271.bat

C:\Users\MACT\AppData\Local\Temp\60e86271.bat appears to try to cover the malware's tracks by deleting the initially unpacked file and then itself.

```
:DELFILE
del "C:\Users\MACT\Desktop\MACT\LhIgoE.exe"
if exist "C:\Users\MACT\Desktop\MACT\LhIgoE.exe" goto :DELFILE
del "C:\Users\MACT\AppData\Local\Temp\60e86271.bat"
```

Figure 73 8F7AB3C56E3FF4A6D23C57D0E07A4D1E C:\Users\MACT\AppData\Local\Temp\60e86271.bat

C:\Users\MACT\AppData\Local\Temp\BUSeJQv.exe creates C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe and then executes it via WinExec. It also seems to process all .exe files.

CopyFileExW	C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe	C:\MACT\Fri Aug 24 175101 2018\Files\Created\6513488170 ivXSsb.exe
CreateFileW	C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe	C0000000
CreateFileA	C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe	C0000000
CreateFileW	C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe	00000080
CreateProcess		C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe
WinExec	C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe	5

Figure 74 8F7AB3C56E3FF4A6D23C57D0E07A4D1E BUSeJQv.exe creating and then executing ivXSsb.exe

C:\Users\MACT\AppData\Local\Temp\ivXSsb.exe had an MD5 of 56B2C3810DBA2E939A8BB9FA36D3CF96.

Serverlog.txt indicated an executable in memory artifact C:\MACT\Fri Aug 24 182558 2018\Mem\210000 VirtualAlloc 7262326221500.bin which had an md5 56B2C3810DBA2E939A8BB9FA36D3CF96. This is likely ivXSsb.exe being created as the hashes are the same.

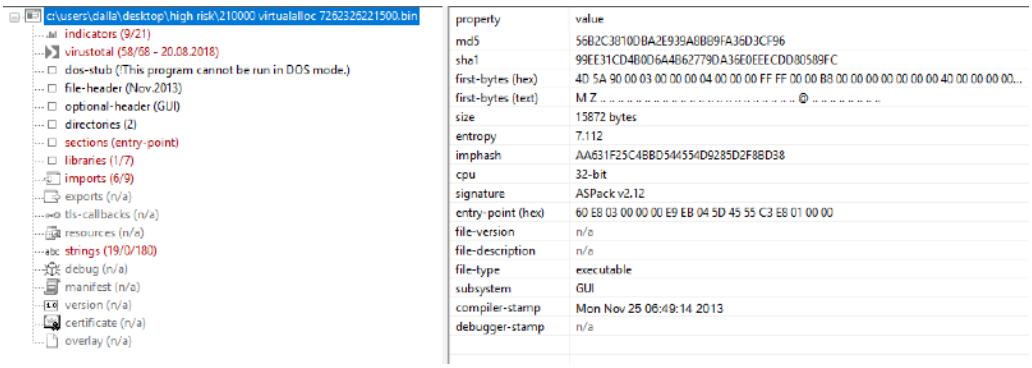


Figure 75 56B2C3810DBA2E939A8BB9FA36D3CF96 in pestudio

Subsequent runs of the same sample executables appear to show different results. For example, the second time ivXSsb.exe runs in the previously corrupted environment it copies and executes itself as iZhNrf.exe which has the same MD5.

CopyFileExW	C:\Users\MACT\AppData\Local\Temp\iZhNrf.exe	C:\MACT\Sat Aug 25 080112 2018\Files\Created\512269173400 iZhNrf.exe
CreateFileW	C:\Users\MACT\AppData\Local\Temp\iZhNrf.exe	C0000000
CreateFileA	C:\Users\MACT\AppData\Local\Temp\iZhNrf.exe	C0000000
CreateFileW	C:\Users\MACT\AppData\Local\Temp\iZhNrf.exe	00000080
WinExec	C:\Users\MACT\AppData\Local\Temp\iZhNrf.exe	5

Figure 76 Creating and executing iZhNrf.exe

Appears to look for the exact same exectables as the Trojan sample 0EC2EF48ECCC4E25FA35C59D3CB4A56E.

lstrcmpiA	WPWIN7.EXE	sample.exe
lstrcmpiA	PRWIN70.EXE	sample.exe
lstrcmpiA	PS80.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	QFINDER.EXE	sample.exe
lstrcmpiA	PFIM80.EXE	sample.exe
lstrcmpiA	UA80.EXE	sample.exe
lstrcmpiA	PDXWIN32.EXE	sample.exe
lstrcmpiA	SITEBUILDER.EXE	sample.exe
lstrcmpiA	HOTDOG4.EXE	sample.exe
lstrcmpiA	RNAAPP.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	PDEXPLO.EXE	sample.exe
lstrcmpiA	SIZEMGR.EXE	sample.exe
lstrcmpiA	SMARTCTR.EXE	sample.exe
lstrcmpiA	WPWIN8.EXE	sample.exe
lstrcmpiA	PRWIN8.EXE	sample.exe
lstrcmpiA	UE32.EXE	sample.exe
lstrcmpiA	PP70.EXE	sample.exe
lstrcmpiA	PP80.EXE	sample.exe
lstrcmpiA	PS80.EXE	sample.exe
lstrcmpiA	ABCMM.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	CORELDRW.EXE	sample.exe
lstrcmpiA	FILLER51.EXE	sample.exe
lstrcmpiA	AUTORUN.EXE	sample.exe
lstrcmpiA	AUTORUN.EXE	sample.exe
lstrcmpiA	POWERPNT.EXE	sample.exe
lstrcmpiA	MSMONEY.EXE	sample.exe
lstrcmpiA	soffice.EXE	sample.exe
lstrcmpiA	WPWIN9.EXE	sample.exe
lstrcmpiA	QPW.EXE	sample.exe
lstrcmpiA	PRWIN9.EXE	sample.exe
lstrcmpiA	DAD9.EXE	sample.exe

Figure 77 8F7AB3C56E3FF4A6D23C57D0E07A4D1E String compare looking for executables.

Results

Sample	Persistence	Files	Registry	Data	Communication
8F7AB3C56E3FF4A6D23C57D0E07A4D1E	X	X	X		

Table 10 8F7AB3C56E3FF4A6D23C57D0E07A4D1E Results

Method of Persistence

Establishes itself via registry entries, makes copies of itself, remains in C:\Users\MACT\AppData\Local\Temp\

Relevant Files Created, Modfied, or Opened

Original Sample:

LhIgoE.exe

ivXSsb.exe

All .exe's do an inventory of .exe files and create copies of themselves and a .dat file.

LhIgoE.exe:

41052161330500 LhIgoE.exe

41053346932600 LhIgoE.exe

41335162639400 counters.dat

BUSeJQv.exe:

4126694390400 BUSeJQv.exe

4127911192500 BUSeJQv.exe

445606908200 counters.dat

ivXSsb.exe:

44526582776400 sample.exe

44527768378500 sample.exe

4482838260400 counters.dat

Registry Elements Created, Modified, or Opened

Original Sample:

>RegOpenKeyEx Key = [SOFTWARE\Microsoft\BidInterface\Loader]

>RegOpenKeyExW Key = [SOFTWARE\ODBC\ODBC.INI\ODBC]

>RegOpenKeyExW Key = [SOFTWARE\ODBC\ODBC.INI\ODBC]

>RegOpenKeyExW Key = [SOFTWARE\ODBC\ODBC.INI\ODBC]

>RegOpenKeyExW Key = [SOFTWARE\ODBC\ODBC.INI\ODBC]

>RegCreateKeyEx Key = [SYSTEM\CurrentControlSet\Services\Rskgwq aqmccyya]

>RegOpenKeyEx Key = [SYSTEM\CurrentControlSet\Services\Rskgwq aqmccyya]

LhIgoE.exe:

>RegOpenKeyExW Key = [FEATURE_MIME_HANDLING]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [LhIgoE.exe]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [*]

>RegOpenKeyExW Key = [Software\Microsoft\Internet Explorer\Main]

>RegOpenKeyExW Key = [FEATURE_BROWSER_EMULATION]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [LhIgoE.exe]

>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]

>RegQueryValueEx Key = [ConnectTimeOut]

>RegQueryValueEx Key = [ConnectTimeOut]

>RegQueryValueEx Key = [SendTimeOut]

>RegQueryValueEx Key = [SendTimeOut]

>RegQueryValueEx Key = [ReceiveTimeOut]

>RegQueryValueEx Key = [ReceiveTimeOut]

>RegCreateKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]

>RegQueryValueEx Key = [SyncMode5]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [SessionStartTimeDefaultDeltaSecs]

>RegGetValueA SubKey = [SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings]

>RegGetValueA Value = [MBCSAPIforCrack]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [LhIgoE.exe]

>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [*]

>RegOpenKeyExW Key = [RETRY_HEADERONLYPOST_ONCONNECTIONRESET]

>RegOpenKeyExW Key = [FEATURE_BYPASS_CACHE_FOR_CREDPOLICY_KB936611]

>RegOpenKeyExW Key = [FEATURE_IGNORE_MAPPINGS_FOR_CREDPOLICY]
>RegOpenKeyExW Key = [FEATURE_INCLUDE_PORT_IN_SPN_KB908209]
>RegOpenKeyExW Key = [FEATURE_BUFFERBREAKING_818408]
>RegOpenKeyExW Key =
[FEATURE_SKIP_POST_RETRY_ON_INTERNETWRITEFILE_KB895954]
>RegOpenKeyExW Key =
[FEATURE_FIX_CHUNKED_PROXY_SCRIPT_DOWNLOAD_KB843289]
>RegOpenKeyExW Key = [FEATURE_USE_CNAME_FOR_SPN_KB911149]
>RegOpenKeyExW Key =
[FEATURE_PERMIT_CACHE_FOR_AUTHENTICATED_FTP_KB910274]
>RegOpenKeyExW Key =
[FEATURE_DISABLE_UNICODE_HANDLE_CLOSING_CALLBACK]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [LhIgoE.exe]
>RegGetValueA Value = [UTF8ServerNameRes-,n@]
>RegQueryValueEx Key = [DisableReadRange]
>RegQueryValueEx Key = [SocketSendBufferLength]
>RegQueryValueEx Key = [SocketReceiveBufferLength]
>RegQueryValueEx Key = [KeepAliveTimeout]
>RegQueryValueEx Key = [MaxHttpRedirects]
>RegQueryValueEx Key = [MaxConnectionsPerServer]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]

BUSeJQv.exe:
>RegOpenKeyExW Key = [FEATURE_MIME_HANDLING]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [BUSeJQv.exe]
>RegOpenKeyEx Key = [PROTOCOLS\Name-Space Handler*\]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [BUSeJQv.exe]
>RegGetValueW SubKey = [NULL]

>RegGetValueW Value = [*]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]
>RegQueryValueEx Key = [ConnectTimeOut]
>RegQueryValueEx Key = [ConnectTimeOut]
>RegQueryValueEx Key = [SendTimeOut]
>RegQueryValueEx Key = [SendTimeOut]
>RegQueryValueEx Key = [ReceiveTimeOut]
>RegQueryValueEx Key = [ReceiveTimeOut]
>RegCreateKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]

ivXSsb.exe:
>RegGetValueW Value = [sample.exe]
>RegOpenKeyExW Key = [FEATURE_BROWSER_EMULATION]
>RegGetValueW SubKey = [NULL]
>RegGetValueW Value = [sample.exe]
>RegOpenKeyEx Key = [SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings]

Type of Information Collected/Purpose

Unknown.

Method of Communication

Unknown.

Sample 10 - Worm

MD5 44B5A3AF895F31E22F6BC4EB66BD3EB7

Link

https://www.virustotal.com/#/file/a98099541168c7f36b107e24e9c80c9125fefb787ae720799b03bb4425aba1a9/detection

Classification Worm

Overview

- Uses GetTickCount anti-analysis. Used at the very beginning of malware execution. Used MACT parameter overriding to bypass.
- Uses QueryPerformanceCounter API for anti-analysis. This API uses the uses a performance counter that can be used for measuring time. Modified MACT to intercept this function to perform parm substitution.
- After bypassing the anti-analysis techniques, the malware tries to cover its tracks and then communicates with its command and control.

CreateProcess		cmd /c del /q "c:\RECYCLER\\waccess.tmp"
CreateProcess		cmd /c del /q "c:\RECYCLER\\waccess.tmp"
CreateProcess		cmd /c reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion" /v ProductName

Figure 78 44B5A3AF895F31E22F6BC4EB66BD3EB7creates a process to delete a file.

- No memory artifacts were captured.
- Some data is stored in a .DAT file. This is marked as a system file and as such is hidden.
- This sample appears to no longer be functioning due to a lack of response from the command and control IP. MACT can can no further information.
-

Results

Sample	Persistence	Files	Registry	Data	Communication
44B5A3AF895F31E22F6BC4EB66BD3EB7		X	X		X

Table 11 44B5A3AF895F31E22F6BC4EB66BD3EB7 Results

Method of Persistence

Unknown.

Files Created, Modified, or Opened

2151443250 sample.exe	Application
217109928400 MACT.tc.dat	DAT File
2055222707500 sample.exe	Application
2055753108500 sample.exe	Application
2056314709500 sample.exe	Application

Figure 79 44B5A3AF895F31E22F6BC4EB66BD3EB7captured file artifacts.

Registry Elements Created, Modified, or Opened

>RegOpenKeyExW Key = [Software\Microsoft\Windows NT\CurrentVersion\VFW]

Type of Information Collected/Purpose

Unknown.

Method of Communication

Cycles waiting for response from command and control (69.64.90.94). Uncertain why the other addresses are there, perhaps as an attempt to obfuscate.

>gethostbyname: microsoft.com
>gethostbyname: www.microsoft.com
>gethostbyname: google.com
>gethostbyname: www.bing.com
>gethostbyname: maktoob.yahoo.com
>gethostbyname: 69.64.90.94
>gethostbyname: microsoft.com
>gethostbyname: www.microsoft.com
>gethostbyname: google.com
>gethostbyname: www.bing.com
>gethostbyname: maktoob.yahoo.com
>gethostbyname: 69.64.90.94
>gethostbyname: www.microsoft.com

>gethostbyname: google.com
>gethostbyname: www.bing.com
>gethostbyname: maktoob.yahoo.com
>gethostbyname: 69.64.90.94
>inet_addr 69.64.90.94
>inet_addr 69.64.90.94
>inet_addr 69.64.90.94
>inet_addr 69.64.90.94
>inet_addr 69.64.90.94
>inet_addr 69.64.90.94
>gethostbyname: google.com
>gethostbyname: www.bing.com
>gethostbyname: maktoob.yahoo.com
>gethostbyname: 69.64.90.94
>gethostbyname: microsoft.com
>gethostbyname: www.microsoft.com
>gethostbyname: google.com
>gethostbyname: www.bing.com
>gethostbyname: maktoob.yahoo.com

Testing Summary

Testing of MACT against the malware samples was performed using a 32-bit Windows 7 virtual machine. The files *mact32.dll*, *serverc.exe*, *sqlite3.dll*, *withdll.exe* and the malware samples were copied the virtual machine. The program *withdll.exe* was used to inject *mact32.dll* into each sample. A server, *serverc.exe*, was running and listing for *mact32.dll* to connect. Once the connection was made to the server it provided the MACT command line and analysis of the sample could begin.

Five objectives of malware analysis were sought. In Table 12 MACT Testing Results the objectives are indicated by the columns persistence, files, registry, data, and communication. There was a range of objectives met by the tested samples. Classifications of samples tested

included Backdoor, Potentially Unwanted Applications (PUA), Ransomware, Trojans, Virus, and Worms.

Sample	Persistence	Files	Registry	Data	Communication
361B481084C41D0DF4C4EB763F268043	X	X	X		
5CB07299CEDD69F096B09358754831E0	X	X	X	X	X
F54D48B019436E28C1AF5BABF247748B	X	X	X		X
830400121A557B0668AE9374CD847C8B	X	X	X	X	X
2F54B592E62E66FBFE7F7DB24F70F36A	X	X	X		
78B661723E5A4DE6446EE585A030E5C1	X	X	X	X	X
7D436F8B7C72498CD5738812D9E0EC61	X	X	X		X
0EC2EF48ECCC4E25FA35C59D3CB4A56E					
8F7AB3C56E3FF4A6D23C57D0E07A4D1E	X	X	X		
44B5A3AF895F31E22F6BC4EB66BD3EB7		X	X		X

Table 12 MACT Testing Results

361B481084C41D0DF4C4EB763F268043 was a .NET assembly and as such there was registry and file activity related to its execution via *coremain.* When analyzing .NET applications these registry and file activities should not be considered part of the malware activity. The sample was observed copying itself to *conhost.exe* in different directories and using the API *RegSetValue* for persistence. The communication for this sample was not detected and neither was the data collected, if any.

Sample 5CB07299CEDD69F096B09358754831E0 uses API *IsUserAnAdmin* to determine if the malware has administrative authority and then executes *SetProcessDepPolicy* to allow data execution as well as other API functions. While executing interactively the malware can be observed using API functions such as *GetWindowText* to gather windows form information. The data catured by *GetWindowText* can include, among other things, user ids and passwords. The data is then transmitted to a host name and IP which are identified by MACT.

F54D48B019436E28C1AF5BABF247748B is also .NET assembly. It persists by copying itself and making registry modifications. Although the sample creates a large number of files, what those files contain is unknown. The sample then attempts to communicate with several IPs and continues to cycle through these IPs. It is possible the servers the malware is attempting to communicate no longer exist.

830400121A557B0668AE9374CD847C8B iterates through the file structure using APIs *FindFirstFile* and *FindNextFile* looking for files with specific extensions. The sample makes

numerous copies of itself and registry modifications to persist. Several program scripts and html files are downloaded. Emails are scanned to examine the email addresses in the TO and FROM fields looking for email addresses, possibly specific email addresses. This malware sets the infected host machine up as a bot. Communication attempts are made to several IPs. Some of the IPs may be command and control but there are several commercial IPs contacted. The commercial IPs may indicate obfuscation or DDOS attacks.

2F54B592E62E66FBFE7F7DB24F70F36A uses string compares to locate anti-virus drivers as well as specific programs to terminate. The sample unpacks at least one additional executable. It creates a new process and runs itself through WMIC. MACT was able to follow this execution by intercepting ShellExecuteExW, which the malware uses to create its new process, and modifying the structure containing the execution parameters to inject itself into the new process. Execution on the second stage halted possibly due to undetected anti-analysis performed on by the malware.

Sample 78B661723E5A4DE6446EE585A030E5C1 appears to generate files with text strings of domain names and string fragments to build domain names. The domain names contain the extension for many different countries and the string fragments for public institutions and infrastructure. This sample may be used for DDOS attacks.

7D436F8B7C72498CD5738812D9E0EC61 uses *GetTickCount* as an initial anti-analysis method which was bypassed using MACT return value substitution. The sample then unpacks a new executable, *MBSSCR.exe*, which executes wscript that runs a visual basic script. Persistence is achieved via the visual basic script. Analysis could not continue as *MBSSCR.exe* reaches a point where the only observable behavior is API looping through *GetTickCount*. The cause of this could not be determined using MACT.

Analysis of 0EC2EF48ECCC4E25FA35C59D3CB4A56E was not successful. The sample started with *GetTickCount* anti-analysis which was bypassed using MACT. It was observed comparing its executable file name against a set list of file names until it matched one of the entries on the list. The file name comparison could be an anti-analysis technique. Modifying the return values for the string compares MACT was able to simulate a match. There is an indication from the memory artifacts that iexplorer was used to execute files, but this provided no useful insight and none of the files for execution were found. None of the five malware objectives sought were identified for this sample.

8F7AB3C56E3FF4A6D23C57D0E07A4D1E appears to look for the exact same exectables as the Trojan sample 0EC2EF48ECCC4E25FA35C59D3CB4A56E. Persistence is achieved via unpacking executables and making windows registry modifications. Some of the unpacked executables are executed using the *WinExec* windows API. Using parameter substitution MACT was able to inject itself into the new processes. MACT's file capture, in some cases, fails to successfully save the artifact. To work around this limitation breakpoints were sent, in this case *WinExec*, at which point the file exists and can be copied. Batch (.bat) files were also captured that helped the malware conceal itself by deleting files. Data transmitted and communications, if any were not detected for this sample.

44B5A3AF895F31E22F6BC4EB66BD3EB7 used *GetTickCount* and *GetTickCount64* for anti-analysis. Additionally, a new form of anti-analysis technique in the use of API *QueryPerformanceCounter* was identified in this sample. *QueryPerformanceCounter* is another form of a timing monitor used to detect debugging. MACT was modified to bypass *QueryPerformanceCounter* in a fashion similar to that of *GetTickCoun*t and *GetTickCount64*. As a result, the timing anti-analysis techniques appear to have been bypassed and executing continued. Communications was captured with what appears to be well known commercial addresses intermixed possibly for obfuscation. A method of persistence or type of data captured or sent was not detected.

Malware anti-analysis techniques and the limitations of MACT itself prevented, in some cases, all of the objectives to be met. However, for the samples analyzed, MACT was able to return useful and actionable data in all but one of the samples. Analyzing the artifacts captured by MACT can also aid the further analysis, if necessary, when using other tools by indicating process flow, breakpoints of interest, exploitation methods, and memory activity. In the next chapter conclusions are drawn regarding the testing, what the exact limitations were, and improvements that could be made to improve the performance of MACT.

CHAPTER FIVE

Conclusions and Recommendations

Overview

As previously stated, malware can cause great financial and reputational harm to an organization. Time is critical when resolving a malware infection. The longer it takes to remediate the greater the potential financial and reputational cost to an organization. Malware analysts are tasked with determining the effects malware has on a system and to remove it. There are many tools employed by malware analysts but the existing tools can be general reverse engineering tools, difficult to set up, lack interactive functionality, be time consuming to use, and may not provide adequate code coverage of the malware being analyzed.

This research introduced a new Malware Analysis and Artifact Capture Tool (MACT) that attempts to address the weaknesses of some of the currently used tools. MACT uses API interception to bypass some anti-analysis techniques, provide for quick setup, interactivity in the analysis process, allow for custom code when intercepting a function, logging, and the capture of file, registry, communication, and memory artifacts. The purpose is to provide the analyst with a tool that will help quickly met five malware analysis objectives:

1. Method of persistence.
2. Files added or modified.
3. Registry elements added or modified.
4. Type of information collected.
5. Method of information transference.

To evaluate the effectiveness of MACT in reaching the malware analysis objectives, MACT was used to analyze several malware samples of various classifications. The malware samples were executed in a Windows 7 32-bit virtual machine. During execution MACT was injected via *mact32.dll* into the malware. A separate server program was run concurrently. The server program accepts messages via socket communications from *mact32.dll* for display and

logging. Also, the server provides the mechanism for interaction between the analyst, MACT, and the malware during malware execution. The results of the execution are collected in a SQLite database, text logs, interactive messages to the user, and captured artifacts. The types of artifacts captured include files, registry, memory, and communication.

After the execution of the malware the server logs, MACT logs and API database were studied as well as the file, memory, registry, and communication artifacts to determine if the five objectives could be met. Looking at all the data and artifacts in total generally provided insight into the tested malware's activity. Depending on the sample not all the objective categories were met but often there was enough relevant information available to guide the analyst toward identifying at least some of the major malware system effects. The overview of each sample tested was provided as well as a tabular summary indicating whether each objective was reached. Anti-analysis techniques, when identified, were documented and circumvention was attempted.

Problems and Limitations

Several implementation issues were identified during testing. These issues were determined to be detrimental to the effectiveness of MACT. By identifying these problem areas, it may be possible to increase the effectiveness of the tool. A summary of the issues is external communication points no longer available, limited function set intercepted, anti-analysis techniques employed by the malware, communication detection, "noise" produced by nested calls of intercepted functions, and process execution timing issues.

An attempt was made to use relatively new samples. However, since the tested samples were already detected and studied some of the functionality of the malware may have already been impaired. Remote external communication points may have been already shut down. These external communication points include command and control locations and other locations from which files may have been downloaded. Without command and control the malware may not receive information regarding its execution or experience failed communication when attempting to upload gathered data. Missing supporting locations may also mean certain phases of the malware were not able to execute because the code for these phases could not be downloaded to the infected machine.

The intercepted functions are the method by which all data and artifacts are captured. Malware analysts often attempt to make their programs difficult to analyize and one method they employ to do this is to use unexpected, more obscure, or undocumented functions. Choosing the intercepted function set in such a way as to provide the necessary scope to detect and document API use is critical. Initial testing identified some key modifications to the initial function interception set. This research provided a proof of concept implementation of MACT. A more fully featured MACT with a more exhaustive function interception set could provide a better foundation for capturing data and artifacts providing more and better information by which the malware analysis could be performed.

Anti-analysis techniques, as previously discussed, are used by the malware author to inhibit and delay malware analysis. MACT provides an overview of the malware at an API level. Clues of anti-analysis techniques can be identified at the API level such as *GetTickCount* calls, *Sleep* calls, spawning of processes and others. One weakness, however, is that some malware may require examination of the non-API logical instructions that evaluate the results from API calls and employ other methods of anti-analysis. For example, malware may use the *GetFileSize* API call and depending on the size returned continue executing or stop if the size was not as expected. Without looking at the instructions that are executed it impossible to determine, using MACT, what size the malware was expecting. Without this information it may not be possible to bypass this check.

"Noise" in the use of MACT is defined by the API calls and artifacts that are captured due to nested function calls or other Windows functionality related to the execution of the sample. Some functions that were intercepted called other functions for support which were also in the interception list. These nested function calls provided no insight into the execution of the malware but, by adding data to the log and artifact that needed to be examined, masked the relevant data. Another example is how Windows executes .NET applications from the metadata in the PE file causing registry and dll accesses. This pre-malware Windows execution also created data and artifacts that obscured the execution properties of the malware.

The interplay of at least three processes was at work during the execution of MACT. These processes are the MACT, server, malware, and any malware spawned threads. MACT has limited control of the timing as processes may continue to execute during the interception of a function although the calling process often waits for a response from the called function. The server must

process data as fast or as slowly as it is received and has no control of the other processes. The malware executes, spawns and communicates with threads and other processes. These simultaneously running threads which perform the framework for the analysis introduced several issues. First, malware sometimes uses spawned threads and processes as a form of anti-analysis by using communicating execution timing between them. Second, the server may not be able to process the data it is receiving quickly enough causing data to be overwritten in the socket communications. Third, some malware spawned process may get out of sync with the initial malware if MACT and/or the server increases the time for function completion to such a degree that the communication and execution of related threads is compromised. A more robust server implementation may help remediate a small subset of these issues.

The effectiveness of MACT, like any other tool, rests in large part on the experience, knowledge, and capability of its user. MACT is interactive and is guided by the analyst therefore a certain level of competence is required to extract and identify relevant information. The logs and artifacts must be interpreted by an experienced analyst who is familiar with the general execution methodologies and exploits employed by malware.

Validation Summary

Regardless of the limitations, MACT can be considered as meeting the objectives outlined. It was able to successfully identify some or all the malware analysis objectives for most of the samples. The data and artficats were directly focused toward the task of malware analysis providing a less obstructive view of the functionality of the malware. Many useful file, memory, and registry elements were identified and captured.

Some of the files included data that was captured by the malware, downloaded programs, and copies of the malware useful for persistence. Memory artfacts were captured that contained executables that were extracted and executed by the malware. Some of the memory artifacts consisted of executables previously detected and uploaded to VirustTotal and some were new samples not previously uploaded. Communication attempts were logged along with the IP's and/or domains being contacted. The registry elements helped to determine the persistence method and objectives of the malware.

MACT was successful at bypassing most, if not all, of the timing anti-analysis techniques employed by the malware samples tested. It allowed for bypassing the termination of services or reporting back analyst controllable parameters and return values affecting the execution paths of the running processes. It was also used to insert itself into spawned processes allowing for more in-depth analysis of the malware.

Recommendations for Future Research

Researching and identifying common anti-analysis techniques and the APIs they use would be useful in instrumenting MACT to include a bypass for such techniques. Such informaton would allow for a more complete execution of the malware providing for the collection of more API calls and artifacts for a given malaware sample. The additional information could be used to analyze malware that was not previously possible to study due to the anti-analysis techniques or provide further analysis of malware which had execution paths that were not able to be followed due to the anti-analysis techniques.

Studying the API functions in malware import tables and the APIs dynamically loaded by a large collection of unique malware samples could provide a more comprehensive list of releavent API calls to intercept. Once such a list is created a method could be defined to determine relavence of the API call. Using this list MACT could be modified to intercept a greater number of relevant functions. This would aid in the analysis of the malware by providing more relevant information upon which the analysis can be based. It would also be necessary to periodically review malware API usage trends and update the list of intercepted APIs.

The prototype of MACT used for this research only includes 32-bit Windows APIs and testing used Windows 7 exlusively. To be a more useful tool MACT should be coded to include 64-bit functions and tested on additional Windows operating systems. It would also be possible to implement a similar tool on other operating systems such as Linux. Such an implementation would require a completely different API function list as the supporting APIs will be from that operating system versus Windows.

For ease of use it would be helpful to replace the command line interface with a GUI interface. A GUI would provide for a friendlier interaction. A GUI would also provide a better interface for the automation of functionality that was implemented manually during testing. This

functionality includes *GetTickCount* capture, toggling specific API interception, timing control, and following of spawned processes.

Conclusions

MACT is not designed to replace the currently available tools but rather supplement them by providing a means to get some quick actionable answers with respect to specific malware. Should an organization be affected by malware, MACT could be used to provide quick answers to management for public relations purposes and to begin remediation. Later, a more time consuming and detailed analysis can be performed, if necessary, using reverse engineering tools to provide additional insight into the workings of the malware.

MACT can also be used with other malware analysis tools by performing an initial analysis of a malware sample to provide information that could focus and speed up the use of some of the other tools. By identifying APIs and their parameters an analyst may be able to identify relevant breakpoints for a more detailed analysis performed with a reverse engineering tool. The additional code coverage provided by the interactivity of MACT may also help an analyst identify execution paths of interest. Captured artifacts may be indicative of behavior known to the analyst. Identification of certain anti-analysis techniques may allow the setting of a breakpoint beyond the check allowing execution to continue uninterrupted.

Based on the results from the samples tested, MACT can provide quick and actionable data. With a more robust implementation, involving the interception of more relevant functions, MACT could be even more effective at providing data and artifacts useful for the analysis of malware. MACT, like other tools, had issues with some of the anti-analysis techniques employed by the malware but unlike some of the existing tools was able to bypass certain techniques such as timing checks and process spawning in some cases.

In several of the test cases MACT was able to provide information concerning the method of persistence, files affected, registry elements modified, type of information collected, purpose of the malware and the communication employed by the malware. Using this information, a malware analyst can remove the malware, remove or repair affected files or registry entries, report on the effects of the malware, and block IPs used for communication. Once the malware is dealt with the company can begin to contend with system availability and reputational issues that may be related

to the malware. MACT's fast set up and ease of use reduces, in many of the malware samples tested, can reduce the cost a business incurs due to the malware.

References

Amin, M. (2016). *A Survey of Financial Losses Due to Malware*. Paper presented at the Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, Udaipur, India.

Botacin, M., L, P., #237, Geus, c. D., Andr, #233, . . . gio. (2018). Enhancing Branch Monitoring for Security Purposes: From Control Flow Integrity to Malware Analysis and Debugging. *ACM Trans. Priv. Secur., 21*(1), 1-30. doi:10.1145/3152162

Bulazel, A., #252, & Yener, l. (2017). *A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web*. Paper presented at the Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, Vienna, Austria.

Caron, R. (2000, February 2000). Coding Techniques and Programming Practices. Retrieved from https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx

Christodorescu, M., & Jha, S. (2004). Testing malware detectors. *SIGSOFT Softw. Eng. Notes, 29*(4), 34-44. doi:10.1145/1013886.1007518

Creswell, J. W. (2013). Research design; qualitative, quantitative, and mixed methods approaches, 4th ed. In (Vol. 28). Portland: Ringgold Inc.

Cuckoo. (2018). Cuckoo. Retrieved from https://cuckoosandbox.org/

Dai, C., Pang, J., Zhao, R., & Ma, X. (2008, 14-16 May 2008). *Static Analysis of the Disassembly against Malicious Code Obfuscated with Conditional Jumps.* Paper presented at the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008).

Dinaburg, A., Royal, P., Sharif, M., & Lee, W. (2008). *Ether: malware analysis via hardware virtualization extensions*. Paper presented at the Proceedings of the 15th ACM conference on Computer and communications security, Alexandria, Virginia, USA.

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR), 44*(2), 6.

Gadhiya, S., & Bhavsar, K. (2013). Techniques for malware analysis. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(4).

Galen Hunt, D. B. (1999). Detours: Binary Interception of Win32 Functions. *Third USENIX Windows NT*.

Gao, Y., Lu, Z., & Luo, Y. (2014, 18-20 Aug. 2014). *Survey on malware anti-analysis.* Paper presented at the Fifth International Conference on Intelligent Control and Information Processing.

Greenberg, A. (2017). Your Guide to Russia's Infrastructure Hacking Teams. Retrieved from https://www.wired.com/story/russian-hacking-teams-infrastructure/

Griffin, M. (2013). *Assessment of run-time malware detection through critical function hooking and process introspection against real-world attacks*: The University of Texas at San Antonio.

Han, Y., Hao, Z., Cui, L., Wang, C., & Sang, Y. (2016, 23-26 Aug. 2016). *A Hybrid Monitoring Mechanism in Virtualized Environments.* Paper presented at the 2016 IEEE Trustcom/BigDataSE/ISPA.

Hunt, G., & Brubacher, D. (1999). *Detours: Binary Interception of Win32 Functions.* Paper presented at the Proceedings of the 3rd USENIX Windows NT Symposium. Seattle, Seattle WA.

Jämthagen, C., Lantz, P., & Hell, M. (2013, 30-30 Oct. 2013). *A new instruction overlapping technique for anti-disassembly and obfuscation of x86 binaries.* Paper presented at the 2013 Workshop on Anti-malware Testing Research.

Kono, K., Phomkeona, S., & Okamura, K. (2018, 23-27 July 2018). *An Unknown Malware Detection Using Execution Registry Access.* Paper presented at the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC).

Krishnamoorthy, N., Debray, S., & Fligg, K. (2009, 13-16 Oct. 2009). *Static Detection of Disassembly Errors.* Paper presented at the 2009 16th Working Conference on Reverse Engineering.

Kumar, A., & Goyal, S. Advance Dynamic Malware Analysis Using Api Hooking.

Kunwar, R. S., & Sharma, P. (2016). *Malware Analysis: Tools and Techniques*. Paper presented at the Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, Udaipur, India.

Lawler, R. (2017). Equifax security breach leaks personal info of 143 million US consumers. Retrieved from https://www.engadget.com/2017/09/07/equifax-hack-143-million/

Lee, J., Jeong, K., & Lee, H. (2010). *Detecting metamorphic malwares using code graphs*. Paper presented at the Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland.

Linn, C., & Debray, S. (2003). *Obfuscation of executable code to improve resistance to static disassembly.* Paper presented at the Proceedings of the 10th ACM conference on Computer and communications security.

Lynch, D. J. (2017). Hackers tapped personal information in SEC breach. Retrieved from https://www.ft.com/content/d767d516-a78a-11e7-ab55-27219df83c97?mhq5j=e6

Mariani, B. (2011). Inline hooking in windows. *by High-Tech Bridge SA dated Sep, 6*, 26.

Marpaung, J. A., Sain, M., & Lee, H.-J. (2012). *Survey on malware evasion techniques: State of the art and challenges.* Paper presented at the Advanced Communication Technology (ICACT), 2012 14th International Conference on.

Mehra, M., & Pandey, D. (2015, 17-20 Dec. 2015). *Event triggered malware: A new challenge to sandboxing.* Paper presented at the 2015 Annual IEEE India Conference (INDICON).

Microsoft. (2018a). GetTickCount64 function (Windows). Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/ms724411(v=vs.85).aspx

Microsoft. (2018b). GetTickCount function (Windows). Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/ms724408(v=vs.85).aspx

Microsoft. (2018c). .NET Assembly File Format. Retrieved from https://docs.microsoft.com/en-us/dotnet/standard/assembly-format

Microsoft. (2018d). QueryPerformanceCounter function (Windows). Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx

Mosli, R., Li, R., Yuan, B., & Pan, Y. (2016, 10-11 May 2016). *Automated malware detection using artifacts in forensic memory images.* Paper presented at the 2016 IEEE Symposium on Technologies for Homeland Security (HST).

Nair, V. P., Jain, H., Golecha, Y. K., Gaur, M. S., & Laxmi, V. (2010). *MEDUSA: MEtamorphic malware dynamic analysis usingsignature from API*. Paper presented at the Proceedings of the 3rd international conference on Security of information and networks, Taganrog, Rostov-on-Don, Russian Federation.

Oliveira, D. A. S. d., Crandall, J. R., Wassermann, G., Wu, S. F., Su, Z., & Chong, F. T. (2006). *ExecRecorder: VM-based full-system replay for attack analysis and system recovery*. Paper

presented at the Proceedings of the 1st workshop on Architectural and system support for improving software dependability, San Jose, California.

Ömer, A., & Samet, R. (2017, Oct. 30 2017-Nov. 3 2017). *Investigation of Possibilities to Detect Malware Using Existing Tools.* Paper presented at the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA).

Pandey, S. K., & Mehtre, B. M. (2014, 8-10 May 2014). *Performance of malware detection tools: A comparison.* Paper presented at the 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies.

Patanaik, C. K., Barbhuiya, F. A., & Nandi, S. (2012). *Obfuscated malware detection using API call dependency*. Paper presented at the Proceedings of the First International Conference on Security of Internet of Things, Kollam, India.

Peffers, K. E. N., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems, 24*(3), 45-77.

Prayudi, Y., & Riadi, I. (2015). Implementation of malware analysis using static and dynamic analysis method. *International Journal of Computer Applications, 117*(6).

Research, M. Detours. Retrieved from https://www.microsoft.com/en-us/research/project/detours/

Roundy, K. A., & Miller, B. P. (2013). Binary-code obfuscations in prevalent packer tools. *ACM Comput. Surv., 46*(1), 1-32. doi:10.1145/2522968.2522972

Russinovich, M. (2018). Strings - Windows Sysinternals. Retrieved from https://docs.microsoft.com/en-us/sysinternals/downloads/strings

Shaid, S. Z. M., & Maarof, M. A. (2015, 21-23 April 2015). *In memory detection of Windows API call hooking technique.* Paper presented at the 2015 International Conference on Computer, Communications, and Control Technology (I4CT).

Singh, A., & Bu, Z. (2013). Hot knives through butter: Evading file-based sandboxes. *Threat Research Blog*.

Smith, A. J., Mills, R. F., Bryant, A. R., Peterson, G. L., & Grimaila, M. R. (2014, 19-23 May 2014). *REDIR: Automated static detection of obfuscated anti-debugging techniques.* Paper presented at the 2014 International Conference on Collaboration Technologies and Systems (CTS).

Times, N. Y. (2017). U.S. Tells 21 States That Hackers Targeted Their Voting Systems. Retrieved from https://www.nytimes.com/2017/09/22/us/politics/us-tells-21-states-that-hackers-targeted-their-voting-systems.html

Uppal, D., Mehra, V., & Verma, V. (2014). Basic survey on malware analysis, tools and techniques. *International Journal on Computational Sciences & Applications (IJCSA), 4*(1), 103.

VirtualBox. (2018). VirtualBox. Retrieved from https://www.virtualbox.org/

VirusShare. (2018). VirusShare.com. Retrieved from https://virusshare.com/

Vmware. (2018). VMware – Official Site. Retrieved from https://www.vmware.com

Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D. A., . . . Viola, I. (2015). *A survey of visualization systems for malware analysis.* Paper presented at the EG Conference on Visualization (EuroVis)-STARs.

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*.

Willems, C., Holz, T., & Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy, 5*(2), 32-39. doi:10.1109/MSP.2007.45

Xie, P., Lu, X., Su, J., Wang, Y., & Li, M. (2013, 28-30 Jan. 2013). *iPanda: A comprehensive malware analysis tool.* Paper presented at the The International Conference on Information Networking 2013 (ICOIN).

Xu, B., Sesma, J., Freeman, R., & Li, W. (2006). System for obfuscating computer code upon disassembly. In: Google Patents.

Yakdan, K., Dechand, S., Gerhards-Padilla, E., & Smith, M. (2016, 22-26 May 2016). *Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study.* Paper presented at the 2016 IEEE Symposium on Security and Privacy (SP).

Ye, Y., Wang, D., Li, T., & Ye, D. (2007). *IMDS: intelligent malware detection system*. Paper presented at the Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, San Jose, California, USA.

Times, N. Y. (2017). U.S. Tells 21 States That Hackers Targeted Their Voting Systems. Retrieved from https://www.nytimes.com/2017/09/22/us/politics/us-tells-21-states-that-hackers-targeted-their-voting-systems.html

Uppal, D., Mehra, V., & Verma, V. (2014). Basic survey on malware analysis, tools and techniques. *International Journal on Computational Sciences & Applications (IJCSA), 4*(1), 103.

VirtualBox. (2018). VirtualBox. Retrieved from https://www.virtualbox.org/

VirusShare. (2018). VirusShare.com. Retrieved from https://virusshare.com/

Vmware. (2018). VMware – Official Site. Retrieved from https://www.vmware.com

Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D. A., . . . Viola, I. (2015). *A survey of visualization systems for malware analysis.* Paper presented at the EG Conference on Visualization (EuroVis)-STARs.

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*.

Willems, C., Holz, T., & Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy, 5*(2), 32-39. doi:10.1109/MSP.2007.45

Xie, P., Lu, X., Su, J., Wang, Y., & Li, M. (2013, 28-30 Jan. 2013). *iPanda: A comprehensive malware analysis tool.* Paper presented at the The International Conference on Information Networking 2013 (ICOIN).

Xu, B., Sesma, J., Freeman, R., & Li, W. (2006). System for obfuscating computer code upon disassembly. In: Google Patents.

Yakdan, K., Dechand, S., Gerhards-Padilla, E., & Smith, M. (2016, 22-26 May 2016). *Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study.* Paper presented at the 2016 IEEE Symposium on Security and Privacy (SP).

Ye, Y., Wang, D., Li, T., & Ye, D. (2007). *IMDS: intelligent malware detection system*. Paper presented at the Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, San Jose, California, USA.

Appendix A APIs of Interest

APIs of Interest	
Category	Function
Memory Management	CopyMemory
	MoveMemory
	GetProcessDEPPolicy
	SetProcessDEPPolicy
	CreateFileMapping
	CreateFileMappingFromApp
	CreateFileMappingNuma
	MapViewOfFile
	MapViewOfFIle2
	MapViewOfFileEx
	MapViewOfFileExW
	MapViewOfFileFromApp
	MapViewOfFileNuma2
	OpenFileMapping
	GetProcessHeap
	GetProcessHeaps
	HeapAlloc
	HeapCompact
	HeapFree

	HeapReAlloc
	VirtualAlloc
	VirtualAllocEx
	VirtualAllocNumExNuma
	VirtualAllocFromApp
	VirtualFree
	VirtualFreeEx
	VirtualProtect
	VirtualProtectEx
	VirtualProtectFromApp
	GlobalAlloc
	LocalAlloc
	GlobalHandle
	LocalHandle
	GlobalReAlloc
	LocalReAlloc
Windows Registry Modification	RegConnectRegistry
	RegCopyTree
	RegCreateKeyEx
	RegCreateKeyTransacted
	RegDeleteKey
	RegDeleteKeyEx
	RegDeleteKeyTransacted
	RegDeleteKeyValue
	RegDeleteTree
	RegDeleteValue

	RegEnumKeyEx
	RegEnumValue
	RegFlushKey
	RegGetKeySecurity
	RegGetValue
	RegLoadKey
	RegLoadMUIString
	RegOpenCurrentUser
	RegOpenKeyEx
	RegOpenKeyTransacted
	RegOpenUserClassesRoot
	RegOverridePredefKey
	RegQueryInfoKey
	RegQueryMultipleValues
	RegQueryValueEx
	RegReplaceKey
	RegRestoreKey
	RegSaveKey
	RegSaveKeyEx
	RegSetKeyValue
	RegSetKeySecurity
	RegSetValueEx
	RegUnLoadKey
Process and Thread	CreateProcess
	CreateProcessAsUser
	CreateProcessWithLogonW

	CreateProcessWithTokenW
	ExitProcess
	GetCurrentProcess
	GetCurrentProcessId
	GetProcessId
	GetProcessIdOfThread
	GetProcessPriorityBoost
	OpenProcess
	SetEnvironmentVariable
	SetPriorityClass
	SetProcessInformation
	SetProcessRestrictionExemption
	TerminateProcess
	EnumProcesses
	Process32First
	Process32Next
	STSEnumerateProcesses
	QueryProtectedPolicy
	SetProtectedPolicy
	CreateRemoteThread
	CreateRemoteThreadEx
	CreateThread
	ExitThread
	GetCurrentThread
	GetCurrentThreadId
	GetThreadId

	GetThreadInformation
	GetThreadPriorityBoost
	OpenThread
	ResumeThread
	SetThreadPriority
	SetThreadPriorityBoost
	Sleep
	SleepEx
	SuspendThread
	SwitchToThread
	TerminateThread
	DeleteProcThreadAttributeList
	InitializeProcThreadAttributeList
	UpdateProcThreadAttribute
File Functions	CopyFile
	CopyFile2
	CopyFileEx
	CopyFileTransacted
	CreateFile
	CreateFile2
	CreateFileTransacted
	DecryptFile
	DeleteFile
	DeleteFileTransacted
	EncryptFile
	FindFirstFile

	FindFirstFileEx
	FindNextFile
	GetTempFileName
	GetTempPath
	LZCopy
	LZInit
	LZOpenFile
	MoveFile
	MoveFileEx
	MoveFileTransacted
	OpenFile
	OpenFileById
	ReplaceFile
	SearchPath
	SetFileAttributes
	SetFileAttributesTransacted
External Communication	accept
	AcceptEx
	Bind
	connect
	ConnectEx
	GetAddressByName
	GetAddrInfoEx
	GetAddrInfoW
	gethostbyaddr
	gethostbyname

	GetHostnameW
	GetNameByType
	getnameinfo
	GetNameInfoW
	listen
	recv
	recvfrom
	SetService
	send
	sendto
	socket
	TransmitFile
	TransmitPackets
Dynamic-Link Library	AddDllDirectory
	DllMain
	GetModuleFileName
	GetModuleFileNameEx
	GetModuleHandle
	GetModuleHandleEx
	GetProcAddress
	LoadLibrary
	LoadLibraryEx
	RemoveDllDirectory
	SetDefaultDllDirectories
	SetDllDirectory

Table 13 APIs of Interest

Appendix B Error Messages

B1. Error: 0001A-001

Message: Invalid command entered.

Source: Command Validator/Parser

Description: An invalid verb/noun combination was entered.

Remediation: Enter a valid command. Valid commands can be identified with the "D C" command.

B2. Error: 0001A-002

Message: Invalid command parameters entered for current API function. The valid parameter list for function <function name> is: <Valid Parameter List>.

Source: Command Validator/Parser

Description: An incorrect list of parameters was entered. The numbers of parameters, types, or length are in error.

Remediation: Enter a correct parameter list. Please refer to the error message to identify the current function and valid parameter list.

Appendix C Modified APIs Intercept Functions

APIs of Interest	
Category	Function
Memory Management	CoTaskMemAlloc
	CoTaskMemFree
	HeapCreate
	VirtualAlloc
	VirtualAllocEx
	VirtualFree
	VirtualFreeEx
	VirtualProtect
	VirtualProtectEx
Windows Registry Modification	RegCreateKeyEx
	RegEnumKeyExA
	RegEnumKeyExW
	RegGetValueA
	RegGetValueW
	RegOpenKeyA
	RegOpenKeyEx
	RegOpenKeyExA
	RegOpenKeyExW
	RegQueryValueEx
	RegSetValueA
	RegSetValueEx

	RtlCreateRegistryKey
	RtlWriteRegistryValue
Process and Thread	AttachThreadInput
	ControlService
	CreateProcess
	CreateProcessW
	CreateRemoteThread
	CreateRemoteThreadEx
	CreateService
	EnumProcesses
	EnumProcessModules
	EnumProcessModulesEx
	GetThreadContext
	OpenProcess
	Process32First
	Process32FirstW
	Process32Next
	Process32NextW
	QueueUserAPC
	ReadProcessMemory
	SetThreadContext
	TerminateProcess
	ResumeThread
	SfcTerminateWatcherThread
	StartServiceCtrlDispatcherA
	StartServiceCtrlDispatcherA

	SuspendThread
	Thread32First
	Thread32Next
	Toolhelp32ReadProcessMemory
	WriteProcessMemory
File Functions	CloseHandle
	CopyFileA
	CopyFileW
	CopyFileExA
	CopyFileExW
	CreateFileW
	CreateFileA
	CreateFileMappingA
	DeleteFileA
	DeleteFileW
	FindFirstFile
	FindFirstFileEx
	FindNextFile
	FlushFileBuffers
	GetFileSize
	GetTempPathA
	GetWindowsDirectory
	MapViewOfFile
	MapViewOfFileEx
	SetFileTime
	WriteFile

	WriteFileEx
External Communication	accept
	bind
	connect
	ConnectNamedPipe
	FtpOpenFileW
	FtpPutFile
	getaddrinfo
	gethostbyname
	gethostname
	HttpOpenRequestW
	HttpSendRequestW
	HttpSendRequestExW
	inet_addr
	InternetOpen
	InternetOpenW
	InternetConnectW
	InternetOpenUrl
	InternetOpenUrlA
	InternetReadFile
	InternetWriteFile
	PeekNamedPipe
	recv
	send
	URLDownloadToFile
	URLDownloadToFileA

	WSASend
	WSAStartup
Dynamic-Link Library	FindResourceA
	FindResourceExA
	GetModuleFileName
	GetModuleFileNameExA
	GetModuleFileNameExW
	GetModuleHandle
	GetModuleHandleEx
	GetProcAddress
	LoadLibrary
	LoadLibraryExA
	LdrLoadDll
	LoadResource
	Module32First
	Module32Next
Miscellanous Categories	GetTickCount
	GetTickCount64
	QueryPerformanceCounter
	Sleep
	SleepEx
	ShellExecuteW
	ShellExecuteExA
	ShellExecuteExW
	_wsystem
	system
	WinExec

	lstrcmpiA
	lstrcmpiW
	lstrcmpW
	CompareStringEx
	AdjustTokenPrivileges
	IsNTAdmin
	IsUserAnAdmin
	SamIConnect
	BitBlt
	CreateToolhelp32Snapshot
	DeviceIoControl
	FindWindow
	FindWindowEx
	GetAdaptersInfo
	GetConsoleWindow
	GetDC
	GetKeyState
	GetSystemDefaultLangID
	GetForegroundWindow
	GetWindowText
	MapVirtualKeyA
	MapVirtualKeyExA
	MapVirtualKeyW
	MapVirtualKeyExW
	RegisterHotKey
	SetWindowsHookEx
	WideCharToMultiByte
	GetStartupInfoA

	GetVersionEx
	IsWow64Process
	LsaEnumerateLogonSessions
	SetProcessDEPPolicy
	OutputDebugString
	OutputDebugStringA
	OutputDebugStringW
	CertOpenSystemStore
	CryptAcquireContextA
	CryptAcquireContextW
	CreateMutex
	CreateMutexEx
	GetAsyncKeyState
	OpenMutexA

Table 14 Final API Intercept Functions

Appendix D serverc.cpp

```
//*********************************************************************************************
// Program: serverc.exe serverc.cpp
//
// Description: This is the MACT server.
//
// Functionality:
//                  Communicates with MACT
//                  Creates Logs
//                  Creates database of API calls
//                  Allows the analyst interaction to MACT
//
//*********************************************************************************************
#undef UNICODE

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>
#include <algorithm>
#include <ctime>
#include <strsafe.h>
#include <chrono>
#include <future>
#include <c:\sqlite\sqlite3.h>
/*
To create the lib file for linking I had to execute:
lib /def:sqlite3.def /out:sqlite3.lib
*/
#pragma comment (lib, "Ws2_32.lib")
#pragma comment(lib, "c:\\sqlite\\sqlite3.lib")

static BOOL MACTSTART  = FALSE;
static BOOL MACTFINISH = FALSE;
static BOOL MACTDEBUG  = FALSE;
static int MACTCallSequence = 0;
static HANDLE SR_HANDLE;
static BOOL   SR_BOOL;
static LPVOID SR_LPVOID;

static std::string MACTdir;
```

```
static std::string MACTdbname;
std::string MACTAPISkip[6];
std::string MACTbp[100];
int iMACTbp = 0;

SOCKET Socket;

//*************************************************************************************************
//
// Function   : callback
// Description: sqlite error callback
//
//*************************************************************************************************
static int callback(void *NotUsed, int argc, char **argv, char **azColName) {
   int i;
   for(i = 0; i<argc; i++) {
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
   printf("\n");
   return 0;
}

//*************************************************************************************************
//
// Function   : MACTCreateDatabase
// Description: Creates SQL database to hold API table.
//
//*************************************************************************************************
void MACTCreateDatabase()
{
    static sqlite3 *MACTdb;
    char *zErrMsg = 0;
    int rc;
    char *sql;

    MACTdbname = MACTdir + "\\MACT.db";

    rc = sqlite3_open(MACTdbname.c_str(), &MACTdb);

    if(rc) {
        printf("Can't open database: %s\n", sqlite3_errmsg(MACTdb));
        exit(0);
    } else {
        printf("Opened database successfully\n");
    }

    /* Create SQL statement */
```

```
    sql = "CREATE TABLE MACT("  \
          "CALLSEQ INT  PRIMARY KEY NOT NULL," \
          "CALLRAW TEXT NOT NULL," \
          "APINAME TEXT NOT NULL," \
          "APIP01 TEXT NOT NULL," \
          "APIP02 TEXT NOT NULL," \
          "APIP03 TEXT NOT NULL," \
          "APIP04 TEXT NOT NULL," \
          "APIP05 TEXT NOT NULL," \
          "APIP06 TEXT NOT NULL," \
          "APIP07 TEXT NOT NULL," \
          "APIP08 TEXT NOT NULL," \
          "APIP09 TEXT NOT NULL," \
          "APIP10 TEXT NOT NULL," \
          "APIP11 TEXT NOT NULL," \
          "APIP12 TEXT NOT NULL," \
          "APIRET TEXT NOT NULL," \
          "APIOV TEXT NOT NULL," \
          "APIOVRET TEXT NOT NULL);";

    /* Execute SQL statement */
    rc = sqlite3_exec(MACTdb, sql, callback, 0, &zErrMsg);

    if(rc != SQLITE_OK){
        printf("SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
        exit(0);
    } else {
        printf("Table created successfully\n");
    }

    sqlite3_close(MACTdb);
}

//*************************************************************************************************
//
// Function   : MACTInsertDatabase
// Description: Adds rows to API Table.
//
//*************************************************************************************************
void MACTInsertDatabase(char* buffer)
{
    static sqlite3 *MACTdb;
    char *zErrMsg = 0;
    int rc;
    char *sql;
```

```
std::string sMACTCallSequence = std::to_string(MACTCallSequence);
std::string sbuffer;

switch(buffer[0]) {
    case '*' :
        sbuffer = buffer;
        sbuffer = sbuffer.substr(1,(strlen(buffer)-1));
        ++MACTCallSequence;
        break;
    case '-' :
        return;
    case '+' :
        return;
    case '>' :
        return;
    case ':' :
        return;
    default :
        printf("Error!!! : %s\n", buffer);
        sbuffer = buffer;
        ++MACTCallSequence;
        break;
}

/* Open database */

rc = sqlite3_open(MACTdbname.c_str(), &MACTdb);

if(rc) {
    printf("Can't open database: %s\n", sqlite3_errmsg(MACTdb));
    exit(0);
}

std::string sSource = sbuffer;

std::replace(sSource.begin(), sSource.end(), '(', ',');
std::replace(sSource.begin(), sSource.end(), ')', ',');


size_t pos = 0;
std::string token;
std::string sTokens[16];
int iToken = 0;
while((pos = sSource.find(',')) != std::string::npos) {
    token = sSource.substr(0, pos);
    sTokens[iToken] = token;
    ++iToken;
```

```
        sSource.erase(0, pos + 1);
    }

    for(int i = 15; i > 4; --i) {
        if(sTokens[i] != "") {
            sTokens[15] = sTokens[i];
            sTokens[14] = sTokens[i-1];
            sTokens[13] = sTokens[i-2];
            if(i < 13)
                sTokens[i] = "";
            if((i-1) < 13)
                sTokens[i-1] = "";
            if((i-2) < 13)
                sTokens[i-2] = "";
            break;
        }
    }


    /* Create SQL statement */

      std::string ssql = "INSERT INTO MACT (CALLSEQ, CALLRAW, APINAME, APIP01, APIP02, "   \
                                           "APIP03, APIP04, APIP05, APIP06, APIP07, "     \
                                           "APIP08, APIP09, APIP10, APIP11, APIP12, "     \
                                           "APIRET, APIOV, APIOVRET) "                   \
                         "VALUES (" + sMACTCallSequence + ", '" + sbuffer.c_str() + "', '" +  \
                                  sTokens[0].c_str()  + "', '"  + sTokens[1].c_str() + "', '" +  \
                                  sTokens[2].c_str()  + "', '"  + sTokens[3].c_str() + "', '" +  \
                                  sTokens[4].c_str()  + "', '"  + sTokens[5].c_str() + "', '" +  \
                                  sTokens[6].c_str()  + "', '"  + sTokens[7].c_str() + "', '"+  \
                                  sTokens[8].c_str()  + "', '"  + sTokens[9].c_str() + "', '" + \
                                  sTokens[10].c_str() + "', '"  + sTokens[11].c_str() + "', '" + \
                                  sTokens[12].c_str() + "', '"  + sTokens[13].c_str() + "', '" + \
                                  sTokens[14].c_str() + "', '"  + sTokens[15].c_str() + "');";

    /* Execute SQL statement */
    rc = sqlite3_exec(MACTdb, ssql.c_str(), callback, 0, &zErrMsg);

    if(rc != SQLITE_OK){
        printf("SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
        exit(0);
    }

    sqlite3_close(MACTdb);
}
```

```
//*************************************************************************************************
//
// Function   : ishex
// Description: Determints if string is a valid hex string.
//
//*************************************************************************************************
BOOL ishex(std::string HexString)
{
    if(HexString.length() == 0)
        return FALSE;

    if((HexString[0] == '0') && (HexString[1] == 'X'))
        HexString = HexString.substr(2, HexString.length());

    if(HexString.length() > 8)
        return FALSE;

    for(size_t i=0; i < HexString.length(); i++)
        if(!isxdigit(static_cast<unsigned char>(HexString[i])))
            return FALSE;

    return TRUE;
}

//*************************************************************************************************
//
// Function   : MACTWriteLog
// Description: Write data passed from MACT to the serverlog
//
//*************************************************************************************************
void MACTWriteLog(char* buffer)
{
    FILE * pFile;
    std::string sFilename = MACTdir + "\\serverlog.txt";

    pFile = fopen(sFilename.c_str(), "a");

    fwrite(buffer, sizeof(char), strlen(buffer), pFile);

    fclose(pFile);
}

//*************************************************************************************************
//
// Function   : MACTSendSocket
// Description: Send data to MACT via the socket.
//
```

```
//*************************************************************************************************
void MACTSendSocket(std::string* sParsedTokens)
{

        fd_set WriteFDs;
        FD_ZERO(&WriteFDs);
        FD_SET(Socket, &WriteFDs);
        char buffer[80];
        std::string strBuffer;

        if(select(0, NULL, &WriteFDs, NULL, 0) > 0) {
            if (FD_ISSET(Socket, &WriteFDs)) {

                memset(buffer, 0, 80);
                buffer[79] = '\0';
                strBuffer   =   sParsedTokens[0]   +   sParsedTokens[1]   +   sParsedTokens[2]   +
sParsedTokens[3];
                strncpy(buffer, strBuffer.c_str(), 80);
                buffer[79] = '\0';
                if(send(Socket, buffer, 80, 0) == SOCKET_ERROR)
                    std::cout<<"Socket error on write.\n";
            }
            else {
                printf("Error Write FD_ISSET\n");
            }
        }
        else {
            printf("Error on select write.\n");
        }
}

//*************************************************************************************************
//
// Function   : DCCommand
// Description: Displays the available MACT commands.
//
//*************************************************************************************************
void DCCommand()
{
    std::cout << "\n\nValid Commands:\n\n";
    std::cout << "B A                                 Breakpoint add.\n";
    std::cout << "B C                                 Breakpoint clear.\n";
    std::cout << "B D                                 Breakpoint delete.\n";
    std::cout << "B L                                 Breakpoint list.\n";
    std::cout << "C                                   Continue executing application.\n";
    std::cout << "C E                                 Continue to end of API.\n";
    std::cout << "D C                                 Display valid commands.\n";
```

```
    std::cout << "D S                                  Display memory construct structure.\n";
    std::cout << "D M <Address> <Length>               Display memory at address.\n";
    std::cout << "M A <Address> <Length>               Get and write memory from location.\n";
    std::cout << "M S <Address> <Length> <FileName>   Initiate the substitution of the an artifact
from the specified File Name to the Address and of the Length specified.\n";
    std::cout << "S P <Parameters>                    Substitute parameters for function call.\n";
    std::cout << "S R <Return Value>                       Substitute return values from function
call.\n";
    std::cout << "\n\n";
}

//*****************************************************************************************
//
// Function   : ExecuteMACTCommand
// Description: Processes the entered command.
//
//*****************************************************************************************
BOOL ExecuteMACTCommand(char *sType, std::string* ParsedTokens)
{

    char sSwitch[2];
    strncpy(sSwitch, ParsedTokens[0].c_str(), 1);

    switch(sSwitch[0]) {
        case 'B' :
            if((ParsedTokens[1]  ==  "A")  &&  (ParsedTokens[2]  !=  "")  &&  ((ParsedTokens[3]  +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                std::cout << "Breakpoint Add\n";
            }
            else if((ParsedTokens[1] == "D") && (ParsedTokens[2] != "") && ((ParsedTokens[3] +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                            std::cout << "Breakpoint Delete\n";
            }
            else  if((ParsedTokens[1]  ==  "C")  &&  ((ParsedTokens[2]  +  ParsedTokens[3]  +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                std::cout << "Breakpoint Clear\n";
            }

            else  if((ParsedTokens[1]  ==  "L")  &&  ((ParsedTokens[2]  +  ParsedTokens[3]  +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                std::cout << "Breakpoint List\n";
            }
            else {
                std::cout << "Invalid Command B.\n";
                return FALSE;
            }
            break;
```

```
        case 'C' :
            if((ParsedTokens[1] + ParsedTokens[2] + ParsedTokens[3] + ParsedTokens[4] +
ParsedTokens[5]) == "") {
                if(MACTDEBUG)
                    std::cout << "Continue executing application.\n";
            }
            else if((ParsedTokens[1] == "E") && ((ParsedTokens[2] + ParsedTokens[3] +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                MACTFINISH = TRUE;
                std::cout << "Continue to end..\n";
            }
            else {
                std::cout << "Invalid Command C.\n";
                return FALSE;
            }
            break;
        case 'D' :
            if((ParsedTokens[1] == "B") && ((ParsedTokens[2] + ParsedTokens[3] + ParsedTokens[4] +
ParsedTokens[5]) == "")) {
                std::cout << "Display Breakpoints.\n";
            }
            else if((ParsedTokens[1] == "C") && ((ParsedTokens[2] + ParsedTokens[3] +
ParsedTokens[4] + ParsedTokens[5]) == ""))
                DCCommand();
            else if((ParsedTokens[1] == "M") && (ishex(ParsedTokens[2]) && ishex(ParsedTokens[3]))
&& ((ParsedTokens[4] + ParsedTokens[5]) == "")) {
                if(MACTDEBUG)
                    std::cout << "Display memory at address.\n";

                std::string sString = ParsedTokens[2];
                if((sString[0] == '0') && (sString[1] == 'X'))
                    sString = sString.substr(2, sString.length());
                sString.insert (0, 8 - sString.length(), '0');
                ParsedTokens[2] = sString;

                sString = ParsedTokens[3];
                if((sString[0] == '0') && (sString[1] == 'X'))
                    sString = sString.substr(2, sString.length());
                sString.insert (0, 8 - sString.length(), '0');
                ParsedTokens[3] = sString;
            }
            else if((ParsedTokens[1] == "S") && ((ParsedTokens[2] + ParsedTokens[3] +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                if(MACTDEBUG)
                    std::cout << "Display memory construct structure.\n";
            }
            else {
```

```
                std::cout << "Invalid Command D.\n";
                return FALSE;
            }
            break;
        case 'M' :
            if((ParsedTokens[1] == "A") && (ishex(ParsedTokens[2]) && ishex(ParsedTokens[3])) &&
((ParsedTokens[4] + ParsedTokens[5]) == "")) {
                std::cout << "Initiate the serialization of an artifact using the address and
length specified.\n";

                std::string sString = ParsedTokens[2];
                if((sString[0] == '0') && (sString[1] == 'X'))
                    sString = sString.substr(2, sString.length());
                sString.insert (0, 8 - sString.length(), '0');
                ParsedTokens[2] = sString;

                sString = ParsedTokens[3];
                if((sString[0] == '0') && (sString[1] == 'X'))
                    sString = sString.substr(2, sString.length());
                sString.insert (0, 8 - sString.length(), '0');
                ParsedTokens[3] = sString;
            }
            else {
                std::cout << "Invalid Command M.\n";
                return FALSE;
            }
            break;
        case 'S' :
            if((ParsedTokens[1] == "R") && (ishex(ParsedTokens[2]) && (ParsedTokens[3] +
ParsedTokens[4] + ParsedTokens[5]) == "")) {
                if(sType == "HANDLE") {
                    UINT cValue;
                    sscanf(ParsedTokens[2].c_str(), "%x", &cValue);
                    SR_HANDLE = (HANDLE)cValue;
                    printf("=======>%p\n", SR_HANDLE);
                }
                else if(sType == "BOOL") {
                    BOOL cValue;
                    sscanf(ParsedTokens[2].c_str(), "%d", &cValue);
                    SR_BOOL = (BOOL)cValue;
                    printf("=======>%d\n", SR_BOOL);
                }
            }
            else {
                std::cout << "Invalid Command S.\n";
                return FALSE;
            }
```

```
            break;
        case 'Q' :
            std::cout << "Quitting.\n";
            exit(0);
            break;
        default :
            std::cout << "Invalid Command X.\n";
            return FALSE;
    }


    return TRUE;

}

//*****************************************************************************************
//
// Function   : MACTGetCommand
// Description: Prompt for user input.
//
//*****************************************************************************************
std::string* MACTGetCommand()
{
    if(MACTDEBUG)
        printf("serverb: In MACTGetCommand\n");

    std::string RawTokens;
    std::string* ParsedTokens = new std::string[6];
    int CurrentToken;

    printf("MACT>> ");

    getline(std::cin, RawTokens);
    std::transform(RawTokens.begin(), RawTokens.end(), RawTokens.begin(), toupper);
    RawTokens += " ";

    CurrentToken = 0;
    size_t pos = 0;
    std::string token;
    while ((pos = RawTokens.find(" ")) != std::string::npos) {
        token = RawTokens.substr(0, pos);
        ParsedTokens[CurrentToken] = token + "\0";
        if(CurrentToken < 5)
            ++CurrentToken;
        RawTokens.erase(0, pos + 1);
    }
```

```
    return ParsedTokens;
}

//*************************************************************************************************
//
// Function   : MACTMain
// Description: Control interface.
//
//*************************************************************************************************
int MACTMain(char* sType)
{
    if(MACTDEBUG)
        printf("serverb: In MACTMAIN\n");

    if(MACTFINISH){
        printf("serverb: In MACTMAIN MACTFINISH\n");
        return 1;
    }

    if(MACTDEBUG)
        printf("serverb: In MACTMAIN point A\n");

    std::string* ParsedTokens = new std::string[6];


    if(strncmp(sType, "CONTINUE", 8) == 0) {
        ParsedTokens[0] = 'C';
        MACTSendSocket(ParsedTokens);
    }
    else {
        BOOL bValid = FALSE;
        while(!bValid) {
            ParsedTokens = MACTGetCommand();
            bValid = ExecuteMACTCommand(sType, ParsedTokens);
        }
        MACTSendSocket(ParsedTokens);
    }


    int ret = 0;

    delete[] ParsedTokens;

    return(ret);
}

//*************************************************************************************************
```

```
//
// Function   : MACTCleanUP
// Description: Clean up any threads before terminating.
//
//*****************************************************************************************
void MACTCleanUp()
{
    exitSignal.set_value();

    for(int i=0; i<THREADCOUNT; i++)
    {
        CloseHandle(hThreadArray[i]);
        if(pDataArray[i] != NULL)
        {
            HeapFree(GetProcessHeap(), 0, pDataArray[i]);
            pDataArray[i] = NULL;    // Ensure address is not reused.
        }
    }
}

//*****************************************************************************************
//
// Function   : main
// Description: Establish socket, prompt for commands, execute commands, terminate.
//
//*****************************************************************************************
void main() {
    WSADATA WsaDat;
    char buffer[512];
    char lastbuffer[512];
    int inDataLength = 0;
    std::string slastbuffer;

    lastbuffer[0]  = '\0';
    MACTAPISkip[0] = "CLOSEHANDLE";

    if(WSAStartup(MAKEWORD(2,2),&WsaDat)!=0) {
        std::cout<<"WSA Initialization failed!\r\n";
        WSACleanup();
        system("PAUSE");
        return;
    }

    Socket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    if(Socket==INVALID_SOCKET) {
        std::cout<<"Socket creation failed.\r\n";
        WSACleanup();
```

```
    system("PAUSE");
    return;
}

SOCKADDR_IN serverInf;
serverInf.sin_family=AF_INET;
serverInf.sin_addr.s_addr=INADDR_ANY;
serverInf.sin_port=htons(27015);

if(bind(Socket,(SOCKADDR*)(&serverInf),sizeof(serverInf))==SOCKET_ERROR) {
    std::cout<<"Unable to bind socket!\r\n";
    WSACleanup();
    system("PAUSE");
    return;
}

listen(Socket, 1);

SOCKET TempSock=SOCKET_ERROR;
while(TempSock==SOCKET_ERROR) {
    std::cout<<"Waiting for incoming connections...\r\n";
    TempSock=accept(Socket,NULL,NULL);
}

// If iMode!=0, non-blocking mode is enabled.
u_long iMode=1;
ioctlsocket(Socket,FIONBIO,&iMode);

Socket=TempSock;
std::cout<<"Client connected!\r\n\r\n";
bool selected = FALSE;
std::string* ParsedTokens = new std::string[6];
ParsedTokens[0] = "C";

for(;;) {

    for(;;) {

        fd_set ReadFDs;
        FD_ZERO(&ReadFDs);
        FD_SET(Socket, &ReadFDs);
        inDataLength = 0;

        if(select(0, &ReadFDs, NULL, NULL, 0) > 0) {
            if (FD_ISSET(Socket, &ReadFDs)) {
                if(MACTDEBUG)
                    printf("serverb: read\n");
```

```
                    memset(buffer, 0, 512);
                    inDataLength = recv(Socket, buffer, 512, 0);
                    if(inDataLength > 0) {
                        buffer[511] = '\0';
                        if(MACTDEBUG) {
                            printf("serverb: Got bytes: %d\n", inDataLength);
                            printf("serverb: Got: %s\n", buffer);
                        }

// Comms may get off due to functions that fail, this resets the communication if two MACTSTARTS
in a row are sent.
                        if((strncmp(lastbuffer, "MACTSTART", 9) == 0) &&
                           (strncmp(buffer, "MACTSTART", 9) == 0)) {
                            strncpy(buffer, ">Resetting comms.\n", 18);
                            MACTSendSocket(ParsedTokens);
                        }
                        else
                            strncpy(lastbuffer, buffer, 512);

                        if(strncmp(buffer, "MACTSTART", 9) == 0) {
                            MACTFINISH = FALSE;
                            break;
                        } else if(strncmp(buffer, "MACTEXIT", 9) == 0) {
                            exit(0);
                        } else if(strncmp(buffer, "MACTINIT", 7) == 0) {
                            printf("First time start up!\n");
                            MACTdir = buffer;
                            size_t idirlength = MACTdir.length() - 9;
                            MACTdir = MACTdir.substr(9, idirlength);
                            MACTCreateDatabase();
                        } else {
                            if(buffer[0] == '=') {
                                std::string sbuffer = buffer;
                                size_t ibuffer = sbuffer.length() - 1;
                                sbuffer = sbuffer.substr(1, ibuffer);
                                memset(buffer, 0, 512);
                                strncpy(buffer, sbuffer.c_str(), ibuffer);
                            }
                            else {
                                MACTWriteLog(buffer);
                                MACTInsertDatabase(buffer);
                            }
                            if(buffer[0] != '*')
                                printf("%s", buffer);
                        }
                    }
                    else {
```

```
                    buffer[0] = '\0';
                    printf("Done.\n");
                    MACTCleanUp();
                    closesocket(Socket);
                    WSACleanup();
                    return;
                }
            }
            else {
                printf("Error FD_ISSET\n");
                MACTCleanUp();
                exit(0);
            }
        }
        else {
            printf("Error on select.\n");
            MACTCleanUp();
            exit(0);
        }

    }

    MACTMain("TEST");
  }
}
```

Appendix E mact.cpp

```
//*********************************************************************************************
// Library: mact.dll mact.cpp
//
// Description: This dll is injected into a process and used to intercept specific Windows API
//              calls.
//
// Functionality:
//                  Communicates with a server application.
//                  Saves logs regarding APIs called, parameters, and return values.
//                  Allows the programmer to override return values.
//                  Saves artifacts such as registry, file, memory and communication.
//
//*********************************************************************************************
//*********************************************************************************************
//
// Define the required libraries.
//
//*********************************************************************************************
#pragma comment(lib, "Ws2_32.lib")
#pragma comment(lib, "shell32.lib")
#pragma comment(lib, "user32.lib")
#pragma comment(lib, "advapi32.lib")
#pragma comment(lib, "crypt32.lib")
#pragma comment(lib, "gdi32.lib")
#pragma comment(lib, "Iphlpapi.lib")
#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "Ole32.lib")
#pragma comment(lib, "samsrv.lib")
#pragma comment(lib, "Secur32.lib")
#pragma comment(lib, "Urlmon.lib")
#pragma comment(lib, "Wininet.lib")

//*********************************************************************************************
//
// Identify the include members.
//
//*********************************************************************************************
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
#include <winreg.h>
#include <C:\\detours\\include\\detours.h>
#include <algorithm>
#include <stdlib.h>
#include <iostream>
```

```
#include <tchar.h>
#include <atlconv.h>
#include <ctime>
#include <chrono>
#include <future>
#include <psapi.h>
#include <strsafe.h>
#include <fcntl.h>
#include <fstream>
#include <Iphlpapi.h>
#include <sys/stat.h>
#include <io.h>
#include <locale>
#include <codecvt>
#include <ntsecapi.h>
#include <Objbase.h>
#include <Shlobj.h>
#include <subauth.h>
#include <ws2tcpip.h>
#include <TlHelp32.h>
#include <Urlmon.h>
#include <vector>
#include <Wincrypt.h>
#include <Wininet.h>
#include <winsock2.h>
#include <mscoree.h>
#include "verify.cpp"

//*********************************************************************************************
//
// Define global variables that need to exist between API calls.
//
//*********************************************************************************************

// Default lengths for socket communication.
#define BUFSIZE 512

// Create Socket variable.
SOCKET ConnectSocket = INVALID_SOCKET;

// Variables containing paths for artifacts.
static std::string MACTdir;
static std::string MACTdirFilesClosed;
static std::string MACTdirFilesDeleted;
static std::string MACTdirFilesMapped;
static std::string MACTdirFilesCreated;
static std::string MACTdirMem;
```

```
// Define variables that keep track of specific states between API calls.
static BOOL  fBroke          = FALSE;
static BOOL  fLog            = FALSE;
static LONG  dwSlept         = 0;
static BOOL  MACTFINISH      = FALSE;
static BOOL  MACTDEBUG       = FALSE;
static BOOL  MACTDEBUG2      = FALSE;
static BOOL  MACTSTARTED     = FALSE;
static BOOL  MACTVERBOSE     = TRUE;
static BOOL  MACTBP          = FALSE;
static BOOL  MACTSEND        = FALSE;
static int   MACTMSGS        = 0;
static int   MACTWRITE       = TRUE;
static BOOL  MACTWIN7        = TRUE;
static int   MACTTICKCOUNT   = 0;
static int   MACTTICKNUM     = 0;
static int   MACTTICKCOUNT64 = 0;
static int   MACTTICKNUM64   = 0;
static int   MACTQPCCOUNT = 0;
static int   MACTQPCNUM      = 0;
static DWORD MACTTICKADJ     = 0;

static std::vector<int> vTicks;
static std::vector<int> vTicks64;
static std::vector<LARGE_INTEGER> vQPC;


// Memory allocation definition.
typedef struct Mem {
    LPVOID            Mem_address;
    size_t            Mem_size;
    int               Mem_type;
    int               Mem_interval;
    std::future<void> Mem_futureObj;
} MEMDATA, *PMEMDATA;
PMEMDATA pDataArray[8096];
DWORD    dwThreadIdArray[8096];
HANDLE   hThreadArray[8096];

// Thread information.  Threads are spawned to tracked changes in memory allocations.
int      THREADCOUNT = 0;
std::promise<void> exitSignal;

// Global variables used for return value substitution.
static HANDLE      SR_HANDLE;
static BOOL        SR_BOOL;
```

```
static LPVOID      SR_LPVOID;
static UINT        SR_UINT;
static HINSTANCE   SR_HINSTANCE;
static LONG        SR_LONG;
static HCERTSTORE  SR_HCERTSTORE;
static SC_HANDLE   SR_SC_HANDLE;
static HRSRC       SR_HRSRC;
static HWND        SR_HWND;
static ULONG       SR_ULONG;
static SHORT       SR_SHORT;
static HDC         SR_HDC;
static hostent     SR_HOSTENT;
static INT         SR_INT;
static DWORD       SR_DWORD;
static HMODULE     SR_HMODULE;
static FARPROC     SR_FARPROC;
static LANGID      SR_LANGID;
static HINTERNET   SR_HINTERNET;
static NTSTATUS    SR_NTSTATUS;
static HGLOBAL     SR_HGLOBAL;
static LSTATUS     SR_LSTATUS;
static HHOOK       SR_HHOOK;
static HRESULT     SR_HRESULT;
static SOCKET      SR_SOCKET;
static ULONGLONG   SR_ULONGLONG;
//26


// Memory construct definition.
struct MACTVA {
    LPVOID       MACTVAAddress;
    SIZE_T       MACTVASize;
    CHAR         MACTVAType;
    std::string  MACTVAStatus;
    DWORD        MACTVAProtect;
};

static const INT MAXMEMCON = 8192;
MACTVA aMemory[MAXMEMCON];
int aMemoryCount = 0;

// Storage to track breakpoint information.
std::string MACTBreakpoints[50];
int MACTBreakpointCount = 0;

// Substitute MACTSTARTUPINFO structure.
typedef struct _MACTSTARTUPINFOA {
```

```
  DWORD  cb;
  LPSTR  lpReserved;
  LPSTR  lpDesktop;
  LPSTR  lpTitle;
  DWORD  dwX;
  DWORD  dwY;
  DWORD  dwXSize;
  DWORD  dwYSize;
  DWORD  dwXCountChars;
  DWORD  dwYCountChars;
  DWORD  dwFillAttribute;
  DWORD  dwFlags;
  WORD   wShowWindow;
  WORD   cbReserved2;
  LPBYTE lpReserved2;
  HANDLE hStdInput;
  HANDLE hStdOutput;
  HANDLE hStdError;
} MACTSTARTUPINFOA, *LPMACTSTARTUPINFOA;

// Function signatures for calls before defined.
BOOL MACTSocketPrint2(CHAR* a1);
BOOL MACTPrint(const CHAR *psz, ...);
static INT MACTReceive(CHAR* sType);
VOID MACTlog(const CHAR *psz, ...);
void MACTCreateThread(LPVOID buffer, size_t msize, int interval, int imemtype);
HANDLE WINAPI MyCreateFileA(LPCSTR a0,
                            DWORD a1,
                            DWORD a2,
                            LPSECURITY_ATTRIBUTES a3,
                            DWORD a4,
                            DWORD a5,
                            HANDLE a6);
HANDLE WINAPI MyCreateFileW(LPCWSTR a0,
                            DWORD a1,
                            DWORD a2,
                            LPSECURITY_ATTRIBUTES a3,
                            DWORD a4,
                            DWORD a5,
                            HANDLE a6);
BOOL WINAPI MyWriteFileEx(HANDLE a0,
                          LPCVOID a1,
                          DWORD a2,
                          LPOVERLAPPED a3,
                          LPOVERLAPPED_COMPLETION_ROUTINE a4);
BOOL WINAPI MyFlushFileBuffers(HANDLE a0);
BOOL WINAPI MyCloseHandle(HANDLE a0);
```

```
//
//*****************************************************************************************
//
// Function   : TrueSleep
// Description: This function suspends the execution of the current thread for a specified
//              interval.
// Relavance  : Used by Malware to appear inactive.
//
//*****************************************************************************************
static VOID (WINAPI * TrueSleep)(DWORD a0) = Sleep;
//*****************************************************************************************
//
// Function   : TrueSleepEx
// Description: This function suspends the execution of the current thread for a specified
//              interval.
// Relavance  : Used by Malware to appear inactive.
//
//*****************************************************************************************
static DWORD (WINAPI * TrueSleepEx)(DWORD a0,
                                    BOOL  a1) = SleepEx;
//*****************************************************************************************
//
// Function   : TrueGetTickCount
// Description: Retrieves the number of milliseconds that have elapsed since the system was
//              started, up to 49.7 days.
// Relavance  : Used by Malware to try and detect being analyzed.
//
//*****************************************************************************************
static DWORD (WINAPI * TrueGetTickCount)(void) = GetTickCount;
//*****************************************************************************************
//
// Function   : TrueGetTickCount64
// Description: Retrieves the number of milliseconds that have elapsed since the system was
//              started.
// Relavance  : Used by Malware to try and detect being analyzed.
//
//*****************************************************************************************
static ULONGLONG (WINAPI * TrueGetTickCount64)(void) = GetTickCount64;
//*****************************************************************************************
//
// Function   : TrueQueryPerformanceCounter
// Description: Retrieves the current value of the performance counter, which is a high
//              resolution (<1us) time stamp that can be used for time-interval measurements.
// Relavance  : Used by Malware to try and detect being analyzed.
//
//*****************************************************************************************
```

```
static    BOOL    (WINAPI    *    TrueQueryPerformanceCounter)(_Out_    LARGE_INTEGER    *a0)    =
QueryPerformanceCounter;
//*****************************************************************************************
//
// Function   : lstrcmpiA
// Description: Compares two character strings.
// Relavance  :
//
//*****************************************************************************************
static INT (WINAPI * TruelstrcmpiA)(LPCSTR a0,
                                    LPCSTR a1) = lstrcmpiA;
//*****************************************************************************************
//
// Function   : lstrcmpiW
// Description: Compares two character strings.
// Relavance  :
//
//*****************************************************************************************
static INT (WINAPI * TruelstrcmpiW)(LPCWSTR a0,
                                    LPCWSTR a1) = lstrcmpiW;
//*****************************************************************************************
//
// Function   : lstrcmpW
// Description: Compares two character strings.
// Relavance  :
//
//*****************************************************************************************
static INT (WINAPI * TruelstrcmpW)(LPCWSTR a0,
                                   LPCWSTR a1) = lstrcmpW;
//*****************************************************************************************
//
// Function   : CompareStringEx
// Description: Compares two Unicode (wide character) strings, for a locale specified by name.
// Relavance  :
//
//*****************************************************************************************
static INT (WINAPI * TrueCompareStringEx)(LPCWSTR                          a0,
                                          DWORD                            a1,
                                          _In_NLS_string_(cchCount1)LPCWCH a2,
                                          int                              a3,
                                          _In_NLS_string_(cchCount2)LPCWCH a4,
                                          int                              a5,
                                          LPNLSVERSIONINFO                 a6,
                                          LPVOID                           a7,
                                          LPARAM                           a8) = CompareStringEx;
//*****************************************************************************************
//
```

```
// Function   : TrueCreateFileA
// Description: Creates or opens a file or I/O device.
// Relavance  :
//
//*****************************************************************************************
static HANDLE (WINAPI * TrueCreateFileA)(LPCSTR a0,
                                         DWORD a1,
                                         DWORD a2,
                                         LPSECURITY_ATTRIBUTES a3,
                                         DWORD a4,
                                         DWORD a5,
                                         HANDLE a6) = CreateFileA;
//*****************************************************************************************
//
// Function   : TrueGetFileSize
// Description: Retrieves the size of the specified file, in bytes.
// Relavance  :
//
//*****************************************************************************************
static DWORD (WINAPI * TrueGetFileSize)(HANDLE  a0,
                                        LPDWORD a1) = GetFileSize;
//*****************************************************************************************
//
// Function   : TrueCreateFileW
// Description: Creates or opens a file or I/O device.
// Relavance  :
//
//*****************************************************************************************
static HANDLE (WINAPI * TrueCreateFileW)(LPCWSTR a0,
                                         DWORD a1,
                                         DWORD a2,
                                         LPSECURITY_ATTRIBUTES a3,
                                         DWORD a4,
                                         DWORD a5,
                                         HANDLE a6) = CreateFileW;
//*****************************************************************************************
//
// Function   : TrueWriteFile
// Description: Writes data to the specified file or input/output (I/O) device.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueWriteFile)(HANDLE       a0,
                                     LPCVOID      a1,
                                     DWORD        a2,
                                     LPDWORD      a3,
                                     LPOVERLAPPED a4) = WriteFile;
```

```
//*************************************************************************************************
//
// Function   : TrueWriteFileEx
// Description: Writes data to the specified file or input/output (I/O) device.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueWriteFileEx)(HANDLE a0,
                                       LPCVOID a1,
                                       DWORD a2,
                                       LPOVERLAPPED a3,
                                       LPOVERLAPPED_COMPLETION_ROUTINE a4) = WriteFileEx;
//*************************************************************************************************
//
// Function   : TrueFlushFileBuffers
// Description: Flushes the buffers of a specified file and causes all buffered data to be
//              written to a file.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueFlushFileBuffers)(HANDLE a0) = FlushFileBuffers;
//*************************************************************************************************
//
// Function   : TrueCloseHandle
// Description: Closes an open object handle.
// Relavance  :
//
//*************************************************************************************************

static BOOL (WINAPI * TrueCloseHandle)(HANDLE a0) = CloseHandle;
//*************************************************************************************************
//
// Function   : TrueCopyFileA
// Description: Copies an existing file to a new file.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueCopyFileA)(LPCSTR a0,
                                     LPCSTR a1,
                                     BOOL a2) = CopyFileA;
//*************************************************************************************************
//
// Function   : TrueCopyFileExA
// Description: Copies an existing file to a new file.
// Relavance  :
//
//*************************************************************************************************
```

```
static BOOL (WINAPI * TrueCopyFileExA)(LPCSTR a0,
                                       LPCSTR a1,
                                       LPPROGRESS_ROUTINE a2,
                                       LPVOID a3,
                                       LPBOOL a4,
                                       DWORD a5) = CopyFileExA;
//*****************************************************************************************
//
// Function   : TrueCopyFileExW
// Description: Copies an existing file to a new file.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueCopyFileExW)(LPCWSTR            a0,
                                       LPCWSTR            a1,
                                       LPPROGRESS_ROUTINE a2,
                                       LPVOID             a3,
                                       LPBOOL             a4,
                                       DWORD              a5) = CopyFileExW;
//*****************************************************************************************
//
// Function   : TrueCopyFileW
// Description: Copies an existing file to a new file.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueCopyFileW)(LPCWSTR a0,
                                     LPCWSTR a1,
                                     BOOL    a2) = CopyFileW;
//*****************************************************************************************
//
// Function   : TrueDeleteFileA
// Description: Deletes the specified file.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueDeleteFileA)(LPCSTR a0) = DeleteFileA;
//*****************************************************************************************
//
// Function   : TrueDeleteFileW
// Description: Deletes the specified file.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueDeleteFileW)(LPCWSTR a0) = DeleteFileW;
//*****************************************************************************************
//
```

```
// Function   : TrueVirtualAlloc
// Description: This function is a memory-allocation routine that can allocate memory in a remote
//              process.
// Relavance  : Malware sometimes uses VirtualAllocEx as part of process injection.
//
//***************************************************************************************************
static LPVOID (WINAPI * TrueVirtualAlloc)(LPVOID a0,
                                          SIZE_T a1,
                                          DWORD a2,
                                          DWORD a3) = VirtualAlloc;
//***************************************************************************************************
//
// Function   : TrueVirtualAllocEx
// Description: This function is a memory-allocation routine that can allocate memory in a remote
//              process.
// Relavance  : Malware sometimes uses VirtualAllocEx as part of process injection.
//
//***************************************************************************************************
static LPVOID (WINAPI * TrueVirtualAllocEx)(HANDLE a0,
                                            LPVOID a1,
                                            SIZE_T a2,
                                            DWORD a3,
                                            DWORD a4) = VirtualAllocEx;
//***************************************************************************************************
//
// Function   : TrueVirtualFree
// Description: Releases, decommits, or releases and decommits a region of pages within the
//              virtual address space of the calling process.
// Relavance  :
//
//***************************************************************************************************
static BOOL (WINAPI * TrueVirtualFree)(LPVOID a0,
                                       SIZE_T a1,
                                       DWORD  a2) = VirtualFree;
//***************************************************************************************************
//
// Function   : TrueVirtualFreeEx
// Description: Releases, decommits, or releases and decommits a region of memory within the
//              virtual address space of a specified process.
// Relavance  :
//
//***************************************************************************************************
static BOOL (WINAPI * TrueVirtualFreeEx)(HANDLE a0,
                                         LPVOID a1,
                                         SIZE_T a2,
                                         DWORD a3) = VirtualFreeEx;
//***************************************************************************************************
```

```
//
// Function   : TrueCoTaskMemAlloc
// Description: Allocates a block of task memory in the same way that IMalloc::Alloc does.
// Relavance  :
//
//*****************************************************************************************
static __drv_allocatesMem(Mem)LPVOID (WINAPI * TrueCoTaskMemAlloc)(SIZE_T a0) = CoTaskMemAlloc;
//*****************************************************************************************
//
// Function   : TrueCoTaskMemFree
// Description: Frees a block of task memory previously allocated through a call to the
//              CoTaskMemAlloc or CoTaskMemRealloc function.
// Relavance  :
//
//*****************************************************************************************
static VOID (WINAPI * TrueCoTaskMemFree)(LPVOID a0) = CoTaskMemFree;
//*****************************************************************************************
//
// Function   : TrueVirtualProtect
// Description: This function is used to change the protection on a region of memory.
// Relavance  : Malware may use this function to change a read-only section of memory to an
//              executable.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueVirtualProtect)(LPVOID a0,
                                          SIZE_T a1,
                                          DWORD a2,
                                          PDWORD a3) = VirtualProtect;
//*****************************************************************************************
//
// Function   : TrueVirtualProtectEx
// Description: Changes the protection on a region of committed pages in the virtual address
//              space of a specified process.
// Relavance  : Malware may use this function to change a read-only section of memory to an
//              executable.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueVirtualProtectEx)(HANDLE a0,
                                            LPVOID a1,
                                            SIZE_T a2,
                                            DWORD a3,
                                            PDWORD a4) = VirtualProtectEx;
//*****************************************************************************************
//
// Function   : TrueWinExec
// Description: This function is used to execute another program.
// Relavance  :
```

```
//
//*************************************************************************************************
static UINT (WINAPI * TrueWinExec)(LPCSTR a0,
                                   UINT a1) = WinExec;
//*************************************************************************************************
//
// Function   : TrueShellExecuteW
// Description: This function is used to execute another program.
// Relavance  :
//
//*************************************************************************************************
static HINSTANCE (WINAPI * TrueShellExecuteW)(HWND a0,
                                              LPCWSTR a1,
                                              LPCWSTR a2,
                                              LPCWSTR a3,
                                              LPCWSTR a4,
                                              INT     a5) = ShellExecuteW;
//*************************************************************************************************
//
// Function   : TrueShellExecuteExA
// Description: This function is used to execute another program.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueShellExecuteExA)(SHELLEXECUTEINFOA *a0) = ShellExecuteExA;
//*************************************************************************************************
//
// Function   : TrueShellExecuteExW
// Description: This function is used to execute another program.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueShellExecuteExW)(SHELLEXECUTEINFOW *a0) = ShellExecuteExW;
//*************************************************************************************************
//
// Function   : TrueRegGetValueA
// Description: Retrieves the type and data for the specified registry value.
// Relavance  :
//
//*************************************************************************************************
static LONG (WINAPI * TrueRegGetValueA)(HKEY    a0,
                                        LPCSTR  a1,
                                        LPCSTR  a2,
                                        DWORD   a3,
                                        LPDWORD a4,
                                        PVOID   a5,
                                        PDWORD  a6) = RegGetValueA;
```

```
//*************************************************************************************************
//
// Function   : TrueRegGetValueW
// Description: Retrieves the type and data for the specified registry value.
// Relavance  :
//
//*************************************************************************************************
static LONG (WINAPI * TrueRegGetValueW)(HKEY    a0,
                                        LPCWSTR a1,
                                        LPCWSTR a2,
                                        DWORD   a3,
                                        LPDWORD a4,
                                        PVOID   a5,
                                        PDWORD  a6) = RegGetValueW;
//*************************************************************************************************
//
// Function   : TrueRegQueryValueEx
// Description: Retrieves the data associated with the default or unnamed value of a specified
//              registry key.
// Relavance  :
//
//*************************************************************************************************
static LONG (WINAPI * TrueRegQueryValueEx)(HKEY    a0,
                                           LPCTSTR a1,
                                           LPDWORD a2,
                                           LPDWORD a3,
                                           LPBYTE  a4,
                                           LPDWORD a5) = RegQueryValueEx;
//*************************************************************************************************
//
// Function   : TrueRegOpenKeyEx
// Description: Opens the specified registry key.
// Relavance  :
//
//*************************************************************************************************
static LONG (WINAPI * TrueRegOpenKeyEx)(HKEY    a0,
                                        LPCTSTR a1,
                                        DWORD   a2,
                                        REGSAM  a3,
                                        PHKEY   a4) = RegOpenKeyEx;
//*************************************************************************************************
//
// Function   : TrueRegSetValueA
// Description: Sets the data and type of a specified value under a registry key.
// Relavance  :
//
//*************************************************************************************************
```

```
static LSTATUS (WINAPI * TrueRegSetValueA)(HKEY   a0,
                                           LPCSTR a1,
                                           DWORD  a2,
                                           LPCSTR a3,
                                           DWORD  a4) = RegSetValueA;
//*****************************************************************************************
//
// Function   : TrueRegSetValueEx
// Description: Sets the data and type of a specified value under a registry key.
// Relavance  :
//
//*****************************************************************************************
static LONG (WINAPI * TrueRegSetValueEx)(HKEY         a0,
                                         LPCTSTR      a1,
                                         DWORD        a2,
                                         DWORD        a3,
                                         const BYTE * a4,
                                         DWORD        a5) = RegSetValueEx;
//*****************************************************************************************
//
// Function   : TrueRegSetValueExW
// Description: Sets the data and type of a specified value under a registry key.
// Relavance  :
//
//*****************************************************************************************
static LSTATUS (WINAPI * TrueRegSetValueExW)(HKEY         a0,
                                             LPCWSTR      a1,
                                             DWORD        a2,
                                             DWORD        a3,
                                             const BYTE * a4,
                                             DWORD        a5) = RegSetValueExW;
//*****************************************************************************************
//
// Function   : TrueRegEnumKeyExA
// Description: Enumerates the subkeys of the specified open registry key. The function retrieves
//              information about one subkey each time it is called.
// Relavance  :
//
//*****************************************************************************************
static LSTATUS (WINAPI * TrueRegEnumKeyExA)(HKEY      a0,
                                            DWORD     a1,
                                            LPSTR     a2,
                                            LPDWORD   a3,
                                            LPDWORD   a4,
                                            LPSTR     a5,
                                            LPDWORD   a6,
                                            PFILETIME a7) = RegEnumKeyExA;
```

```
//*************************************************************************************************
//
// Function   : TrueRegEnumKeyExW
// Description: Enumerates the subkeys of the specified open registry key. The function retrieves
//              information about one subkey each time it is called.
// Relavance  :
//
//*************************************************************************************************
static LSTATUS (WINAPI * TrueRegEnumKeyExW)(HKEY      a0,
                                            DWORD     a1,
                                            LPWSTR    a2,
                                            LPDWORD   a3,
                                            LPDWORD   a4,
                                            LPWSTR    a5,
                                            LPDWORD   a6,
                                            PFILETIME a7) = RegEnumKeyExW;
//*************************************************************************************************
//
// Function   : TrueRegCreateKeyEx
// Description: Creates the specified registry key. If the key already exists, the function opens
it.
// Relavance  :
//
//*************************************************************************************************
static LONG (WINAPI * TrueRegCreateKeyEx)(HKEY                  a0,
                                          LPCTSTR               a1,
                                          DWORD                 a2,
                                          LPTSTR                a3,
                                          DWORD                 a4,
                                          REGSAM                a5,
                                          LPSECURITY_ATTRIBUTES a6,
                                          PHKEY                 a7,
                                          LPDWORD               a8) = RegCreateKeyEx;
//*************************************************************************************************
//
// Function   : TrueAdjustTokenPrivileges
// Description: This function is used to enable or disable specific access privileges.
// Relavance  : In a process injection attack, this function is used by malware to gain additional
//              permissions.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueAdjustTokenPrivileges)(HANDLE           a0,
                                                 BOOL             a1,
                                                 PTOKEN_PRIVILEGES a2,
                                                 DWORD            a3,
                                                 PTOKEN_PRIVILEGES a4,
                                                 PDWORD           a5) = AdjustTokenPrivileges;
```

```
//*************************************************************************************************
//
// Function   : TrueAttachThreadInput
// Description: This function attaches the input processing from one thread to another so that
//              the second thread receives input events such as keyboard and mouse events.
// Relavance  : Keyloggers and other spyware use this function.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueAttachThreadInput)(DWORD a0,
                                             DWORD a1,
                                             BOOL  a2) = AttachThreadInput;
//*************************************************************************************************
//
// Function   : TrueBitBlt
// Description: This function is used to copy graphic data from one device to another.
// Relavance  : Spyware sometimes uses this function to capture screenshots.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueBitBlt)(HDC   a0,
                                  int   a1,
                                  int   a2,
                                  int   a3,
                                  int   a4,
                                  HDC   a5,
                                  int   a6,
                                  int   a7,
                                  DWORD a8) = BitBlt;
//*************************************************************************************************
//
// Function   : TrueCertOpenSystemStore
// Description: This function is used to access the certificates stored on the local system.
// Relavance  :
//
//*************************************************************************************************
static HCERTSTORE (WINAPI * TrueCertOpenSystemStore)(HCRYPTPROV_LEGACY a0,
                                                     LPCTSTR           a1) = CertOpenSystemStore;
//*************************************************************************************************
//
// Function   : TrueControlService
// Description: This function is used to start, stop, modify, or send a signal to a running
//              service.
// Relavance  : If malware is using its own malicious service, code needs to be analyzed that
//              implements the service in order to determine the purpose of the call.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueControlService)(SC_HANDLE        a0,
                                          DWORD            a1,
```

```
                                        LPSERVICE_STATUS a2) = ControlService;
//*********************************************************************************************
//
// Function   : TrueCreateMutex
// Description: This function creates a mutual exclusion object
// Relavance  : Can be used by malware to ensure that only a single instance of the malware is
//              running on a system at any given time.  Malware often uses fixed names for
//              mutexes, which can be good host-based indicators to detect additional
//              installations of the malware.
//
//*********************************************************************************************
static HANDLE (WINAPI * TrueCreateMutex)(LPSECURITY_ATTRIBUTES a0,
                                         BOOL                  a1,
                                         LPCTSTR               a2) = CreateMutex;
//*********************************************************************************************
//
// Function   : TrueCreateMutexEx
// Description: This function creates a mutual exclusion object
// Relavance  : Can be used by malware to ensure that only a single instance of the malware is
//              running on a system at any given time.  Malware often uses fixed names for
//              mutexes, which can be good host-based indicators to detect additional
//              installations of the malware.
//
//*********************************************************************************************
static HANDLE (WINAPI  * TrueCreateMutexEx)(LPSECURITY_ATTRIBUTES a0,
                                            LPCTSTR               a1,
                                            DWORD                 a2,
                                            DWORD                 a3) = CreateMutexEx;
//*********************************************************************************************
//
// Function   : TrueCreateProcess
// Description: This function creates and launches a new process.
// Relavance  : If malware creates a new process, new process needs to be analyzed as well.
//
//*********************************************************************************************
static BOOL (WINAPI * TrueCreateProcess)(LPCTSTR               a0,
                                         LPTSTR                a1,
                                         LPSECURITY_ATTRIBUTES a2,
                                         LPSECURITY_ATTRIBUTES a3,
                                         BOOL                  a4,
                                         DWORD                 a5,
                                         LPVOID                a6,
                                         LPCTSTR               a7,
                                         LPSTARTUPINFO         a8,
                                         LPPROCESS_INFORMATION a9) = CreateProcess;
//*********************************************************************************************
//
```

```
// Function   : TrueCreateProcessW
// Description: This function creates and launches a new process.
// Relavance  : If malware creates a new process, new process needs to be analyzed as well.
//
//********************************************************************************************
static BOOL (WINAPI * TrueCreateProcessW)(LPCWSTR              a0,
                                          LPWSTR               a1,
                                          LPSECURITY_ATTRIBUTES a2,
                                          LPSECURITY_ATTRIBUTES a3,
                                          BOOL                 a4,
                                          DWORD                a5,
                                          LPVOID               a6,
                                          LPCWSTR              a7,
                                          LPSTARTUPINFOW       a8,
                                          LPPROCESS_INFORMATION a9) = CreateProcessW;
//********************************************************************************************
//
// Function   : TrueTerminateProcess
// Description: Terminates the specified process and all of its threads.
// Relavance  :
//
//********************************************************************************************
static BOOL (WINAPI * TrueTerminateProcess)(HANDLE a0,
                                            UINT   a1) = TerminateProcess;
//********************************************************************************************
//
// Function   : TrueCreateRemoteThread
// Description: This function is used to start a thread in a remote process.
// Relavance  : Launchers and stealth malware use CreateRemoteThread to inject code into a
//              different process.
//
//********************************************************************************************
static HANDLE (WINAPI * TrueCreateRemoteThread)(HANDLE                 a0,
                                                LPSECURITY_ATTRIBUTES  a1,
                                                SIZE_T                 a2,
                                                LPTHREAD_START_ROUTINE a3,
                                                LPVOID                 a4,
                                                DWORD                  a5,
                                                LPDWORD                a6) = CreateRemoteThread;
//********************************************************************************************
//
// Function   : TrueCreateRemoteThread
// Description: This function is used to start a thread in a remote process.
// Relavance  : Launchers and stealth malware use CreateRemoteThread to inject code into a
//              different process.
//
//********************************************************************************************
```

```
static HANDLE (WINAPI * TrueCreateRemoteThreadEx)(HANDLE                       a0,
                                                  LPSECURITY_ATTRIBUTES        a1,
                                                  SIZE_T                       a2,
                                                  LPTHREAD_START_ROUTINE       a3,
                                                  LPVOID                       a4,
                                                  DWORD                        a5,
                                                  LPPROC_THREAD_ATTRIBUTE_LIST a6,
                                                  LPDWORD                                 a7)  =
CreateRemoteThreadEx;
//*************************************************************************************************
//
// Function   : TrueCreateService
// Description: This function is used to create a service that can be started at boot time.
// Relavance  : Malware uses CreateService for persistence, stealth, or to load kernel drivers.
//
//*************************************************************************************************
static SC_HANDLE (WINAPI * TrueCreateService)(SC_HANDLE a0,
                                              LPCTSTR   a1,
                                              LPCTSTR   a2,
                                              DWORD     a3,
                                              DWORD     a4,
                                              DWORD     a5,
                                              DWORD     a6,
                                              LPCTSTR   a7,
                                              LPCTSTR   a8,
                                              LPDWORD   a9,
                                              LPCTSTR   a10,
                                              LPCTSTR   a11,
                                              LPCTSTR   a12) = CreateService;
//*************************************************************************************************
//
// Function   : TrueCreateToolhelp32Snapshot
// Description: This function is used to create a snapshot of processes, heaps, threads, and
//              modules.
// Relavance  : Malware often uses this function as part of code that iterates through processes
//              or threads.
//
//*************************************************************************************************
static HANDLE (WINAPI * TrueCreateToolhelp32Snapshot)(DWORD a0,
                                                      DWORD a1) = CreateToolhelp32Snapshot;
//*************************************************************************************************
//
// Function   : TrueCryptAcquireContextA
// Description: The CryptAcquireContext function is used to acquire a handle to a particular key
//              container within a particular cryptographic service provider (CSP).
// Relavance  : This function is often the first function used by malware to initialize the use
//              of Windows encryption.
```

```
//
//*************************************************************************************************
static BOOL (WINAPI * TrueCryptAcquireContextA)(HCRYPTPROV *a0,
                                                LPCTSTR    a1,
                                                LPCTSTR    a2,
                                                DWORD      a3,
                                                DWORD      a4) = CryptAcquireContext;
//*************************************************************************************************
//
// Function   : TrueCryptAcquireContextW
// Description: The CryptAcquireContext function is used to acquire a handle to a particular key
//              container within a particular cryptographic service provider (CSP).
// Relavance  : This function is often the first function used by malware to initialize the use
//              of Windows encryption.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueCryptAcquireContextW)(HCRYPTPROV *a0,
                                                LPCWSTR    a1,
                                                LPCWSTR    a2,
                                                DWORD      a3,
                                                DWORD      a4) = CryptAcquireContextW;
//*************************************************************************************************
//
// Function   : TrueDeviceIoControl
// Description: This function sends a control message from user space to a device driver.
// Relavance  : Kernel malware that needs to pass information between user space and kernel space
//              often use this function.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueDeviceIoControl)(HANDLE       a0,
                                           DWORD        a1,
                                           LPVOID       a2,
                                           DWORD        a3,
                                           LPVOID       a4,
                                           DWORD        a5,
                                           LPDWORD      a6,
                                           LPOVERLAPPED a7) = DeviceIoControl;
//*************************************************************************************************
//
// Function   : TrueEnumProcesses
// Description: This function is used to enumerate through running processes on the system.
// Relavance  : Malware often enumerates through processes to find a process into which to
//              inject.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueEnumProcesses)(DWORD *a0,
                                         DWORD a1,
```

```
                                        DWORD *a2) = EnumProcesses;
//*************************************************************************************************
//
// Function   : TrueEnumProcessModules

// Relavance  : Malware enumerates through modules when doing an injection.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueEnumProcessModules)(HANDLE  a0,
                                              HMODULE *a1,
                                              DWORD   a2,
                                              LPDWORD a3) = EnumProcessModules;
//*************************************************************************************************
//
// Function   : TrueEnumProcessModulesEx
// Description: This function is used to enumerate the loaded modules (executables and DLLs) for
//              a given process.
// Relavance  : Malware enumerates through modules when doing an injection.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueEnumProcessModulesEx)(HANDLE  a0,
                                                HMODULE *a1,
                                                DWORD   a2,
                                                LPDWORD a3,
                                                DWORD   a4) = EnumProcessModulesEx;
//*************************************************************************************************
//
// Function   : TrueFindFirstFile
// Description: This function is used to search through a directory and enumerate the file
//              system.
// Relavance  :
//
//*************************************************************************************************
static HANDLE (WINAPI * TrueFindFirstFile)(LPCTSTR           a0,
                                           LPWIN32_FIND_DATA a1) = FindFirstFile;
//*************************************************************************************************
//
// Function   : TrueFindFirstFileEx
// Description: This function is used to search through a directory and enumerate the file
//              system.
// Relavance  :
//
//*************************************************************************************************
static HANDLE (WINAPI * TrueFindFirstFileEx)(LPCTSTR            a0,
                                             FINDEX_INFO_LEVELS a1,
                                             LPVOID             a2,
                                             FINDEX_SEARCH_OPS  a3,
```

```
                                        LPVOID             a4,
                                        DWORD              a5) = FindFirstFileEx;
//*************************************************************************************************
//
// Function   : TrueFindNextFile
// Description: This function is used to search through a directory and enumerate the file
//              system.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueFindNextFile)(HANDLE             a0,
                                        LPWIN32_FIND_DATAA a1) = FindNextFile;
//*************************************************************************************************
//
// Function   : TrueNextNextFileEx
// Description: This function is used to search through a directory and enumerate the file
//              system.
// Relavance  :
//
//*************************************************************************************************
static HRSRC (WINAPI * TrueFindResourceA)(HMODULE a0,
                                          LPCSTR  a1,
                                          LPCSTR  a2) = FindResourceA;
//*************************************************************************************************
//
// Function   : TrueNextNextFileEx
// Description: This function is used to search through a directory and enumerate the file
//              system.
// Relavance  :
//
//*************************************************************************************************
static HRSRC (WINAPI * TrueFindResourceExA)(HMODULE a0,
                                            LPCSTR  a1,
                                            LPCSTR  a2,
                                            WORD    a3) = FindResourceExA;
//*************************************************************************************************
//
// Function   : TrueFindWindow
// Description: This function is used to search for an open window on the desktop.
// Relavance  : Sometimes this function is used as an anti-debugging technique to search for
//              OllyDbg windows.
//
//*************************************************************************************************
static HWND (WINAPI * TrueFindWindow)(LPCTSTR a0,
                                      LPCTSTR a1) = FindWindow;
//*************************************************************************************************
//
```

```
// Function   : TrueFindWindow
// Description: This function is used to search for an open window on the desktop.
// Relavance  : Sometimes this function is used as an anti-debugging technique to search for
//              OllyDbg windows.
//
//*************************************************************************************************
static HWND (WINAPI * TrueFindWindowEx)(HWND    a0,
                                        HWND    a1,
                                        LPCTSTR a2,
                                        LPCTSTR a3) = FindWindowExA;
//*************************************************************************************************
//
// Function   : TrueFtpOpenFileW
// Description: Initiates access to a remote file on an FTP server for reading or writing.
// Relavance  :
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueFtpOpenFileW)(HINTERNET a0,
                                             LPCWSTR   a1,
                                             DWORD     a2,
                                             DWORD     a3,
                                             DWORD_PTR a4) = FtpOpenFileW;
//*************************************************************************************************
//
// Function   : TrueFtpPutFile
// Description: This function is used to upload a file to remote FTP server.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueFtpPutFile)(HINTERNET a0,
                                      LPCTSTR a1,
                                      LPCTSTR a2,
                                      DWORD a3,
                                      DWORD a4) = FtpPutFile;
//*************************************************************************************************
//
// Function   : TrueGetAdaptersInfo
// Description: This function is used to obtain information about the network adapters on the
//              system.
// Relavance  : Backdoors sometimes call GetAdaptersInfo in the information-gathering phase to
//              gather information about infected machines. In some cases, it's used to gather
//              MAC addresses to check for VMware as part of anti-virtual machine techniques.
//
//*************************************************************************************************
static ULONG (WINAPI * TrueGetAdaptersInfo)(PIP_ADAPTER_INFO a0,
                                            PULONG           a1) = GetAdaptersInfo;
//*************************************************************************************************
```

```
//
// Function   : TrueGetAsyncKeyState
// Description: This function is used to determine whether a particular key is being pressed.
// Relavance  : Malware sometimes uses this function to implement a keylogger.
//
//***************************************************************************************************
static SHORT (WINAPI * TrueGetAsyncKeyState)(int a0) = GetAsyncKeyState;
//***************************************************************************************************
//
// Function   : TrueGetDC
// Description: This function returns a handle to a device context for a window or the whole
//              screen.
// Relavance  : Spyware that takes screen captures often uses this function.
//
//***************************************************************************************************
static HDC (WINAPI * TrueGetDC)(HWND a0) = GetDC;
//***************************************************************************************************
//
// Function   : TrueGetForegroundWindow
// Description: This function returns a handle to the window currently in the foreground of the
//              desktop.
// Relavance  : Keyloggers commonly use this function to determine in which window the user is
//              entering his keystrokes.
//
//***************************************************************************************************
static HWND (WINAPI * TrueGetForegroundWindow)(void) = GetForegroundWindow;
//***************************************************************************************************
//
// Function   : TrueGetWindowText
// Description: Copies the text of the specified window's title bar (if it has one) into a
//              buffer. If the specified window is a control, the text of the control is copied.
// Relavance  : Can be used to get text from forms.
//
//***************************************************************************************************
static INT (WINAPI * TrueGetWindowText)(HWND   a0,
                                        LPTSTR a1,
                                        int    a2) = GetWindowText;
//***************************************************************************************************
//
// Function   : Truegethostbyname
// Description: This function is used to perform a DNS lookup on a particular hostname prior to
//              making an IP connection to a remote host.
// Relavance  : Hostnames that serve as command and-control servers often make good
//              network-based signatures.
//
//***************************************************************************************************
static hostent *(WINAPI * Truegethostbyname)(const char *a0) = gethostbyname;
```

```
//*************************************************************************************************
//
// Function   : Truegethostname
// Description: This function is used to retrieve the hostname of the computer.
// Relavance  : Backdoors sometimes use gethostname in information gathering phase of the
//              victim machine.
//
//*************************************************************************************************
static int (WINAPI * Truegethostname)(char *a0,
                                      int  a1) = gethostname;
//*************************************************************************************************
//
// Function   : Truegetaddrinfo
// Description: This function translates from an ANSI host name to an address.
// Relavance  :
//
//*************************************************************************************************
static INT (WINAPI * Truegetaddrinfo)(PCSTR            a0,
                                      PCSTR            a1,
                                      const ADDRINFOA *a2,
                                      PADDRINFOA      *a3) = getaddrinfo;
//*************************************************************************************************
//
// Function   : TrueGetKeyState
// Description: Obtain the status of a particular key on the keyboard.
// Relavance  : This function is used by keyloggers to obtain the status of a particular key.
//
//*************************************************************************************************
static SHORT (WINAPI * TrueGetKeyState)(int a0) = GetKeyState;
//*************************************************************************************************
//
// Function   : GetModuleFileName
// Description: This function returns the filename of a module that is loaded in the current
//              process.
// Relavance  : Malware can use this function to modify or copy files in the currently running
//              process.
//
//*************************************************************************************************
static DWORD (WINAPI * TrueGetModuleFileName)(HMODULE a0,
                                              LPTSTR  a1,
                                              DWORD   a2) = GetModuleFileName;
//*************************************************************************************************
//
// Function   : GetModuleFileNameExA
// Description: This function returns the filename of a module that is loaded in the current
//              process.
// Relavance  : Malware can use this function to modify or copy files in the currently running
```

```
//              process.
//
//*************************************************************************************************
static DWORD (WINAPI * TrueGetModuleFileNameExA)(HANDLE  a0,
                                                 HMODULE a1,
                                                 LPSTR   a2,
                                                 DWORD   a3) = GetModuleFileNameExA;
//*************************************************************************************************
//
// Function   : GetModuleFileNameExW
// Description: This function returns the filename of a module that is loaded in the current
//              process.
// Relavance  : Malware can use this function to modify or copy files in the currently running
//              process.
//
//*************************************************************************************************
static DWORD (WINAPI * TrueGetModuleFileNameExW)(HANDLE  a0,
                                                 HMODULE a1,
                                                 LPWSTR  a2,
                                                 DWORD   a3) = GetModuleFileNameExW;
//*************************************************************************************************
//
// Function   : TrueGetModuleHandle
// Description: This function is used to obtain a handle to an already loaded module.

//
//*************************************************************************************************
static HMODULE (WINAPI * TrueGetModuleHandle)(LPCTSTR a0) = GetModuleHandle;
//*************************************************************************************************
//
// Function   : TrueGetModuleHandleEx
// Description: This function is used to obtain a handle to an already loaded module.
// Relavance  : Malware may use GetModuleHandle to locate and modify code in a loaded module or
//              to search for a good location to inject code.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueGetModuleHandleEx)(DWORD    a0,
                                             LPCTSTR  a1,
                                             HMODULE *a2) = GetModuleHandleEx;
//*************************************************************************************************
//
// Function   : TrueGetProcAddress
// Description: This function is used to retrieve the address of a function in a DLL loaded into
//              memory.
// Relavance  : This is used to import functions from other DLLs in addition to the functions
//              imported in the PE file header.
//
```

```
//*************************************************************************************************
static FARPROC (WINAPI * TrueGetProcAddress)(HMODULE a0,
                                             LPCSTR  a1) = GetProcAddress;
//*************************************************************************************************
//
// Function   : TrueGetStartupInfoA
// Description: This function is used to retrieve a structure containing details about how the
//              current process was configured to run, such as where the standard handles are
//              directed.
// Relavance  :
//
//*************************************************************************************************
static VOID (WINAPI * TrueGetStartupInfoA)(LPSTARTUPINFOA a0) = GetStartupInfoA;
//*************************************************************************************************
//
// Function   : TrueGetSystemDefaultLangID
// Description: This function returns the default language settings for the system.
// Relavance  : These are used by malwares by specifically designed for region-based attacks.
//
//*************************************************************************************************
static LANGID (WINAPI * TrueGetSystemDefaultLangID)(void) = GetSystemDefaultLangID;
//*************************************************************************************************
//
// Function   : TrueGetTempPathA
// Description: This function returns the temporary file path.
// Relavance  : If malware call this function, check whether it reads or writes any files in the
//              temporary file path.
//
//*************************************************************************************************
static DWORD (WINAPI * TrueGetTempPathA)(DWORD a0,
                                         LPSTR a1) = GetTempPathA;
//*************************************************************************************************
//
// Function   : TrueGetThreadContext
// Description: This function returns the context structure of a given thread.
// Relavance  : The context for a thread stores all the thread information, such as the register
//              values and current state.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueGetThreadContext)(HANDLE    a0,
                                            LPCONTEXT a1) = GetThreadContext;
//*************************************************************************************************
//
// Function   : TrueGetVersionEx
// Description: This function returns information about which version of Windows is currently
//              running.
// Relavance  : This can be used as part of a victim survey, or to select between different
```

```
//              offsets for undocumented structures that have changed between different versions
//              of Windows.
//
//*********************************************************************************************
static BOOL (WINAPI * TrueGetVersionEx)(LPOSVERSIONINFO a0) = GetVersionEx;
//*********************************************************************************************
//
// Function   : TrueGetWindowsDirectory
// Description: This function returns the file path to the Windows directory (usually C:\Windows)
// Relavance  : Malware sometimes uses this call to determine into which directory to install
//              additional malicious programs.
//
//*********************************************************************************************
static UINT (WINAPI * TrueGetWindowsDirectory)(LPTSTR a0,
                                               UINT   a1) = GetWindowsDirectory;
//*********************************************************************************************
//
// Function   : Trueinet_addr
// Description: This function converts an IP address string like 127.0.0.1 so that it can be used
//              by functions such as connect. The string specified can sometimes be used as a
//              network-based signature.
// Relavance  :
//
//*********************************************************************************************
static ULONG (WINAPI * Trueinet_addr)(const char * a0) = inet_addr;
//*********************************************************************************************
//
// Function   : TrueInternetOpen
// Description: This function initializes the high-level Internet access functions from WinINet,
//              such as InternetOpenUrl and InternetReadFile. Searching for InternetOpen is a
//              good way to find the start of Internet access functionality. One of the
//              parameters to InternetOpen is the User-Agent, which can sometimes make a good
//              network-based signature.
// Relavance  :
//
//*********************************************************************************************
static HINTERNET (WINAPI * TrueInternetOpen)(LPCTSTR a0,
                                             DWORD   a1,
                                             LPCTSTR a2,
                                             LPCTSTR a3,
                                             DWORD   a4) = InternetOpen;
//*********************************************************************************************
//
// Function   : TrueInternetOpenW
// Description: This function initializes the high-level Internet access functions from WinINet,
//              such as InternetOpenUrl and InternetReadFile. Searching for InternetOpen is a
//              good way to find the start of Internet access functionality. One of the
```

```
//              parameters to InternetOpen is the User-Agent, which can sometimes make a good
//              network-based signature.
// Relavance  :
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueInternetOpenW)(LPCWSTR a0,
                                              DWORD   a1,
                                              LPCWSTR a2,
                                              LPCWSTR a3,
                                              DWORD   a4) = InternetOpenW;
//*************************************************************************************************
//
// Function   : TrueInternetOpenUrl
// Description: This function opens a specific URL for a connection using FTP, HTTP, or HTTPS.
//              URLs, if fixed, can often be good network-based signatures.
// Relavance  :
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueInternetOpenUrl)(HINTERNET a0,
                                                LPCTSTR   a1,
                                                LPCTSTR   a2,
                                                DWORD     a3,
                                                DWORD     a4,
                                                DWORD_PTR a5) = InternetOpenUrl;
//*************************************************************************************************
//
// Function   : TrueInternetOpenUrlA
// Description: This function opens a specific URL for a connection using FTP, HTTP, or HTTPS.
//              URLs, if fixed, can often be good network-based signatures.
// Relavance  :
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueInternetOpenUrlA)(HINTERNET a0,
                                                 LPCSTR    a1,
                                                 LPCSTR    a2,
                                                 DWORD     a3,
                                                 DWORD     a4,
                                                 DWORD_PTR a5) = InternetOpenUrlA;
//*************************************************************************************************
//
// Function   : TrueInternetConnectW
// Description: This function opens an FTP or HTTP session for a specified site.
// Relavance  : C&C Communication and downloading payloads.
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueInternetConnectW)(HINTERNET     a0,
                                                 LPCWSTR       a1,
```

```
                                                INTERNET_PORT a2,
                                                LPCWSTR       a3,
                                                LPCWSTR       a4,
                                                DWORD         a5,
                                                DWORD         a6,
                                                DWORD         a7) = InternetConnectW;
//*************************************************************************************************
//
// Function   : TrueHttpOpenRequestW
// Description: Creates an HTTP request handle.
// Relavance  : Communication
//
//*************************************************************************************************
static HINTERNET (WINAPI * TrueHttpOpenRequestW)(HINTERNET  a0,
                                                LPCWSTR    a1,
                                                LPCWSTR    a2,
                                                LPCWSTR    a3,
                                                LPCWSTR    a4,
                                                LPCWSTR   *a5,
                                                DWORD      a6,
                                                DWORD_PTR  a7) = HttpOpenRequestW;
//*************************************************************************************************
//
// Function   : TrueHttpSendRequestW
// Description: Creates an HTTP request handle.
// Relavance  : Communication
//
//*************************************************************************************************
static BOOL (WINAPI * TrueHttpSendRequestW)(HINTERNET a0,
                                            LPCWSTR   a1,
                                            DWORD     a2,
                                            LPVOID    a3,
                                            DWORD     a4) = HttpSendRequestW;
//*************************************************************************************************
//
// Function   : TrueHttpSendRequestExW
// Description: Creates an HTTP request handle.
// Relavance  : Communication
//
//*************************************************************************************************
static BOOL (WINAPI * TrueHttpSendRequestExW)(HINTERNET           a0,
                                              LPINTERNET_BUFFERSW a1,
                                              LPINTERNET_BUFFERSW a2,
                                              DWORD               a3,
                                              DWORD_PTR           a4) = HttpSendRequestExW;
//*************************************************************************************************
//
```

```
// Function   : TrueInternetReadFile
// Description: This function reads data from a previously opened URL.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueInternetReadFile)(HINTERNET a0,
                                            LPVOID    a1,
                                            DWORD     a2,
                                            LPDWORD   a3) = InternetReadFile;
//*************************************************************************************************
//
// Function   : TrueInternetWriteFile
// Description: This function reads data from a previously opened URL.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueInternetWriteFile)(HINTERNET a0,
                                             LPCVOID   a1,
                                             DWORD     a2,
                                             LPDWORD   a3) = InternetWriteFile;
//*************************************************************************************************
//
// Function   : TrueIsWow64Process
// Description: This function is used by a 32-bit process to determine if it is running on a
//              64-bit operating system.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueIsWow64Process)(HANDLE a0,
                                          PBOOL a1) = IsWow64Process;
//*************************************************************************************************
//
// Function   : LdrLoadDll
// Description: This is a low-level function to load a DLL into a process, just like LoadLibrary.
// Relavance  : Normal programs use LoadLibrary, and the presence of this import may indicate a
//              program that is attempting to be stealthy.
//
//*************************************************************************************************
typedef NTSTATUS (WINAPI *fLdrLoadDll)
(
    IN PWCHAR PathToFile OPTIONAL,
    IN ULONG Flags OPTIONAL,
    IN PUNICODE_STRING ModuleFileName,
    OUT PHANDLE ModuleHandle
);
// Not part of export table of ntdll.dll, have to access this way.
HMODULE hmodule = GetModuleHandleA("ntdll.dll");
```

```
fLdrLoadDll   _LdrLoadDll = (fLdrLoadDll) TrueGetProcAddress ( hmodule, "LdrLoadDll" );
static NTSTATUS (WINAPI * TrueLdrLoadDll)(PWCHAR           a0,
                                          ULONG            a1,
                                          PUNICODE_STRING  a2,
                                          PHANDLE          a3) = _LdrLoadDll;
//*********************************************************************************************
//
// Function   : RtlCreateRegistryKey
// Description: This function is used to create a registry from kernel-mode code.
// Relavance  :
//
//*********************************************************************************************
typedef NTSTATUS (WINAPI *fRtlCreateRegistryKey)
(
    IN ULONG RelativeTo,
    IN PWSTR Path
);
// Not part of export table of ntdll.dll, have to access this way.
HMODULE hmodule2 = GetModuleHandleA("ntdll.dll");
fRtlCreateRegistryKey _RtlCreateRegistryKey = (fRtlCreateRegistryKey) TrueGetProcAddress(hmodule2,
"RtlCreateRegistryKey");
static NTSTATUS (WINAPI * TrueRtlCreateRegistryKey)(ULONG a0,
                                                    PWSTR a1) = _RtlCreateRegistryKey;
//*********************************************************************************************
//
// Function   : RtlWriteRegistryValue
// Description: This function is used to write a value to the registry from kernel-mode code.
// Relavance  :
//
//*********************************************************************************************
typedef NTSTATUS (WINAPI *fRtlWriteRegistryValue)
(
    IN ULONG  RelativeTo,
    IN PCWSTR Path,
    IN PCWSTR ValueName,
    IN ULONG  ValueType,
    IN PVOID  ValueData,
    IN ULONG  ValueLength
);
// Not part of export table of ntdll.dll, have to access this way.
HMODULE hmodule3 = GetModuleHandleA("ntdll.dll");
fRtlWriteRegistryValue          _RtlWriteRegistryValue          =          (fRtlWriteRegistryValue)
TrueGetProcAddress(hmodule3, "RtlWriteRegistryValue");
static NTSTATUS (WINAPI * TrueRtlWriteRegistryValue)(ULONG  a0,
                                                     PCWSTR a1,
                                                     PCWSTR a2,
                                                     ULONG  a3,
```

```
                                                    PVOID  a4,
                                                    ULONG  a5) = _RtlWriteRegistryValue;
//*************************************************************************************************
//
// Function   : TrueLoadResource
// Description: This function loads a resource from a PE file into memory.
// Relavance  : Malware sometimes uses resources to store strings, configuration information, or
//              other malicious files.
//
//*************************************************************************************************
static HGLOBAL (WINAPI * TrueLoadResource)(HMODULE a0,
                                           HRSRC   a1) = LoadResource;
//*************************************************************************************************
//
// Function   : TrueLsaEnumerateLogonSessions
// Description: This function is used to enumerate through logon sessions on the current system.
// Relavance  : Can be used as part of a credential stealer.
//
//*************************************************************************************************
static NTSTATUS (WINAPI * TrueLsaEnumerateLogonSessions)(PULONG a0,
                                                         PLUID  *a1) = LsaEnumerateLogonSessions;
//*************************************************************************************************
//
// Function   : TrueMapViewOfFile
// Description: This function is used to map a file into memory and makes the contents of the
//              file accessible via memory addresses.
// Relavance  : Launchers, loaders, and injectors use this function to read and modify PE files.
//              By using MapViewOfFile, the malware can avoid using WriteFile to modify the
//              contents of a file.
//
//*************************************************************************************************
static LPVOID (WINAPI * TrueMapViewOfFile)(HANDLE a0,
                                           DWORD  a1,
                                           DWORD  a2,
                                           DWORD  a3,
                                           SIZE_T a4) = MapViewOfFile;
//*************************************************************************************************
//
// Function   : TrueMapViewOfFileEx
// Description: This function is used to map a file into memory and makes the contents of the
//              file accessible via memory addresses.
// Relavance  : Launchers, loaders, and injectors use this function to read and modify PE files.
//              By using MapViewOfFile, the malware can avoid using WriteFile to modify the
//              contents of a file.
//
//*************************************************************************************************
static LPVOID (WINAPI * TrueMapViewOfFileEx)(HANDLE a0,
```

```
                                        DWORD a1,
                                        DWORD a2,
                                        DWORD a3,
                                        SIZE_T a4,
                                        LPVOID a5) = MapViewOfFileEx;
//*************************************************************************************************
//
// Function   : TrueMapVirtualKeyA
// Description: This function is used to translate a virtual-key code into a character value.
// Relavance  : It is often used by keylogging malware.
//
//*************************************************************************************************
static UINT (WINAPI * TrueMapVirtualKeyA)(UINT a0,
                                      UINT a1) = MapVirtualKeyA;
//*************************************************************************************************
//
// Function   : TrueMapVirtualKeyExA
// Description: This function is used to translate a virtual-key code into a character value.
// Relavance  : It is often used by keylogging malware
//
//*************************************************************************************************
static UINT (WINAPI * TrueMapVirtualKeyExA)(UINT a0,
                                        UINT a1,
                                        HKL  a2) = MapVirtualKeyExA;
//*************************************************************************************************
//
// Function   : TrueMapVirtualKeyW
// Description: This function is used to translate a virtual-key code into a character value.
// Relavance  : It is often used by keylogging malware
//
//*************************************************************************************************
static UINT (WINAPI * TrueMapVirtualKeyW)(UINT a0,
                                      UINT a1) = MapVirtualKeyW;
//*************************************************************************************************
//
// Function   : TrueMapVirtualKeyExW
// Description: This function is used to translate a virtual-key code into a character value.
// Relavance  : It is often used by keylogging malware
//
//*************************************************************************************************
static UINT (WINAPI * TrueMapVirtualKeyExW)(UINT a0,
                                        UINT a1,
                                        HKL  a2) = MapVirtualKeyExW;
//*************************************************************************************************
//
// Function   : TrueModule32First
// Description: This function is used to enumerate through modules loaded into a process.
```

```
// Relavance  : Injectors use this function to determine where to inject code.
//
//*********************************************************************************************
static BOOL (WINAPI * TrueModule32First)(HANDLE          a0,
                                         LPMODULEENTRY32 a1) = Module32First;
//*********************************************************************************************
//
// Function   : TrueModule32Next
// Description: This function is used to enumerate through modules loaded into a process.
// Relavance  : Injectors use this function to determine where to inject code.
//
//*********************************************************************************************
static BOOL (WINAPI * TrueModule32Next)(HANDLE          a0,
                                        LPMODULEENTRY32 a1) = Module32Next;
//*********************************************************************************************
//
// Function   : TrueOpenMutexA
// Description: This function opens a handle to a mutual exclusion object that can be used by
//              malware to ensure that only a single instance of malware is running on a system
//              at any given time.
// Relavance  : Malware often uses fixed names for mutexes, which can be good host-based
//              indicators.
//
//*********************************************************************************************
static HANDLE (WINAPI * TrueOpenMutexA)(DWORD  a0,
                                        BOOL   a1,
                                        LPCSTR a2) = OpenMutexA;
//*********************************************************************************************
//
// Function   : TrueOpenProcess
// Description: This function is used to open a handle to another process running on the system.
// Relavance  : This handle can be used to read and write to the other process memory or to
//              inject code into the other process.
//
//*********************************************************************************************
static HANDLE (WINAPI * TrueOpenProcess)(DWORD a0,
                                         BOOL  a1,
                                         DWORD a2) = OpenProcess;
//*********************************************************************************************
//
// Function   : TrueOutputDebugString
// Description: This function is used to output a string to a debugger if one is attached.
// Relavance  : This can be used as an anti-debugging technique.
//
//*********************************************************************************************
static VOID (WINAPI * TrueOutputDebugString)(LPCTSTR a0) = OutputDebugString;
//*********************************************************************************************
```

```
//
// Function   : TrueOutputDebugStringA
// Description: This function is used to output a string to a debugger if one is attached.
// Relavance  : This can be used as an anti-debugging technique.
//
//********************************************************************************************
static VOID (WINAPI * TrueOutputDebugStringA)(LPCSTR a0) = OutputDebugStringA;
//********************************************************************************************
//
// Function   : TruePeekNamedPipe
// Description: This function is used to output a string to a debugger if one is attached.
// Relavance  : This can be used as an anti-debugging technique.
//
//********************************************************************************************
static VOID (WINAPI * TrueOutputDebugStringW)(LPCWSTR a0) = OutputDebugStringW;
//********************************************************************************************
//
// Function   : TruePeekNamedPipe
// Description: This function is used to copy data from a named pipe without removing data from
//              the pipe.
// Relavance  : This function is popular with reverse shells.
//
//********************************************************************************************
static BOOL (WINAPI * TruePeekNamedPipe)(HANDLE  a0,
                                         LPVOID  a1,
                                         DWORD   a2,
                                         LPDWORD a3,
                                         LPDWORD a4,
                                         LPDWORD a5) = PeekNamedPipe;
//********************************************************************************************
//
// Function   : TrueProcess32First
// Description: This function is used to begin enumerating processes from a previous call to
//              CreateToolhelp32Snapshot.
// Relavance  : Malware often enumerates through processes to find a process into which to
//              inject.
//
//********************************************************************************************
static BOOL (WINAPI * TrueProcess32First)(HANDLE           a0,
                                          LPPROCESSENTRY32 a1) = Process32First;
//********************************************************************************************
//
// Function   : TrueProcess32FirstW
// Description: This function is used to begin enumerating processes from a previous call to
//              CreateToolhelp32Snapshot.
// Relavance  : Malware often enumerates through processes to find a process into which to
//              inject.
```

```
//
//*****************************************************************************************
static BOOL (WINAPI * TrueProcess32FirstW)(HANDLE            a0,
                                           LPPROCESSENTRY32W a1) = Process32FirstW;
//*****************************************************************************************
//
// Function   : TrueProcess32Next
// Description: This function is used to begin enumerating processes from a previous call to
//              CreateToolhelp32Snapshot.
// Relavance  : Malware often enumerates through processes to find a process into which to
//              inject.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueProcess32Next)(HANDLE           a0,
                                         LPPROCESSENTRY32 a1) = Process32Next;
//*****************************************************************************************
//
// Function   : TrueProcess32NextW
// Description: This function is used to begin enumerating processes from a previous call to
//              CreateToolhelp32Snapshot..
// Relavance  : Malware often enumerates through processes to find a process into which to
//              inject.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueProcess32NextW)(HANDLE            a0,
                                          LPPROCESSENTRY32W a1) = Process32NextW;
//*****************************************************************************************
//
// Function   : TrueQueueUserAPC
// Description: This function is used to execute code for a different thread.
// Relavance  : Malware sometimes uses QueueUserAPC to inject code into another process.
//
//*****************************************************************************************
static DWORD (WINAPI * TrueQueueUserAPC)(PAPCFUNC  a0,
                                         HANDLE    a1,
                                         ULONG_PTR a2) = QueueUserAPC;
//*****************************************************************************************
//
// Function   : TrueReadProcessMemory
// Description: This function is used to read the memory of a remote process.
// Relavance  :
//
//*****************************************************************************************
static BOOL (WINAPI * TrueReadProcessMemory)(HANDLE  a0,
                                             LPCVOID a1,
                                             LPVOID  a2,
                                             SIZE_T  a3,
```

```
                                          SIZE_T  *a4) = ReadProcessMemory;
//*************************************************************************************************
//
// Function   : TrueRegisterHotKey
// Description: This function is used to register a handler to be notified anytime a user enters
//              a particular key combination (like CTRL-ALT-J), regardless of which window is
//              active when the user presses the key combination.
// Relavance  : This function is sometimes used by spyware that remains hidden from the user
//              until the key combination is pressed.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueRegisterHotKey)(HWND a0,
                                          int  a1,
                                          UINT a2,
                                          UINT a3) = RegisterHotKey;
//*************************************************************************************************
//
// Function   : TrueRegOpenKeyA
// Description: This function is used to open a handle to a registry key for reading and editing.
//              Registry keys are sometimes written as a way for software to achieve persistence
//              on a host. The registry also contains a whole host of operating system and
//              application setting information.
// Relavance  :
//
//*************************************************************************************************
static LSTATUS (WINAPI * TrueRegOpenKeyA)(HKEY   a0,
                                          LPCSTR a1,
                                          PHKEY  a2) = RegOpenKeyA;
//*************************************************************************************************
//
// Function   : TrueRegOpenKeyExA
// Description: This function is used to open a handle to a registry key for reading and editing.
//              Registry keys are sometimes written as a way for software to achieve persistence
//              on a host. The registry also contains a whole host of operating system and
//              application setting information.
// Relavance  :
//
//*************************************************************************************************
static LSTATUS (WINAPI * TrueRegOpenKeyExA)(HKEY   a0,
                                            LPCSTR a1,
                                            DWORD  a2,
                                            REGSAM a3,
                                            PHKEY  a4) = RegOpenKeyExA;
//*************************************************************************************************
//
// Function   : TrueRegOpenKeyExW
// Description: This function is used to open a handle to a registry key for reading and editing.
```

```
//              Registry keys are sometimes written as a way for software to achieve persistence
//              on a host. The registry also contains a whole host of operating system and
//              application setting information.
// Relavance  :
//
//*************************************************************************************************
static LSTATUS (WINAPI * TrueRegOpenKeyExW)(HKEY    a0,
                                            LPCWSTR a1,
                                            DWORD   a2,
                                            REGSAM  a3,
                                            PHKEY   a4) = RegOpenKeyExW;
//*************************************************************************************************
//
// Function   : TrueResumeThread
// Description: This function is used to resume a previously suspended thread.
// Relavance  : ResumeThread is used as part of several injection techniques.
//
//*************************************************************************************************
static DWORD (WINAPI * TrueResumeThread)(HANDLE a0) = ResumeThread;
//*************************************************************************************************
//
// Function   : TrueSamIConnect
// Description: This function is used to connect to the Security Account Manager (SAM) in order
//              to make future calls that access credential information.
// Relavance  : Hash-dumping programs access the SAM database in order to retrieve the hash of
//              users' login passwords.
//
//*************************************************************************************************
typedef LPVOID (WINAPI *SAMICONNECT)
    (
         IN PDWORD a0,
         IN PDWORD a1,
         IN PDWORD a2
    );

// Needs to be reached via ordinal.
SAMICONNECT Dallas0()
{
    USES_CONVERSION;
    TCHAR szSystemDir[MAX_PATH+1];
    int nSize = GetSystemDirectory(szSystemDir,MAX_PATH);
    szSystemDir[nSize] = '\0';
    TCHAR szSFCOS[MAX_PATH+1];
    _tcscpy(szSFCOS,szSystemDir);
    _tcscat(szSFCOS,_T("\\samsrv.dll"));
    HMODULE hSFSModule=::LoadLibrary(szSFCOS);
```

```
    SAMICONNECT pFnSamIConnect;
    pFnSamIConnect = (SAMICONNECT) TrueGetProcAddress(hSFSModule, (LPCSTR)5);

    return (pFnSamIConnect);
}
static LPVOID (WINAPI * TrueSamIConnect)(PDWORD a0,
                                         PDWORD a1,
                                         PDWORD a2) = Dallas0();

//*****************************************************************************************
//
// Function   : TrueSetFileTime
// Description: This function is used to modify the creation, access, or last modified time of a
//              file.
// Relavance  : Malware often uses this function to conceal malicious activity.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueSetFileTime)(HANDLE         a0,
                                       CONST FILETIME *a1,
                                       CONST FILETIME *a2,
                                       CONST FILETIME *a3) = SetFileTime;
//*****************************************************************************************
//
// Function   : TrueSetThreadContext
// Description: This function is used to modify the context of a given thread.
// Relavance  : Some injection techniques use SetThreadContext.
//
//*****************************************************************************************
static BOOL (WINAPI * TrueSetThreadContext)(HANDLE  a0,
                                            const CONTEXT *a1) = SetThreadContext;
//*****************************************************************************************
//
// Function   : TrueSetWindowsHookEx
// Description: This function is used to set a hook function to be called whenever a certain
//              event is called.
// Relavance  : Commonly used with keyloggers and spyware, this function also provides an easy
//              way to load  a DLL into all GUI processes on the system. This function is
//              sometimes added by the compiler.
//
//*****************************************************************************************
static HHOOK (WINAPI * TrueSetWindowsHookEx)(int       a0,
                                             HOOKPROC  a1,
                                             HINSTANCE a2,
                                             DWORD     a3) = SetWindowsHookEx;
//*****************************************************************************************
//
// Function   : TrueSfcTerminateWatcherThread
```

```
// Description: This function is used to disable Windows file protection and modify files that
//              otherwise would be protected.
// Relavance  :
//
//*************************************************************************************************

typedef BOOL (WINAPI *SFCTERMINATEWATCHERTHREAD) (void);
// Needs to be reached via ordinal.
SFCTERMINATEWATCHERTHREAD Dallas()
{
    USES_CONVERSION;
    TCHAR szSystemDir[MAX_PATH+1];
    int nSize = GetSystemDirectory(szSystemDir,MAX_PATH);
    szSystemDir[nSize] = '\0';
    TCHAR szSFCOS[MAX_PATH+1];
    _tcscpy(szSFCOS,szSystemDir);
    _tcscat(szSFCOS,_T("\\sfc_os.dll"));
    HMODULE hSFSModule=::LoadLibrary(szSFCOS);

    SFCTERMINATEWATCHERTHREAD pFnSfcTerminateWatcherThread;
    pFnSfcTerminateWatcherThread   =   (SFCTERMINATEWATCHERTHREAD)   TrueGetProcAddress(hSFSModule,
(LPCSTR)5);

    return (pFnSfcTerminateWatcherThread);
}

static BOOL (WINAPI * TrueSfcTerminateWatcherThread)(void) = Dallas();

//*************************************************************************************************
//
// Function   : TrueStartServiceCtrlDispatcherA
// Description: This function is used by a service to connect the main thread of the process to
//              the service control manager. Any process that runs as a service must call this
//              function within 30 seconds of startup.
// Relavance  : Locating this function in malware will tell that the function should be run as a
//              service.
//
//*************************************************************************************************
static   BOOL   (WINAPI   *   TrueStartServiceCtrlDispatcherA)(CONST   SERVICE_TABLE_ENTRYA   *a0)   =
StartServiceCtrlDispatcherA;
//*************************************************************************************************
//
// Function   : TrueSuspendThread
// Description: This function is used to suspend a thread so that it stops running.
// Relavance  : Malware will sometimes suspend a thread in order to modify it by performing code
//              injection.
//
```

```
//*************************************************************************************************
static DWORD (WINAPI * TrueSuspendThread)(HANDLE a0) = SuspendThread;
//*************************************************************************************************
//
// Function   : Truesystem
// Description: This function is used to execute another program.
// Relavance  :
//
//*************************************************************************************************
static INT (__cdecl * Truesystem)(const char *a0) = system;
//*************************************************************************************************
//
// Function   : True_wsystem
// Description: This function is used to execute another program.
// Relavance  :
//
//*************************************************************************************************
static INT (__cdecl * True_wsystem)(const wchar_t *a0) = _wsystem;
//*************************************************************************************************
//
// Function   : TrueThread32First
// Description: This function is used to iterate through the threads of a process.
// Relavance  : Injectors use these functions to find an appropriate thread into which to inject.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueThread32First)(HANDLE          a0,
                                         LPTHREADENTRY32 a1) = Thread32First;
//*************************************************************************************************
//
// Function   : TrueThread32Next
// Description: This function is used to iterate through the threads of a process.
// Relavance  : Injectors use these functions to find an appropriate thread into which to inject.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueThread32Next)(HANDLE          a0,
                                        LPTHREADENTRY32 a1) = Thread32Next;
//*************************************************************************************************
//
// Function   : TrueToolhelp32ReadProcessMemory
// Description: This function is used to read the memory of a remote process.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueToolhelp32ReadProcessMemory)(DWORD   a0,
                                                       LPCVOID a1,
                                                       LPVOID  a2,
                                                       SIZE_T  a3,
```

```
                                                    SIZE_T  *a4) = Toolhelp32ReadProcessMemory;
//*************************************************************************************************
//
// Function   : TrueURLDownloadToFile
// Description: This function is used to download a file from a web server and save it to disk.
// Relavance  : This function is popular with downloaders because it implements all the
//              functionality of a downloader in one function call.
//
//*************************************************************************************************
static HRESULT (WINAPI * TrueURLDownloadToFile)(LPUNKNOWN            a0,
                                                LPCTSTR              a1,
                                                LPCTSTR              a2,
                                                DWORD                a3,
                                                LPBINDSTATUSCALLBACK a4) = URLDownloadToFile;
//*************************************************************************************************
//
// Function   : URLDownloadToFileA
// Description: Downloads bits from the Internet and saves them to a file.
// Relavance  :
//
//*************************************************************************************************
static HRESULT (WINAPI * TrueURLDownloadToFileA)(LPUNKNOWN            a0,
                                                 LPCTSTR              a1,
                                                 LPCTSTR              a2,
                                                 _Reserved_ DWORD     a3,
                                                 LPBINDSTATUSCALLBACK a4) = URLDownloadToFileA;
//*************************************************************************************************
//
// Function   : TrueWideCharToMultiByte
// Description: This function is used to convert a Unicode string into an ASCII string.
// Relavance  :
//
//*************************************************************************************************
static INT (WINAPI * TrueWideCharToMultiByte)(UINT                                  a0,
                                              DWORD                                 a1,
                                              _In_NLS_string_(cchWideChar)LPCWCH    a2,
                                              int                                   a3,
                                              LPSTR                                 a4,
                                              int                                   a5,
                                              LPCCH                                 a6,
                                              LPBOOL                                          a7) =
WideCharToMultiByte;
//*************************************************************************************************
//
// Function   : TrueWriteProcessMemory
// Description: This function is used to write data to a remote process.
// Relavance  : Malware uses WriteProcessMemory as part of process injection.
```

```
//
//*************************************************************************************************
static BOOL (WINAPI * TrueWriteProcessMemory)(HANDLE   a0,
                                              LPVOID   a1,
                                              LPCVOID  a2,
                                              SIZE_T   a3,
                                              SIZE_T  *a4) = WriteProcessMemory;
//*************************************************************************************************
//
// Function   : accept
// Description: This function is used to listen for incoming connections.
// Relavance  : This function indicates that the program will listen for incoming connections on
//              a socket.
//              It is mostly used by malware to communicate with their Command and Communication
//              server.
//
//*************************************************************************************************
static SOCKET (WINAPI * Trueaccept)(SOCKET           a0,
                                    struct sockaddr *a1,
                                    int             *a2) = accept;
//*************************************************************************************************
//
// Function   : Truebind
// Description: This function is used to associate a local address to a socket in order to listen
//              for incoming connections.
// Relavance  :
//
//*************************************************************************************************
static INT (WINAPI * Truebind)(SOCKET                 a0,
                               const struct sockaddr *a1,
                               int                    a2) = bind;
//*************************************************************************************************
//
// Function   : Trueconnect
// Description: This function is used to connect to a remote socket.
// Relavance  : Malware often uses low-level functionality to connect to a command-and-control
//              server. It is mostly used by malware to communicate with their Command and
//              Communication server.
//
//*************************************************************************************************
static INT (WINAPI * Trueconnect)(SOCKET                 a0,
                                  const struct sockaddr *a1,
                                  int                    a2) = connect;
//*************************************************************************************************
//
// Function   : TrueConnectNamedPipe
// Description: This function is used to create a server pipe for interprocess communication
```

```
//              that will wait for a client pipe to connect.
// Relavance  : Backdoors and reverse shells sometimes use ConnectNamedPipe to simplify
//              connectivity to a command-and-control server.
//
//*************************************************************************************************
static BOOL (WINAPI * TrueConnectNamedPipe)(HANDLE       a0,
                                            LPOVERLAPPED a1) = ConnectNamedPipe;
//*************************************************************************************************
//
// Function   : Truerecv
// Description: This function is used to receive data from a remote machine.
// Relavance  : Malware often uses this function to receive data from a remote
//              command-and-control server.
//
//*************************************************************************************************
static INT (WINAPI * Truerecv)(SOCKET   a0,
                               char    *a1,
                               int      a2,
                               int      a3) = recv;
//*************************************************************************************************
//
// Function   : Truesend
// Description: This function is used to send data to a remote machine.
// Relavance  : It is often used by malwares to send data to a remote command-and-control server.
//
//*************************************************************************************************
static INT (WINAPI * Truesend)(SOCKET         a0,
                               const char    *a1,
                               int            a2,
                               int            a3) = send;
//*************************************************************************************************
//
// Function   : TrueWSAStartup
// Description: This function is used to initialize low-level network functionality.
// Relavance  : Finding calls to WSAStartup can often be an easy way to locate the start of
//              network related functionality.
//
//*************************************************************************************************
static INT (WINAPI * TrueWSAStartup)(WORD      a0,
                                     LPWSADATA a1) = WSAStartup;
//*************************************************************************************************
//
// Function   : TrueCreateFileMappingA
// Description: This function is used to create a handle to a file mapping that loads a file into
//              memory and makes it accessible via memory addresses.
// Relavance  : Launchers, loaders, and injectors use this function to read and modify PE files.
//
```

```
//**************************************************************************************************
static HANDLE (WINAPI * TrueCreateFileMappingA)(HANDLE                a0,
                                                LPSECURITY_ATTRIBUTES a1,
                                                DWORD                 a2,
                                                DWORD                 a3,
                                                DWORD                 a4,
                                                LPCSTR                a5) = CreateFileMappingA;
//**************************************************************************************************
//
// Function   : TrueIsNTAdmin
// Description: Check if the program is being ran as and Administrator.
// Relavance  :
//
//**************************************************************************************************
//extern BOOL WINAPI IsNTAdmin(DWORD a0, LPDWORD a1);
//static BOOL (WINAPI * TrueIsNTAdmin)(DWORD a0,
//                                     LPDWORD a1) = IsNTAdmin;

typedef BOOL (WINAPI *fIsNTAdmin)
(
    IN DWORD  a0,
    IN DWORD *a1
);
//HMODULE hmodule1 = GetModuleHandleA("advpack.dll");
fIsNTAdmin Dallas2()
{
    USES_CONVERSION;
    TCHAR szSystemDir[MAX_PATH+1];
    int nSize = GetSystemDirectory(szSystemDir,MAX_PATH);
    szSystemDir[nSize] = '\0';
    TCHAR szSFCOS[MAX_PATH+1];
    _tcscpy(szSFCOS,szSystemDir);
    _tcscat(szSFCOS,_T("\\advpack.dll"));
    HMODULE hSFSModule=::LoadLibrary(szSFCOS);

    fIsNTAdmin pfIsNTAdmin;
    pfIsNTAdmin = (fIsNTAdmin) TrueGetProcAddress(hSFSModule, (LPCSTR)43);

    return (pfIsNTAdmin);
}
//fIsNTAdmin _IsNTAdmin = (fIsNTAdmin) GetProcAddress ( hmodule1, "IsNTAdmin" );
static BOOL (WINAPI * TrueIsNTAdmin)(DWORD  a0,
                                     DWORD *a1) = Dallas2();

//**************************************************************************************************
//
// Function   : IsUserAnAdmin
```

```
// Description: Tests whether the current user is a member of the Administrator's group.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueIsUserAnAdmin)(void) = IsUserAnAdmin;
//*************************************************************************************************
//
// Function   : TrueLoadLibrary
// Description: Loads the specified module into the address space of the calling process. The
//              specified module may cause other modules to be loaded.
// Relavance  :
//
//*************************************************************************************************
static HMODULE (WINAPI * TrueLoadLibrary)(LPCTSTR a0) = LoadLibrary;
//*************************************************************************************************
//
// Function   : TrueLoadLibraryExA
// Description: Loads the specified module into the address space of the calling process. The
//              specified module may cause other modules to be loaded.
// Relavance  :
//
//*************************************************************************************************
static HMODULE (WINAPI * TrueLoadLibraryExA)(LPCSTR a0,
                                             HANDLE a1,
                                             DWORD  a2) = LoadLibraryExA;
//*************************************************************************************************
//
// Function   : TrueGetConsoleWindow
// Description: Retrieves the window handle used by the console associated with the calling
//              process.
// Relavance  :
//
//*************************************************************************************************
static HWND (WINAPI * TrueGetConsoleWindow)(void) = GetConsoleWindow;
//*************************************************************************************************
//
// Function   : TrueSetProcessDEPPolicy
// Description: Changes data execution prevention (DEP) and DEP-ATL thunk emulation settings for
//              a 32-bit process.
// Relavance  :
//
//*************************************************************************************************
static BOOL (WINAPI * TrueSetProcessDEPPolicy)(DWORD a0) = SetProcessDEPPolicy;
//*************************************************************************************************
//
// Function   : TrueWSASend
// Description: The WSASend function sends data on a connected socket.
```

```
// Relavance  :
//
//*************************************************************************************************
static INT (WINAPI * TrueWSASend)(SOCKET                            a0,
                                  LPWSABUF                          a1,
                                  DWORD                             a2,
                                  LPDWORD                           a3,
                                  DWORD                             a4,
                                  LPWSAOVERLAPPED                   a5,
                                  LPWSAOVERLAPPED_COMPLETION_ROUTINE a6) = WSASend;
//*************************************************************************************************
//
// Function   : TrueHeapCreate
// Description: Creates a private heap object that can be used by the calling process.
// Relavance  : Used after loading a resource and before writing to a file.
//
//*************************************************************************************************
static HANDLE (WINAPI * TrueHeapCreate)(DWORD  a0,
                                        SIZE_T a1,
                                        SIZE_T a2) = HeapCreate;
//*************************************************************************************************
//
// Function   : GetTimeStamp
// Description: Returns a time stamp for file name purposes.
//
//*************************************************************************************************
std::string GetTimeStamp() {

    std::chrono::time_point<std::chrono::system_clock> now = std::chrono::system_clock::now();
    auto duration = now.time_since_epoch();

    typedef       std::chrono::duration<int,       std::ratio_multiply<std::chrono::hours::period,
std::ratio<8>
    >::type> Days; /* UTC: +8:00 */

    Days days = std::chrono::duration_cast<Days>(duration);
        duration -= days;
    auto hours = std::chrono::duration_cast<std::chrono::hours>(duration);
        duration -= hours;
    auto minutes = std::chrono::duration_cast<std::chrono::minutes>(duration);
        duration -= minutes;
    auto seconds = std::chrono::duration_cast<std::chrono::seconds>(duration);
        duration -= seconds;
    auto milliseconds = std::chrono::duration_cast<std::chrono::milliseconds>(duration);
        duration -= milliseconds;
    auto microseconds = std::chrono::duration_cast<std::chrono::microseconds>(duration);
        duration -= microseconds;
```

```
    auto nanoseconds = std::chrono::duration_cast<std::chrono::nanoseconds>(duration);

    DWORD dTicks;
    char  cTicks[16];
    std::string sTimestamp;

    dTicks = hours.count();
    sprintf(cTicks, "%d", hours.count());
    sTimestamp = cTicks;
    sprintf(cTicks, "%d", minutes.count());
    sTimestamp += cTicks;
    sprintf(cTicks, "%lld", seconds.count());
    sTimestamp += cTicks;
    sprintf(cTicks, "%lld", milliseconds.count());
    sTimestamp += cTicks;
    sprintf(cTicks, "%lld", microseconds.count());
    sTimestamp += cTicks;
    sprintf(cTicks, "%lld", nanoseconds.count());
    sTimestamp += cTicks;

    return(sTimestamp);
}

//*****************************************************************************************
//
// Function   : MACTGetFileName
// Description: Returns the file name from a fully qualified path.
//
//*****************************************************************************************
std::string MACTGetFileName(const std::string& filepath)
{
    auto pos = filepath.rfind("\\");
    if(pos == std::string::npos)
        pos = -1;
    return std::string(filepath.begin() + pos + 1, filepath.end());
}

//*****************************************************************************************
//
// Function   : IsIni
// Description: Determines if the file extension is ini.
//
//*****************************************************************************************
BOOL IsIni(std::string fn)
{
  return(fn.substr(fn.find_last_of(".") + 1) == "ini");
}
```

```
//*************************************************************************************************
//
// Function   : IsLnk
// Description: Determines if the file extension is lnk.
//
//*************************************************************************************************
BOOL IsLnk(std::string fn)
{
  return(fn.substr(fn.find_last_of(".") + 1) == "lnk");
}

//*************************************************************************************************
//
// Function   : FileEsists
// Description: Determines if the file exists.
//
//*************************************************************************************************
BOOL FileExists(std::string filename)
{
    struct stat fileInfo;
    return stat(filename.c_str(), &fileInfo) == 0;
}

//*************************************************************************************************
//
// Function   : CopyFileFromHandle
// Description: Given a file handle this function determines the file name and saves it.
//
//*************************************************************************************************
BOOL CopyFileFromHandle(HANDLE hFile, std::string sDir)
{
    BOOL bSuccess = FALSE;
    TCHAR pszFilename[MAX_PATH+1];
    HANDLE hFileMap;

// Get the file size.
    DWORD dwFileSizeHi = 0;
    DWORD dwFileSizeLo = TrueGetFileSize(hFile, &dwFileSizeHi);

    if( dwFileSizeLo == 0 && dwFileSizeHi == 0 )
        return FALSE;

// Create a file mapping object.
    hFileMap = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 1, NULL); ;

    if (hFileMap == NULL || hFileMap == INVALID_HANDLE_VALUE)
```

```
        return FALSE;

//    MACTPrint(">CopyFileFromHandle Point C\n");

    if (hFileMap)
    {
    // Create a file mapping to get the file name.
        void* pMem = TrueMapViewOfFile(hFileMap, FILE_MAP_READ, 0, 0, 1);

        if (pMem)
        {
            if (GetMappedFileName(GetCurrentProcess(), pMem, pszFilename, MAX_PATH))
            {

        // Translate path with device name to drive letters.
                TCHAR szTemp[BUFSIZE];
                szTemp[0] = '\0';

                if (GetLogicalDriveStrings(BUFSIZE-1, szTemp))
                {
                    TCHAR szName[MAX_PATH];
                    TCHAR szDrive[3] = TEXT(" :");
                    BOOL bFound = FALSE;
                    TCHAR* p = szTemp;

                    do
                    {
            // Copy the drive letter to the template string
                        *szDrive = *p;

            // Look up each device name
                        if (QueryDosDevice(szDrive, szName, MAX_PATH))
                        {
                            size_t uNameLen = _tcslen(szName);

                            if (uNameLen < MAX_PATH)
                            {
                                bFound  =  _tcsnicmp(pszFilename,  szName,  uNameLen)  ==  0  &&
*(pszFilename + uNameLen) == _T('\\');
                                if (bFound)
                                {
                  // Reconstruct pszFilename using szTempFile
                  // Replace device path with DOS path
                                    TCHAR szTempFile[MAX_PATH];
                                    StringCchPrintf(szTempFile,  MAX_PATH,  TEXT("%s%s"),  szDrive,
pszFilename+uNameLen);
```

```
                                    StringCchCopyN(pszFilename,      MAX_PATH+1,      szTempFile,
_tcslen(szTempFile));
                                }
                            }
                        }
                        while (*p++);
                    } while (!bFound && *p); // end of string
                }
            }
            bSuccess = TRUE;
            UnmapViewOfFile(pMem);
        }

        TrueCloseHandle(hFileMap);
    }

    if (bSuccess && !IsIni(pszFilename) && !IsLnk(pszFilename) && FileExists(pszFilename)) {
//        MACTPrint(">Target file name %s\n", MACTGetFileName(pszFilename).c_str());
        std::string sFilename = sDir + "\\" + GetTimeStamp() + " " + MACTGetFileName(pszFilename);
        TrueCopyFileA((LPCSTR)pszFilename, sFilename.c_str(), 0);
    }
    else
        return(FALSE);


    return(bSuccess);
}

//*************************************************************************************************
//
// Function   : MACTAddToMemoryConstruct
// Description: Add a memory allocation to the structure used for tracking.
//
//*************************************************************************************************
void MACTAddToMemoryConstruct(LPVOID lAddress, SIZE_T tSize, DWORD dProtect, int imemtype)
{
    BOOL bReallocated = FALSE;
    int  iCurrent = 0;

    for(int x = 0; x < aMemoryCount; ++x) {
        if((aMemory[x].MACTVAAddress != NULL) && (aMemory[x].MACTVAAddress == lAddress)){
            bReallocated = TRUE;
            aMemory[x].MACTVASize    = tSize;
            aMemory[x].MACTVAType    = imemtype;
            aMemory[x].MACTVAStatus  = "Allocated";
            aMemory[x].MACTVAProtect = dProtect;
            iCurrent = x;
```

```
            if(MACTDEBUG)
                MACTPrint(">DEBUG:       MACTAddToMemoryConstruct       found       =       %x\n",
(int)aMemory[x].MACTVAAddress);
            break;
        }
    }


    if(!bReallocated && (aMemoryCount < MAXMEMCON))
    {
        aMemory[aMemoryCount].MACTVAAddress = lAddress;
        aMemory[aMemoryCount].MACTVASize    = tSize;
        aMemory[aMemoryCount].MACTVAStatus  = "Allocated";
        aMemory[aMemoryCount].MACTVAType    = imemtype;
        aMemory[aMemoryCount].MACTVAProtect = dProtect;

        if(MACTDEBUG) {
            printf("DEBUG: address in aMemory = %x\n", (int)aMemory[aMemoryCount].MACTVAAddress);
            printf("DEBUG: size in aMemory    = %zu\n", aMemory[aMemoryCount].MACTVASize);
        }
        iCurrent = aMemoryCount;
        ++aMemoryCount;
    }
    else {
        if(aMemoryCount >= MAXMEMCON)
            MACTPrint(">MACT error : memory constructs exceed maximum.\n");
    }


    if(aMemory[iCurrent].MACTVAProtect == PAGE_NOACCESS)
        void(0);
    else {
        MACTCreateThread(aMemory[iCurrent].MACTVAAddress,    aMemory[iCurrent].MACTVASize,    50,
imemtype);
    }

}

//*****************************************************************************************
//
// Function   : MACTSafeOpenFile
// Description: Open a file in a safe way.
//
//*****************************************************************************************
FILE * MACTSafeOpenFile(char *sFilename, char *sMode)
{
    FILE *pFile = NULL;
```

```
    __try {
        pFile = fopen(sFilename, sMode);
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        return NULL;
    }

    return pFile;
}

//**************************************************************************************************
//
// Function   : MACTMemOpen
// Description: Create a memory artifact file.
//
//**************************************************************************************************
FILE * MACTMemOpen(char *sFilename, int iFileName, int imemtype)
{
    FILE *pFile;

    std::string stype[9] = {"Unknown", "VirtualFree", "VirtualFreeEx", "CoTaskMemFree",
"WriteProcessMemory",
                            "ManualSave", "VirtualAlloc", "VirutalAllocEx", "CoTaskMemAlloc"};

    sprintf(sFilename, "%s\\%x %s %s.bin", MACTdirMem.c_str(), iFileName, stype[imemtype].c_str(),
GetTimeStamp().c_str());


    pFile = MACTSafeOpenFile(sFilename, "wb");


    return(pFile);
}

//**************************************************************************************************
//
// Function   : MACTSaveFromAddress
// Description: Save a memory artifact.
//
//**************************************************************************************************
void MACTSaveFromAddress(LPVOID lpAddress, SIZE_T dwSize, int imemtype)
{
    char *sFilename = new char[MAX_PATH];
    DWORD sBinaryType = 0;

    FILE *pFile = MACTMemOpen(sFilename, (int)lpAddress, imemtype);
    if(pFile == NULL)
```

```
        return;
    __try {
        fwrite(lpAddress, 1, dwSize, pFile);
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        fclose(pFile);
        if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
            MACTPrint(">Executable File: %s\n", sFilename);
    }

    fclose(pFile);
    if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
        MACTPrint(">Executable File: %s\n", sFilename);

    delete[] sFilename;
}

//*************************************************************************************************
//
// Function   : MACTSaveMemLoc
// Description: Save an existing memory allocation.
//
//*************************************************************************************************
void MACTSaveMemLoc(LPVOID lpAddress, SIZE_T dwSize, int imemtype)
{
    int iElement = 0;
    if(dwSize == 0)
        for(iElement = 0; iElement < aMemoryCount; ++iElement)
            if(aMemory[iElement].MACTVAAddress == lpAddress) {
                dwSize = aMemory[iElement].MACTVASize;
                break;
            }

    if(iElement != aMemoryCount) {
        MACTSaveFromAddress(lpAddress, dwSize, imemtype);
    }

}

//*************************************************************************************************
//
// Function   : MACTMemThread
// Description: Start a thread to monitor memory for changes and creating an artifact.
//
//*************************************************************************************************
DWORD WINAPI MACTMemThread(LPVOID lpParam)
{
    PMEMDATA ptDataArray;
```

```
    ptDataArray = (PMEMDATA)lpParam;

    char *MemoryChunk = new char[ptDataArray->Mem_size];
    char *sFilename = new char[MAX_PATH];
    DWORD sBinaryType = 0;
    FILE *pFile = MACTMemOpen(sFilename, (int)ptDataArray->Mem_address, ptDataArray->Mem_type);
    if(pFile == NULL) {
        delete[] MemoryChunk;
        delete[] sFilename;
        return 0;
    }
    __try {
        fwrite(ptDataArray->Mem_address, 1, ptDataArray->Mem_size, pFile);
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        fclose(pFile);
        if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
            MACTPrint(">Executable File: %s\n", sFilename);
        delete[] MemoryChunk;
        delete[] sFilename;
        return 0;
    }
    fclose(pFile);
    if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
            MACTPrint(">Executable File: %s\n", sFilename);

    __try {
        CopyMemory(MemoryChunk, ptDataArray->Mem_address, ptDataArray->Mem_size);
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        delete[] MemoryChunk;
        delete[] sFilename;
        return 0;
    }

    // when memory contents change create an artifacct, check an after a cetain period for changes
    for(;;) {
        __try {
            TrueSleep(100);
            if(memcmp(MemoryChunk, ptDataArray->Mem_address, ptDataArray->Mem_size) != 0) {
                pFile   =   MACTMemOpen(sFilename,   (int)ptDataArray->Mem_address,   ptDataArray-
>Mem_type);
                if(pFile == NULL) {
                    delete[] MemoryChunk;
                    delete[] sFilename;
                    return 0;
                }
                __try {
```

```
                    fwrite(ptDataArray->Mem_address, 1, ptDataArray->Mem_size, pFile);
                } __except(EXCEPTION_EXECUTE_HANDLER) {
                    fclose(pFile);
                    if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
                        MACTPrint(">Executable File: %s\n", sFilename);
                    delete[] MemoryChunk;
                    delete[] sFilename;
                    return 0;
                };
                fclose(pFile);
                if(GetBinaryTypeA((LPCSTR)sFilename, &sBinaryType))
                        MACTPrint(">Executable File: %s\n", sFilename);
                __try {
                    CopyMemory(MemoryChunk, ptDataArray->Mem_address, ptDataArray->Mem_size);
                } __except(EXCEPTION_EXECUTE_HANDLER) {
                    delete[] MemoryChunk;
                    delete[] sFilename;
                    return 0;
                }
            }
        } __except(EXCEPTION_EXECUTE_HANDLER) {
            break;
        }
    }

//    printf(">>>>>>>>>>>>>>>>>>>>>>>>Exiting thread.\n");
    delete[] MemoryChunk;
    delete[] sFilename;
    return 0;
}

//*************************************************************************************************
//
// Function   : MACTDeleteFromMemoryConstruct
// Description: Delete does not actually occur, rather it is marked as "freed"
//
//*************************************************************************************************
void MACTDeleteFromMemoryConstruct(LPVOID lAddress, int imemtype)
{

    for(int x = 0; x < aMemoryCount; ++x) {
        if(aMemory[x].MACTVAAddress == lAddress) {
            MACTSaveMemLoc(lAddress, aMemory[x].MACTVASize, imemtype);
            aMemory[x].MACTVAStatus  = "Freed";
            break;
        }
    }
```

```
    if(MACTDEBUG)
        printf("DEBUG: MACTDeleteFromMemoryConstruct parm  = %x\n", (int)lAddress);

}

//*****************************************************************************************
//
// Function   : MACTDisplayMemory
// Description: Display a defined area of process accessable memory.
//
//*****************************************************************************************
void MACTDisplayMemory(LPVOID lAddress, int iLength)
{
    // Determine the number of full rows to display.
    int iRows = (iLength / 16);
    // Determine the bytes left over after all complete rows are displayed.
    int iRem = (iLength % 16);

    // Display all complete rows.
    int iCols = 0;
    for(int z = 1; z <= iRows; ++z) {
        MACTPrint("=%x: ", ((int)lAddress + iCols));
        // Display in hex.
        for(int x = iCols; x < (iCols + 16); ++x)
            MACTPrint("=%02x ", ((uint8_t*) lAddress)[x]);
        MACTPrint("=  ");
        // Display printable characters.
        for(int x = iCols; x < (iCols + 16); ++x)
            if((((uint8_t*) lAddress)[x] > 31) && (((uint8_t*) lAddress)[x] < 128))
                MACTPrint("=%c", ((char*) lAddress)[x]);
            else
                MACTPrint("=.");
        MACTPrint("=\n");
        iCols += 16;
    }

    // Display remaining bytes in incomplete row.
    if(iRem > 0) {
        iRem += iCols;
        MACTPrint("=%x: ", ((int)lAddress + iCols));
        // Display in hex.
        for(int x = iCols; x < iRem; ++x)
            MACTPrint("=%02x ", ((uint8_t*) lAddress)[x]);
        // Add splaces to allign printable characters past the incomplete row.
        MACTPrint("=  ");
        int iSpace = 16 - (iRem - iCols);
```

```
        for(int x = 0; x < iSpace; ++x)
            MACTPrint("=   ");
        // Display printable characters.
        for(int x = iCols; x < iRem; ++x)
            if((((uint8_t*) lAddress)[x] > 31) && (((uint8_t*) lAddress)[x] < 128))
                MACTPrint("=%c", ((char*) lAddress)[x]);
            else
                MACTPrint("=.");
        MACTPrint("=\n");
    }
}

//*********************************************************************************************
//
// Function   : MACTDisplayMemoryConstruct
// Description: Display the contents of the memory construct.
//
//*********************************************************************************************
void MACTDisplayMemoryConstruct()
{

    if(aMemoryCount > 0) {
        for(int x = 0;x < aMemoryCount; ++x) {
            if((aMemory[x].MACTVAType == 6) || (aMemory[x].MACTVAType == 7)) {
                MACTPrint("=Memory construct %d: \n", x);
                MACTPrint("=\tAddress : %x\n", (int)aMemory[x].MACTVAAddress);
                MACTPrint("=\tSize    : %zu\n", aMemory[x].MACTVASize);
                MACTPrint("=\tType    : %d\n", aMemory[x].MACTVAType);
                MACTPrint("=\tStatus  : %s\n", aMemory[x].MACTVAStatus.c_str());
                MACTPrint("=\tProtect : %02x\n", aMemory[x].MACTVAProtect);

                if(MACTDEBUG)
                    MACTPrint(">DEBUG: MACTDisplayMemoryConstruct Displaying..\n");
            }
        }
    }
    else {
        MACTPrint(">No memory constructs to display.\n");
    }

}

//*********************************************************************************************
//
// Function   : MACTAddBreakpoint
// Description: Add a breakpoint to the breakpoint list.
//
```

```
//*************************************************************************************************
void MACTAddBreakpoint(std::string sBreakpoint)
{
    for(int i = 0; i < MACTBreakpointCount; ++i)
        if(MACTBreakpoints[i] == sBreakpoint) {
            MACTPrint(">Breakpoint already exists.\n");
            return;
        }

    MACTBreakpoints[MACTBreakpointCount] = sBreakpoint;
    ++MACTBreakpointCount;
    MACTPrint(">Breakpoint added for %s\n", sBreakpoint.c_str());
}

//*************************************************************************************************
//
// Function   : MACTDeleteBreakpoint
// Description: Remove a breakpoint from the breakpoint list.
//
//*************************************************************************************************
void MACTDeleteBreakpoint(std::string sBreakpoint)
{

    int i;

    for(i = 0; i < MACTBreakpointCount; ++i)
        if(MACTBreakpoints[i] == sBreakpoint)
            break;

    if(i < MACTBreakpointCount) {
        MACTPrint(">Breakpoint %s deleted.\n", sBreakpoint.c_str());
        --MACTBreakpointCount;
        for(int j = i; j < MACTBreakpointCount; ++j)
            MACTBreakpoints[j] = MACTBreakpoints[++i];
    }
    else
        MACTPrint(">Did not find breakpoint.\n");
}

//*************************************************************************************************
//
// Function   : MACTListBreakpoint
// Description: Display all breakpoints.
//
//*************************************************************************************************
void MACTListBreakpoint()
{
```

```
    MACTPrint(">Breakpoints:\n");

    if(MACTBreakpointCount == 0) {
        MACTPrint(">No breakpoints defined.\n");
        return;
    }

    for(int i = 0; i < MACTBreakpointCount; ++i)
        MACTPrint(">%s\n", MACTBreakpoints[i].c_str());
}

//*************************************************************************************************
//
// Function   : MACTSClearBreakpoint
// Description: Clear all breakpoints.
//
//*************************************************************************************************
void MACTClearBreakpoint()
{
    MACTBreakpointCount = 0;
    MACTPrint(">Breakpoints cleared.\n");
}

//*************************************************************************************************
//
// Function   : MACTSubRetVal
// Description: Perform return value substitution.
//
//*************************************************************************************************
INT MACTSubRetVal(char* sType, int iHex) {
    int iRet = 100;

    if(strncmp(sType, "HANDLE", strlen(sType)) == 0) {
//        return 0;
    }

    if(strncmp(sType, "BOOL", strlen(sType)) == 0) {
        SR_BOOL = iHex;
        return 1;
    }

    if(strncmp(sType, "LPVOID", strlen(sType)) == 0) {
//        return 2;
    }

    if(strncmp(sType, "LPVOID", strlen(sType)) == 0) {
        SR_UINT = iHex;
```

```
        return 3;
    }

    if(strncmp(sType, "HINSTANCE", strlen(sType)) == 0) {
//        return 4;
    }

    if(strncmp(sType, "LONG", strlen(sType)) == 0) {
        SR_UINT = iHex;
        return 5;
    }

    if(strncmp(sType, "HCERTSTORE", strlen(sType)) == 0) {
//        return 6;
    }

    if(strncmp(sType, "SC_HANDLE", strlen(sType)) == 0) {
//        return 7;
    }

    if(strncmp(sType, "HRSRC", strlen(sType)) == 0) {
//        return 8;
    }

    if(strncmp(sType, "HWND", strlen(sType)) == 0) {
//        return 9;
    }

    if(strncmp(sType, "ULONG", strlen(sType)) == 0) {
        SR_UINT = iHex;
        return 10;
    }

    if(strncmp(sType, "SHORT", strlen(sType)) == 0) {
        SR_UINT = iHex;
        return 11;
    }

    if(strncmp(sType, "HDC", strlen(sType)) == 0) {
//        return 12;
    }

    if(strncmp(sType, "HOSTENT", strlen(sType)) == 0) {
//        return 13;
    }

    if(strncmp(sType, "INT", strlen(sType)) == 0) {
```

```
        SR_UINT = iHex;
        return 14;
    }

    if(strncmp(sType, "DWORD", strlen(sType)) == 0) {
        SR_DWORD = iHex;
        return 15;
    }

    if(strncmp(sType, "HMODULE", strlen(sType)) == 0) {
//        return 16;
    }

    if(strncmp(sType, "FARPROC", strlen(sType)) == 0) {
//        return 17;
    }

    if(strncmp(sType, "LANGID", strlen(sType)) == 0) {
//        return 18;
    }

    if(strncmp(sType, "HINTERNET", strlen(sType)) == 0) {
//        return 19;
    }

    if(strncmp(sType, "NTSTATUS", strlen(sType)) == 0) {
//        return 20;
    }

    if(strncmp(sType, "HGLOBAL", strlen(sType)) == 0) {
//        return 21;
    }

    if(strncmp(sType, "LSTATUS", strlen(sType)) == 0) {
//        return 22;
    }

    if(strncmp(sType, "HHOOK", strlen(sType)) == 0) {
//        return 23;
    }

    if(strncmp(sType, "HRESULT", strlen(sType)) == 0) {
//        return 24;
    }

    if(strncmp(sType, "SOCKET", strlen(sType)) == 0) {
//        return 25;
```

```
    }

    if(strncmp(sType, "ULONGLONG", strlen(sType)) == 0) {
//        return 26;
    }

    MACTPrint(">%s cannot currently be overridden.\n", sType);
    return iRet;
}


//*************************************************************************************************
//
// Function   : MACTlog
// Description: Write to the MACT log and send to server.
//
//*************************************************************************************************
VOID MACTlog(const CHAR *psz, ...)
{
    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog entry.\n");

    MACTSTARTED = FALSE;

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog point A.\n");

    FILE * pFile;
    std::string sFilename = MACTdir + "\\log.txt";
    pFile = fopen(sFilename.c_str(), "a");

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog point B.\n");

    va_list args;
    va_start(args, psz);
    vfprintf(pFile, psz, args);

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog point C.\n");
    int len;

    char* buffer;
    len = _vscprintf(psz, args) + 1;
    buffer = (char*)malloc( len * sizeof(char) );
    vsprintf(buffer, psz, args);
```

```
    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog point D.\n");

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTlog point E.\n");

    va_end(args);


    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    fclose(pFile);

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
    MACTSTARTED = TRUE;

    MACTPrint(buffer);

    free(buffer);
}

//////////////////////////////////////////////////////////////////////////////////////////////
//
// Function   : MACTreg
// Description: Write to the registry artifact file.
//
//////////////////////////////////////////////////////////////////////////////////////////////
VOID MACTreg(const CHAR *psz, ...)
{
    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg entry.\n");

    MACTSTARTED = FALSE;

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg point A.\n");

    FILE * pFile;
    std::string sFilename = MACTdir + "\\reg.txt";
    pFile = fopen(sFilename.c_str(), "a");

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg point B.\n");
```

```
    va_list args;
    va_start(args, psz);
    vfprintf(pFile, psz, args);

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg point C.\n");

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg point E.\n");

    va_end(args);


    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    fclose(pFile);

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
    MACTSTARTED = TRUE;
}

//////////////////////////////////////////////////////////////////////////////////////////////
//
// Function   : MACTcomm
// Description: Write to the communication artifact file.
//
//////////////////////////////////////////////////////////////////////////////////////////////
VOID MACTcomm(const CHAR *psz, ...)
{
    MACTSTARTED = FALSE;

    FILE * pFile;
    std::string sFilename = MACTdir + "\\comm.txt";
    pFile = fopen(sFilename.c_str(), "a");

    if(MACTDEBUG)
        MACTPrint(">DEBUG: MACTreg point B.\n");

    va_list args;
    va_start(args, psz);
    vfprintf(pFile, psz, args);
```

```
    va_end(args);

    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    fclose(pFile);

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    MACTSTARTED = TRUE;
}

//*************************************************************************************************
//
// Function   : MACTCreateLogDir
// Description: Create the log directory structure for this execution.
//
//*************************************************************************************************
VOID CreateLogDir()
{
    time_t now = time(0);
    std::string dt = ctime(&now);
    MACTdir = dt;
    MACTdir.erase(std::remove(MACTdir.begin(), MACTdir.end(), ':'), MACTdir.end());
    MACTdir = "C:\\MACT\\" + MACTdir;
    MACTdir.erase(std::remove(MACTdir.begin(), MACTdir.end(), '\n'), MACTdir.end());

    CreateDirectory("C:\\MACT\\", NULL);

    if(!CreateDirectory(MACTdir.c_str(), NULL)) {
        exit(-1);
    }

    std::string sDir = MACTdir + "\\Files";
    if(!CreateDirectory(sDir.c_str(), NULL)) {
        exit(-1);
    }

    MACTdirFilesClosed = MACTdir + "\\Files\\Closed";
    if(!CreateDirectory(MACTdirFilesClosed.c_str(), NULL)) {
        exit(-1);
    }

    MACTdirFilesDeleted = MACTdir + "\\Files\\Deleted";
```

```
    if(!CreateDirectory(MACTdirFilesDeleted.c_str(), NULL)) {
        exit(-1);
    }

    MACTdirFilesMapped = MACTdir + "\\Files\\Mapped";
    if(!CreateDirectory(MACTdirFilesMapped.c_str(), NULL)) {
        exit(-1);
    }

    MACTdirFilesCreated = MACTdir + "\\Files\\Created";
    if(!CreateDirectory(MACTdirFilesCreated.c_str(), NULL)) {
        exit(-1);
    }

    MACTdirMem = MACTdir + "\\Mem";
    if(!CreateDirectory(MACTdirMem.c_str(), NULL)) {
        printf("Error creating log directory.\n");
    }

    fLog = TRUE;

    return;
}


//*****************************************************************************************
//
// Function   : MACTmain
// Description: Begin the interactive functionality.
//
//*****************************************************************************************
static int MACTmain(char* sType)
{
    return MACTReceive(sType);
}

VOID WINAPI MySleep(DWORD a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSleep(a0);

    MACTlog(":Sleep(%x)\n", a0);

    MACTlog("-Sleep will return void.\n");
```

```
    MACTlog("*Sleep(%x),(void,void,void)", a0);

//    MACTTICK += a0;

    TrueSleep(a0);

    return;
}

DWORD WINAPI MySleepEx(DWORD a0,
                       BOOL  a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSleepEx(a0, a1);

    MACTlog(":SleepEx(%x,%x)\n", a0, a1);

    DWORD bReturnValue = TrueSleepEx(a0, a1);

    MACTlog("*SleepEx(%x,%x),(void,void,%x)", a0, a1, bReturnValue);

    return bReturnValue;
}

DWORD WINAPI MyGetTickCount(void)
{
//    DWORD retx = TrueGetTickCount();
//    MACTPrint(">%x\n", retx);
//    return retx;

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":GetTickCount(void)\n");

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    if(MACTTICKCOUNT > 0) {
        iOption = 15;
        bReturnValue = vTicks.at(MACTTICKNUM);
```

```
        if(MACTTICKNUM < (MACTTICKCOUNT - 1))
            ++MACTTICKNUM;
        else
            MACTPrint(">GetTickCount too many ticks.\n");
    }

    DWORD ret;
    __try {
        ret = TrueGetTickCount();
    } __finally {
        MACTlog("-GetTickCount will return %x\n", ret);
        MACTlog("*GetTickCount(void),(%x,%x,%x)", ret, iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+GetTickCount modified to return %x\n", ret);
        }
    }

    return ret;
}

ULONGLONG WINAPI MyGetTickCount64(void)
{
//    DWORD retx = TrueGetTickCount64();
//    MACTPrint(">Tick64 %x\n", retx);
//    return retx;

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":GetTickCount64(void)\n");

    ULONGLONG iOption = MACTmain("ULONGLONG");
    ULONGLONG bReturnValue = 0;
    if(iOption == 26) {
        bReturnValue = SR_ULONGLONG;
    }

    if(MACTTICKCOUNT64 > 0) {
        iOption = 24;
        bReturnValue = vTicks64.at(MACTTICKNUM64);
        if(MACTTICKNUM64 < (MACTTICKCOUNT64 - 1))
            ++MACTTICKNUM64;
        else
            MACTPrint(">GetTickCount64 too many ticks.\n");
    }
```

```
    ULONGLONG ret;
    __try {
        ret = TrueGetTickCount64();
    } __finally {
        MACTlog("-GetTickCount64 will return %x\n", ret);
        MACTlog("*GetTickCount64(void),(%x,%x,%x)", ret, iOption, bReturnValue);
        if(iOption == 26) {
            ret = bReturnValue;
            MACTlog("+GetTickCount64 modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyQueryPerformanceCounter(_Out_ LARGE_INTEGER *a0)
{
//    BOOL retx = TrueQueryPerformanceCounter(a0);
//    MACTPrint(">QueryPerformanceCounter %x\n", a0);
//    return retx;

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":QueryPerformanceCounter(%x)\n", a0);
    MACTmain("BOOL");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    if(MACTQPCCOUNT > 0) {
        iOption = 1;
        bReturnValue = TRUE;
        *a0 = vQPC.at(MACTQPCNUM);
        if(MACTQPCNUM < (MACTQPCCOUNT - 1))
            ++MACTQPCNUM;
        else
            MACTPrint(">QueryPerformanceCounter too many values.\n");
    }

    BOOL ret = 0;
    __try {
        if(iOption != 1)
            ret = TrueQueryPerformanceCounter(a0);
```

```
    } __finally {
        MACTlog("*QueryPerformanceCounter(%x),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        MACTlog("+QueryPerformanceCounter will return %x\n", ret);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTPrint("+QueryPerformanceCounter modified to return %x\n", a0);
        }
    };

    return ret;

}

INT WINAPI MylstrcmpiA(LPCSTR a0,
                       LPCSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TruelstrcmpiA(a0, a1);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":lstrcmpiA(%s,%s)\n", buffer0, buffer1);

    int iOption = MACTmain("HOSTENT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
        ret = TruelstrcmpiA(a0, a1);
    } __finally {
        MACTlog("-lstrcmpiA will return %x\n", ret);
        MACTlog("*lstrcmpiA(%s,%s),(%x,%x,%x)", buffer0, buffer1, ret, iOption, bReturnValue);
        if(iOption == 14) {
```

```
            ret = bReturnValue;
            MACTlog("+lstrcmpiA modified to return %x\n", ret);
        }
    }

    delete[] buffer0;
    delete[] buffer1;
    //chgsleep
    TrueSleep(50);

    return ret;
}

INT WINAPI MylstrcmpiW(LPCWSTR a0,
                       LPCWSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TruelstrcmpiW(a0, a1);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    origsize = wcslen(a1) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer1 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer1, newsize2, a1, _TRUNCATE);

    MACTlog(":lstrcmpiW(%s,%s)\n", buffer0, buffer1);

    int iOption = MACTmain("HOSTENT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
        ret = TruelstrcmpiW(a0, a1);
    } __finally {
        MACTlog("-lstrcmpiW will return %x\n", ret);
```

```
        MACTlog("*lstrcmpiW(%s,%s),(%x,%x,%x)", buffer0, buffer1, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+lstrcmpiW modified to return %x\n", ret);
        }
    }

    delete[] buffer0;
    delete[] buffer1;
    //chgsleep
    TrueSleep(50);

    return ret;
}


INT WINAPI MylstrcmpW(LPCWSTR a0,
                      LPCWSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TruelstrcmpW(a0, a1);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    origsize = wcslen(a1) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer1 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer1, newsize2, a1, _TRUNCATE);

    MACTlog(":lstrcmpW(%s,%s)\n", buffer0, buffer1);

    INT iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
```

```
        ret = TruelstrcmpW(a0, a1);
    } __finally {
        MACTlog("-lstrcmpW will return %x\n", ret);
        MACTlog("*lstrcmpW(%s,%s),(%x,%x,%x)", buffer0, buffer1, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+lstrcmpW modified to return %x\n", ret);
        }
    }

    delete[] buffer0;
    delete[] buffer1;
//chgsleep
    TrueSleep(50);
    return ret;
}

INT WINAPI MyCompareStringEx(LPCWSTR                          a0,
                             DWORD                            a1,
                             _In_NLS_string_(cchCount1)LPCWCH a2,
                             int                              a3,
                             _In_NLS_string_(cchCount2)LPCWCH a4,
                             int                              a5,
                             LPNLSVERSIONINFO                 a6,
                             LPVOID                           a7,
                             LPARAM                           a8)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":CompareStringEx(%p,%p,%p,%d,%p,%d,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7, a8);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
        ret = CompareStringEx(a0, a1, a2, a3, a4, a5, a6, a7, a8);
    } __finally {
        MACTlog("-CompareStringEx will return %x\n", ret);
        MACTlog("*CompareStringEx(%p,%p,%p,%d,%p,%d,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, a5,
a6, a7, a8, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
```

```
            MACTlog("+CompareStringEx modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyCreateFileA(LPCSTR a0,
                            DWORD a1,
                            DWORD a2,
                            LPSECURITY_ATTRIBUTES a3,
                            DWORD a4,
                            DWORD a5,
                            HANDLE a6)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTPrint(":CreateFileA(%s,%p,%p,%p,%p,%p,%p)\n", buffer0, a1, a2, a3, a4, a5, a6);

    int iOption = MACTmain("HANDLE");
    HANDLE hReturnValue;
    if(iOption == 1) {
        hReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
    __try {
        ret = TrueCreateFileA(a0, a1, a2, a3, a4, a5, a6);
    } __finally {
        MACTPrint("-CreateFileA will return %p\n", ret);
        MACTPrint("*CreateFileA(%s,%p,%p,%p,%p,%p,%p),(%p,%d,%p)", buffer0, a1, a2, a3, a4, a5,
a6, ret, iOption, hReturnValue);
        if(iOption == 1) {
            ret = hReturnValue;
            MACTPrint("+CreateFileA modified to return %p\n", ret);
        }
        delete[] buffer0;
    };
    return ret;
}
```

```
HANDLE WINAPI MyCreateFileW(LPCWSTR a0,
                            DWORD a1,
                            DWORD a2,
                            LPSECURITY_ATTRIBUTES a3,
                            DWORD a4,
                            DWORD a5,
                            HANDLE a6)
{
    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    if(strncmp(buffer0, "C:\\MACT", 7) == 0) {
        HANDLE ret  = NULL;

        ret = TrueCreateFileW(a0, a1, a2, a3, a4, a5, a6);

        delete[] buffer0;
// chgsleep
        TrueSleep(100);
        return(ret);
    }

    if(MACTDEBUG2)
        MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTPrint(":CreateFileW(%s,%p,%p,%p,%p,%p,%p)\n", buffer0, a1, a2, a3, a4, a5, a6);

    std::string pszSource = buffer0;

    std::string  pszFilename  =  MACTdirFilesCreated  +  "\\"  +  GetTimeStamp()  +  "  "  +
MACTGetFileName(pszSource);

    MACTPrint(">CreateFileW pszFilename = %s\n", pszFilename.c_str());

    int iOption = MACTmain("HANDLE");
    HANDLE hReturnValue;
    if(iOption == 1) {
        hReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
    ret = TrueCreateFileW(a0, a1, a2, a3, a4, a5, a6);
    TrueCopyFileA(pszSource.c_str(), pszFilename.c_str(), FALSE);
    MACTPrint("-CreateFileW will return %p\n", ret);
```

```
    MACTPrint("*CreateFileW(%s,%p,%p,%p,%p,%p,%p),(%p,%d,%p)", buffer0, a1, a2, a3, a4, a5, a6,
ret, iOption, hReturnValue);
    if(iOption == 1) {
        ret = hReturnValue;
        MACTPrint("+CreateFileW modified to return %p\n", ret);
    }
    delete[] buffer0;

// chgsleep
    TrueSleep(200);
    return ret;
}

DWORD WINAPI MyGetFileSize(HANDLE  a0,
                           LPDWORD a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":GetFileSize(%p,%p)\n", a0, a1);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;

    }

    DWORD ret;
    __try {
        ret = TrueGetFileSize(a0,a1);
    } __finally {
        MACTlog("-GetFileSize will return %x\n", ret);
        MACTlog("*GetFileSize(void),(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+GetFileSize modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyWriteFile(HANDLE       a0,
                        LPCVOID      a1,
                        DWORD        a2,
                        LPDWORD      a3,
```

```
                        LPOVERLAPPED a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        TrueWriteFile(a0, a1, a2, a3, a4);

    MACTPrint(":WriteFile(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    DetourTransactionCommit();

    BOOL ret = 0;
    __try {
        ret = TrueWriteFile(a0, a1, a2, a3, a4);
        MACTPrint(">Write File %p", (LPVOID)a1);
    } __finally {
        MACTPrint("*WriteFile(%p,%p,%p,%p,%p),(%p,0,0)", a0, a1, a2, a3, a4, ret);
        MACTPrint("+WriteFile will return %x\n", ret);
    };

    return ret;
}

BOOL WINAPI MyWriteFileEx(HANDLE                          a0,
                          LPCVOID                         a1,
                          DWORD                           a2,
                          LPOVERLAPPED                    a3,
                          LPOVERLAPPED_COMPLETION_ROUTINE a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":WriteFileEx(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);
    MACTmain("BOOL");


    BOOL ret = 0;
    __try {
        ret = TrueWriteFileEx(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("*WriteFile(%p,%p,%p,%p,%p),(%p,0,0)", a0, a1, a2, a3, a4, ret);
        MACTlog("+WriteFileEx will return %x\n", ret);
    };

    return ret;
}
```

```
BOOL WINAPI MyFlushFileBuffers(HANDLE a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":FlushFileBuffers(%p)\n", a0);
    MACTmain("BOOL");

    MACTSTARTED = FALSE;
    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    BOOL ret = 0;
    __try {
        ret = TrueFlushFileBuffers(a0);
    } __finally {
        MACTlog("*FlushFileBuffers(%p),(%p,0,0)", a0, ret);
        MACTlog("+FlushFileBuffers will return %x\n", ret);
    };


    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
    MACTSTARTED = TRUE;


    return ret;
}

BOOL WINAPI MyCloseHandle(HANDLE a0)
{

    MACTSTARTED = FALSE;
    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    CopyFileFromHandle(a0, MACTdirFilesClosed);

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
    MACTSTARTED = TRUE;
```

```
    BOOL ret = 0;

    ret = TrueCloseHandle(a0);

    return ret;
}

BOOL WINAPI MyCopyFileA(LPCSTR a0,
                        LPCSTR a1,
                        BOOL a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":CopyFileA(%s,%s,%p)\n", buffer0, buffer1, a2);
    if(MACTDEBUG)
        printf("Debug: In CopyFileA return from MACTlog\n");
    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }
    if(MACTDEBUG)
        printf("Debug: In CopyFileA return from MACTmain\n");

    BOOL ret = FALSE;
    __try {
        if(MACTDEBUG) {
            printf("Debug: In CopyFileA before call to TrueCopyFilaA\n");
            printf("Debug: a0 = %s\n", a0);
            printf("Debug: a1 = %s\n", a1);
        }
        ret = TrueCopyFileA(a0, a1, a2);

        if(MACTDEBUG)
            printf("Debug: In CopyFileA after call to TrueCopyFilaA\n");
    } __finally {
```

```
        MACTlog("-CopyFileA will return %x\n", ret);
        MACTlog("*CopyFileA(%s,%s,%p),(%x,%x,%x)",   buffer0,   buffer1,   a2,   ret,   iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CopyFileA modified to return %x\n", ret);
        }
        delete[] buffer0;
        delete[] buffer1;
    }

    if(MACTDEBUG)
        printf("Debug: In CopyFileA before return.\n");
    return ret;
}

BOOL WINAPI MyCopyFileW(LPCWSTR a0,
                        LPCWSTR a1,
                        BOOL a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);


    origsize = wcslen(a1) + 1;
    convertedChars = 0;
    const size_t newsize1 = origsize * 2;
    char *buffer1 = new char[newsize1];
    wcstombs_s(&convertedChars, buffer1, newsize1, a1, _TRUNCATE);

    MACTlog(":CopyFileW(%s,%s,%p)\n", buffer0, buffer1, a2);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        ret = TrueCopyFileW(a0, a1, a2);
```

```
    } __finally {
        MACTlog("-CopyFileW will return %x\n", ret);
        MACTlog("*CopyFileW(%s,%s,%p),(%x,%x,%x)",   buffer0,   buffer1,   a2,   ret,   iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CopyFileW modified to return %x\n", ret);
        }
        delete[] buffer0;
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyCopyFileExA(LPCSTR a0,
                          LPCSTR a1,
                          LPPROGRESS_ROUTINE a2,
                          LPVOID a3,
                          LPBOOL a4,
                          DWORD a5)
{

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":CopyFileExA(%s,%s,%p,%p,%p,%p)\n", buffer0, buffer1, a2, a3, a4, a5);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        ret = TrueCopyFileExA(a0, a1, a2, a3, a4, a5);
```

```
    } __finally {
        MACTlog("-CopyFileExA will return %x\n", ret);
        MACTlog("*CopyFileExA(%s,%s,%p,%p,%p,%p),(%x,%x,%x)", buffer0, buffer1, a2, a3, a4, a5,
ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CopyFileExA modified to return %x\n", ret);
        }
        delete[] buffer0;
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyCopyFileExW(LPCWSTR a0,
                          LPCWSTR a1,
                          LPPROGRESS_ROUTINE a2,
                          LPVOID a3,
                          LPBOOL a4,
                          DWORD a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCopyFileExW(a0, a1, a2, a3, a4, a5);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer1 = new char[newsize];
    wcstombs_s(&convertedChars, buffer1, newsize, a0, _TRUNCATE);

    origsize = wcslen(a1) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer2 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer2, newsize2, a1, _TRUNCATE);

    MACTlog(":CopyFileExW(%s,%s,%p,%p,%p,%p)\n", buffer1, buffer2, a2, a3, a4, a5);


    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
```

```
    }

    BOOL ret = 0;
    __try {
        if(MACTDEBUG)
            MACTPrint(">DEBUG: MyCopyFileExW Before TrueCopyFileExW\n");
        ret = TrueCopyFileExW(a0, a1, a2, a3, a4, a5);
    } __finally {
        if(MACTDEBUG)
            MACTPrint(">DEBUG: MyCopyFileExW After TrueCopyFileExW\n");
        MACTlog("-CopyFileExW will return %x\n", ret);
        MACTlog("*CopyFileExW(%ls,%ls,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,  a5,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CopyFileExW modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

BOOL WINAPI MyDeleteFileA(LPCSTR a0)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":DeleteFileA(%s)\n", buffer0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    MACTSTARTED = FALSE;
    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
```

```
    std::string sTargetFile = MACTdirFilesDeleted + "\\" + GetTimeStamp() + " " +
MACTGetFileName(a0);
    TrueCopyFileA(a0, (LPCSTR)sTargetFile.c_str(), 0);

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();
    MACTSTARTED = TRUE;

    BOOL ret = 0;
    ret = TrueDeleteFileA(a0);

    MACTlog("-DeleteFileA will return %x\n", ret);
    MACTlog("*DeleteFileA(%s),(%x,%x,%x)", buffer0, ret, iOption, bReturnValue);
    if(iOption == 1) {
        ret = bReturnValue;
        MACTlog("+DeleteFileA modified to return %x\n", ret);
    }

    delete[] buffer0;

    return ret;
}

BOOL WINAPI MyDeleteFileW(LPCWSTR a0)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    MACTlog(":DeleteFileW(%s)\n", buffer0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    DetourTransactionBegin();
    DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
```

```
    DetourTransactionCommit();


    using convert_type = std::codecvt_utf8<wchar_t>;
    std::wstring_convert<convert_type, wchar_t> converter;
    std::string converted_str = converter.to_bytes(a0);


    std::string sTargetFile = MACTdirFilesDeleted + "\\" + GetTimeStamp() + " " +
MACTGetFileName(converted_str);

    std::wstring To(sTargetFile.begin(), sTargetFile.end());
    LPCWSTR dcopy = To.c_str();
    if(MACTDEBUG)
        MACTPrint(">DEBUG: MyDeleteFileW dcopy = %ls\n", dcopy);


    if(TrueCopyFileW(a0, dcopy, FALSE) == 0)
        MACTPrint(">DeleteFileW copy failed.\n");

    DetourTransactionBegin();
    DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
    DetourTransactionCommit();

    BOOL ret = 0;
    ret = TrueDeleteFileW(a0);

    MACTlog("-DeleteFileW will return %x\n", ret);
    MACTlog("*DeleteFileW(%s),(%x,%x,%x)", buffer0, ret, iOption, bReturnValue);
    if(iOption == 1) {
        ret = bReturnValue;
        MACTlog("+DeleteFileW modified to return %x\n", ret);
    }

    delete[] buffer0;

    return ret;
}

LPVOID WINAPI MyVirtualAlloc(LPVOID a0,
                             SIZE_T a1,
                             DWORD a2,
                             DWORD a3)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);
```

```
    MACTlog(":VirtualAlloc(%p,%p,%p,%p)\n", a0, a1, a2, a3);

    int iOption = MACTmain("LPVOID");
    LPVOID bReturnValue = NULL;
    if(iOption == 2) {
        bReturnValue = SR_LPVOID;
    }

    LPVOID ret = 0;
    __try {
        if(MACTDEBUG)
            MACTPrint(">DEBUG: VirtualAlloc about to call TrueVirtualAlloc\n");
        ret = TrueVirtualAlloc(a0, a1, a2, a3);
        if(MACTDEBUG)
            MACTPrint(">DEBUG: VirtualAlloc called TrueVirtualAlloc\n");
    } __finally {
        MACTlog("-VirtualAlloc will return %p\n", ret);
        MACTlog("*VirtualAlloc(%p,%p,%p,%p),(%p,%x,%p)",   a0,   a1,   a2,   a3,   ret,   iOption,
bReturnValue);
        MACTAddToMemoryConstruct(ret, a1, a3, 6);
        if(iOption == 2) {
            ret = bReturnValue;
            MACTlog("+VirtualAlloc modified to return %p\n", (int)&ret);
        }
    }

    return ret;
}

LPVOID WINAPI MyVirtualAllocEx(HANDLE a0,
                               LPVOID a1,
                               SIZE_T a2,
                               DWORD  a3,
                               DWORD  a4)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":VirtualAllocEx(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("LPVOID");
    LPVOID bReturnValue = NULL;
    if(iOption == 2) {
        bReturnValue = SR_LPVOID;
    }
```

```
    LPVOID ret = 0;
    __try {
        ret = TrueVirtualAllocEx(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-VirtualAllocEx will return %p\n", ret);
        MACTlog("*VirtualAllocEx(%p,%p,%p,%p,%p),(%p,%x,%p)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
        MACTAddToMemoryConstruct(ret, a2, a4, 7);
        if(iOption == 2) {
            ret = bReturnValue;
            MACTlog("+VirtualAllocEx modified to return %p\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyVirtualProtect(LPVOID a0,
                             SIZE_T a1,
                             DWORD a2,
                             PDWORD a3)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

// A lot of virtual protects.  Turn on if you wish.
//          if((a2   ==   PAGE_EXECUTE_READ)   ||   (a2   ==   PAGE_EXECUTE_READWRITE)   ||   (a2   ==
PAGE_EXECUTE_WRITECOPY))
//        (void)0;
//    else
//        return(TrueVirtualProtect(a0, a1, a2, a3));
    if(!MACTSTARTED)
        return(TrueVirtualProtect(a0, a1, a2, a3));

    if(MACTDEBUG)
        printf(">DEBUG: MyVirtualProtect A\n");

    MACTPrint(":VirtualProtect(%p,%p,%p,%p)\n", a0, a1, a2, a3);

    if(MACTDEBUG)//
        printf(">DEBUG: MyVirtualProtect B\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
```

```
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        ret = TrueVirtualProtect(a0, a1, a2, a3);
    } __finally {
        MACTPrint("-VirtualProtect will return %x\n", ret);
        MACTPrint("*VirtualProtect(%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  ret,  iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTPrint("+VirtualProtect modified to return %x\n", (int)&ret);
        }
    }

    return ret;
}

BOOL WINAPI MyVirtualProtectEx(HANDLE a0,
                               LPVOID a1,
                               SIZE_T a2,
                               DWORD a3,
                               PDWORD a4)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        MACTlog(":VirtualProtectEx(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        ret = TrueVirtualProtectEx(a0, a1, a2, a3, a4);
        MACTlog("*VirtualProtect(%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
    } __finally {
        MACTlog("-VirtualProtectEx will return %x\n", (int)&ret);
        if(iOption == 2) {
            ret = bReturnValue;
```

```
            MACTlog("+VirtualProtectEx modified to return %x\n", (int)&ret);
        }
    }

    return ret;
}

BOOL WINAPI  MyVirtualFree(LPVOID a0,
                           SIZE_T a1,
                           DWORD  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":VirtualFree(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        MACTDeleteFromMemoryConstruct(a0, 1);
        ret = TrueVirtualFree(a0, a1, a2);
    } __finally {
        MACTlog("-VirtualFree will return %x\n", ret);
        MACTlog("*VirtualFree(%p,%p,%p),(%x,%x,%x)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+VirtualFree modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyVirtualFreeEx(HANDLE a0,
                            LPVOID a1,
                            SIZE_T a2,
                            DWORD a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog(":VirtualFreeEx(%p,%p,%p,%p)\n", a0, a1, a2, a3);
```

```
    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = 0;
    __try {
        MACTDeleteFromMemoryConstruct(a1, 2);
        ret = TrueVirtualFreeEx(a0, a1, a2, a3);
    } __finally {
        MACTlog("-VirtualFreeEx will return %x\n", ret);
        MACTlog("*VirtualFreeEx(%p,%p,%p,%p),(%x,%x,%x)",   a0,   a1,   a2,   a3,   ret,   iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+VirtualFreeEx modified to return %x\n", ret);
        }
    }

    return ret;
}

__drv_allocatesMem(Mem)LPVOID WINAPI MyCoTaskMemAlloc(SIZE_T a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTPrint(":CoTaskMemAlloc(%x)\n", a0);

    __drv_allocatesMem(Mem)LPVOID ret = 0;
    __try {
        ret = TrueCoTaskMemAlloc(a0);
    } __finally {
        MACTPrint("-CoTaskMemAlloc will return %x\n", ret);
        MACTPrint("*CoTaskMemAlloc(%x),(%p,%x,%p)", a0, ret, 0, 0);
        MACTAddToMemoryConstruct((LPVOID)ret, a0, 0, 8);
    }
//chgsleep
    TrueSleep(50);
    return ret;
}

VOID WINAPI MyCoTaskMemFree(LPVOID a0)
{
    if(MACTDEBUG2)
```

```
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);


    MACTPrint(":CoTaskMemFree(%p)\n", a0);

    __try {
        MACTDeleteFromMemoryConstruct(a0, 3);
        TrueCoTaskMemFree(a0);
    } __finally {
        MACTPrint("-CoTaskMemFree will return VOID\n");
        MACTPrint("*CoTaskMemFree(%p),(%x,%x,%x)", a0, 0, 0, 0);
    }
//chgsleep
    TrueSleep(50);
    return;
}

UINT WINAPI MyWinExec(LPCSTR a0,
                      UINT a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":WinExec(%s,%u)\n", buffer0, a1);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_UINT;
    }
/*
    if(strncmp(a0, "C:\\Users\\MACT\\AppData\\Local\\Temp\\LhIgoE.exe", sizeof(a0)) == 0) {
        a0              =              "C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe              -
d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\Desktop\\MACT\\LhIgoE.exe";
        MACTPrint(">WinExec override to C:\\Users\\MACT\\AppData\\Local\\Temp\\LhIgoE.exe");
    }

    if(strncmp(a0, "C:\\Users\\MACT\\AppData\\Local\\Temp\\BUSeJQv.exe", sizeof(a0)) == 0) {
        a0              =              "C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe              -
d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\Desktop\\MACT\\BUSeJQv.exe";
        MACTPrint(">WinExec override to C:\\Users\\MACT\\AppData\\Local\\Temp\\BUSeJQv.exe");
    }
```

```
    if(strncmp(a0, "C:\\Users\\MACT\\AppData\\Local\\Temp\\ivXSsb.exe", sizeof(a0)) == 0) {
        a0              =              "C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe              -
d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\AppData\\Local\\Temp\\ivXSsb.exe";
        MACTPrint(">WinExec override to C:\\Users\\MACT\\AppData\\Local\\Temp\\ivXSsb.exe");
    }
*/
    UINT ret = 0;
    __try {
        ret = TrueWinExec(a0, a1);
    } __finally {
        MACTlog("-WinExec will return %x\n", (int)&ret);
        MACTlog("*WinExec(%s,%u),(%u,%x,%u)", buffer0, a1, ret, iOption, bReturnValue);
        if(iOption == 3) {
            ret = bReturnValue;
            MACTlog("+WinExec modified to return %x\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
}

HINSTANCE WINAPI MyShellExecuteW(HWND a0,
                                 LPCWSTR a1,
                                 LPCWSTR a2,
                                 LPCWSTR a3,
                                 LPCWSTR a4,
                                 INT a5)
{
//    MACTPrint(">>SEONLY ShellExecuteW");
//    HINSTANCE retx = TrueShellExecuteW(a0, a1, a2, a3, a4, a5);
//    return retx;

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize1 = origsize * 2;
    char *buffer1 = new char[newsize1];
    wcstombs_s(&convertedChars, buffer1, newsize1, a1, _TRUNCATE);

    origsize = wcslen(a2) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer2 = new char[newsize2];
```

```
    wcstombs_s(&convertedChars, buffer2, newsize2, a2, _TRUNCATE);

    origsize = wcslen(a3) + 1;
    convertedChars = 0;
    const size_t newsize3 = origsize * 2;
    char *buffer3 = new char[newsize3];
    wcstombs_s(&convertedChars, buffer3, newsize3, a3, _TRUNCATE);

    origsize = wcslen(a4) + 1;
    convertedChars = 0;
    const size_t newsize4 = origsize * 2;
    char *buffer4 = new char[newsize4];
    wcstombs_s(&convertedChars, buffer4, newsize4, a4, _TRUNCATE);

    MACTlog(":ShellExecuteW(%p,%s,%s,%s,%s,%d)\n", a0, buffer1, buffer2, buffer3, buffer4, a5);

    int iOption = MACTmain("HINSTANCE");
    HINSTANCE bReturnValue = NULL;
    if(iOption == 4) {
        bReturnValue = SR_HINSTANCE;
    }

    HINSTANCE ret = 0;
    __try {
        ret = TrueShellExecuteW(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-ShellExecuteW will return %x\n", (int)ret);
        MACTlog("*ShellExecuteW(%p,%s,%s,%s,%s,%d),(%x,%x,%x)",  a0,  buffer1,  buffer2,  buffer3,
buffer4, a5, ret, iOption, bReturnValue);
        if(iOption == 4) {
            ret = bReturnValue;
            MACTlog("+ShellExecuteW modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
        delete[] buffer3;
        delete[] buffer4;
    }

    return ret;
}

BOOL WINAPI MyShellExecuteExA(SHELLEXECUTEINFOA *a0)
{

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);
```

```
if(!MACTSTARTED)
    return TrueShellExecuteExA(a0);

typedef struct _MACTSHELLEXECUTEINFOA {
  DWORD     cbSize;
  ULONG     fMask;
  HWND      hwnd;
  LPCSTR    lpVerb;
  LPCSTR    lpFile;
  LPCSTR    lpParameters;
  LPCSTR    lpDirectory;
  int       nShow;
  HINSTANCE hInstApp;
  void      *lpIDList;
  LPCSTR    lpClass;
  HKEY      hkeyClass;
  DWORD     dwHotKey;
  union {
    HANDLE hIcon;
    HANDLE hMonitor;
  } DUMMYUNIONNAME;
  HANDLE    hProcess;
} MACTSHELLEXECUTEINFOA, *LPMACTSHELLEXECUTEINFOA;

int la = 0;

BOOL wcsfail = FALSE;
__try {
    la = lstrlenA(a0->lpVerb);
} __except(EXCEPTION_EXECUTE_HANDLER) {
    wcsfail = TRUE;
    la = 5;
}
char *buffer0 = new char[la+1];
if(wcsfail) {
    strncpy(buffer0, "NULL\0", 5);
}
else {
    strncpy(buffer0, a0->lpVerb, la);
    buffer0[la] = '\0';
}

la = lstrlenA(a0->lpFile);
char *buffer1 = new char[la+1];
strncpy(buffer1, a0->lpFile, la);
buffer1[la] = '\0';
```

```
 wcsfail = FALSE;
__try {
    la = lstrlenA(a0->lpParameters);
} __except(EXCEPTION_EXECUTE_HANDLER) {
    wcsfail = TRUE;
    la = 5;
}
char *buffer2 = new char[la+1];
if(wcsfail) {
    strncpy(buffer2, "NULL\0", 5);
}
else {
    strncpy(buffer2, a0->lpParameters, la);
    buffer2[la] = '\0';
}

wcsfail = FALSE;
__try {
    la = lstrlenA(a0->lpDirectory);
} __except(EXCEPTION_EXECUTE_HANDLER) {
    wcsfail = TRUE;
    la = 5;
}
char *buffer3 = new char[la+1];
if(wcsfail) {
    strncpy(buffer3, "NULL\0", 5);
}
else {
    strncpy(buffer3, a0->lpDirectory, la);
    buffer3[la] = '\0';
}

MACTPrint(">ShellExecuteExA lpVerb       = [%s]\n", buffer0);
MACTPrint(">ShellExecuteExA lpFile       = [%s]\n", buffer1);
MACTPrint(">ShellExecuteExA lpParameters = [%s]\n", buffer2);
MACTPrint(">ShellExecuteExA lpDirectory  = [%s]\n", buffer3);

LPMACTSHELLEXECUTEINFOA MACTShellExecuteExA = (LPMACTSHELLEXECUTEINFOA)a0;
//    MACTShellExecuteExW->lpFile = L"cmd.exe";
//        MACTShellExecuteExW->lpParameters  =  L"C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe  -
d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\AppData\\Local\\Temp/MBSSCR.exe";
//     MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start withdll.exe -
d:mact32.dll sample.exe\" ";
//     MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start C:\\wdll.exe -
d:C:\\mac.dll c:\\sam.exe\" ";
```

```
//ok          MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\sample.exe\"";
//ok          MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\serverc.exe\"";
    //      C:\Users\MACT\Desktop\MACT\withdll.exe      -d:C:\Users\MACT\Desktop\MACT\mact32.dll
C:\Users\MACT\Desktop\MACT\sample.exe
//             MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe         -d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll
C:\\Users\\MACT\\Desktop\\MACT\\serverc.exe\"";
//       MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start withdll.exe -
d:mact32.dll sample.exe\"";
// process call create "cmd /c start withdll.exe -d:mact32.dll sample.exe"  "

//             MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe       -d:C:\\Users\\MACT\\Desktop\\MACT\\mactsc32.dll
C:\\Users\\MACT\\Desktop\\MACT\\sample.exe\"";
//    MACTShellExecuteExA->lpFile = L"C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe";
//         MACTShellExecuteExA->lpParameters  =  L"-d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll
C:\\Users\\MACT\\AppData\\Local\\Temp/MBSSCR.exe";

    a0 = (LPSHELLEXECUTEINFOA)MACTShellExecuteExA;

    MACTlog(":ShellExecuteExA(%p)\n", a0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueShellExecuteExA(a0);
    } __finally {
        MACTlog("-ShellExecuteExA will return %x\n", ret);
        MACTlog("*ShellExecuteExA(%p),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+ShellExecuteExA modified to return %x\n", ret);
        }
    }

    delete[] buffer0;
    delete[] buffer1;
    delete[] buffer2;
    delete[] buffer3;
```

```
    return ret;
}

BOOL WINAPI MyShellExecuteExW(SHELLEXECUTEINFOW *a0)
{

    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueShellExecuteExW(a0);

    typedef struct _MACTSHELLEXECUTEINFOW {
      DWORD     cbSize;
      ULONG     fMask;
      HWND      hwnd;
      LPCWSTR   lpVerb;
      LPCWSTR   lpFile;
      LPCWSTR   lpParameters;
      LPCWSTR   lpDirectory;
      int       nShow;
      HINSTANCE hInstApp;
      void      *lpIDList;
      LPCWSTR   lpClass;
      HKEY      hkeyClass;
      DWORD     dwHotKey;
      union {
        HANDLE hIcon;
        HANDLE hMonitor;
      } DUMMYUNIONNAME;
      HANDLE    hProcess;
    } MACTSHELLEXECUTEINFOW, *LPMACTSHELLEXECUTEINFOW;


    size_t origsize = 0;

    BOOL wcsfail = FALSE;
    __try {
        origsize = wcslen(a0->lpVerb) + 1;
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        wcsfail = TRUE;
        origsize = 5;
    }
    size_t convertedChars = 0;
    const size_t newsize0 = origsize * 2;
    char *buffer0 = new char[newsize0];
    if(wcsfail) {
```

```
    strncpy(buffer0, "NULL\0", 5);
}
else
    wcstombs_s(&convertedChars, buffer0, newsize0, a0->lpVerb, _TRUNCATE);

origsize = wcslen(a0->lpFile) + 1;
convertedChars = 0;
const size_t newsize1 = origsize * 2;
char *buffer1 = new char[newsize1];
wcstombs_s(&convertedChars, buffer1, newsize1, a0->lpFile, _TRUNCATE);

wcsfail = FALSE;
__try {
    origsize = wcslen(a0->lpParameters) + 1;
} __except(EXCEPTION_EXECUTE_HANDLER) {
    wcsfail = TRUE;
    origsize = 5;
}
convertedChars = 0;
const size_t newsize2 = origsize * 2;
char *buffer2 = new char[newsize2];
if(wcsfail) {
    strncpy(buffer2, "NULL\0", 5);
}
else
    wcstombs_s(&convertedChars, buffer2, newsize2, a0->lpParameters, _TRUNCATE);

wcsfail = FALSE;
__try {
    origsize = wcslen(a0->lpDirectory) + 1;
} __except(EXCEPTION_EXECUTE_HANDLER) {
    wcsfail = TRUE;
    origsize = 5;
}
convertedChars = 0;
const size_t newsize3 = origsize * 2;
char *buffer3 = new char[newsize3];
if(wcsfail) {
    strncpy(buffer3, "NULL\0", 5);
}
else
    wcstombs_s(&convertedChars, buffer3, newsize3, a0->lpDirectory, _TRUNCATE);

MACTPrint(">ShellExecuteExW lpVerb       = [%s]\n", buffer0);
MACTPrint(">ShellExecuteExW lpFile       = [%s]\n", buffer1);
MACTPrint(">ShellExecuteExW lpParameters = [%s]\n", buffer2);
MACTPrint(">ShellExecuteExW lpDirectory  = [%s]\n", buffer3);
```

```
    LPMACTSHELLEXECUTEINFOW MACTShellExecuteExW = (LPMACTSHELLEXECUTEINFOW)a0;
//    MACTShellExecuteExW->lpFile = L"cmd.exe";
//         MACTShellExecuteExW->lpParameters  =  L"C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe  -
d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll C:\\Users\\MACT\\AppData\\Local\\Temp/MBSSCR.exe";
//      MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start withdll.exe -
d:mact32.dll sample.exe\" ";
//     MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start C:\\wdll.exe -
d:C:\\mac.dll c:\\sam.exe\" ";
//ok          MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\sample.exe\"";
//ok          MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\serverc.exe\"";
    //       C:\Users\MACT\Desktop\MACT\withdll.exe       -d:C:\Users\MACT\Desktop\MACT\mact32.dll
C:\Users\MACT\Desktop\MACT\sample.exe
//             MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe         -d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll
C:\\Users\\MACT\\Desktop\\MACT\\serverc.exe\"";
//       MACTShellExecuteExW->lpParameters = L"process call create \"cmd /c start withdll.exe -
d:mact32.dll sample.exe\"";
// process call create "cmd /c start withdll.exe -d:mact32.dll sample.exe"  "

//             MACTShellExecuteExW->lpParameters  =  L"process  call  create  \"cmd  /c  start
C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe       -d:C:\\Users\\MACT\\Desktop\\MACT\\mactsc32.dll
C:\\Users\\MACT\\Desktop\\MACT\\sample.exe\"";
//    MACTShellExecuteExW->lpFile = L"C:\\Users\\MACT\\Desktop\\MACT\\withdll.exe";
//          MACTShellExecuteExW->lpParameters  =  L"-d:C:\\Users\\MACT\\Desktop\\MACT\\mact32.dll
C:\\Users\\MACT\\AppData\\Local\\Temp/MBSSCR.exe";

    a0 = (LPSHELLEXECUTEINFOW)MACTShellExecuteExW;

    MACTlog(":ShellExecuteExW(%p)\n", a0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        MACTPrint(">ShellExecuteW before.\n");
        ret = TrueShellExecuteExW(a0);
        MACTPrint(">ShellExecuteW after.\n");
    } __finally {
        MACTlog("-ShellExecuteExW will return %x\n", ret);
        MACTlog("*ShellExecuteExW(%p),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
```

```
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+ShellExecuteExW modified to return %x\n", ret);
        }
    }

    delete[] buffer0;
    delete[] buffer1;
    delete[] buffer2;
    delete[] buffer3;

    return ret;
}

LONG WINAPI MyRegGetValueA(HKEY    a0,
                           LPCSTR  a1,
                           LPTSTR  a2,
                           DWORD   a3,
                           LPDWORD a4,
                           PVOID   a5,
                           PDWORD  a6)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la;
    if(a1 == NULL)
        la = 4;
    else
        la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    if(a1 == NULL)
        strncpy(buffer1, "NULL", la);
    else
        strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    if(a2 == NULL)
        la = 4;
    else
        la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    if(a2 == NULL)
        strncpy(buffer2, "NULL", la);
    else
        strncpy(buffer2, a2, la);
```

```
    MACTlog(":RegGetValueA(%p,%s,%s,%p,%p,%p,%p)\n", a0, buffer1, buffer2, a3, a4, a5, a6);

    MACTreg(">RegGetValueA SubKey = [%s]\n", buffer1);

    MACTreg(">RegGetValueA Value  = [%s]\n", buffer2);

    int iOption = MACTmain("HINSTANCE");
    LONG bReturnValue = 0;
    if(iOption == 6) {
        bReturnValue = SR_LONG;
    }

    LONG ret = 0;
    __try {
        ret = TrueRegGetValueA(a0, a1, a2, a3, a4, a5, a6);
    } __finally {
        MACTlog("-RegGetValueA will return %x\n", (int)ret);
        MACTlog("*RegGetValueA(%p,%s,%s,%p,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, buffer2, a3, a4,
a5, a6,
            ret, iOption, bReturnValue);
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+RegGetValueA modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

LONG WINAPI MyRegGetValueW(HKEY    a0,
                           LPCWSTR a1,
                           LPCWSTR a2,
                           DWORD   a3,
                           LPDWORD a4,
                           PVOID   a5,
                           PDWORD  a6)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize;
    if(a1 == NULL)
        origsize = 4;
    else
        origsize = wcslen(a1) + 1;
```

```
size_t convertedChars = 0;
const size_t newsize1 = origsize * 2;
char *buffer1 = new char[newsize1];

if(a1 == NULL) {
    strncpy(buffer1, "NULL", origsize);
    buffer1[origsize] = '\0';
}
else
    wcstombs_s(&convertedChars, buffer1, newsize1, a1, _TRUNCATE);


if(a2 == NULL)
    origsize = 4;
else
    origsize = wcslen(a2) + 1;

convertedChars = 0;
const size_t newsize2 = origsize * 2;
char *buffer2 = new char[newsize2];

if(a2 == NULL) {
    strncpy(buffer2, "NULL", origsize);
    buffer2[origsize] = '\0';
}
else
    wcstombs_s(&convertedChars, buffer2, newsize2, a2, _TRUNCATE);

MACTlog(":RegGetValueW(%p,%s,%s,%p,%p,%p,%p)\n", a0, buffer1, buffer2, a3, a4, a5, a6);

MACTreg(">RegGetValueW SubKey = [%s]\n", buffer1);

MACTreg(">RegGetValueW Value  = [%s]\n", buffer2);

int iOption = MACTmain("HINSTANCE");
LONG bReturnValue = 0;
if(iOption == 6) {
    bReturnValue = SR_LONG;
}

LONG ret = 0;
__try {
    ret = TrueRegGetValueW(a0, a1, a2, a3, a4, a5, a6);
} __finally {
    MACTlog("-RegGetValueW will return %x\n", (int)ret);
```

```
        MACTlog("*RegGetValueW(%p,%s,%s,%p,%p,%p,%p),(%x,%x,%x)", a0, buffer1, buffer2, a3, a4,
a5, a6,
            ret, iOption, bReturnValue);
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+RegGetValueW modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

LONG WINAPI MyRegQueryValueEx(HKEY    a0,
                              LPCTSTR a1,
                              LPDWORD a2,
                              LPDWORD a3,
                              LPBYTE  a4,
                              LPDWORD a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":RegQueryValueEx(%p,%s,%p,%p,%p,%p)\n", a0, buffer1, a2, a3, a4, a5);

    MACTreg(">RegQueryValueEx Key = [%s]\n", buffer1);

    int iOption = MACTmain("HINSTANCE");
    LONG bReturnValue = 0;
    if(iOption == 6) {
        bReturnValue = SR_LONG;
    }

    LONG ret = 0;

    ret = TrueRegQueryValueEx(a0, a1, a2, a3, a4, a5);

    MACTlog("-RegQueryValueEx will return %x\n", ret);
    MACTlog("*RegQueryValueEx(%p,%s,%p,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, a2, a3, a4, a5, ret,
iOption, bReturnValue);
    if(iOption == 6) {
```

```
        ret = bReturnValue;
        MACTlog("+RegQueryValueEx modified to return %x\n", ret);
    }

// chgsleep
    TrueSleep(100);
    delete[] buffer1;
    return ret;
}

LONG WINAPI MyRegOpenKeyEx(HKEY    a0,
                           LPCTSTR a1,
                           DWORD   a2,
                           REGSAM  a3,
                           PHKEY   a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":RegOpenKeyEx(%p,%s,%p,%p,%p)\n", a0, buffer1, a2, a3, a4);

    MACTreg(">RegOpenKeyEx Key = [%s]\n", buffer1);


    int iOption = MACTmain("HINSTANCE");
    LONG bReturnValue = FALSE;
    if(iOption == 6) {
        bReturnValue = SR_LONG;
    }

    LONG ret = 0;
    __try {
        ret = TrueRegOpenKeyEx(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-RegOpenKeyEx will return %x\n", (int)ret);
        MACTlog("*GetOpenKeyEx(%p,%s,%p,%p,%p),(%ld,%x,%ld)",  a0,  buffer1,  a2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+RegOpenKeyEx modified to return %x\n", ret);
        }
    }
```

```
// chgsleep
    TrueSleep(100);
    delete[] buffer1;
    return ret;
}

LSTATUS WINAPI MyRegSetValueA(HKEY   a0,
                              LPCSTR a1,
                              DWORD  a2,
                              LPCSTR a3,
                              DWORD  a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la;
    if(a1 == NULL)
        la = 4;
    else
        la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    if(a1 == NULL)
        strncpy(buffer1, "NULL", la);
    else
        strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a3);
    char *buffer3 = new char[la+1];
    strncpy(buffer3, a3, la);
    buffer3[la] = '\0';

    MACTlog(":RegSetValueA(%p,%s,%p,%s,%p)\n", a0, buffer1, a2, buffer3, a4);

    MACTreg(">RegSetValueA SubKey = [%s]\n", buffer1);
    MACTreg(">RegSetValueA Data   = [%s]\n", buffer3);

    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = 0;
    __try {
        ret = TrueRegSetValueA(a0, a1, a2, a3, a4);
    } __finally {
```

```
        MACTlog("-RegSetValueA will return %x\n", ret);
        MACTlog("*RegSetValueA(%p,%s,%p,%s,%p),(%x,%x,%x)",  a0,  buffer1,  a2,  buffer3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegSetValueA modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer3;
    }

    return ret;
}

LONG WINAPI MyRegSetValueEx(HKEY        a0,
                            LPCTSTR     a1,
                            DWORD       a2,
                            DWORD       a3,
                            const BYTE* a4,
                            DWORD       a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":RegSetValueEx(%p,%s,%p,%p,%p,%p)\n", a0, buffer1, a2, a3, a4, a5);

    MACTreg(">RegSetValueEx Key = [%s]\n", buffer1);


    int iOption = MACTmain("HINSTANCE");
    LONG bReturnValue = FALSE;
    if(iOption == 6) {
        bReturnValue = SR_LONG;
    }

    LONG ret = 0;
    __try {
        ret = TrueRegSetValueEx(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-RegSetValueEx will return %x\n", (int)ret);
        MACTlog("*RegSetValueEx(%p,%s,%p,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, a2, a3, a4, a5, ret,
iOption, bReturnValue);
```

```
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+RegSetValueEx modified to return %x\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

LSTATUS WINAPI MyRegSetValueExW(HKEY        a0,
                                LPCWSTR     a1,
                                DWORD       a2,
                                DWORD       a3,
                                const BYTE* a4,
                                DWORD       a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer1 = new char[newsize];
    wcstombs_s(&convertedChars, buffer1, newsize, a1, _TRUNCATE);

    MACTlog(":RegSetValueExW(%p,%s,%p,%p,%p,%p)\n", a0, buffer1, a2, a3, a4, a5);

    MACTreg(">RegSetValueExW Key = [%s]\n", buffer1);


    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = 0;
    __try {
        ret = TrueRegSetValueExW(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-RegSetValueExW will return %x\n", (int)ret);
        MACTlog("*RegSetValueExW(%p,%s,%p,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, a2, a3, a4, a5,
ret, iOption, bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegSetValueExW modified to return %x\n", ret);
```

```
        }
        delete[] buffer1;
    }

    return ret;
}

LSTATUS WINAPI MyRegEnumKeyExA(HKEY      a0,
                               DWORD     a1,
                               LPSTR     a2,
                               LPDWORD   a3,
                               LPDWORD   a4,
                               LPSTR     a5,
                               LPDWORD   a6,
                               PFILETIME a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegEnumKeyExA(a0, a1, a2, a3, a4, a5, a6, a7);

    int la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTreg(">RegEnumKeyExA Subkey = [%s]\n", buffer2);

    MACTlog(":RegEnumKeyExA(%p,%p,%s,%p,%p,%p,%p,%p)\n", a0, a1, buffer2, a3, a4, a5, a6, a7);

    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = NULL;
    __try {
        ret = TrueRegEnumKeyExA(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-RegEnumKeyExA will return %p\n", ret);
        MACTlog("*RegEnumKeyExA(%p,%p,%s,%p,%p,%p,%p,%p)(%p,%x,%p)", a0, a1, buffer2, a3, a4, a5,
a6, a7, ret, iOption, bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegEnumKeyExA modified to return %p\n", ret);
```

```
        }
        delete[] buffer2;
    }

//    TrueSleep(50);
    return ret;
}

LSTATUS WINAPI MyRegEnumKeyExW(HKEY      a0,
                               DWORD     a1,
                               LPWSTR    a2,
                               LPDWORD   a3,
                               LPDWORD   a4,
                               LPWSTR    a5,
                               LPDWORD   a6,
                               PFILETIME a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegEnumKeyExW(a0, a1, a2, a3, a4, a5, a6, a7);

    size_t origsize = wcslen(a2) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer2 = new char[newsize];
    wcstombs_s(&convertedChars, buffer2, newsize, a2, _TRUNCATE);

    MACTreg(">RegEnumKeyExW Subkey = [%s]\n", buffer2);

    MACTlog(":RegEnumKeyExW(%p,%p,%s,%p,%p,%p,%p,%p)\n", a0, a1, buffer2, a3, a4, a5, a6, a7);

    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = NULL;
    __try {
        ret = TrueRegEnumKeyExW(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-RegEnumKeyExW will return %p\n", ret);
        MACTlog("*RegEnumKeyExW(%p,%p,%s,%p,%p,%p,%p,%p)(%p,%x,%p)", a0, a1, buffer2, a3, a4, a5,
a6, a7, ret, iOption, bReturnValue);
        if(iOption == 22) {
```

```
            ret = bReturnValue;
            MACTlog("+RegEnumKeyExW modified to return %p\n", ret);
        }
        delete[] buffer2;
    }

//    TrueSleep(50);
    return ret;
}

LONG WINAPI  MyRegCreateKeyEx(HKEY                  a0,
                              LPCTSTR               a1,
                              DWORD                 a2,
                              LPTSTR                a3,
                              DWORD                 a4,
                              REGSAM                a5,
                              LPSECURITY_ATTRIBUTES a6,
                              PHKEY                 a7,
                              LPDWORD               a8)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":RegCreateKeyEx(%p,%s,%p,%p,%p,%p,%p,%p,%p)\n", a0, buffer1, a2, a3, a4, a5, a6, a7,
a8);

    MACTreg(">RegCreateKeyEx Key = [%s]\n", buffer1);


    int iOption = MACTmain("HINSTANCE");
    LONG bReturnValue = FALSE;
    if(iOption == 6) {
        bReturnValue = SR_LONG;
    }

    LONG ret = 0;
    __try {
        ret = TrueRegCreateKeyEx(a0, a1, a2, a3, a4, a5, a6, a7, a8);
    } __finally {
        MACTlog("-RegCreateKeyEx will return %x\n", (int)ret);
        MACTlog("*RegCreateKeyEx(%p,%s,%p,%p,%p,%p,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, a2, a3,
a4,
```

```
            a5, a6, a7, a8, ret, iOption, bReturnValue);
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+RegCreateKeyEx modified to return %x\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyAdjustTokenPrivileges(HANDLE            a0,
                                    BOOL              a1,
                                    PTOKEN_PRIVILEGES a2,
                                    DWORD             a3,
                                    PTOKEN_PRIVILEGES a4,
                                    PDWORD            a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueAdjustTokenPrivileges(a0, a1, a2, a3, a4, a5);

    MACTlog(":AdjustTokenPrivileges(%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueAdjustTokenPrivileges(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-AdjustTokenPrivileges will return %x\n", (int)ret);
        MACTlog("*AdjustTokenPrivileges(%p,%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, a5,
ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+AdjustTokenPrivileges modified to return %x\n", ret);
        }
    }

    return ret;
}
```

```
BOOL WINAPI MyAttachThreadInput(DWORD a0,
                                DWORD a1,
                                BOOL  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueAttachThreadInput(a0, a1, a2);

    MACTPrint(":AttachThreadInput(%p,%p,%x)\n", a0, a1, (int)a2);
    return TrueAttachThreadInput(a0, a1, a2);

    MACTlog(":AttachThreadInput(%p,%p,%x)\n", a0, a1, (int)a2);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueAttachThreadInput(a0, a1, a2);
    } __finally {
        MACTlog("-AttachThreadInput will return %x\n", (int)ret);
        MACTlog("*AttachThreadInput(%p,%p,%x),(%x,%x,%x)",    a0,    a1,    a2,    ret,    iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+AttachThreadInput modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyBitBlt(HDC   a0,
                     int    a1,
                     int    a2,
                     int    a3,
                     int    a4,
                     HDC    a5,
                     int    a6,
                     int    a7,
                     DWORD  a8)
```

```
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

// This is called a lot and causes a lot of noise so it is shut off.
// The TrueSleep is necessary as the program dies before everything is processed.
//    return TrueBitBlt(a0, a1, a2, a3, a4, a5, a6, a7, a8);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueBitBlt(a0, a1, a2, a3, a4, a5, a6, a7, a8);

    if(MACTVERBOSE)
        MACTPrint(":BitBlt(%p,%x,%x,%x,%x,%p,%x,%x,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7, a8);

    MACTPrint("*BitBlt(%p,%x,%x,%x,%x,%p,%x,%x,%p),(%ld,%d,%ld)", a0, a1, a2, a4, a5, a6, a7, a8,
                0, 0, 0);
    TrueSleep(50);
    return TrueBitBlt(a0, a1, a2, a3, a4, a5, a6, a7, a8);

    MACTlog(":BitBlt(%p,%x,%x,%x,%x,%p,%x,%x,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7, a8);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueBitBlt(a0, a1, a2, a3, a4, a5, a6, a7, a8);
    } __finally {
        MACTlog("-BitBlt will return %x\n", (int)ret);
        MACTlog("*BitBlt(%p,%x,%x,%x,%x,%p,%x,%x,%p),(%x,%x,%x)", a0, a1, a2, a4, a5, a6, a7, a8,
                ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+BitBlt modified to return %x\n", ret);
        }
    }

    return ret;
}

HCERTSTORE WINAPI MyCertOpenSystemStore(HCRYPTPROV_LEGACY a0,
                                        LPCTSTR           a1)
{
    if(MACTDEBUG2)
```

```
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCertOpenSystemStore(a0, a1);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":CertOpenSystemStore(%p,%s)\n", a0, buffer1);

    int iOption = MACTmain("HCERTSTORE");
    HCERTSTORE bReturnValue = NULL;
    if(iOption == 6) {
        bReturnValue = SR_HCERTSTORE;
    }

    HCERTSTORE ret;
    __try {
        ret = TrueCertOpenSystemStore(a0, a1);
    } __finally {
        MACTlog("-CertOpenSystemStore will return %p\n", ret);
        MACTlog("*CertOpenSystemStore(%p,%s),(%p,%x,%p)",    a0,    buffer1,    ret,    iOption,
bReturnValue);
        if(iOption == 6) {
            ret = bReturnValue;
            MACTlog("+CertOpenSystemStore modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyControlService(SC_HANDLE        a0,
                             DWORD            a1,
                             LPSERVICE_STATUS a2)

{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueControlService(a0, a1, a2);

    MACTlog(":ControlService(%p,%p,%p)\n", a0, a1, a2);
```

```
    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueControlService(a0, a1, a2);
    } __finally {
        MACTlog("-ControlService will return %xx", (int)ret);
        MACTlog("*ControlService(%p,%p,%p),(%x,%x,%x)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+ControlService modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyCreateMutex(LPSECURITY_ATTRIBUTES a0,
                            BOOL                  a1,
                            LPCTSTR               a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateMutex(a0, a1, a2);

    int la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":CreateMutex(%p,%p,%s)\n", a0, a1, buffer2);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = FALSE;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = FALSE;
    __try {
```

```
        ret = TrueCreateMutex(a0, a1, a2);
    } __finally {
        MACTlog("-CreateMutex will return %p\n", ret);
        MACTlog("*CreateMutex(%p,%p,%s),(%p,%x,%p)", a0, a1, buffer2, ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+CreateMutex modified to return %x\n", ret);
        }
        delete[] buffer2;
    }

    return ret;
}

HANDLE WINAPI MyCreateMutexEx(LPSECURITY_ATTRIBUTES a0,
                              LPCTSTR               a1,
                              DWORD                 a2,
                              DWORD                 a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateMutexEx(a0, a1, a2, a3);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":CreateMutexEx(%p,%s,%p,%p)\n", a0, buffer1, a2, a3);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = FALSE;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = FALSE;
    __try {
        ret = TrueCreateMutexEx(a0, a1, a2, a3);
    } __finally {
        MACTlog("-CreateMutexEx will return %p\n", ret);
        MACTlog("*CreateMutexEx(%p,%s,%p,%p),(%p,%x,%p)",  a0,  buffer1,  a2,  a3,  ret,  iOption,
bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
```

```
            MACTlog("+CreateMutexEx modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyCreateProcess(LPCTSTR               a0,
                            LPTSTR                a1,
                            LPSECURITY_ATTRIBUTES a2,
                            LPSECURITY_ATTRIBUTES a3,
                            BOOL                  a4,
                            DWORD                 a5,
                            LPVOID                a6,
                            LPCTSTR               a7,
                            LPSTARTUPINFO         a8,
                            LPPROCESS_INFORMATION a9)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateProcess(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a7);
    char *buffer7 = new char[la+1];
    strncpy(buffer7, a7, la);
    buffer7[la] = '\0';

    MACTlog(":CreateProcess(%s,%s,%p,%p,%x,%p,%p,%s,%p,%p)\n", buffer0, (char *) a1, a2, a3, a4,
a5, a6, buffer7, a8, a9);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueCreateProcess(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9);
```

```
    } __finally {
        MACTlog("-CreateProcess will return %x\n", (int)ret);
        MACTlog("*CreateProcess(%s,%s,%p,%p,%x,%p,%p,%s,%p,%p),(%p,%x,%p)", buffer0, (char *) a1,
a2, a4, a5, a6, buffer7, a8, a9,
                ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CreateProcess modified to return %x\n", ret);
        }
        delete[] buffer0;
        delete[] buffer7;
    }

    return ret;
}


BOOL WINAPI MyCreateProcessW(LPCWSTR               a0,
                             LPWSTR                a1,
                             LPSECURITY_ATTRIBUTES a2,
                             LPSECURITY_ATTRIBUTES a3,
                             BOOL                  a4,
                             DWORD                 a5,
                             LPVOID                a6,
                             LPCWSTR               a7,
                             LPSTARTUPINFOW        a8,
                             LPPROCESS_INFORMATION a9)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateProcessW(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    origsize = wcslen(a7) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer7 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer7, newsize2, a7, _TRUNCATE);
```

```
    MACTlog(":CreateProcessW(%s,%p,%p,%p,%x,%p,%p,%s,%p,%p)\n", buffer0, a1, a2, a3, a4, a5, a6,
buffer7, a8, a9);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueCreateProcessW(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9);
    } __finally {
        MACTlog("-CreateProcessW will return %x\n", (int)ret);
        MACTlog("*CreateProcessW(%s,%p,%p,%p,%x,%p,%p,%s,%p,%p),(%p,%x,%p)", buffer0, a1, a2, a4,
a5, a6, buffer7, a8, a9,
                ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CreateProcessW modified to return %x\n", ret);
        }
        delete[] buffer0;
        delete[] buffer7;
    }

    return ret;
}

BOOL WINAPI MyTerminateProcess(HANDLE a0,
                               UINT   a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueTerminateProcess(a0, a1);

    MACTlog(":TerminateProcess(%p,%p,)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
```

```
        ret = TrueTerminateProcess(a0, a1);
    } __finally {
        MACTlog("-TerminateProcess will return %x\n", (int)ret);
        MACTlog("*TerminateProcess(%p,%p),(%p,%x,%p)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+TerminateProcess modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyCreateRemoteThread(HANDLE                 a0,
                                   LPSECURITY_ATTRIBUTES  a1,
                                   SIZE_T                 a2,
                                   LPTHREAD_START_ROUTINE a3,
                                   LPVOID                 a4,
                                   DWORD                  a5,
                                   LPDWORD                a6)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateRemoteThread(a0, a1, a2, a3, a4, a5, a6);

    MACTPrint(":CreateRemoteThread(%p,%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = 0;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
    __try {
        ret = TrueCreateRemoteThread(a0, a1, a2, a3, a4, a5, a6);
    } __finally {
        MACTPrint("-CreateRemoteThread will return %p\n", ret);
        MACTPrint("*CreateRemoteThread(%p,%p,%p,%p,%p,%p,%p),(%p,%x,%p)", a0, a1, a2, a3, a4, a5,
a6, ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTPrint("+CreateRemoteThread modified to return %p\n", ret);
        }
    }
```

```
    return ret;
}

HANDLE WINAPI MyCreateRemoteThreadEx(HANDLE                       a0,
                                     LPSECURITY_ATTRIBUTES        a1,
                                     SIZE_T                       a2,
                                     LPTHREAD_START_ROUTINE       a3,
                                     LPVOID                       a4,
                                     DWORD                        a5,
                                     LPPROC_THREAD_ATTRIBUTE_LIST a6,
                                     LPDWORD                      a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateRemoteThreadEx(a0, a1, a2, a3, a4, a5, a6, a7);

    MACTlog(":CreateRemoteThreadEx(%p,%p,%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
    __try {
        ret = TrueCreateRemoteThreadEx(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-CreateRemoteThreadEx will return %p\n", ret);
        MACTlog("*CreateRemoteThreadEx(%p,%p,%p,%p,%p,%p,%p,%p),(%p,%x,%p)", a0, a1, a2, a3, a4,
a5, a6, a7, ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+CreateRemoteThreadEx modified to return %p\n", ret);
        }
    }

    return ret;
}

SC_HANDLE WINAPI MyCreateService(SC_HANDLE a0,
                                 LPCTSTR   a1,
                                 LPCTSTR   a2,
                                 DWORD     a3,
```

```
                                DWORD     a4,
                                DWORD     a5,
                                DWORD     a6,
                                LPCTSTR   a7,
                                LPCTSTR   a8,
                                LPDWORD   a9,
                                LPCTSTR   a10,
                                LPCTSTR   a11,
                                LPCTSTR   a12)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateService(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    la = lstrlenA(a7);
    char *buffer7 = new char[la+1];
    strncpy(buffer7, a7, la);
    buffer7[la] = '\0';

    la = lstrlenA(a8);
    char *buffer8 = new char[la+1];
    strncpy(buffer8, a8, la);
    buffer8[la] = '\0';

    la = lstrlenA(a10);
    char *buffer10 = new char[la+1];
    strncpy(buffer10, a10, la);
    buffer10[la] = '\0';

    la = lstrlenA(a11);
    char *buffer11 = new char[la+1];
    strncpy(buffer11, a11, la);
    buffer11[la] = '\0';

    la = lstrlenA(a12);
```

```
    char *buffer12 = new char[la+1];
    strncpy(buffer12, a12, la);
    buffer12[la] = '\0';

    MACTlog(":CreateService(%p,%s,%s,%x,%x,%x,%x,%s,%s,%p,%s,%s,%s)\n",
        a0, buffer1, buffer2, a3, a4, a5, a6, buffer7, buffer8, a9, buffer10, buffer11, buffer12);

    int iOption = MACTmain("SC_HANDLE");
    SC_HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_SC_HANDLE;
    }

    SC_HANDLE ret = FALSE;
    __try {
        ret = TrueCreateService(a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
    } __finally {
        MACTlog("-CreateService will return %p\n", ret);
        MACTlog("*CreateService(%p,%s,%s,%x,%x,%x,%x,%s,%s,%p,%s,%s,%s),(%p,%x,%p)",
            a0, buffer1, buffer2, a3, a4, a5, a6, buffer7, buffer8, a9, buffer10, buffer11,
buffer12,
            ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+CreateService modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
        delete[] buffer7;
        delete[] buffer8;
        delete[] buffer10;
        delete[] buffer11;
        delete[] buffer12;
    }

    return ret;
}

HANDLE WINAPI MyCreateToolhelp32Snapshot(DWORD a0,
                                         DWORD a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCreateToolhelp32Snapshot(a0, a1);
```

```
    MACTlog(":CreateToolhelp32Snapshot(%x,%x)\n", a0, a1);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = FALSE;
    __try {
        ret = TrueCreateToolhelp32Snapshot(a0, a1);
    } __finally {
        MACTlog("-CreateToolhelp32Snapshot will return %p\n", ret);
        MACTlog("*CreateToolhelp32Snapshot(%x,%x),(%p,%x,%p)", a0, a1,ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+CreateToolhelp32Snapshot modified to return %p\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyCryptAcquireContextA(HCRYPTPROV *a0,
                                   LPCTSTR    a1,
                                   LPCTSTR    a2,
                                   DWORD      a3,
                                   DWORD      a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCryptAcquireContextA(a0, a1, a2, a3, a4);

    MACTlog(":CryptAcquireContextA(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueCryptAcquireContextA(a0, a1, a2, a3, a4);
    } __finally {
```

```
        MACTlog("-CryptAcquireContextA will return %x\n", (int)ret);
        MACTlog("*CryptAcquireContextA(%p,%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CryptAcquireContextA modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyCryptAcquireContextW(HCRYPTPROV *a0,
                                   LPCWSTR     a1,
                                   LPCWSTR     a2,
                                   DWORD       a3,
                                   DWORD       a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueCryptAcquireContextW(a0, a1, a2, a3, a4);

    MACTlog(":CryptAcquireContextW(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueCryptAcquireContextW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-CryptAcquireContextW will return %x\n", (int)ret);
        MACTlog("*CryptAcquireContextW(%p,%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+CryptAcquireContextW modified to return %x\n", ret);
        }
    }

    return ret;
}
```

```
BOOL WINAPI MyDeviceIoControl(HANDLE       a0,
                              DWORD        a1,
                              LPVOID       a2,
                              DWORD        a3,
                              LPVOID       a4,
                              DWORD        a5,
                              LPDWORD      a6,
                              LPOVERLAPPED a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueDeviceIoControl(a0, a1, a2, a3, a4, a5, a6, a7);

    MACTPrint(":DeviceIoControl(%p,%p,%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7);
    MACTPrint("*DeviceIoControl(%p,%p,%p,%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, a5, a6,
a7,
            0, 0, 0);

    return TrueDeviceIoControl(a0, a1, a2, a3, a4, a5, a6, a7);

    MACTlog(":DeviceIoControl(%p,%p,%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueDeviceIoControl(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-DeviceIoControl will return %x\n", (int)ret);
        MACTlog("*DeviceIoControl(%p,%p,%p,%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, a5,
a6, a7,
            ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+DeviceIoControl modified to return %x\n", ret);
        }
    }

    return ret;
}
```

```
BOOL WINAPI MyEnumProcesses(DWORD *a0,
                            DWORD a1,
                            DWORD *a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueEnumProcesses(a0, a1, a2);

    MACTlog(":EnumProcesses(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueEnumProcesses(a0, a1, a2);
    } __finally {
        MACTlog("-EnumProcesses will return %x\n", (int)ret);
        MACTlog("*EnumProcesses(%p,%p,%p),(%x,%x,%x)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+EnumProcesses modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyEnumProcessModules(HANDLE  a0,
                                 HMODULE *a1,
                                 DWORD   a2,
                                 LPDWORD a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueEnumProcessModules(a0, a1, a2, a3);

    MACTlog(":EnumProcessModules(%p,%p,%p,%p)\n", a0, a1, a2, a3);
```

```
    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueEnumProcessModules(a0, a1, a2, a3);
    } __finally {
        MACTlog("-EnumProcessModules will return %x\n", (int)ret);
        MACTlog("*EnumProcessModules(%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  ret,  iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+EnumProcessModules modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyEnumProcessModulesEx(HANDLE  a0,
                                   HMODULE *a1,
                                   DWORD   a2,
                                   LPDWORD a3,
                                   DWORD   a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueEnumProcessModulesEx(a0, a1, a2, a3, a4);

    MACTlog(":EnumProcessModulesEx(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret;
    __try {
        ret = TrueEnumProcessModulesEx(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-EnumProcessModulesEx will return %x\n", (int)ret);
```

```
        MACTlog("*EnumProcessModulesEx(%p,%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+EnumProcessModulesEx modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyFindFirstFile(LPCTSTR           a0,
                              LPWIN32_FIND_DATA a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindFirstFile(a0, a1);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":FindFirstFile(%s,%p)\n", buffer0, a1);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret;
    __try {
        ret = TrueFindFirstFile(a0, a1);
    } __finally {
        MACTlog("-FindFirstFile will return %p\n", ret);
        MACTlog("*FindFirstFile(%s,%p),(%p,%x,%p)", buffer0, a1, ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+FindFirstFile modified to return %p\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
```

```
}

HANDLE WINAPI MyFindFirstFileEx(LPCTSTR            a0,
                                FINDEX_INFO_LEVELS a1,
                                LPVOID             a2,
                                FINDEX_SEARCH_OPS  a3,
                                LPVOID             a4,
                                DWORD              a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindFirstFileEx(a0, a1, a2, a3, a4, a5);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":FindFirstFileEx(%s,%d,%p,%d,%p,%p)\n", buffer0, (int)a1, a2, (int)a3, a4, a5);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret;
    __try {
        ret = TrueFindFirstFileEx(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-FindFirstFileEx will return %p\n", ret);
        MACTlog("*FindFirstFileEx(%s,%d,%p,%d,%p,%p),(%p,%x,%p)", buffer0, (int)a1, a2, (int)a3,
a4, a5,
            ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+FindFirstFileEx modified to return %p\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
}

BOOL WINAPI MyFindNextFile(HANDLE             a0,
```

```
                            LPWIN32_FIND_DATAA a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindNextFile(a0, a1);

    MACTlog(":FindNextFile(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueFindNextFile(a0, a1);
    } __finally {
        MACTlog("-FindNextFile will return %p\n", ret);
        MACTlog("*FindNextFile(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+FindNextFile modified to return %p\n", ret);
        }
    }

    return ret;
}

HRSRC WINAPI MyFindResourceA(HMODULE a0,
                             LPCSTR  a1,
                             LPCSTR  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindResourceA(a0, a1, a2);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
```

```
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":FindResourceA(%p,%s,%s)\n", a0, buffer1, buffer2);

    int iOption = MACTmain("HRSRC");
    HRSRC bReturnValue = NULL;
    if(iOption == 8) {
        bReturnValue = SR_HRSRC;
    }

    HRSRC ret;
    __try {
        ret = TrueFindResourceA(a0, a1, a2);
    } __finally {
        MACTlog("-FindResourceA will return %p\n", ret);
        MACTlog("*FindResourceA(%p,%s,%s),(%p,%x,%p)",  a0,  buffer1,  buffer2,  ret,  iOption,
bReturnValue);
        if(iOption == 8) {
            ret = bReturnValue;
            MACTlog("+FindResourceA modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

HRSRC WINAPI MyFindResourceExA(HMODULE a0,
                               LPCSTR  a1,
                               LPCSTR  a2,
                               WORD    a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindResourceExA(a0, a1, a2, a3);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
```

```
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';


    MACTlog(":FindResourceExA(%p,%s,%s,%p)\n", a0, buffer1, buffer2, a3);

    int iOption = MACTmain("HRSRC");
    HRSRC bReturnValue = NULL;
    if(iOption == 8) {
        bReturnValue = SR_HRSRC;
    }

    HRSRC ret;
    __try {
        ret = TrueFindResourceExA(a0, a1, a2, a3);
    } __finally {
        MACTlog("-FindResourceExA will return %p\n", ret);
        MACTlog("*FindResourceExA(%p,%s,%s,%p),(%p,%x,%p)",  a0,  buffer1,  buffer2,  a3,  ret,
iOption, bReturnValue);
        if(iOption == 8) {
            ret = bReturnValue;
            MACTlog("+FindResourceExA modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

HWND WINAPI MyFindWindow(LPCTSTR a0,
                         LPCTSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindWindow(a0, a1);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
```

```
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTlog(":FindWindow(%s,%s)\n", buffer0, buffer1);

    int iOption = MACTmain("HRWND");
    HWND bReturnValue = NULL;
    if(iOption == 9) {
        bReturnValue = SR_HWND;
    }

    HWND ret;
    __try {
        ret = TrueFindWindow(a0, a1);
    } __finally {
        MACTlog("-FindWindow will return %p\n", ret);
        MACTlog("*FindWindow(%s,%s),(%p,%x,%p)", buffer0, buffer1, ret, iOption, bReturnValue);
        if(iOption == 9) {
            ret = bReturnValue;
            MACTlog("+FindWindow modified to return %p\n", ret);
        }
        delete[] buffer0;
        delete[] buffer1;
    }

    return ret;
}

HWND WINAPI MyFindWindowEx(HWND    a0,
                           HWND    a1,
                           LPCTSTR a2,
                           LPCTSTR a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFindWindowEx(a0, a1, a2, a3);

    int la = lstrlenA(a3);
    char *buffer3 = new char[la+1];
    strncpy(buffer3, a3, la);
    buffer3[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
```

```
    buffer2[la] = '\0';


    MACTlog(":FindWindowEx(%p,%p,%s,%s)\n", a0, a1, buffer2, buffer3);

    int iOption = MACTmain("HWND");
    HWND bReturnValue = NULL;
    if(iOption == 9) {
        bReturnValue = SR_HWND;
    }

    HWND ret;
    __try {
        ret = TrueFindWindowEx(a0, a1, a2, a3);
    } __finally {
        MACTlog("-FindWindowEx will return %p\n", ret);
        MACTlog("*FindWindowEx(%p,%p,%s,%s),(%p,%x,%p)", a0, a1, buffer2, buffer3, ret, iOption,
bReturnValue);
        if(iOption == 9) {
            ret = bReturnValue;
            MACTlog("+FindWindowEx modified to return %p\n", ret);
        }
        delete[] buffer2;
        delete[] buffer3;
    }

    return ret;
}

HINTERNET WINAPI MyFtpOpenFileW(HINTERNET a0,
                                LPCWSTR   a1,
                                DWORD     a2,
                                DWORD     a3,
                                DWORD_PTR a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueFtpOpenFileW(a0, a1, a2, a3, a4);
    }

    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize1 = origsize * 2;
    char *buffer1 = new char[newsize1];
    wcstombs_s(&convertedChars, buffer1, newsize1, a1, _TRUNCATE);
```

```
    MACTlog(":FtpOpenFileW(%s,%p,%s,%s,%p)\n", a0, buffer1, a2, a3, a4);
    MACTcomm(">FtpOpenFileW- File Name = [%s]\n", buffer1);

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueFtpOpenFileW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-FtpOpenFileW will return (void)\n");
        MACTlog("*FtpOpenFileW(%s,%p,%s,%s,%p),(%llu,%x,%llu)",  a0,  buffer1,  a2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+FtpOpenFileW modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

BOOL WINAPI MyFtpPutFile(HINTERNET a0,
                         LPCTSTR a1,
                         LPCTSTR a2,
                         DWORD a3,
                         DWORD a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueFtpPutFile(a0, a1, a2, a3, a4);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
```

```
    buffer2[la] = '\0';

    MACTlog(":FtpPutFile(%p,%s,%s,%p,%p)\n", a0, buffer1, buffer2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueFtpPutFile(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-FtpPutFile will return %p\n", ret);
        MACTlog("*FtpPutFile(%p,%s,%s,%p,%p),(%x,%x,%x)",  a0,  buffer1,  buffer2,  a3,  a4,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+FtpPutFile modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

ULONG WINAPI MyGetAdaptersInfo(PIP_ADAPTER_INFO a0,
                               PULONG           a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetAdaptersInfo(a0, a1);

    MACTlog(":GetAdaptersInfo(%p,%x)\n", a0, a1);

    int iOption = MACTmain("ULONG");
    ULONG bReturnValue = 0;
    if(iOption == 10) {
        bReturnValue = SR_ULONG;
    }

    ULONG ret;
    __try {
```

```
        ret = TrueGetAdaptersInfo(a0, a1);
    } __finally {
        MACTlog("-GetAdaptersInfo will return %p\n", ret);
        MACTlog("*GetAdaptersInfo(%p,%x),(%ld,%x,%ld)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 10) {
            ret = bReturnValue;
            MACTlog("+GetAdaptersInfo modified to return %p\n", ret);
        }
    }

    return ret;
}

SHORT WINAPI MyGetAsyncKeyState(int a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetAsyncKeyState(a0);

    MACTlog(":GetAsyncKeyState(%x)\n", a0);

    int iOption = MACTmain("SHORT");
    SHORT bReturnValue = 0;
    if(iOption == 11) {
        bReturnValue = SR_SHORT;
    }

    SHORT ret;
    __try {
        ret = TrueGetAsyncKeyState(a0);
    } __finally {
        MACTlog("-GetAsyncKeyState will return %x\n", ret);
        MACTlog("*GetAsyncKeyState(%x),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 11) {
            ret = bReturnValue;
            MACTlog("+GetAsyncKeyState modified to return %x\n", ret);
        }
    }

    return ret;
}

HDC WINAPI MyGetDC(HWND a0)
{
    if(MACTDEBUG2)
```

```
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueGetDC(a0);

    MACTlog(":GetDC(%p)\n", a0);

    HDC ret = TrueGetDC(a0);
    MACTlog("*GetDC(%p),(%ld,%d,%ld)", a0, 0, ret, 0);
    TrueSleep(50);
    return ret;

    MACTlog(":GetDC(%p)\n", a0);

    int iOption = MACTmain("HDC");
    HDC bReturnValue = NULL;
    if(iOption == 12) {
        bReturnValue = SR_HDC;
    }

    ret;
    __try {
        ret = TrueGetDC(a0);
    } __finally {
        MACTlog("-GetDC will return %p\n", ret);
        MACTlog("*GetDC(%p),(%p,%x,%p)", a0, ret, iOption, bReturnValue);
        if(iOption == 12) {
            ret = bReturnValue;
            MACTlog("+GetDC modified to return %p\n", ret);
        }
    }

    TrueSleep(100);
    return ret;
}

HWND WINAPI MyGetForegroundWindow(void)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueGetForegroundWindow();

    MACTlog(":GetForegroundWindow()\n");

    int iOption = MACTmain("HWND");
```

```
    HWND bReturnValue = NULL;
    if(iOption == 12) {
        bReturnValue = SR_HWND;
    }

    HWND ret;
    __try {
        ret = TrueGetForegroundWindow();
    } __finally {
        MACTlog("-GetForegroundWindow will return %p\n", ret);
        MACTlog("*GetForegroundWindow(),(%p,%x,%p)", ret, iOption, bReturnValue);
        if(iOption == 12) {
            ret = bReturnValue;
            MACTlog("+GetForegroundWindow modified to return %p\n", ret);
        }
    }

    return ret;
}

INT WINAPI MyGetWindowText(HWND   a0,
                           LPTSTR a1,
                           int    a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetWindowText(a0, a1, a2);

    MACTlog(":GetWindowText(%p,%p,%x)\n", a0, a1, a2);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret = 0;

    ret = TrueGetWindowText(a0, a1, a2);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';
```

```
    MACTlog("-GetWindowText will return %x\n", ret);
    MACTlog("*GetWindowText(%p,%s,%x),(%x,%x,%x)", a0, buffer1, a2, ret, iOption, bReturnValue);
    if(iOption == 14) {
        ret = bReturnValue;
        MACTlog("+GetWindowText modified to return %x\n", ret);
    }
    delete[] buffer1;

    TrueSleep(100);
    return ret;
}


HOSTENT * WINAPI Mygethostbyname(const char *a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return Truegethostbyname(a0);

    MACTlog(":gethostbyname(%s)\n", a0);

    MACTcomm(">gethostbyname: %s\n\n", a0);

    int iOption = MACTmain("HOSTENT");
    hostent bReturnValue;
    if(iOption == 13) {
        bReturnValue = SR_HOSTENT;
    }

    hostent *ret;
    __try {
        ret = Truegethostbyname(a0);
    } __finally {
        MACTlog("-gethostbyname will return %p\n", ret);
        MACTlog("*gethostbyname(%s),(%p,%x,%p)", a0, ret, iOption, bReturnValue);
        if(iOption == 13) {
            ret = &bReturnValue;
            MACTlog("+gethostbyname modified to return %p\n", ret);
        }
    }

    return ret;
}

int WINAPI Mygethostname(const char *a0,
```

```
                        int         a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return Truegethostname((char *)a0, a1);

    MACTlog(":gethostname(%s,%x)\n", a0, a1);
    MACTcomm(">gethostname: %s\n\n", a0);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
        ret = Truegethostname((char *)a0, a1);
    } __finally {
        MACTlog("-gethostname will return %x\n", ret);
        MACTlog("*gethostname(%s,%x),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+gethostname modified to return %x\n", ret);
        }
    }

    return ret;
}

static INT WINAPI Mygetaddrinfo(PCSTR            a0,
                                PCSTR            a1,
                                const ADDRINFOA *a2,
                                PADDRINFOA      *a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return Truegetaddrinfo(a0, a1, a2, a3);

    MACTlog(":getaddrinfo(%s,%s,%p,%p)\n", a0, a1, a2, a3);
    MACTcomm(">getaddrinfo: Node:%s Service:%s\n\n", a0, a1);

    int iOption = MACTmain("INT");
```

```
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret;
    __try {
        ret = Truegetaddrinfo(a0, a1, a2, a3);
    } __finally {
        MACTlog("-getaddrinfo will return %x\n", ret);
        MACTlog("*ggetaddrinfo(%s,%s,%p,%p),(%x,%x,%x)",   a0,   a1,   a2,   a3,   ret,   iOption,
bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+getaddrinfo modified to return %x\n", ret);
        }
    }

    return ret;
}

SHORT WINAPI MyGetKeyState(int a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueGetKeyState(a0);

    MACTlog(":GetKeyState(%x)\n", a0);

    int iOption = MACTmain("SHORT");
    SHORT bReturnValue = 0;
    if(iOption == 11) {
        bReturnValue = SR_SHORT;
    }

    SHORT ret;
    __try {
        ret = TrueGetKeyState(a0);
    } __finally {
        MACTlog("-GetKeyState will return %x\n", ret);
        MACTlog("*GetKeyState(%x),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 11) {
            ret = bReturnValue;
            MACTlog("+GetKeyState modified to return %x\n", ret);
        }
```

```
    }

    TrueSleep(100);
    return ret;
}

DWORD WINAPI MyGetModuleFileName(HMODULE a0,
                                 LPTSTR  a1,
                                 DWORD   a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetModuleFileName(a0, a1, a2);

    MACTlog(":GetModuleFileName(%p,%p,%x)\n", a0, a1, a2);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;

    ret = TrueGetModuleFileName(a0, a1, a2);

    size_t la1 = strlen(a1);
    char *buffer = new char[la1+1];
    strncpy(buffer, a1, la1);
    buffer[la1] = '\0';

    MACTlog("-GetModuleFileName will return %ld with %s\n", ret, buffer);
    MACTlog("*GetModuleFileName(%p,%s,%x),(%ld,%ld,%ld)",   a0,   buffer,   a2,   ret,   iOption,
bReturnValue);
    if(iOption == 15) {
        ret = bReturnValue;
        MACTlog("+GetModuleFileName modified to return %ld\n", ret);
    }
    delete[] buffer;

    return ret;
}

DWORD WINAPI MyGetModuleFileNameExA(HANDLE  a0,
                                    HMODULE a1,
```

```
                                    LPSTR   a2,
                                    DWORD   a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetModuleFileNameExA(a0, a1, a2, a3);

    int la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":GetModuleFileNameExA(%p,%p,%s,%x)\n", a0, a1, buffer2, a3);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;
    __try {
        ret = TrueGetModuleFileNameExA(a0, a1, a2, a3);
    } __finally {
        MACTlog("-GetModuleFileNameExA will return %ld with %s\n", ret, buffer2);
        MACTlog("*GetModuleFileNameExA(%p,%p,%s,%x),(%x,%x,%x)",  a0,  a1,  buffer2,  a3,  ret,
iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+GetModuleFileNameExA modified to return %x\n", ret);
        }
        delete[] buffer2;
    }

    return ret;
}

DWORD WINAPI MyGetModuleFileNameExW(HANDLE  a0,
                                    HMODULE a1,
                                    LPWSTR  a2,
                                    DWORD   a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);
```

```
    if(!MACTSTARTED)
        return TrueGetModuleFileNameExW(a0, a1, a2, a3);

    size_t origsize = wcslen(a2) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer2 = new char[newsize];
    wcstombs_s(&convertedChars, buffer2, newsize, a2, _TRUNCATE);

    MACTlog(":GetModuleFileNameExW(%p,%p,%s,%x)\n", a0, a1, a2, a3);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;
    __try {
        ret = TrueGetModuleFileNameExW(a0, a1, a2, a3);
    } __finally {
        MACTlog("-GetModuleFileNameExW will return %ld with %s\n", ret, buffer2);
        MACTlog("*GetModuleFileNameExW(%p,%p,%s,%x),(%x,%x,%x)",  a0,  a1,  buffer2,  a3,  ret,
iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+GetModuleFileNameExW modified to return %x\n", ret);
        }
        delete[] buffer2;
    }

    return ret;
}

HMODULE WINAPI MyGetModuleHandle(LPCTSTR a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetModuleHandle(a0);


    int la;
    if(a0 == NULL)
        la = 4;
    else
```

```
        la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    if(a0 == NULL)
        strncpy(buffer0, "NULL", la);
    else
        strncpy(buffer0, a0, la);
    buffer0[la] = '\0';


    MACTlog(":GetModuleHandle(%s)\n", buffer0);

    int iOption = MACTmain("HMODULE");
    HMODULE bReturnValue = NULL;
    if(iOption == 16) {
        bReturnValue = SR_HMODULE;
        MACTPrint(">Returning from MACTmain %x\n", bReturnValue);
    }

    HMODULE ret;
    __try {
        ret = TrueGetModuleHandle(a0);
    } __finally {
        MACTlog("-GetModuleHandle will return %p\n", ret);
        MACTlog("*GetModuleHandle(%s),(%p,%x,%p)", buffer0, ret, iOption, bReturnValue);
        if(iOption == 16) {
            ret = bReturnValue;
            MACTPrint(">Returning from MACTmain %x\n", bReturnValue);
            MACTPrint(">Returning from MACTmain %x\n", ret);
            MACTlog("+GetModuleHandle modified to return %p\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
}

BOOL WINAPI MyGetModuleHandleEx(DWORD    a0,
                                LPCTSTR  a1,
                                HMODULE *a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetModuleHandleEx(a0, a1, a2);

    int la = lstrlenA(a1);
```

```
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    if(a0 == GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS)
        MACTlog(":GetModuleHandleEx(%p,%x,%p)\n", a0, a1, a2);
    else {
        MACTlog(":GetModuleHandleEx(%p,%s,%p)\n", a0, buffer1, a2);
    }


    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueGetModuleHandleEx(a0, a1, a2);
    } __finally {
        MACTlog("-GetModuleHandleEx will return %p\n", ret);
        if(a0 == GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS)
            MACTlog("*GetModuleHandleEx(%p,%x,%p),(%x,%x,%x)",   a0,   a1,   a2,   ret,   iOption,
bReturnValue);
        else
            MACTlog("*GetModuleHandleEx(%p,%s,%p),(%x,%x,%x)",  a0,  buffer1,  a2,  ret,  iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+GetModuleHandleEx modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

    return ret;
}

int GetBufferLength(LPCSTR a1)
{
    int ia1 = (int)a1;

    int la;
    if(ia1 <= 65535)
        la = std::to_string(ia1).length();
    else
        la = lstrlenA(a1);
```

```
    return(la);
}

void GetBuffer(LPCSTR a1, char *buffer1, int la)
{
    int ia1 = (int)a1;

    if(ia1 <= 65535) {
        strncpy(buffer1, std::to_string(ia1).c_str(), la);
    }
    else {
        strncpy(buffer1, a1, la);
    }

    buffer1[la] = '\0';
}

FARPROC WINAPI MyGetProcAddress(HMODULE a0,
                                LPCSTR  a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueGetProcAddress(a0, a1);

    int la = GetBufferLength(a1);
    char *buffer1 = new char[la+1];

    GetBuffer(a1, buffer1, la);

    MACTPrint(":GetProcAddress(%p,%s)\n", a0, buffer1);

    int iOption = MACTmain("FARPROC");
    FARPROC bReturnValue = NULL;
    if(iOption == 17) {
        bReturnValue = SR_FARPROC;
    }

    FARPROC ret = NULL;
    __try {
        ret = TrueGetProcAddress(a0, a1);
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        MACTPrint(">>>>>>>>>>GetProcAddress E\n");
        MACTPrint("-GetProcAddress failed.\n");
```

```
        MACTPrint("*GetProcAddress(%p,%s),(%s,%x,%p)",    a0,    buffer1,    "FAIL",    iOption,
bReturnValue);
//        TrueSleep(200);
        delete[] buffer1;
        return ret;
    }

    MACTPrint("-GetProcAddress will return %p\n", ret);
    MACTPrint("*GetProcAddress(%p,%s),(%p,%x,%p)", a0, buffer1, ret, iOption, bReturnValue);
    if(iOption == 17) {
        ret = bReturnValue;
        MACTPrint("+GetProcAddress modified to return %p\n", ret);
    }

// chgsleep
    TrueSleep(200);
    delete[] buffer1;
    return ret;
}

VOID WINAPI MyGetStartupInfoA(LPSTARTUPINFOA a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        TrueGetStartupInfoA(a0);
        return;
    }

    TrueGetStartupInfoA(a0);

    LPMACTSTARTUPINFOA MACTstartup = (LPMACTSTARTUPINFOA) a0;
    MACTPrint(">-------------lpTitle = %s\n", a0->lpTitle);

    a0 = (LPSTARTUPINFOA)MACTstartup;

    MACTlog(":GetStartupInfoA(%p)\n", a0);

    __try {
        TrueGetStartupInfoA(a0);
    } __finally {
        MACTlog("-GetStartupInfoA will return (void)\n");
        MACTlog("*GetStartupInfoA(%p),(void,void,void)", a0);
    }

    return;
```

```
}

LANGID WINAPI MyGetSystemDefaultLangID(void)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetSystemDefaultLangID();

    MACTlog(":GetSystemDefaultLangID()\n");

    int iOption = MACTmain("LANGID");
    LANGID bReturnValue = NULL;
    if(iOption == 18) {
        bReturnValue = SR_LANGID;
    }

    LANGID ret;
//    __try {
        ret = TrueGetSystemDefaultLangID();
//    } __finally {
        MACTlog("-GetSystemDefaultLangID will return %p\n", ret);
        MACTlog("*GetSystemDefaultLangID(),(%p,%x,%p)", ret, iOption, bReturnValue);
        if(iOption == 18) {
            ret = bReturnValue;
            MACTlog("+GetSystemDefaultLangID modified to return %p\n", ret);
        }
//    }

    return ret;
}

DWORD WINAPI MyGetTempPathA(DWORD a0,
                            LPSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetTempPathA(a0, a1);

    MACTlog(":GetTempPathA(%p,%p)\n", a0, a1);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
```

```
        bReturnValue = SR_DWORD;
    }

    DWORD ret = 0;

    ret = TrueGetTempPathA(a0, a1);

    size_t la1 = strlen(a1);
    char *buffer = new char[la1+1];
    strncpy(buffer, a1, la1);
    buffer[la1] = '\0';

    MACTlog("-GetTempPathA will return %ld with %s\n", ret, buffer);
    MACTlog("*GetTempPathA(%p,%s),(%ld,%ld,%ld)", a0, buffer, ret, iOption, bReturnValue);
    if(iOption == 15) {
        ret = bReturnValue;
        MACTlog("+GetTempPathA modified to return %ld\n", ret);
    }
    delete[] buffer;

    return ret;
}

BOOL WINAPI MyGetThreadContext(HANDLE    a0,
                               LPCONTEXT a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetThreadContext(a0, a1);

    MACTlog(":GetThreadContext(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueGetThreadContext(a0, a1);
    } __finally {
        MACTlog("-GetThreadContext will return %p\n", ret);
        MACTlog("*GetThreadContext(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
```

```
            ret = bReturnValue;
            MACTlog("+GetThreadContext modified to return %p\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyGetVersionEx(LPOSVERSIONINFO a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetVersionEx(a0);

    MACTlog(":GetVersionEx(%x)\n", a0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueGetVersionEx(a0);
    } __finally {
        MACTlog("-GetVersionEx will return %p\n", ret);
        MACTlog("*GetVersionEx(%x),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+GetVersionEx modified to return %x\n", ret);
        }
    }

    return ret;
}

UINT WINAPI MyGetWindowsDirectory(LPTSTR a0,
                                  UINT   a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueGetWindowsDirectory(a0, a1);
```

```
    MACTlog(":GetWindowsDirectory(%p,%p,%x)\n", a0, a1);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_DWORD;
    }

    UINT ret;

    ret = TrueGetWindowsDirectory(a0, a1);

    size_t la0 = strlen(a0);
    char *buffer = new char[la0+1];
    strncpy(buffer, a0, la0);
    buffer[la0] = '\0';

    MACTlog("-GetWindowsDirectory will return %x with %s\n", ret, buffer);
    MACTlog("*GetWindowsDirectory(%s,%p),(%x,%x,%x)", a0, buffer, ret, iOption, bReturnValue);
    if(iOption == 3) {
        ret = bReturnValue;
        MACTlog("+GetWindowsDirectory modified to return %x\n", ret);
    }
    delete[] buffer;

    return ret;
}

ULONG WINAPI Myinet_addr(const char * a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return Trueinet_addr(a0);
    }

    MACTlog(":inet_addr(%p)\n", a0);
    MACTcomm(">inet_addr %s\n", a0);

    int iOption = MACTmain("ULONG");
    ULONG bReturnValue = 0;
    if(iOption == 10) {
        bReturnValue = SR_ULONG;
    }
```

```
    ULONG ret;
    __try {
        ret = Trueinet_addr(a0);
    } __finally {
        MACTlog("-inet_addr will return (void)\n");
        MACTlog("*inet_addr(%p),(%ld,%x,%ld)", a0, ret, iOption, bReturnValue);
        if(iOption == 10) {
            ret = bReturnValue;
            MACTlog("+inet_addr modified to return %p\n", ret);
        }
    }

    return ret;
}

HINTERNET WINAPI MyInternetOpen(LPCTSTR a0,
                                DWORD   a1,
                                LPCTSTR a2,
                                LPCTSTR a3,
                                DWORD   a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueInternetOpen(a0, a1, a2, a3, a4);
    }

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    la = lstrlenA(a3);
    char *buffer3 = new char[la+1];
    strncpy(buffer3, a3, la);
    buffer3[la] = '\0';


    MACTlog(":InternetOpen(%s,%p,%s,%s,%p)\n", buffer0, a1, buffer2, buffer3, a4);
    MACTcomm(">InternetOpen\n");
```

```
    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueInternetOpen(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-InternetOpen will return (void)\n");
        MACTlog("*InternetOpen(%s,%p,%s,%s,%p),(%llu,%x,%llu)", buffer0, a1, buffer2, buffer3, a4,
ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+InternetOpen modified to return %p\n", ret);
        }
        delete[] buffer0;
        delete[] buffer2;
        delete[] buffer3;
    }

    return ret;
}


HINTERNET WINAPI MyInternetOpenW(LPCWSTR a0,
                                 DWORD   a1,
                                 LPCWSTR a2,
                                 LPCWSTR a3,
                                 DWORD   a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueInternetOpenW(a0, a1, a2, a3, a4);
    }

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize0 = origsize * 2;
    char *buffer0 = new char[newsize0];
    wcstombs_s(&convertedChars, buffer0, newsize0, a0, _TRUNCATE);

    origsize = wcslen(a2) + 1;
    convertedChars = 0;
```

```
    const size_t newsize2 = origsize * 2;
    char *buffer2 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer2, newsize2, a2, _TRUNCATE);

    origsize = wcslen(a3) + 1;
    convertedChars = 0;
    const size_t newsize3 = origsize * 2;
    char *buffer3 = new char[newsize3];
    wcstombs_s(&convertedChars, buffer3, newsize3, a3, _TRUNCATE);

    MACTlog(":InternetOpenW(%s,%p,%s,%s,%p)\n", buffer0, a1, buffer2, buffer3, a4);
    MACTcomm(">InternetOpenW\n");

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueInternetOpenW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-InternetOpenW will return (void)\n");
        MACTlog("*InternetOpenW(%s,%p,%s,%s,%p),(%llu,%x,%llu)",  buffer0,  a1,  buffer2,  buffer3,
a4, ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+InternetOpenW modified to return %p\n", ret);
        }
        delete[] buffer0;
        delete[] buffer2;
        delete[] buffer3;
    }

    return ret;
}

HINTERNET WINAPI MyInternetOpenUrl(HINTERNET a0,
                                   LPCTSTR   a1,
                                   LPCTSTR   a2,
                                   DWORD     a3,
                                   DWORD     a4,
                                   DWORD_PTR a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);
```

```
    if(!MACTSTARTED) {
        return TrueInternetOpenUrl(a0, a1, a2, a3, a4, a5);
    }

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":InternetOpenUrl(%p,%s,%s,%p,%p,%p)\n", a0, buffer1, buffer2, a3, a4, a5);
    MACTcomm(">InternetOpenUrl\n\tURL =[%s]\n\tHeaders = [%s]\n", buffer1, buffer2);

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueInternetOpenUrl(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-InternetOpenUrl will return (void)\n");
        MACTlog("*InternetOpenUrl(%p,%s,%s,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, buffer2, a3, a4,
a5, ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+InternetOpenUrl modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

HINTERNET WINAPI MyInternetOpenUrlA(HINTERNET a0,
                                    LPCSTR    a1,
                                    LPCSTR    a2,
                                    DWORD     a3,
                                    DWORD     a4,
```

```
                                    DWORD_PTR a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueInternetOpenUrlA(a0, a1, a2, a3, a4, a5);
    }

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":InternetOpenUrlA(%p,%s,%s,%p,%p,%p)\n", a0, buffer1, buffer2, a3, a4, a5);
    MACTcomm(">InternetOpenUrlA\n\tURL =[%s]\n\tHeaders = [%s]\n", buffer1, buffer2);

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueInternetOpenUrlA(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-InternetOpenUrlA will return (void)\n");
        MACTlog("*InternetOpenUrlA(%p,%s,%s,%p,%p,%p),(%ld,%x,%ld)", a0, buffer1, buffer2, a3, a4,
a5, ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+InternetOpenUrlA modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

HINTERNET WINAPI MyInternetConnectW(HINTERNET     a0,
```

```
                                    LPCWSTR       a1,
                                    INTERNET_PORT a2,
                                    LPCWSTR       a3,
                                    LPCWSTR       a4,
                                    DWORD         a5,
                                    DWORD         a6,
                                    DWORD         a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueInternetConnectW(a0, a1, a2, a3, a4, a5, a6, a7);
    }

    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize1 = origsize * 2;
    char *buffer1 = new char[newsize1];
    wcstombs_s(&convertedChars, buffer1, newsize1, a1, _TRUNCATE);

    origsize = wcslen(a3) + 1;
    convertedChars = 0;
    const size_t newsize3 = origsize * 2;
    char *buffer3 = new char[newsize3];
    wcstombs_s(&convertedChars, buffer3, newsize3, a3, _TRUNCATE);

    origsize = wcslen(a4) + 1;
    convertedChars = 0;
    const size_t newsize4 = origsize * 2;
    char *buffer4 = new char[newsize4];
    wcstombs_s(&convertedChars, buffer4, newsize4, a4, _TRUNCATE);

    MACTlog(":InternetConnectW(%p,%s,%p,%s,%s,%p,%p,%p)\n", a0, buffer1, a2, buffer3, buffer4, a5,
a6, a7);
    MACTcomm(">InternetConnectW\n\tServer =[%s]\n\tUser = [%s]\n\tPassword = [%s]\n", buffer1,
buffer3, buffer4);

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueInternetConnectW(a0, a1, a2, a3, a4, a5, a6, a7);
```

```
    } __finally {
        MACTlog("-InternetConnectW will return (void)\n");
        MACTlog("*InternetConnectW(%p,%s,%p,%s,%s,%p,%p,%p),(%ld,%x,%ld)",   a0,   buffer1,   a2,
buffer3, buffer4, a5, a6, a7, ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+InternetConnectW modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer3;
        delete[] buffer4;
    }

    return ret;
}

HINTERNET WINAPI MyHttpOpenRequestW(HINTERNET  a0,
                                    LPCWSTR    a1,
                                    LPCWSTR    a2,
                                    LPCWSTR    a3,
                                    LPCWSTR    a4,
                                    LPCWSTR   *a5,
                                    DWORD      a6,
                                    DWORD_PTR  a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED) {
        return TrueHttpOpenRequestW(a0, a1, a2, a3, a4, a5, a6, a7);
    }

    LPCWSTR a1n = a1;
    if(a1 == NULL) {
        a1n = L"GET\0";
    }
    size_t origsize = wcslen(a1n) + 1;
    size_t convertedChars = 0;
    const size_t newsize1 = origsize * 2;
    char *buffer1 = new char[newsize1];
    wcstombs_s(&convertedChars, buffer1, newsize1, a1n, _TRUNCATE);

    origsize = wcslen(a2) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer2 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer2, newsize2, a2, _TRUNCATE);
```

```
    origsize = wcslen(a3) + 1;
    convertedChars = 0;
    const size_t newsize3 = origsize * 2;
    char *buffer3 = new char[newsize3];
    wcstombs_s(&convertedChars, buffer3, newsize3, a3, _TRUNCATE);

    MACTlog(":HttpOpenRequestW(%p,%s,%s,%s,%p,%p,%p,%p)\n", a0, buffer1, buffer2, buffer3, a4, a5,
a6, a7);
    MACTcomm(">HttpOpenRequestW\n\tVerb  =[%s]\n\tName  =  [%s]\n\tVersion  =  [%s]\n",  buffer1,
buffer2, buffer3);

    int iOption = MACTmain("HINTERNET");
    HINTERNET bReturnValue = NULL;
    if(iOption == 19) {
        bReturnValue = SR_HINTERNET;
    }

    HINTERNET ret;
    __try {
        ret = TrueHttpOpenRequestW(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-HttpOpenRequestW will return (void)\n");
        MACTlog("*HttpOpenRequestW(%p,%s,%s,%s,%p,%p,%p,%p)\n", a0, buffer1, buffer2, buffer3, a4,
a5, a6, a7, ret, iOption, bReturnValue);
        if(iOption == 19) {
            ret = bReturnValue;
            MACTlog("+HttpOpenRequestW modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
        delete[] buffer3;
    }

    return ret;
}

BOOL WINAPI MyHttpSendRequestW(HINTERNET a0,
                               LPCWSTR   a1,
                               DWORD     a2,
                               LPVOID    a3,
                               DWORD     a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
```

```
        return TrueHttpSendRequestW(a0, a1, a2, a3, a4);

    MACTlog(":HttpSendRequestW(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);
    MACTcomm(">HttpSendRequestW\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueHttpSendRequestW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-HttpSendRequestW will return %p\n", ret);
        MACTlog("*HttpSendRequestW(%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+HttpSendRequestW modified to return %p\n", ret);
        }
    }

//    TrueSleep(50);
    return ret;
}


BOOL WINAPI MyHttpSendRequestExW(HINTERNET           a0,
                                 LPINTERNET_BUFFERSW a1,
                                 LPINTERNET_BUFFERSW a2,
                                 DWORD               a3,
                                 DWORD_PTR           a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueHttpSendRequestExW(a0, a1, a2, a3, a4);

    MACTlog(":HttpSendRequestExW(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);
    MACTcomm(">HttpSendRequestExW\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
```

```
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueHttpSendRequestExW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-HttpSendRequestExW will return %p\n", ret);
        MACTlog("*HttpSendRequestExW(%p,%p,%p,%p,%p),(%x,%x,%x)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+HttpSendRequestExW modified to return %p\n", ret);
        }
    }

//    TrueSleep(50);
    return ret;
}

BOOL WINAPI MyInternetReadFile(HINTERNET a0,
                               LPVOID    a1,
                               DWORD     a2,
                               LPDWORD   a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueInternetReadFile(a0, a1, a2, a3);

    MACTlog(":InternetReadFile(%p,%p,%p,%p)\n", a0, a1, a2, a3);
    MACTcomm(">InternetReadFile\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueInternetReadFile(a0, a1, a2, a3);
    } __finally {
        MACTlog("-InternetReadFile will return %p\n", ret);
        MACTlog("*InternetReadFile(%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  ret,  iOption,
bReturnValue);
```

```
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+InternetReadFile modified to return %p\n", ret);
        }
    }

//    TrueSleep(50);
    return ret;
}

BOOL WINAPI MyInternetWriteFile(HINTERNET a0,
                                LPCVOID   a1,
                                DWORD     a2,
                                LPDWORD   a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueInternetWriteFile(a0, a1, a2, a3);

    MACTlog(":InternetWriteFile(%p,%p,%p,%p)\n", a0, a1, a2, a3);
    MACTcomm(">InternetWriteFile\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueInternetWriteFile(a0, a1, a2, a3);
    } __finally {
        MACTlog("-InternetWriteFile will return %p\n", ret);
        MACTlog("*InternetWriteFile(%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  ret,  iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+InternetWriteFile modified to return %p\n", ret);
        }
    }

    return ret;
}

//Doesnt matter, as this is only running on a 32 bit machine
```

```
BOOL WINAPI MyIsWow64Process(HANDLE a0,
                             PBOOL  a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueIsWow64Process(a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueIsWow64Process(a0, a1);
    } __finally {
        if(iOption == 1) {
            ret = bReturnValue;
        }
    }

    return ret;
}

NTSTATUS WINAPI MyLdrLoadDll(PWCHAR            a0,
                             ULONG             a1,
                             PUNICODE_STRING   a2,
                             PHANDLE           a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueLdrLoadDll(a0, a1, a2, a3);

// #include <codecvt>
    std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>,wchar_t> convert;
    std::string sStr = convert.to_bytes((wchar_t*)a2->Buffer);

    size_t origsize;
    if(a0 == NULL)
        origsize = 4;
    else
        origsize = wcslen(a0) + 1;
```

```
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];

    if(a0 == NULL) {
        strncpy(buffer0, "NULL", origsize);
        buffer0[origsize] = '\0';
    }
    else
        wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    MACTlog(":LdrLoadDll(%s,%p,%s,%p)\n", buffer0, a1, sStr.c_str(), a3);

    int iOption = MACTmain("NTSTATUS");
    NTSTATUS bReturnValue = NULL;
    if(iOption == 20) {
        bReturnValue = SR_NTSTATUS;
    }

    NTSTATUS ret;

    ret = TrueLdrLoadDll(a0, a1, a2, a3);

    MACTlog("-LdrLoadDll will return (void)\n");
    MACTlog("*LdrLoadDll(%s,%p,%s,%p),(%p,%x,%p)", buffer0, a1, sStr.c_str(), a3, ret, iOption,
bReturnValue);
    if(iOption == 20) {
        ret = bReturnValue;
        MACTlog("+LdrLoadDll modified to return %p\n", ret);

    }
    delete[] buffer0;

//    TrueSleep(250);
    return ret;
}

NTSTATUS WINAPI MyRtlCreateRegistryKey(ULONG a0,
                                       PWSTR a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueRtlCreateRegistryKey(a0, a1);
```

```
    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer1 = new char[newsize];
    wcstombs_s(&convertedChars, buffer1, newsize, a1, _TRUNCATE);

    MACTlog(":RtlCreateRegistryKey(%p,%s)\n", a0, buffer1);

    MACTreg(">RtlCreateRegistryKey = [%s]\n", buffer1);

    int iOption = MACTmain("NTSTATUS");
    NTSTATUS bReturnValue = NULL;
    if(iOption == 20) {
        bReturnValue = SR_NTSTATUS;
    }

    NTSTATUS ret;
    __try {
        ret = TrueRtlCreateRegistryKey(a0, a1);
    } __finally {
        MACTlog("-RtlCreateRegistryKey will return (void)\n");
        MACTlog("*RtlCreateRegistryKey(%p,%s),(%p,%x,%p)",    a0,    buffer1,    ret,    iOption,
bReturnValue);
        if(iOption == 20) {
            ret = bReturnValue;
            MACTlog("+RtlCreateRegistryKey modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

//    TrueSleep(50);
    return ret;
}

NTSTATUS WINAPI MyRtlWriteRegistryValue(ULONG  a0,
                                        PCWSTR a1,
                                        PCWSTR a2,
                                        ULONG  a3,
                                        PVOID  a4,
                                        ULONG  a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueRtlWriteRegistryValue(a0, a1, a2, a3, a4, a5);
```

```
    size_t origsize = wcslen(a1) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer1 = new char[newsize];
    wcstombs_s(&convertedChars, buffer1, newsize, a1, _TRUNCATE);

    origsize = wcslen(a2) + 1;
    convertedChars = 0;
    const size_t newsize2 = origsize * 2;
    char *buffer2 = new char[newsize2];
    wcstombs_s(&convertedChars, buffer2, newsize2, a2, _TRUNCATE);

    MACTlog(":RtlWriteRegistryValue(%p,%s,%s,%p,%p,%p)\n", a0, buffer1, buffer2, a3, a4, a5);

    MACTreg(">RtlWriteRegistryValue Path = [%s] Name = [%s]\n", buffer1, buffer2);

    int iOption = MACTmain("NTSTATUS");
    NTSTATUS bReturnValue = NULL;
    if(iOption == 20) {
        bReturnValue = SR_NTSTATUS;
    }

    NTSTATUS ret;
    __try {
        ret = TrueRtlWriteRegistryValue(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-RtlWriteRegistryValue will return (void)\n");
        MACTlog("*RtlWriteRegistryValue(%p,%s,%s,%p,%p,%p),(%p,%x,%p)", a0, buffer1, buffer2, a3,
a4, a5, ret, iOption, bReturnValue);
        if(iOption == 20) {
            ret = bReturnValue;
            MACTlog("+RtlWriteRegistryValue modified to return %p\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

//    TrueSleep(50);
    return ret;
}

HGLOBAL WINAPI MyLoadResource(HMODULE a0,
                              HRSRC   a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);
```

```
    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueLoadResource(a0, a1);

    MACTlog(":LoadResource(%p,%p)\n", a0, a1);

    int iOption = MACTmain("HGLOBAL");
    HGLOBAL bReturnValue = NULL;
    if(iOption == 21) {
        bReturnValue = SR_HGLOBAL;
    }

    HGLOBAL ret = 0;
    __try {
        ret = TrueLoadResource(a0, a1);
    } __finally {
        MACTlog("-LoadResource will return (void)\n");
        MACTlog("*LoadResource(%p,%p),(%p,%x,%p)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 21) {
            ret = bReturnValue;
            MACTlog("+LoadResource modified to return %p\n", ret);
        }
    }

//    TrueSleep(250);
    return ret;
}

NTSTATUS WINAPI MyLsaEnumerateLogonSessions(PULONG a0,
                                            PLUID  *a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueLsaEnumerateLogonSessions(a0, a1);

    MACTlog(":LsaEnumerateLogonSessions(%p,%p)\n", a0, a1);

    int iOption = MACTmain("NTSTATUS");
    NTSTATUS bReturnValue = NULL;
    if(iOption == 20) {
        bReturnValue = SR_NTSTATUS;
    }

    NTSTATUS ret;
    __try {
        ret = TrueLsaEnumerateLogonSessions(a0, a1);
```

```
    } __finally {
        MACTlog("-LsaEnumerateLogonSessions will return (void)\n");
        MACTlog("*LsaEnumerateLogonSessions(%p,%p),(%p,%x,%p)",    a0,    a1,    ret,    iOption,
bReturnValue);
        if(iOption == 20) {
            ret = bReturnValue;
            MACTlog("+LsaEnumerateLogonSessions modified to return %p\n", ret);
        }
    }

//    TrueSleep(250);
    return ret;
}

LPVOID WINAPI MyMapViewOfFile(HANDLE a0,
                              DWORD  a1,
                              DWORD  a2,
                              DWORD  a3,
                              SIZE_T a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueMapViewOfFile(a0, a1, a2, a3, a4);

    CopyFileFromHandle(a0, MACTdirFilesClosed);

    MACTlog(":MapViewOfFile(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("LPVOID");
    LPVOID bReturnValue = NULL;
    if(iOption == 2) {
        bReturnValue = SR_LPVOID;
    }

    LPVOID ret = 0;
    ret = TrueMapViewOfFile(a0, a1, a2, a3, a4);
    MACTlog("-MapViewOfFile will return %x\n", ret);
    MACTlog("*MapViewOfFile(%p,%p,%p,%p,%p),(%p,%x,%p)",  a0,  a1,  a2,  a3,  a4,  ret,  iOption,
bReturnValue);
    if(iOption == 2) {
        ret = bReturnValue;
        MACTlog("+MapViewOfFile modified to return %x\n", ret);
    }

//    TrueSleep(250);
```

```
    return ret;
}

LPVOID WINAPI MyMapViewOfFileEx(HANDLE a0,
                                DWORD  a1,
                                DWORD  a2,
                                DWORD  a3,
                                SIZE_T a4,
                                LPVOID a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueMapViewOfFileEx(a0, a1, a2, a3, a4, a5);

    CopyFileFromHandle(a0, MACTdirFilesMapped);

    MACTlog(":MapViewOfFileEx(%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5);

    int iOption = MACTmain("LPVOID");
    LPVOID bReturnValue = NULL;
    if(iOption == 2) {
        bReturnValue = SR_LPVOID;
    }

    LPVOID ret = 0;
    ret = TrueMapViewOfFileEx(a0, a1, a2, a3, a4, a5);
    MACTlog("-MapViewOfFileEx will return %p\n", ret);
    MACTlog("*MapViewOfFileEx(%p,%p,%p,%p,%p,%p),(%p,%x,%p)", a0, a1, a2, a3, a4, a5, ret, iOption,
bReturnValue);
    if(iOption == 2) {
        ret = bReturnValue;
        MACTlog("+MapViewOfFileEx modified to return %p\n", ret);
    }

    return ret;
}

UINT WINAPI MyMapVirtualKeyA(UINT a0,
                             UINT a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueMapVirtualKeyA(a0, a1);
```

```
    MACTlog(":MapVirtualKeyA(%p,%p)\n", a0, a1);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_UINT;
    }

    UINT ret = 0;
    __try {
        ret = TrueMapVirtualKeyA(a0, a1);
    } __finally {
        MACTlog("-MapVirtualKeyA will return %x\n", (int)&ret);
        MACTlog("*MapVirtualKeyA(%p,%p),(%u,%x,%u)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 3) {
            ret = bReturnValue;
            MACTlog("+MapVirtualKeyA modified to return %x\n", ret);
        }
    }

    return ret;
}

UINT WINAPI MyMapVirtualKeyExA(UINT a0,
                               UINT a1,
                               HKL  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueMapVirtualKeyExA(a0, a1, a2);

    MACTlog(":MapVirtualKeyExA(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_UINT;
    }

    UINT ret = 0;
    __try {
        ret = TrueMapVirtualKeyExA(a0, a1, a2);
    } __finally {
        MACTlog("-MapVirtualKeyExA will return %x\n", (int)&ret);
```

```
        MACTlog("*MapVirtualKeyExA(%p,%p,%p),(%u,%x,%u)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 3) {
            ret = bReturnValue;
            MACTlog("+MapVirtualKeyExA modified to return %x\n", ret);
        }
    }

    return ret;
}

UINT WINAPI MyMapVirtualKeyW(UINT a0,
                             UINT a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueMapVirtualKeyW(a0, a1);

    MACTlog(":MapVirtualKeyW(%p,%p)\n", a0, a1);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_UINT;
    }

    UINT ret = 0;
    __try {
        ret = TrueMapVirtualKeyW(a0, a1);
    } __finally {
        MACTlog("-MapVirtualKeyW will return %x\n", (int)&ret);
        MACTlog("*MapVirtualKeyW(%p,%p),(%u,%x,%u)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 3) {
            ret = bReturnValue;
            MACTlog("+MapVirtualKeyW modified to return %x\n", ret);
        }
    }

    return ret;
}

UINT WINAPI MyMapVirtualKeyExW(UINT a0,
                               UINT a1,
                               HKL  a2)
{
    if(MACTDEBUG2)
```

```
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueMapVirtualKeyExW(a0, a1, a2);

    MACTlog(":MapVirtualKeyExW(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("UINT");
    UINT bReturnValue = 0;
    if(iOption == 3) {
        bReturnValue = SR_UINT;
    }

    UINT ret = 0;
    __try {
        ret = TrueMapVirtualKeyExW(a0, a1, a2);
    } __finally {
        MACTlog("-MapVirtualKeyExW will return %x\n", (int) &ret);
        MACTlog("*MapVirtualKeyExW(%p,%p,%p),(%u,%x,%u)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 3) {
            ret = bReturnValue;
            MACTlog("+MapVirtualKeyExW modified to return %x\n", (int) &ret);
        }
    }

    return ret;
}

BOOL WINAPI MyModule32First(HANDLE          a0,
                            LPMODULEENTRY32 a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueModule32First(a0, a1);

    MACTlog(":Module32First(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
```

```
        ret = TrueModule32First(a0, a1);
    } __finally {
        MACTlog("-Module32First will return %x\n", ret);
        MACTlog("*Module32First(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Module32First modified to return %x\n", ret);
        }
    }

    return ret;
}


BOOL WINAPI MyModule32Next(HANDLE          a0,
                           LPMODULEENTRY32 a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueModule32Next(a0, a1);

    MACTlog(":Module32Next(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueModule32Next(a0, a1);
    } __finally {
        MACTlog("-Module32Next will return %x\n", ret);
        MACTlog("*Module32Next(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Module32Next modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyOpenMutexA(DWORD  a0,
```

```
                            BOOL   a1,
                            LPCSTR a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueOpenMutexA(a0, a1, a2);

    int la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":OpenMutexA(%p,%x,%s)\n", a0, a1, buffer2);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = NULL;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    MACTlog("-OpenMutexA will return. Returned before override.");
    MACTlog("*OpenMutexA(%p,%x,%s),(0,0,0)", a0, a1, buffer2);
    delete[] buffer2;
    return TrueOpenMutexA(a0, a1, a2);

    HANDLE ret = NULL;
    __try {
        ret = TrueOpenMutexA(a0, a1, a2);
    } __finally {
        MACTlog("-OpenMutexA will return %p\n", ret);
        MACTlog("*OpenMutexA(%p,%x,%s),(0,0,0)", a0, a1, buffer2);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+OpenMutexA modified to return %p\n", ret);
        }
    }

//    TrueSleep(100);
    return ret;
}

HANDLE WINAPI MyOpenProcess(DWORD a0,
                            BOOL  a1,
                            DWORD a2)
{
```

```
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueOpenProcess(a0, a1, a2);

    MACTlog(":OpenProcess(%p,%p,%d)\n", a0, a1, a2);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = FALSE;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
    __try {
        ret = TrueOpenProcess(a0, a1, a2);
    } __finally {
        MACTlog("-OpenProcess will return %p\n", ret);
        MACTlog("*OpenProcess(%p,%p,%d),(%p,%x,%p)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+OpenProcess modified to return %p\n", ret);
        }
    }

//    TrueSleep(100);
    return ret;
}

VOID WINAPI MyOutputDebugString(LPCTSTR a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    MACTlog("*OutputDebugString(%p),(void,void,void)", a0);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":OutputDebugString(%s)\n", buffer0);

    MACTlog("-OutputDebugString will return (void)\n");
    MACTlog("*OutputDebugString(%s),(void,void,void)", buffer0);
    delete[] buffer0;
```

```
    return TrueOutputDebugString(a0);
}

VOID WINAPI MyOutputDebugStringA(LPCSTR a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    int la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":OutputDebugStringA(%s)\n", buffer0);

    MACTlog("-OutputDebugStringA will return (void)\n");
    MACTlog("*OutputDebugStringA(%s),(void,void,void)", buffer0);

    delete[] buffer0;
    return TrueOutputDebugStringA(a0);;
}

VOID WINAPI MyOutputDebugStringW(LPCWSTR a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    size_t origsize = wcslen(a0) + 1;
    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
    char *buffer0 = new char[newsize];
    wcstombs_s(&convertedChars, buffer0, newsize, a0, _TRUNCATE);

    MACTlog(":OutputDebugStringW(%s)\n", buffer0);

    MACTlog("-OutputDebugStringW will return (void)\n");
    MACTlog("*OutputDebugStringW(%s),(void,void,void)", buffer0);

    delete[] buffer0;
    return TrueOutputDebugStringW(a0);;
}

BOOL WINAPI MyPeekNamedPipe(HANDLE  a0,
                            LPVOID  a1,
                            DWORD   a2,
                            LPDWORD a3,
```

```
                            LPDWORD a4,
                            LPDWORD a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TruePeekNamedPipe(a0, a1, a2, a3, a4, a5);

    MACTlog(":PeekNamedPipe(%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TruePeekNamedPipe(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-PeekNamedPipe will return %x\n", ret);
        MACTlog("*PeekNamedPipe(%p,%p,%p,%p,%p,%p,(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,  a5,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+PeekNamedPipe modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyProcess32First(HANDLE           a0,
                             LPPROCESSENTRY32 a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueProcess32First(a0, a1);

    MACTlog(":Process32First(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
```

```
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueProcess32First(a0, a1);
    } __finally {
        MACTlog("-Process32First will return %x\n", ret);
        MACTlog("*Process32First(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Process32First modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyProcess32FirstW(HANDLE           a0,
                              LPPROCESSENTRY32W a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueProcess32FirstW(a0, a1);

    MACTlog(":Process32FirstW(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueProcess32FirstW(a0, a1);
    } __finally {
        MACTlog("-Process32FirstW will return %x\n", ret);
        MACTlog("*Process32FirstW(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Process32FirstW modified to return %x\n", ret);
        }
    }
```

```
    return ret;
}

BOOL WINAPI MyProcess32Next(HANDLE           a0,
                            LPPROCESSENTRY32 a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueProcess32Next(a0, a1);

    MACTlog(":Process32Next(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueProcess32Next(a0, a1);
    } __finally {
        MACTlog("-Process32Next will return %x\n", ret);
        MACTlog("*Process32Next(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Process32Next modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyProcess32NextW(HANDLE           a0,
                             LPPROCESSENTRY32W a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueProcess32NextW(a0, a1);

    MACTlog(":Process32NextW(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
```

```
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueProcess32NextW(a0, a1);
    } __finally {
        MACTlog("-Process32NextW will return %x\n", ret);
        MACTlog("*Process32NextW(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Process32NextW modified to return %x\n", ret);
        }
    }

    return ret;
}

DWORD WINAPI MyQueueUserAPC(PAPCFUNC  a0,
                            HANDLE    a1,
                            ULONG_PTR a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueQueueUserAPC(a0, a1, a2);

    MACTlog(":QueueUserAPC(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;

    __try {
    ret = TrueQueueUserAPC(a0, a1, a2);
    } __finally {
        MACTlog("-QueueUserAPC will return %ld\n", ret);
        MACTlog("*QueueUserAPC(%p,%p %p),(%ld,%ld,%ld)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
```

```
            MACTlog("+QueueUserAPC modified to return %ld\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyReadProcessMemory(HANDLE  a0,
                                LPCVOID a1,
                                LPVOID  a2,
                                SIZE_T  a3,
                                SIZE_T  *a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueReadProcessMemory(a0, a1, a2, a3, a4);

    MACTlog(":ReadProcessMemory(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueReadProcessMemory(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-ReadProcessMemory will return %x\n", ret);
        MACTlog("*ReadProcessMemory(%p,%p,%p,%p,%p)(%x,%x,%x)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+ReadProcessMemory modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyRegisterHotKey(HWND a0,
                             int  a1,
                             UINT a2,
                             UINT a3)
```

```
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegisterHotKey(a0, a1, a2, a3);

    MACTlog(":RegisterHotKey(%p,%p,%p,%p)\n", a0, a1, a2, a3);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueRegisterHotKey(a0, a1, a2, a3);
    } __finally {
        MACTlog("-RegisterHotKey will return %x\n", ret);
        MACTlog("*RegisterHotKey(%p,%p,%p,%p)(%x,%x,%x)",   a0,   a1,   a2,   a3,   ret,   iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+RegisterHotKey modified to return %x\n", ret);
        }
    }

    return ret;
}

LSTATUS WINAPI MyRegOpenKeyA(HKEY   a0,
                             LPCSTR a1,
                             PHKEY  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegOpenKeyA(a0, a1, a2);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTreg(">RegOpenKeyA Key = [%s]\n", buffer1);
```

```
    MACTlog(":RegOpenKeyA(%p,%s,%p)\n", a0, buffer1, a2);

    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = NULL;
    __try {
        ret = TrueRegOpenKeyA(a0, a1, a2);
    } __finally {
        MACTlog("-RegOpenKeyA will return %p\n", ret);
        MACTlog("*RegOpenKeyA(%p,%s,%p)(%p,%x,%p)", a0, buffer1, a2, ret, iOption, bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegOpenKeyA modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

//    TrueSleep(50);
    return ret;
}

LSTATUS WINAPI MyRegOpenKeyExA(HKEY   a0,
                               LPCSTR a1,
                               DWORD  a2,
                               REGSAM a3,
                               PHKEY  a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegOpenKeyExA(a0, a1, a2, a3, a4);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    MACTreg(">RegOpenKeyExA Key = [%s]\n", buffer1);

    MACTlog(":RegOpenKeyExA(%p,%s,%p)\n", a0, buffer1, a2, a3, a4);
```

```
    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = NULL;
    __try {
        ret = TrueRegOpenKeyExA(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-RegOpenKeyExA will return %p\n", ret);
        MACTlog("*RegOpenKeyExA(%p,%s,%p)(%p,%x,%p)",  a0,  buffer1,  a2,  a3,  a4,  ret,  iOption,
bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegOpenKeyExA modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

//    TrueSleep(50);
    return ret;
}

LSTATUS WINAPI MyRegOpenKeyExW(HKEY    a0,
                               LPCWSTR a1,
                               DWORD   a2,
                               REGSAM  a3,
                               PHKEY   a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueRegOpenKeyExW(a0, a1, a2, a3, a4);

    size_t origsize;
    BOOL wcsfail = FALSE;
    __try {
        origsize = wcslen(a1) + 1;
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        wcsfail = TRUE;
        origsize = 5;
    }

    size_t convertedChars = 0;
    const size_t newsize = origsize * 2;
```

```
    char *buffer1 = new char[newsize];
    if(wcsfail) {
        strncpy(buffer1, "NULL\0", 5);
    }
    else
        wcstombs_s(&convertedChars, buffer1, newsize, a1, _TRUNCATE);

    MACTreg(">RegOpenKeyExW Key = [%s]\n", buffer1);

    MACTlog(":RegOpenKeyExW(%p,%s,%p,%p,%p)\n", a0, buffer1, a2, a3, a4);

    int iOption = MACTmain("LSTATUS");
    LSTATUS bReturnValue = FALSE;
    if(iOption == 22) {
        bReturnValue = SR_LSTATUS;
    }

    LSTATUS ret = NULL;
    __try {
        ret = TrueRegOpenKeyExW(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-RegOpenKeyExW will return %p\n", ret);
        MACTlog("*RegOpenKeyExW(%p,%s,%p,%p,%p)(%p,%x,%p)", a0, buffer1, a2, a3, a4, ret, iOption,
bReturnValue);
        if(iOption == 22) {
            ret = bReturnValue;
            MACTlog("+RegOpenKeyExW modified to return %p\n", ret);
        }
        delete[] buffer1;
    }

//    TrueSleep(50);
    return ret;
}

DWORD WINAPI MyResumeThread(HANDLE a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueResumeThread(a0);

    MACTlog(":ResumeThread(%p)\n", a0);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
```

```
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;
    __try {
        ret = TrueResumeThread(a0);
    } __finally {
        MACTlog("-ResumeThread will return %ld\n", ret);
        MACTlog("*ResumeThread(%p),(%ld,%ld,%ld)", a0, ret, iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+ResumeThread modified to return %ld\n", ret);
        }
    }

    return ret;
}

LPVOID WINAPI MySamIConnect(PDWORD a0,
                            PDWORD a1,
                            PDWORD a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSamIConnect(a0, a1, a2);

    MACTlog(":SamIConnect(%p,%p,%p)\n", a0, a1, a2);

    int iOption = MACTmain("LPVOID");
    LPVOID bReturnValue = NULL;
    if(iOption == 2) {
        bReturnValue = SR_LPVOID;
    }

    LPVOID ret;
    __try {
        ret = TrueSamIConnect(a0, a1, a2);
    } __finally {
        MACTlog("-SamIConnect will return %p\n", ret);
        MACTlog("*SamIConnect(%p,%p,%p),(%p,%x,%p)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 2) {
            ret = bReturnValue;
            MACTlog("+SamIConnect modified to return %p\n", ret);
        }
```

```
    }

    return ret;
}


BOOL WINAPI MySetFileTime(HANDLE         a0,
                          CONST FILETIME *a1,
                          CONST FILETIME *a2,
                          CONST FILETIME *a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSetFileTime(a0, a1, a2, a3);

    char Buffer1[20];
    char Buffer2[20];
    char Buffer3[20];

    if(a1 != NULL) {
        SYSTEMTIME sta1;
        FileTimeToSystemTime(a1, &sta1);
        sprintf(Buffer1,  "%02d/%02d/%d  %02d:%02d:%02d",  sta1.wMonth,  sta1.wDay,  sta1.wYear,
sta1.wHour, sta1.wMinute, sta1.wSecond);
    }
    else
        sprintf(Buffer1, "NULL");

    if(a2 != NULL) {
        SYSTEMTIME sta1;
        FileTimeToSystemTime(a2, &sta1);
        sprintf(Buffer2,  "%02d/%02d/%d  %02d:%02d:%02d",  sta1.wMonth,  sta1.wDay,  sta1.wYear,
sta1.wHour, sta1.wMinute, sta1.wSecond);
    }
    else
        sprintf(Buffer2, "NULL");

    if(a3 != NULL) {
        SYSTEMTIME sta1;
        FileTimeToSystemTime(a3, &sta1);
        sprintf(Buffer3,  "%02d/%02d/%d  %02d:%02d:%02d",  sta1.wMonth,  sta1.wDay,  sta1.wYear,
sta1.wHour, sta1.wMinute, sta1.wSecond);
    }
    else
        sprintf(Buffer3, "NULL");
```

```
    MACTlog(":SetFileTime(%p,%s,%s,%s)\n", a0, Buffer1, Buffer2, Buffer3);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueSetFileTime(a0, a1, a2, a3);
    } __finally {
        MACTlog("-SetFileTime will return %x\n", ret);

        MACTlog("*SetFileTime(%p,%s,%s,%s)(%x,%x,%x)",  a0,  Buffer1,  Buffer2,  Buffer3,  ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+SetFileTime modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MySetThreadContext(HANDLE         a0,
                               const CONTEXT *a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSetThreadContext(a0, a1);

    MACTlog(":SetThreadContext(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueSetThreadContext(a0, a1);
    } __finally {
```

```
        MACTlog("-SetThreadContext will return %x\n", ret);
        MACTlog("*SetThreadContext(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+SetThreadContext modified to return %x\n", ret);
        }
    }

    return ret;
}

HHOOK WINAPI MySetWindowsHookEx(int       a0,
                                HOOKPROC  a1,
                                HINSTANCE a2,
                                DWORD     a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSetWindowsHookEx(a0, a1, a2, a3);

    MACTlog(":SetWindowsHookEx(%p,%p,%p,%p)\n", a0, a1, a2, a3);

    int iOption = MACTmain("HHOOK");
    HHOOK bReturnValue = NULL;
    if(iOption == 23) {
        bReturnValue = SR_HHOOK;
    }

    HHOOK ret = FALSE;
    __try {
        ret = TrueSetWindowsHookEx(a0, a1, a2, a3);
    } __finally {
        MACTlog("-SetWindowsHookEx will return %p\n", ret);
        MACTlog("*SetWindowsHookEx(%p,%p,%p,%p)(%p,%x,%p)",  a0,  a1,  a2,  a3,  ret,  iOption,
bReturnValue);
        if(iOption == 23) {
            ret = bReturnValue;
            MACTlog("+SetWindowsHookEx modified to return %p\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MySfcTerminateWatcherThread(void)
```

```
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSfcTerminateWatcherThread();

    MACTlog(":SfcTerminateWatcherThread(void)\n");

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueSfcTerminateWatcherThread();
    } __finally {
        MACTlog("-SfcTerminateWatcherThread will return %x\n", ret);
        MACTlog("*SfcTerminateWatcherThread(void)(%x,%x,%x)", ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+SfcTerminateWatcherThread modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyStartServiceCtrlDispatcherA(CONST SERVICE_TABLE_ENTRYA *a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueStartServiceCtrlDispatcherA(a0);

    MACTlog(":StartServiceCtrlDispatcherA(%p)\n", a0);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
```

```
    __try {
        ret = TrueStartServiceCtrlDispatcherA(a0);
    } __finally {
        MACTlog("-StartServiceCtrlDispatcherA will return %x\n", ret);
        MACTlog("*StartServiceCtrlDispatcherA(%p)(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+StartServiceCtrlDispatcherA modified to return %x\n", ret);
        }
    }

    return ret;
}

DWORD WINAPI MySuspendThread(HANDLE a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSuspendThread(a0);

    MACTlog(":SuspendThread(%p)\n", a0);

    int iOption = MACTmain("DWORD");
    DWORD bReturnValue = 0;
    if(iOption == 15) {
        bReturnValue = SR_DWORD;
    }

    DWORD ret;
    __try {
        ret = TrueSuspendThread(a0);
    } __finally {
        MACTlog("-SuspendThread will return %ld\n", ret);
        MACTlog("*SuspendThread(%p),(%ld,%ld,%ld)", a0, ret, iOption, bReturnValue);
        if(iOption == 15) {
            ret = bReturnValue;
            MACTlog("+SuspendThread modified to return %ld\n", ret);
        }
    }

    return ret;
}

INT __cdecl Mysystem(const char *a0)
{
```

```
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return Truesystem(a0);

    MACTlog(":system(%s)\n", a0);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_UINT;
    }

    INT ret = 0;
    __try {
        ret = Truesystem(a0);
    } __finally {
        MACTlog("-system will return %x\n", ret);
        MACTlog("*system(%s),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+system modified to return %x\n", ret);
        }
    }

    return ret;
}

INT __cdecl My_wsystem(const wchar_t *a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return True_wsystem(a0);

    MACTlog(":_wsystem(%s)\n", a0);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_UINT;
    }

    INT ret = 0;
    __try {
```

```
        ret = True_wsystem(a0);
    } __finally {
        MACTlog("-_wsystem will return %x\n", ret);
        MACTlog("*_wsystem(%s,%p,%p),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+_wsystem modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL (WINAPI MyThread32First)(HANDLE          a0,
                              LPTHREADENTRY32 a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueThread32First(a0, a1);

    MACTlog(":Thread32First(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueThread32First(a0, a1);
    } __finally {
        MACTlog("-Thread32First will return %x\n", ret);
        MACTlog("*Thread32First(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Thread32First modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL (WINAPI MyThread32Next)(HANDLE          a0,
                             LPTHREADENTRY32 a1)
```

```
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueThread32Next(a0, a1);

    MACTlog(":Thread32Next(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueThread32Next(a0, a1);
    } __finally {
        MACTlog("-Thread32Next will return %x\n", ret);
        MACTlog("*Thread32Next(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Thread32Next modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyToolhelp32ReadProcessMemory(DWORD   a0,
                                          LPCVOID a1,
                                          LPVOID  a2,
                                          SIZE_T  a3,
                                          SIZE_T  *a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueToolhelp32ReadProcessMemory(a0, a1, a2, a3, a4);

    MACTlog(":Toolhelp32ReadProcessMemory(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
```

```
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueToolhelp32ReadProcessMemory(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-Toolhelp32ReadProcessMemory will return %x\n", ret);
        MACTlog("*Toolhelp32ReadProcessMemory(%p,%p,%p,%p,%p)(%x,%x,%x)", a0, a1, a2, a3, a4, ret,
iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+Toolhelp32ReadProcessMemory modified to return %x\n", ret);
        }
    }

    return ret;
}

HRESULT WINAPI MyURLDownloadToFile(LPUNKNOWN            a0,
                                   LPCTSTR              a1,
                                   LPCTSTR              a2,
                                   DWORD                a3,
                                   LPBINDSTATUSCALLBACK a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueURLDownloadToFile(a0, a1, a2, a3, a4);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":URLDownloadToFile(%p,%s,%s,%p,%p)\n", a0, buffer1, buffer2, a3, a4);
    MACTcomm(">URLDownloadToFile :[%s] [%s]\n", buffer1, buffer2);

    int iOption = MACTmain("HRESULT");
    HRESULT bReturnValue = NULL;
    if(iOption == 24) {
```

```
        bReturnValue = SR_HRESULT;
    }

    HRESULT ret = FALSE;
    __try {
        ret = TrueURLDownloadToFile(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-URLDownloadToFile will return %x\n", ret);
        MACTlog("*URLDownloadToFile(%p,%s,%s,%p,%p)(%x,%x,%x)", a0, buffer1, buffer2, a3, a4, ret,
iOption, bReturnValue);
        if(iOption == 24) {
            ret = bReturnValue;
            MACTlog("+URLDownloadToFile modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

HRESULT WINAPI MyURLDownloadToFileA(LPUNKNOWN            a0,
                                    LPCTSTR              a1,
                                    LPCTSTR              a2,
                                    _Reserved_ DWORD     a3,
                                    LPBINDSTATUSCALLBACK a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueURLDownloadToFileA(a0, a1, a2, a3, a4);

    int la = lstrlenA(a1);
    char *buffer1 = new char[la+1];
    strncpy(buffer1, a1, la);
    buffer1[la] = '\0';

    la = lstrlenA(a2);
    char *buffer2 = new char[la+1];
    strncpy(buffer2, a2, la);
    buffer2[la] = '\0';

    MACTlog(":URLDownloadToFileA(%p,%s,%s,%p,%p)\n", a0, buffer1, buffer2, a3, a4);
    MACTcomm(">URLDownloadToFileA :[%s] [%s]\n", buffer1, buffer2);

    int iOption = MACTmain("HRESULT");
```

```
    HRESULT bReturnValue = NULL;
    if(iOption == 24) {
        bReturnValue = SR_HRESULT;
    }

    HRESULT ret = FALSE;
    __try {
        ret = TrueURLDownloadToFileA(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-URLDownloadToFileA will return %x\n", ret);
        MACTlog("*URLDownloadToFileA(%p,%s,%s,%p,%p)(%x,%x,%x)",  a0,  buffer1,  buffer2,  a3,  a4,
ret, iOption, bReturnValue);
        if(iOption == 24) {
            ret = bReturnValue;
            MACTlog("+URLDownloadToFileA modified to return %x\n", ret);
        }
        delete[] buffer1;
        delete[] buffer2;
    }

    return ret;
}

INT WINAPI MyWideCharToMultiByte(UINT                                a0,
                                 DWORD                               a1,
                                 _In_NLS_string_(cchWideChar)LPCWCH a2,
                                 int                                 a3,
                                 LPSTR                               a4,
                                 int                                 a5,
                                 LPCCH                               a6,
                                 LPBOOL                              a7)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueWideCharToMultiByte(a0, a1, a2, a3, a4, a5, a6, a7);

    MACTlog(":WideCharToMultiByte(%p,%p,%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6, a7);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    INT ret = 0;
```

```
    __try {
        ret = TrueWideCharToMultiByte(a0, a1, a2, a3, a4, a5, a6, a7);
    } __finally {
        MACTlog("-WideCharToMultiByte will return %x\n", ret);
        MACTlog("*WideCharToMultiByte(%p,%p,%p,%p,%p,%p,%p,%p),(%x,%x,%x)",  a0,  a1,  a2,  a3,  a4,
a5, a6, a7, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+WideCharToMultiByte modified to return %x\n", ret);
        }
    }

    TrueSleep(200);
    return ret;
}

BOOL WINAPI MyWriteProcessMemory(HANDLE  a0,
                                 LPVOID  a1,
                                 LPCVOID a2,
                                 SIZE_T  a3,
                                 SIZE_T  *a4)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueWriteProcessMemory(a0, a1, a2, a3, a4);

    MACTlog(":WriteProcessMemory(%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }

    BOOL ret = FALSE;
    __try {
        MACTSaveFromAddress((LPVOID)a2, a3, 4);
        ret = TrueWriteProcessMemory(a0, a1, a2, a3, a4);
    } __finally {
        MACTlog("-WriteProcessMemory will return %x\n", ret);
        MACTlog("*WriteProcessMemory(%p,%p,%p,%p,%p)(%x,%x,%x)", a0, a1, a2, a3, a4, ret, iOption,
bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+WriteProcessMemory modified to return %x\n", ret);
```

```
        }
    }

    return ret;
}

SOCKET WINAPI Myaccept(SOCKET           a0,
                       struct sockaddr *a1,
                       int             *a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return Trueaccept(a0, a1, a2);

    MACTlog(":accept(%p,%p,%x)\n", a0, a1, a2);

    MACTcomm(">accept\n");
    int iOption = MACTmain("SOCKET");
    SOCKET bReturnValue = NULL;
    if(iOption == 25) {
        bReturnValue = SR_SOCKET;
    }

    SOCKET ret = NULL;
    __try {
        MACTPrint(">accept before\n");
        ret = Trueaccept(a0, a1, a2);
        MACTPrint(">accept after\n");
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        MACTlog("-accept failed\n");
        MACTlog("*accept(%p,%p,%x)(NULL,%x,NULL)", a0, a1, a2, iOption);
        return NULL;
    }

    MACTlog("-accept will return %p\n", ret);
    MACTlog("*accept(%p,%p,%x)(%p,%x,%p)", a0, a1, a2, ret, iOption, bReturnValue);
    if(iOption == 25) {
        ret = bReturnValue;
        MACTlog("+accept modified to return %p\n", ret);
    }

    return ret;
}

INT WINAPI Mybind(SOCKET                a0,
```

```
                const struct sockaddr *a1,
                int                    a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return Truebind(a0, a1, a2);

    MACTlog(":bind(%p,%p,%x)\n", a0, a1, a2);

    char *strPort = new char[6];
    SOCKADDR_IN *sa1 = (SOCKADDR_IN *)a1;
    char *ip = inet_ntoa(sa1->sin_addr);
    MACTPrint(">bind IP: %s Port: %d\n", ip, sa1->sin_port);
    MACTcomm(">bind IP: %s Port: %d\n\n", ip, sa1->sin_port);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    UINT ret = 0;
    __try {
        ret = Truebind(a0, a1, a2);
    } __finally {
        MACTlog("-bind will return %x\n", ret);
        MACTlog("*bind(%p,%p,%x),(%x,%x,%x)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+bind modified to return %x\n", ret);
        }
    }

    return ret;
}

INT WINAPI Myconnect(SOCKET                a0,
                     const struct sockaddr *a1,
                     int                    a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return Trueconnect(a0, a1, a2);
```

```
    MACTlog(":connect(%p,%p,%x)\n", a0, a1, a2);

    SOCKADDR_IN *sa1 = (SOCKADDR_IN *)a1;
    char *ip = inet_ntoa(sa1->sin_addr);

    MACTPrint(">connect IP: %s Port: %hu\n", ip, sa1->sin_port);
    MACTcomm(">connect IP: %s Port: %hu\n\n", ip, sa1->sin_port);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    UINT ret = 0;
    __try {
        ret = Trueconnect(a0, a1, a2);
    } __finally {
        MACTlog("-connect will return %x\n", ret);
        MACTlog("*connect(%p,%p,%x),(%x,%x,%x)", a0, a1, a2, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+connect modified to return %x\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyConnectNamedPipe(HANDLE       a0,
                               LPOVERLAPPED a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueConnectNamedPipe(a0, a1);

    MACTlog(":ConnectNamedPipe(%p,%p)\n", a0, a1);

    int iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_BOOL;
    }
```

```
    BOOL ret = FALSE;
    __try {
        ret = TrueConnectNamedPipe(a0, a1);
    } __finally {
        MACTlog("-ConnectNamedPipe will return %x\n", ret);
        MACTlog("*ConnectNamedPipe(%p,%p)(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+ConnectNamedPipe modified to return %x\n", ret);
        }
    }

    return ret;
}

INT WINAPI Myrecv(SOCKET   a0,
                  char    *a1,
                  int      a2,
                  int      a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return Truerecv(a0, a1, a2, a3);

    MACTlog(":recv(%p,%p,%x,%x)\n", a0, a1, a2, a3);

    MACTcomm(">recv:\n%s\n", a1);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    UINT ret = 0;
    __try {
        ret = Truerecv(a0, a1, a2, a3);
    } __finally {
        MACTlog("-recv will return %x\n", ret);
        MACTlog("*recv(%p,%p,%x,%x),(%x,%x,%x)", a0, a1, a2, a3, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+recv modified to return %x\n", ret);
        }
    }
```

```
    return ret;
}

INT WINAPI Mysend(SOCKET         a0,
                  const char    *a1,
                  int            a2,
                  int            a3)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return Truesend(a0, a1, a2, a3);

    MACTlog(":send(%p,%p,%x,%x)\n", a0, a1, a2, a3);

    MACTcomm(">send:\n%s\n", a1);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    UINT ret = 0;
    __try {
        ret = Truesend(a0, a1, a2, a3);
    } __finally {
        MACTlog("-send will return %x\n", ret);
        MACTlog("*send(%p,%p,%x,%x)", a0, a1, a2, a3, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+send modified to return %x\n", ret);
        }
    }

    return ret;
}

INT WINAPI MyWSAStartup(WORD      a0,
                        LPWSADATA a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
```

```
        return TrueWSAStartup(a0, a1);

    MACTlog(":WSAStartup(%x,%p)\n", a0, a1);

    int iOption = MACTmain("INT");
    INT bReturnValue = 0;
    if(iOption == 14) {
        bReturnValue = SR_INT;
    }

    UINT ret = 0;
    __try {
        ret = TrueWSAStartup(a0, a1);
    } __finally {
        MACTlog("-WSAStartup will return %x\n", ret);
        MACTlog("*WSAStartup(%x,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 14) {
            ret = bReturnValue;
            MACTlog("+WSAStartup modified to return %x\n", ret);
        }
    }

    return ret;
}

HANDLE WINAPI MyCreateFileMappingA(HANDLE                a0,
                                   LPSECURITY_ATTRIBUTES a1,
                                   DWORD                 a2,
                                   DWORD                 a3,
                                   DWORD                 a4,
                                   LPCSTR                a5)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueCreateFileMappingA(a0, a1, a2, a3, a4, a5);

    MACTlog(":CreateFileMappingA(%p,%p,%p,%p,%p,%p)\n", a0, a1, a2, a3, a4, a5);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = FALSE;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = 0;
```

```
    __try {
        ret = TrueCreateFileMappingA(a0, a1, a2, a3, a4, a5);
    } __finally {
        MACTlog("-CreateFileMappingA will return %p\n", ret);
        MACTlog("*CreateFileMappingA(%p,%p,%p,%p,%p,%p)(%p,%x,%p)", a0, a1, a2, a3, a4, a5, ret,
iOption, bReturnValue);

        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+CreateFileMappingA modified to return %p\n", ret);
        }
    }

    return ret;
}

BOOL WINAPI MyIsNTAdmin(DWORD  a0,
                        DWORD *a1)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueIsNTAdmin(a0, a1);

    MACTlog(":IsNTAdmin(%p,%p)\n", a0, a1);

    BOOL iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_LONG;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueIsNTAdmin(a0, a1);
    } __finally {
        MACTlog("-IsNTAdmin will return %x\n", ret);
        MACTlog("*IsNTAdmin(%p,%p),(%x,%x,%x)", a0, a1, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+IsNTAdmin modified to return %x\n", ret);
        }
    }

    return ret;
}
```

```
BOOL WINAPI MyIsUserAnAdmin(void)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueIsUserAnAdmin();

    MACTlog(":IsUserAnAdmin(void)\n");

    BOOL iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_LONG;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueIsUserAnAdmin();
    } __finally {
        MACTlog("-IsUserAnAdmin will return %x\n", ret);
        MACTlog("*IsUserAnAdmin(void),(%x,%x,%x)", ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+IsUserAnAdmin modified to return %x\n", ret);
        }
    }

    return ret;
}

HMODULE WINAPI MyLoadLibrary(LPCTSTR a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueLoadLibrary(a0);

    int la;
    if(a0 == NULL)
        la = 4;
    else
        la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
```

```
    if(a0 == NULL)
        strncpy(buffer0, "NULL", la);
    else
        strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":LoadLibrary(%s)\n", buffer0);

    int iOption = MACTmain("HMODULE");
    HMODULE bReturnValue = NULL;
    if(iOption == 16) {
        bReturnValue = SR_HMODULE;
    }

    HMODULE ret;
    __try {
        ret = TrueLoadLibrary(a0);
    } __finally {
        MACTlog("-LoadLibrary will return (void)\n");
        MACTlog("*LoadLibrary(%s),(%p,%x,%p)", buffer0, ret, iOption, bReturnValue);
        if(iOption == 16) {
            ret = bReturnValue;
            MACTlog("+LoadLibrary modified to return %p\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
}

HMODULE WINAPI MyLoadLibraryExA(LPCSTR a0,
                                HANDLE a1,
                                DWORD  a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return TrueLoadLibraryExA(a0, a1, a2);

    int la;
    if(a0 == NULL)
        la = 4;
    else
        la = lstrlenA(a0);
    char *buffer0 = new char[la+1];
    if(a0 == NULL)
```

```
        strncpy(buffer0, "NULL", la);
    else
        strncpy(buffer0, a0, la);
    buffer0[la] = '\0';

    MACTlog(":LoadLibraryExA(%s,%p,%p)\n", buffer0, a1, a2);

    int iOption = MACTmain("HMODULE");
    HMODULE bReturnValue = NULL;
    if(iOption == 16) {
        bReturnValue = SR_HMODULE;
    }

    HMODULE ret;
    __try {
        ret = TrueLoadLibraryExA(a0, a1, a2);
    } __finally {
        MACTlog("-LoadLibraryExA will return (void)\n");
        MACTlog("*LoadLibraryExA(%s,%p,%p),(%p,%x,%p)",   buffer0,   a1,   a2,   ret,   iOption,
bReturnValue);
        if(iOption == 16) {
            ret = bReturnValue;
            MACTlog("+LoadLibraryExA modified to return %p\n", ret);
        }
        delete[] buffer0;
    }

    return ret;
}

HWND WINAPI MyGetConsoleWindow(void)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED || !MACTVERBOSE)
        return NULL;

    MACTlog(":GetConsoleWindow(void)\n");


    MACTlog("-GetConsoleWindow will return NULL\n");
    MACTlog("*GetConsoleWindow(void),(0,0,NULL)");
    MACTlog("+GetConsoleWindow modified to return NULL\n");

    return NULL;
}
```

```
BOOL WINAPI MySetProcessDEPPolicy(DWORD a0)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueSetProcessDEPPolicy(a0);

    MACTlog(":SetProcessDEPPolicy(%x)\n", a0);

    BOOL iOption = MACTmain("BOOL");
    BOOL bReturnValue = FALSE;
    if(iOption == 1) {
        bReturnValue = SR_LONG;
    }

    BOOL ret = FALSE;
    __try {
        ret = TrueSetProcessDEPPolicy(a0);
    } __finally {
        MACTlog("-SetProcessDEPPolicy will return %x\n", ret);
        MACTlog("*SetProcessDEPPolicy(%x),(%x,%x,%x)", a0, ret, iOption, bReturnValue);
        if(iOption == 1) {
            ret = bReturnValue;
            MACTlog("+SetProcessDEPPolicy modified to return %x\n", ret);
        }
    }

    return ret;
}

INT WINAPI MyWSASend(SOCKET                            a0,
                     LPWSABUF                          a1,
                     DWORD                             a2,
                     LPDWORD                           a3,
                     DWORD                             a4,
                     LPWSAOVERLAPPED                   a5,
                     LPWSAOVERLAPPED_COMPLETION_ROUTINE a6)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueWSASend(a0, a1, a2, a3, a4, a5, a6);

    MACTlog(":WSASend(%p,%p,%lu,%p,%lu,%p,%p)\n", a0, a1, a2, a3, a4, a5, a6);
```

```
    u_long imaxbuf = 0;
    for(int x = 0; x < a2; ++x)
        imaxbuf = max(imaxbuf, a1[x].len);

    char *buffer = new char[imaxbuf+1];
    for(int x = 0; x < a2; ++x) {
        snprintf(buffer, a1[x].len, "%s", a1[x].buf);
        MACTcomm(">WSASend:\n%s\n", buffer);
    }

    INT ret = FALSE;
    __try {
        ret = TrueWSASend(a0, a1, a2, a3, a4, a5, a6);
    } __finally {
        MACTlog("-WSASend will return %x\n", ret);
        MACTlog("*WSASend(%p,%p,%lu,%p,%lu,%p,%p),(%p,0,0)", a0, a1, a2, a3, a4, a5, a6, ret);
        delete[] buffer;
    }

    return ret;
}

HANDLE WINAPI MyHeapCreate(DWORD  a0,
                           SIZE_T a1,
                           SIZE_T a2)
{
    if(MACTDEBUG2)
       MACTPrint(">>DEBUG Function: %s\n", __FUNCTION__);

    if(!MACTSTARTED)
        return TrueHeapCreate(a0, a1, a2);

    MACTlog(":HeapCreate(%x,%d,%d)\n", a0, a1, a2);

    int iOption = MACTmain("HANDLE");
    HANDLE bReturnValue = FALSE;
    if(iOption == 0) {
        bReturnValue = SR_HANDLE;
    }

    HANDLE ret = FALSE;
    __try {
        ret = TrueHeapCreate(a0, a1, a2);
    } __finally {
        MACTlog("-HeapCreate will return %p\n", ret);
        MACTlog("*HeapCreate(%x,%d,%d),(%p,%x,%p)", a0, a1, a2, ret, iOption, bReturnValue);
```

```
        if(iOption == 0) {
            ret = bReturnValue;
            MACTlog("+HeapCreate modified to return %x\n", ret);
        }
    }

    return ret;
}

//*************************************************************************************************
//
// Function   : MACTGetSocket
// Description: Establishes the connection to the socket for server communications
//
//*************************************************************************************************
BOOL MACTGetSocket(u_short uPort)
{
    WSADATA WsaDat;
    if(TrueWSAStartup(MAKEWORD(2,2),&WsaDat)!=0) {
        printf("Winsock error - Winsock initialization failed\r\n");
        WSACleanup();
        return FALSE;
    }

    // Create our socket
    ConnectSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    if(ConnectSocket==INVALID_SOCKET)
    {
        printf("Winsock error - Socket creation Failed!\r\n");
        WSACleanup();
        return FALSE;
    }

    // Resolve IP address for hostname
    struct hostent *host;
//    if((host=gethostbyname("localhost"))==NULL)
    if((host=Truegethostbyname("localhost"))==NULL)
    {
        printf("Failed to resolve hostname.\r\n");
        WSACleanup();
        return FALSE;
    }

    // Setup our socket address structure
    SOCKADDR_IN SockAddr;
    SockAddr.sin_port=htons(uPort);
    SockAddr.sin_family=AF_INET;
```

```
    SockAddr.sin_addr.s_addr=*((unsigned long*)host->h_addr);

    // Attempt to connect to server
    if(Trueconnect(ConnectSocket,(SOCKADDR*)(&SockAddr),sizeof(SockAddr))!=0)
    {
        printf("Failed to establish connection with server\r\n");
        WSACleanup();
//        system("PAUSE");
        return FALSE;
    }
    else
        printf("Connected to server.\n");

    // If iMode!=0, non-blocking mode is enabled.
    u_long iMode=1;
    ioctlsocket(ConnectSocket, FIONBIO, &iMode);

    return TRUE;
}

//*************************************************************************************************
//
// Function   : MACTSocketPrint
// Description: Send data through the socket connection to the server.
//
//*************************************************************************************************
BOOL MACTSocketPrint(char *szMessage)
{

        ++MACTMSGS;

        MACTSEND = TRUE;

        fd_set WriteFDs;
        FD_ZERO(&WriteFDs);
        FD_SET(ConnectSocket, &WriteFDs);

        char buffer[512];
        memset(buffer, 0, 512);
        strncpy(buffer, szMessage, strlen(szMessage));
        buffer[511] = '\0';

        if(select(0, NULL, &WriteFDs, NULL, 0) > 0) {
            if (FD_ISSET(ConnectSocket, &WriteFDs)) {
                int iBytesSent = Truesend(ConnectSocket, buffer, 512, 0);
                if(iBytesSent < 0) {
                    int nError=WSAGetLastError();
```

```
                    printf("Winsock error code: %d\r\n", nError);
                    printf("Server disconnected!\r\n");
                    shutdown(ConnectSocket,SD_SEND);
                    closesocket(ConnectSocket);
                    exit(0);
                }
            }
            else {
                printf("Error Write FD_ISSET\n");
            }
        }
        else {
            printf("Error on select write.\n");
            int nError=WSAGetLastError();
            printf("Winsock error code: %d\r\n", nError);
            printf("Server disconnected!\r\n");
            shutdown(ConnectSocket,SD_SEND);
            closesocket(ConnectSocket);
            exit(0);
        }

        MACTSEND = FALSE;
        --MACTMSGS;

        return TRUE;
}

//*************************************************************************************************
//
// Function   : MACTExecuteCommand
// Description: Execute the users command sent from the server.
//
//*************************************************************************************************
INT MACTExecuteCommand(char* buffer, char* sType)
{
    int iRet = 100;

    if(strncmp(buffer, "CE", 2) == 0) {
        MACTFINISH = TRUE;
    } else if(strncmp(buffer, "C\0", 2) == 0 ) {
        (void)0;
    } else if(strncmp(buffer, "DC", 2) == 0) {
        MACTPrint(">Displaying Commands mact.cpp.\n");
    } else if(strncmp(buffer, "DM", 2) == 0) {
        std::string sAddress = buffer;
        sAddress = sAddress.substr(2, 8);
        std::string sLength = buffer;
```

```
        sLength  = sLength.substr(10, 8);
        MACTDisplayMemory((LPVOID)std::stoi(sAddress, NULL, 16), std::stoi(sLength, NULL, 16));
    } else if(strncmp(buffer, "DS", 2) == 0) {
        MACTDisplayMemoryConstruct();
    } else if(strncmp(buffer, "BA", 2) == 0) {
        std::string sBreakpoint;
        sBreakpoint = buffer;
        MACTAddBreakpoint(sBreakpoint.substr(2, sBreakpoint.length() - 2));
    } else if(strncmp(buffer, "BC", 2) == 0) {
        MACTClearBreakpoint();
    } else if(strncmp(buffer, "BD", 2) == 0) {
        std::string sBreakpoint;
        sBreakpoint = buffer;
        MACTDeleteBreakpoint(sBreakpoint.substr(2, sBreakpoint.length() - 2));
    } else if(strncmp(buffer, "BL", 2) == 0) {
        MACTListBreakpoint();
    } else if(strncmp(buffer, "MA", 2) == 0) {
        std::string sAddress = buffer;
        sAddress = sAddress.substr(2, 8);
        std::string sLength = buffer;
        sLength  = sLength.substr(10, 8);
        MACTSaveFromAddress((LPVOID)std::stoi(sAddress, NULL, 16), std::stoi(sLength, NULL, 16),
5);
    } else if(strncmp(buffer, "SR", 2) == 0) {
        std::string sHex;
        sHex = buffer;
        sHex = sHex.substr(2, sHex.length() - 2);
        iRet = MACTSubRetVal(sType, std::stoi(sHex, NULL, 16));
    }
    else
        MACTPrint(">Invalid command.\n");

    return iRet;

}

//*****************************************************************************************
//
// Function   : MACTReceive
// Description: Receive data from the server.
//
//*****************************************************************************************
static INT MACTReceive(char * sType)
{

    int iRet = 100;
```

```
if(MACTFINISH) {
    return iRet;
}

char buffer[80];
memset(buffer, 0, 80);


MACTBP = TRUE;

while(buffer[0] != 'C') {
    MACTPrint("MACTSTART\n");

    fd_set ReadFDs;
    FD_ZERO(&ReadFDs);
    FD_SET(ConnectSocket, &ReadFDs);

    memset(buffer, 0, 80);

    if(select(0, &ReadFDs, NULL, NULL, 0) > 0) {
        if (FD_ISSET(ConnectSocket, &ReadFDs)) {
            memset(buffer, 0, 80);
            int inDataLength = Truerecv(ConnectSocket, buffer, 80, 0);
            if(inDataLength > 0) {
                buffer[79] = '\0';
                if(MACTDEBUG) {
                    printf("Debug: Received %d bytes.\n", inDataLength);
                    printf("Debug: Received from server: <%s>\n", buffer);
                }
                iRet = MACTExecuteCommand(buffer, sType);
            }
        }
        else {
            MACTPrint(">Error FD_ISSET\n");
        }
    }
    else {
        MACTPrint(">Error on select.\n");
    }

    switch(buffer[0]) {
        case 'S' :
            MACTBP = FALSE;
            return iRet;
            break;
        case 'Q' :
            exit(0);
```

```
                break;
            }
        }


    MACTBP = FALSE;
    return iRet;
}

//*************************************************************************************************
//
// Function   : MACTPrint
// Description: Send a message to the server.
//
//*************************************************************************************************
BOOL MACTPrint(const CHAR *psz, ...)
{

    if(MACTDEBUG)
        printf(">DEBUG: MACTPrint\n");

    va_list args;
    va_start(args, psz);

    int     len;
    char    *buffer;

    len = _vscprintf(psz, args) + 1;

    buffer = (char*)malloc( len * sizeof(char) );
    vsprintf(buffer, psz, args);

    MACTSocketPrint(buffer);

    if(MACTFINISH && (buffer[0] == ':')) {
        std::string sbuffer = buffer;
        sbuffer = sbuffer.substr(1, sbuffer.length() - 1);
        sbuffer = sbuffer.substr(0, sbuffer.find('(', 0));
        std::transform(sbuffer.begin(), sbuffer.end(), sbuffer.begin(), toupper);
        for(int i = 0; i < MACTBreakpointCount; ++i)
            if(sbuffer == MACTBreakpoints[i]) {
                MACTFINISH = FALSE;
                break;
            }
    }

    vprintf(psz, args);
```

```
    va_end(args);

    free(buffer);

    return TRUE;

}

//*************************************************************************************************
//
// Function   : MACTCreateThread
// Description: Create a thread for monitoring memory allocations.
//
//*************************************************************************************************
void MACTCreateThread(LPVOID buffer, size_t msize, int interval, int imemtype)
{
    pDataArray[THREADCOUNT]    =    (PMEMDATA)    HeapAlloc(GetProcessHeap(),    HEAP_ZERO_MEMORY,
sizeof(MEMDATA));

    if( pDataArray[THREADCOUNT] == NULL )
        ExitProcess(2);

    pDataArray[THREADCOUNT]->Mem_address   = buffer;
    pDataArray[THREADCOUNT]->Mem_size      = msize;
    pDataArray[THREADCOUNT]->Mem_interval  = interval;
    pDataArray[THREADCOUNT]->Mem_type      = imemtype;

    hThreadArray[THREADCOUNT] = CreateThread(NULL, 0, MACTMemThread, pDataArray[THREADCOUNT], 0,
&dwThreadIdArray[THREADCOUNT]);

    if (hThreadArray[THREADCOUNT] == NULL)
           ExitProcess(3);

    ++THREADCOUNT;
}

//*************************************************************************************************
//
// Function   : MACTLoadTicks
// Description: Load GetTickCount substitution values from file.
//
//*************************************************************************************************
int MACTLoadTicks()
{
    using namespace std;

    string line;
```

```
    ifstream myfile ("getticks.txt");

    if(myfile.is_open()) {
        while(getline(myfile, line))
          vTicks.push_back(stoi(line, NULL, 16));
        myfile.close();
    } else
        return 0;

    int iGetTickCount = 0;
    for(vector<int>::iterator it = vTicks.begin(); it != vTicks.end(); ++it) {
        cout << *it << endl;
        ++iGetTickCount;
    }

    return iGetTickCount;
}

//*************************************************************************************************
//
// Function   : MACTLoadTicks64
// Description: Load GetTickCount64 substitution values from file.
//
//*************************************************************************************************
int MACTLoadTicks64()
{
    using namespace std;

    string line;
    ifstream myfile ("getticks64.txt");

    if(myfile.is_open()) {
        while(getline(myfile, line))
          vTicks64.push_back(stoi(line, NULL, 16));
        myfile.close();
    } else
        return 0;

    int iGetTickCount = 0;
    for(vector<int>::iterator it = vTicks64.begin(); it != vTicks64.end(); ++it) {
        cout << *it << endl;
        ++iGetTickCount;
    }

    return iGetTickCount;
}
```

```
//**************************************************************************************************
//
// Function   : MACTLoadQPC
// Description: Load QueryPerformanceCounter substitution values from file.
//
//**************************************************************************************************
int MACTLoadQPC()
{
    using namespace std;

    string line;
    ifstream myfile ("qpc.txt");

    if(myfile.is_open()) {
        printf("File is open.");
        LARGE_INTEGER liTemp;
        while(getline(myfile, line)) {
            liTemp.QuadPart = stoi(line, NULL, 16);
            vQPC.push_back(liTemp);
        }
        myfile.close();
    } else
        return 0;

    int iQPCCount = 0;
    for(vector<LARGE_INTEGER>::iterator it = vQPC.begin(); it != vQPC.end(); ++it) {
        ++iQPCCount;
    }

    return iQPCCount;
}

//**************************************************************************************************
//
// Function   : DllMain
// Description: Verify, Attach and Detach functions.
//
//**************************************************************************************************
BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
{
    LONG error;
    (void)hinst;
    (void)reserved;

    if (DetourIsHelperProcess()) {
        return TRUE;
```

```
}

if (dwReason == DLL_PROCESS_ATTACH) {

    if (!fLog) {
        CreateLogDir();
    }

    DetourRestoreAfterWith();


    if(!MACTGetSocket(27015))
        if(!MACTGetSocket(27014))
            MACTGetSocket(27013);

    std::string sStart = "MACTINIT " + MACTdir;
    MACTPrint(sStart.c_str());

    MACTPrint(">Starting...\n");

    MACTTICKCOUNT = MACTLoadTicks();
    MACTPrint(">MACTTICKCOUNT   = %d\n", MACTTICKCOUNT);
    MACTTICKCOUNT64 = MACTLoadTicks64();
    MACTPrint(">MACTTICKCOUNT64 = %d\n", MACTTICKCOUNT64);
    MACTQPCCOUNT = MACTLoadQPC();
    MACTPrint(">MACTQPCCOUNT = %d\n", MACTQPCCOUNT);

    MACTPrint(">mact" DETOURS_STRINGIFY(DETOURS_BITS) ".dll: "
           " Starting.\n");
    PVOID pbExeEntry = DetourGetEntryPoint(NULL);
    PVOID pbDllEntry = DetourGetEntryPoint(hinst);
    MACTPrint(">mact" DETOURS_STRINGIFY(DETOURS_BITS) ".dll: "
           " ExeEntry=%p, DllEntry=%p\n", pbExeEntry, pbDllEntry);

    Verify("GetTickCount", (PVOID)GetTickCount);
    Verify("GetTickCount64", (PVOID)GetTickCount64);
    Verify("QueryPerformanceCounter", (PVOID)QueryPerformanceCounter);
    Verify("Sleep", (PVOID)Sleep);
    Verify("SleepEx", (PVOID)SleepEx);
    Verify("lstrcmpiA", (PVOID)lstrcmpiA);
    Verify("lstrcmpiW", (PVOID)lstrcmpiW);
    Verify("lstrcmpW", (PVOID)lstrcmpW);
    Verify("CompareStringEx", (PVOID)CompareStringEx);
    Verify("CreateFileW", (HANDLE)CreateFileW);
    Verify("CreateFileA", (HANDLE)CreateFileA);
    Verify("GetFileSize", (PVOID)GetFileSize);
    Verify("WriteFile", (PVOID)WriteFile);
```

```
Verify("WriteFileEx", (PVOID)WriteFileEx);
Verify("FlushFileBuffers", (PVOID)FlushFileBuffers);
Verify("CloseHandle", (PVOID)CloseHandle);
Verify("CopyFileA", (PVOID)CopyFileA);
Verify("CopyFileW", (PVOID)CopyFileW);
Verify("CopyFileExA", (PVOID)CopyFileExA);
Verify("CopyFileExW", (PVOID)CopyFileExW);
Verify("DeleteFileA", (PVOID)DeleteFileA);
Verify("DeleteFileW", (PVOID)DeleteFileW);
Verify("VirtualAlloc", (LPVOID)VirtualAlloc);
Verify("VirtualAllocEx", (LPVOID)VirtualAllocEx);
Verify("VirtualProtect", (PVOID)VirtualProtect);
Verify("VirtualProtectEx", (PVOID)VirtualProtectEx);
Verify("VirtualFree", (PVOID)VirtualFree);
Verify("VirtualFreeEx", (PVOID)VirtualFreeEx);
Verify("WinExec", (PVOID)WinExec);
Verify("ShellExecuteW", (PVOID)ShellExecuteW);
Verify("ShellExecuteExA", (PVOID)ShellExecuteExA);
Verify("ShellExecuteExW", (PVOID)ShellExecuteExW);
Verify("RegGetValueA", (PVOID)RegGetValueA);
Verify("RegGetValueW", (PVOID)RegGetValueW);
Verify("RegQueryValueEx", (PVOID)RegQueryValueEx);
Verify("RegOpenKeyEx", (PVOID)RegOpenKeyEx);
Verify("RegSetValueA", (PVOID)RegSetValueA);
Verify("RegSetValueEx", (PVOID)RegSetValueEx);
Verify("RegSetValueExW", (PVOID)RegSetValueExW);
Verify("RegEnumKeyExA", (PVOID)RegEnumKeyExA);
Verify("RegEnumKeyExW", (PVOID)RegEnumKeyExW);
Verify("RegCreateKeyEx", (PVOID)RegCreateKeyEx);
Verify("AdjustTokenPrivileges", (PVOID)AdjustTokenPrivileges);
Verify("AttachThreadInput", (PVOID)AttachThreadInput);
Verify("BitBlt", (PVOID)BitBlt);
Verify("CertOpenSystemStore", (LPVOID)CertOpenSystemStore);
Verify("ControlService", (PVOID)ControlService);
Verify("CreateMutex", (HANDLE)CreateMutex);
Verify("CreateMuteEx", (HANDLE)CreateMutexEx);
Verify("CreateProcess", (PVOID)CreateProcess);
Verify("CreateProcessW", (PVOID)CreateProcessW);
Verify("TerminateProcess", (PVOID)TerminateProcess);
Verify("CreateRemoteThread", (HANDLE)CreateRemoteThread);
Verify("CreateRemoteThreadEx", (HANDLE)CreateRemoteThreadEx);
Verify("CreateService", (HANDLE)CreateService);
Verify("CreateToolhelp32Snapshot", (HANDLE)CreateToolhelp32Snapshot);
Verify("CryptAcquireContextA", (PVOID)CryptAcquireContextA);
Verify("CryptAcquireContextW", (PVOID)CryptAcquireContextW);
Verify("DeviceIoControl", (PVOID)DeviceIoControl);
Verify("EnumProcesses", (PVOID)EnumProcesses);
```

```
        Verify("EnumProcessModules", (PVOID)EnumProcessModules);
        Verify("EnumProcessModulesEx", (PVOID)EnumProcessModulesEx);
        Verify("FindFirstFile", (HANDLE)FindFirstFile);
        Verify("FindFirstFileEx", (HANDLE)FindFirstFileEx);
        Verify("FindNextFile", (PVOID)FindNextFile);
        Verify("FindResourceA", (HANDLE)FindResourceA);
        Verify("FindResourceExA", (HANDLE)FindResourceExA);
        Verify("FindWindow", (HANDLE)FindWindow);
        Verify("FindWindowEx", (HANDLE)FindWindowEx);
        Verify("FtpOpenFileW", (HANDLE)FtpOpenFileW);
        Verify("FtpPutFile", (PVOID)FtpPutFile);
        Verify("GetAdaptersInfo", (PVOID)GetAdaptersInfo);
        Verify("GetAsyncKeyState", (PVOID)GetAsyncKeyState);
        Verify("GetDC", (PVOID)GetDC);
        Verify("GetForegroundWindow", (PVOID)GetForegroundWindow);
        Verify("GetWindowText", (PVOID)GetWindowText);
        Verify("gethostbyname", (PVOID)gethostbyname);
        Verify("getaddrinfo", (PVOID)getaddrinfo);
        Verify("gethostname", (PVOID)gethostname);
        Verify("GetKeyState", (PVOID)GetKeyState);
        Verify("GetModuleFileName", (PVOID)GetModuleFileName);
        Verify("GetModuleFileNameExA", (PVOID)GetModuleFileNameExA);
        Verify("GetModuleFileNameExW", (PVOID)GetModuleFileNameExW);
        Verify("GetModuleHandle", (HANDLE)GetModuleHandle);
        Verify("GetModuleHandle", (PVOID)GetModuleHandleEx);
        Verify("GetProcAddress", (PVOID)GetProcAddress);
        Verify("GetStartupInfoA", (PVOID)GetStartupInfoA);
        Verify("GetSystemDefaultLangID", (PVOID)GetSystemDefaultLangID);
        Verify("GetTempPathA", (PVOID)GetTempPathA);
        Verify("GetThreadContext", (PVOID)GetThreadContext);
        Verify("GetVersionEx", (PVOID)GetVersionEx);
        Verify("GetWindowsDirectory", (PVOID)GetWindowsDirectory);
        Verify("inet_addr", (PVOID)inet_addr);
        Verify("InternetOpen", (HANDLE)InternetOpen);
        Verify("InternetOpenW", (HANDLE)InternetOpenW);
        Verify("InternetConnectW", (HANDLE)InternetConnectW);
        Verify("HttpOpenRequestW", (HANDLE)HttpOpenRequestW);
        Verify("HttpSendRequestW", (HANDLE)HttpSendRequestW);
        Verify("HttpSendRequestExW", (HANDLE)HttpSendRequestExW);
        Verify("InternetOpenUrl", (HANDLE)InternetOpenUrl);
        Verify("InternetOpenUrlA", (HANDLE)InternetOpenUrlA);
        Verify("InternetReadFile", (PVOID)InternetReadFile);
        Verify("InternetWriteFile", (PVOID)InternetWriteFile);
        Verify("IsWow64Process", (PVOID)IsWow64Process);
//        Verify("LdrLoadDll", (PVOID)LdrLoadDll);
        Verify("LoadResource", (HANDLE)LoadResource);
        Verify("LsaEnumerateLogonSessions", (PVOID)LsaEnumerateLogonSessions);
```

```
        Verify("MapViewOfFile", (PVOID)MapViewOfFile);
        Verify("MapViewOfFileEx", (PVOID)MapViewOfFileEx);
        Verify("MapVirtualKeyA", (PVOID)MapVirtualKeyA);
        Verify("MapVirtualKeyExA", (PVOID)MapVirtualKeyExA);
        Verify("MapVirtualKeyW", (PVOID)MapVirtualKeyW);
        Verify("MapVirtualKeyExW", (PVOID)MapVirtualKeyExW);
        Verify("Module32First", (PVOID)Module32First);
        Verify("Module32Next", (PVOID)Module32Next);
        Verify("OpenMutexA", (HANDLE)OpenMutexA);
        Verify("OpenProcess", (HANDLE)OpenProcess);
        Verify("OutputDebugString", (PVOID)OutputDebugString);
        Verify("OutputDebugStringA", (PVOID)OutputDebugStringA);
        Verify("OutputDebugStringW", (PVOID)OutputDebugStringW);
        Verify("PeekNamedPipe", (PVOID)PeekNamedPipe);
        Verify("Process32First", (PVOID)Process32First);
        Verify("Process32FirstW", (PVOID)Process32FirstW);
        Verify("Process32Next", (PVOID)Process32Next);
        Verify("Process32NextW", (PVOID)Process32NextW);
        Verify("QueueUserAPC", (PVOID)QueueUserAPC);
        Verify("ReadProcessMemory", (PVOID)ReadProcessMemory);
        Verify("RegisterHotKey", (PVOID)RegisterHotKey);
        Verify("RegOpenKeyA", (PVOID)RegOpenKeyA);
        Verify("RegOpenKeyExA", (PVOID)RegOpenKeyExA);
        Verify("RegOpenKeyExW", (PVOID)RegOpenKeyExW);
        Verify("ResumeThread", (PVOID)ResumeThread);
//        Verify("TrueSamIConnect", (LPVOID)TrueSamIConnect);
        Verify("TrueLdrLoadDll", (PVOID)TrueLdrLoadDll);
        Verify("TrueRtlCreateRegistryKey", (PVOID)TrueRtlCreateRegistryKey);
        Verify("TrueRtlWriteRegistryValue", (PVOID)TrueRtlWriteRegistryValue);
        Verify("SetFileTime", (PVOID)SetFileTime);
        Verify("SetThreadContext", (PVOID)SetThreadContext);
        Verify("SetWindowsHookEx", (PVOID)SetWindowsHookEx);

        Verify("TrueSfcTerminateWatcherThread", (PVOID)TrueSfcTerminateWatcherThread);

        Verify("StartServiceCtrlDispatcherA", (PVOID)StartServiceCtrlDispatcherA);
        Verify("SuspendThread", (PVOID)SuspendThread);
        Verify("system", (PVOID)system);
        Verify("_wsystem", (PVOID)_wsystem);
        Verify("Thread32First", (PVOID)Thread32First);
        Verify("Thread32Next", (PVOID)Thread32Next);
        Verify("Toolhelp32ReadProcessMemory", (PVOID)Toolhelp32ReadProcessMemory);
        Verify("URLDownloadToFile", (PVOID)URLDownloadToFile);
        Verify("URLDownloadToFileA", (PVOID)URLDownloadToFileA);
        Verify("WideCharToMultiByte", (PVOID)WideCharToMultiByte);
        Verify("WriteProcessMemory", (PVOID)WriteProcessMemory);
        Verify("accept", (PVOID)accept);
```

```
        Verify("bind", (PVOID)bind);
        Verify("connect", (PVOID)connect);
        Verify("ConnectNamedPipe", (PVOID)ConnectNamedPipe);
        Verify("recv", (PVOID)recv);
        Verify("send", (PVOID)send);
        Verify("WSAStartup", (PVOID)WSAStartup);
        Verify("CreateFileMappingA", (HANDLE)CreateFileMappingA);

        Verify("TrueIsNTAdmin", (PVOID)TrueIsNTAdmin);

        Verify("IsUserAnAdmin", (PVOID)IsUserAnAdmin);
        Verify("LoadLibrary", (PVOID)LoadLibrary);
        Verify("LoadLibraryExA", (PVOID)LoadLibraryExA);
        Verify("GetConsoleWindow", (PVOID)GetConsoleWindow);
        Verify("SetProcessDEPPolicy", (PVOID)SetProcessDEPPolicy);
        Verify("CoTaskMemAlloc", (PVOID)CoTaskMemAlloc);
        Verify("CoTaskMemFree", (PVOID)CoTaskMemFree);
        Verify("WSASend", (PVOID)WSASend);
        Verify("HeapCreate", (HANDLE)HeapCreate);

        fflush(stdout);

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());


        DetourAttach(&(PVOID&)TrueGetTickCount, MyGetTickCount);
        DetourAttach(&(PVOID&)TrueGetTickCount64, MyGetTickCount64);
        DetourAttach(&(PVOID&)TrueQueryPerformanceCounter, MyQueryPerformanceCounter);

        DetourAttach(&(PVOID&)TrueShellExecuteW, MyShellExecuteW);
        DetourAttach(&(PVOID&)TrueShellExecuteExA, MyShellExecuteExA);
        DetourAttach(&(PVOID&)TrueShellExecuteExW, MyShellExecuteExW);


        DetourAttach(&(PVOID&)TrueSleep, MySleep);
        DetourAttach(&(PVOID&)TrueSleepEx, MySleepEx);
        DetourAttach(&(PVOID&)TruelstrcmpiA, MylstrcmpiA);
        DetourAttach(&(PVOID&)TruelstrcmpiW, MylstrcmpiW);
        DetourAttach(&(PVOID&)TruelstrcmpW, MylstrcmpW);
        DetourAttach(&(PVOID&)TrueCompareStringEx, MyCompareStringEx);
//        DetourAttach(&(PVOID&)TrueVirtualProtect, MyVirtualProtect);
//        DetourAttach(&(PVOID&)TrueWriteFile, MyWriteFile);
        DetourAttach(&(HANDLE&)TrueCreateFileW, MyCreateFileW);
        DetourAttach(&(HANDLE&)TrueCreateFileA, MyCreateFileA);
        DetourAttach(&(PVOID&)TrueGetFileSize, MyGetFileSize);
 //       DetourAttach(&(LPVOID&)TrueWriteFileEx, MyWriteFileEx);
```

```
DetourAttach(&(PVOID&)TrueFlushFileBuffers, MyFlushFileBuffers);
DetourAttach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
DetourAttach(&(PVOID&)TrueCopyFileA, MyCopyFileA);
DetourAttach(&(PVOID&)TrueCopyFileW, MyCopyFileW);
DetourAttach(&(PVOID&)TrueCopyFileExA, MyCopyFileExA);
DetourAttach(&(PVOID&)TrueCopyFileExW, MyCopyFileExW);
DetourAttach(&(PVOID&)TrueDeleteFileA, MyDeleteFileA);
DetourAttach(&(PVOID&)TrueDeleteFileW, MyDeleteFileW);
DetourAttach(&(LPVOID&)TrueVirtualAlloc, MyVirtualAlloc);
DetourAttach(&(LPVOID&)TrueVirtualAllocEx, MyVirtualAllocEx);
DetourAttach(&(PVOID&)TrueVirtualProtectEx, MyVirtualProtectEx);
DetourAttach(&(PVOID&)TrueVirtualFree, MyVirtualFree);
DetourAttach(&(PVOID&)TrueVirtualFreeEx, MyVirtualFreeEx);
DetourAttach(&(PVOID&)TrueCoTaskMemAlloc, MyCoTaskMemAlloc);
DetourAttach(&(PVOID&)TrueCoTaskMemFree, MyCoTaskMemFree);
DetourAttach(&(PVOID&)TrueWinExec, MyWinExec);
DetourAttach(&(PVOID&)TrueRegGetValueA, MyRegGetValueA);
DetourAttach(&(PVOID&)TrueRegGetValueW, MyRegGetValueW);
DetourAttach(&(PVOID&)TrueRegQueryValueEx, MyRegQueryValueEx);
DetourAttach(&(PVOID&)TrueRegSetValueA, MyRegSetValueA);
DetourAttach(&(PVOID&)TrueRegSetValueEx, MyRegSetValueEx);
DetourAttach(&(PVOID&)TrueRegSetValueExW, MyRegSetValueExW);
DetourAttach(&(PVOID&)TrueRegEnumKeyExA, MyRegEnumKeyExA);
DetourAttach(&(PVOID&)TrueRegEnumKeyExW, MyRegEnumKeyExW);
DetourAttach(&(PVOID&)TrueRegOpenKeyEx, MyRegOpenKeyEx);
DetourAttach(&(PVOID&)TrueRegCreateKeyEx, MyRegCreateKeyEx);
DetourAttach(&(PVOID&)TrueAdjustTokenPrivileges, MyAdjustTokenPrivileges);
DetourAttach(&(PVOID&)TrueAttachThreadInput, MyAttachThreadInput);
DetourAttach(&(PVOID&)TrueBitBlt, MyBitBlt);
DetourAttach(&(LPVOID&)TrueCertOpenSystemStore, MyCertOpenSystemStore);
DetourAttach(&(PVOID&)TrueControlService, MyControlService);
DetourAttach(&(HANDLE&)TrueCreateMutex, MyCreateMutex);
DetourAttach(&(HANDLE&)TrueCreateMutexEx, MyCreateMutexEx);
DetourAttach(&(PVOID&)TrueCreateProcess, MyCreateProcess);
DetourAttach(&(PVOID&)TrueCreateProcessW, MyCreateProcessW);
DetourAttach(&(PVOID&)TrueTerminateProcess, MyTerminateProcess);
DetourAttach(&(HANDLE&)TrueCreateRemoteThread, MyCreateRemoteThread);
DetourAttach(&(HANDLE&)TrueCreateRemoteThreadEx, MyCreateRemoteThreadEx);
DetourAttach(&(HANDLE&)TrueCreateService, MyCreateService);
DetourAttach(&(HANDLE&)TrueCreateToolhelp32Snapshot, MyCreateToolhelp32Snapshot);
DetourAttach(&(PVOID&)TrueCryptAcquireContextA, MyCryptAcquireContextA);
DetourAttach(&(PVOID&)TrueCryptAcquireContextW, MyCryptAcquireContextW);
DetourAttach(&(PVOID&)TrueDeviceIoControl, MyDeviceIoControl);
DetourAttach(&(PVOID&)TrueEnumProcesses, MyEnumProcesses);
DetourAttach(&(PVOID&)TrueEnumProcessModules, MyEnumProcessModules);
DetourAttach(&(PVOID&)TrueEnumProcessModulesEx, MyEnumProcessModulesEx);
DetourAttach(&(HANDLE&)TrueFindFirstFile, MyFindFirstFile);
```

```
DetourAttach(&(HANDLE&)TrueFindFirstFileEx, MyFindFirstFileEx);
DetourAttach(&(PVOID&)TrueFindNextFile, MyFindNextFile);
DetourAttach(&(HANDLE&)TrueFindResourceA, MyFindResourceA);
DetourAttach(&(HANDLE&)TrueFindResourceExA, MyFindResourceExA);
DetourAttach(&(HANDLE&)TrueFindWindow, MyFindWindow);
DetourAttach(&(HANDLE&)TrueFindWindowEx, MyFindWindowEx);
DetourAttach(&(HANDLE&)TrueFtpOpenFileW, MyFtpOpenFileW);
DetourAttach(&(PVOID&)TrueFtpPutFile, MyFtpPutFile);
DetourAttach(&(PVOID&)TrueGetAdaptersInfo, MyGetAdaptersInfo);
DetourAttach(&(PVOID&)TrueGetAsyncKeyState, MyGetAsyncKeyState);
DetourAttach(&(PVOID&)TrueGetDC, MyGetDC);
DetourAttach(&(PVOID&)TrueGetForegroundWindow, MyGetForegroundWindow);
DetourAttach(&(PVOID&)TrueGetWindowText, MyGetWindowText);
DetourAttach(&(PVOID&)Truegethostbyname, Mygethostbyname);
DetourAttach(&(PVOID&)Truegetaddrinfo, Mygetaddrinfo);
DetourAttach(&(PVOID&)Truegethostname, Mygethostname);
DetourAttach(&(PVOID&)TrueGetModuleFileName, MyGetModuleFileName);
DetourAttach(&(PVOID&)TrueGetModuleFileNameExA, MyGetModuleFileNameExA);
DetourAttach(&(PVOID&)TrueGetModuleFileNameExW, MyGetModuleFileNameExW);
DetourAttach(&(HANDLE&)TrueGetModuleHandle, MyGetModuleHandle);
DetourAttach(&(PVOID&)TrueGetModuleHandleEx, MyGetModuleHandleEx);
DetourAttach(&(PVOID&)TrueGetProcAddress, MyGetProcAddress);
DetourAttach(&(PVOID&)TrueGetStartupInfoA, MyGetStartupInfoA);
DetourAttach(&(PVOID&)TrueGetSystemDefaultLangID, MyGetSystemDefaultLangID);
DetourAttach(&(PVOID&)TrueGetTempPathA, MyGetTempPathA);
DetourAttach(&(PVOID&)TrueGetThreadContext, MyGetThreadContext);
DetourAttach(&(PVOID&)TrueGetVersionEx, MyGetVersionEx);
DetourAttach(&(PVOID&)TrueGetWindowsDirectory, MyGetWindowsDirectory);
DetourAttach(&(PVOID&)Trueinet_addr, Myinet_addr);
DetourAttach(&(HANDLE&)TrueInternetOpen, MyInternetOpen);
DetourAttach(&(HANDLE&)TrueInternetOpenW, MyInternetOpenW);
DetourAttach(&(HANDLE&)TrueInternetConnectW, MyInternetConnectW);
DetourAttach(&(HANDLE&)TrueHttpOpenRequestW, MyHttpOpenRequestW);
DetourAttach(&(HANDLE&)TrueHttpSendRequestW, MyHttpSendRequestW);
DetourAttach(&(HANDLE&)TrueHttpSendRequestExW, MyHttpSendRequestExW);
DetourAttach(&(HANDLE&)TrueInternetOpenUrl, MyInternetOpenUrl);
DetourAttach(&(HANDLE&)TrueInternetOpenUrlA, MyInternetOpenUrlA);
DetourAttach(&(PVOID&)TrueInternetReadFile, MyInternetReadFile);
DetourAttach(&(PVOID&)TrueInternetWriteFile, MyInternetWriteFile);
DetourAttach(&(PVOID&)TrueIsWow64Process, MyIsWow64Process);
DetourAttach(&(PVOID&)TrueLdrLoadDll, MyLdrLoadDll);
DetourAttach(&(HANDLE&)TrueLoadResource, MyLoadResource);
DetourAttach(&(PVOID&)TrueRtlCreateRegistryKey, MyRtlCreateRegistryKey);
DetourAttach(&(PVOID&)TrueRtlWriteRegistryValue, MyRtlWriteRegistryValue);
DetourAttach(&(PVOID&)TrueLsaEnumerateLogonSessions, MyLsaEnumerateLogonSessions);
DetourAttach(&(PVOID&)TrueMapViewOfFile, MyMapViewOfFile);
DetourAttach(&(PVOID&)TrueMapViewOfFileEx, MyMapViewOfFileEx);
```

```
        DetourAttach(&(PVOID&)TrueMapVirtualKeyA, MyMapVirtualKeyA);
        DetourAttach(&(PVOID&)TrueMapVirtualKeyExA, MyMapVirtualKeyExA);
        DetourAttach(&(PVOID&)TrueMapVirtualKeyW, MyMapVirtualKeyW);
        DetourAttach(&(PVOID&)TrueMapVirtualKeyExW, MyMapVirtualKeyExW);
        DetourAttach(&(PVOID&)TrueModule32First, MyModule32First);
        DetourAttach(&(PVOID&)TrueModule32Next, MyModule32Next);
        DetourAttach(&(HANDLE&)TrueOpenMutexA, MyOpenMutexA);
        DetourAttach(&(HANDLE&)TrueOpenProcess, MyOpenProcess);
        DetourAttach(&(PVOID&)TrueOutputDebugString, MyOutputDebugString);
        DetourAttach(&(PVOID&)TrueOutputDebugStringA, MyOutputDebugStringA);
        DetourAttach(&(PVOID&)TrueOutputDebugStringW, MyOutputDebugStringW);
        DetourAttach(&(PVOID&)TruePeekNamedPipe, MyPeekNamedPipe);
        DetourAttach(&(PVOID&)TrueProcess32First, MyProcess32First);
        DetourAttach(&(PVOID&)TrueProcess32FirstW, MyProcess32FirstW);
        DetourAttach(&(PVOID&)TrueProcess32Next, MyProcess32Next);
        DetourAttach(&(PVOID&)TrueProcess32NextW, MyProcess32NextW);
        DetourAttach(&(PVOID&)TrueQueueUserAPC, MyQueueUserAPC);
        DetourAttach(&(PVOID&)TrueReadProcessMemory, MyReadProcessMemory);
        DetourAttach(&(PVOID&)TrueRegisterHotKey, MyRegisterHotKey);
        DetourAttach(&(PVOID&)TrueRegOpenKeyA, MyRegOpenKeyA);
        DetourAttach(&(PVOID&)TrueRegOpenKeyExA, MyRegOpenKeyExA);
        DetourAttach(&(PVOID&)TrueRegOpenKeyExW, MyRegOpenKeyExW);
        DetourAttach(&(PVOID&)TrueResumeThread, MyResumeThread);
        DetourAttach(&(PVOID&)TrueSetFileTime, MySetFileTime);
        DetourAttach(&(PVOID&)TrueSetThreadContext, MySetThreadContext);
        DetourAttach(&(PVOID&)TrueSetWindowsHookEx, MySetWindowsHookEx);
        DetourAttach(&(PVOID&)TrueSfcTerminateWatcherThread, MySfcTerminateWatcherThread);
        DetourAttach(&(PVOID&)TrueStartServiceCtrlDispatcherA, MyStartServiceCtrlDispatcherA);
        DetourAttach(&(PVOID&)TrueSuspendThread, MySuspendThread);
        DetourAttach(&(PVOID&)Truesystem, Mysystem);
        DetourAttach(&(PVOID&)True_wsystem, My_wsystem);
        DetourAttach(&(PVOID&)TrueThread32First, MyThread32First);
        DetourAttach(&(PVOID&)TrueThread32Next, MyThread32Next);
        DetourAttach(&(PVOID&)TrueToolhelp32ReadProcessMemory, MyToolhelp32ReadProcessMemory);
        DetourAttach(&(PVOID&)TrueURLDownloadToFile, MyURLDownloadToFile);
        DetourAttach(&(PVOID&)TrueURLDownloadToFileA, MyURLDownloadToFileA);
// Too much noise
//        DetourAttach(&(PVOID&)TrueGetKeyState, MyGetKeyState);
//        DetourAttach(&(PVOID&)TrueWideCharToMultiByte, MyWideCharToMultiByte);
        DetourAttach(&(PVOID&)TrueWriteProcessMemory, MyWriteProcessMemory);
        DetourAttach(&(PVOID&)Trueaccept, Myaccept);
        DetourAttach(&(PVOID&)Truebind, Mybind);
        DetourAttach(&(PVOID&)Trueconnect, Myconnect);
        DetourAttach(&(PVOID&)TrueConnectNamedPipe, MyConnectNamedPipe);
        DetourAttach(&(PVOID&)Truerecv, Myrecv);
        DetourAttach(&(PVOID&)Truesend, Mysend);
        DetourAttach(&(PVOID&)TrueWSAStartup, MyWSAStartup);
```

```
//        DetourAttach(&(HANDLE&)TrueCreateFileMappingA, MyCreateFileMappingA);
        DetourAttach(&(HANDLE&)TrueLoadLibrary, MyLoadLibrary);
        DetourAttach(&(HANDLE&)TrueLoadLibraryExA, MyLoadLibraryExA);
        DetourAttach(&(HANDLE&)TrueGetConsoleWindow, MyGetConsoleWindow);
        DetourAttach(&(HANDLE&)TrueSetProcessDEPPolicy, MySetProcessDEPPolicy);
        DetourAttach(&(PVOID&)TrueIsUserAnAdmin, MyIsUserAnAdmin);
        DetourAttach(&(PVOID&)TrueWSASend, MyWSASend);
        DetourAttach(&(HANDLE&)TrueHeapCreate, MyHeapCreate);
//       DetourAttach(&(LPVOID&)TrueSamIConnect, MySamIConnect);

       DetourAttach(&(PVOID&)TrueIsNTAdmin, MyIsNTAdmin);


        MACTPrint(">All attached!\n");
        error = DetourTransactionCommit();
        MACTSTARTED = TRUE;

        MACTReceive("START");

    }
    else if (dwReason == DLL_PROCESS_DETACH) {
//        MACTSocketPrint();
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)TrueGetTickCount, MyGetTickCount);
        DetourDetach(&(PVOID&)TrueGetTickCount64, MyGetTickCount64);
        DetourDetach(&(PVOID&)TrueQueryPerformanceCounter, MyQueryPerformanceCounter);
        DetourDetach(&(PVOID&)TrueSleep, MySleep);
        DetourDetach(&(PVOID&)TrueSleepEx, MySleepEx);
        DetourDetach(&(PVOID&)TruelstrcmpiA, MylstrcmpiA);
        DetourDetach(&(PVOID&)TruelstrcmpiW, MylstrcmpiW);
        DetourDetach(&(PVOID&)TruelstrcmpW, MylstrcmpW);
        DetourDetach(&(PVOID&)TrueCompareStringEx, MyCompareStringEx);
        DetourDetach(&(HANDLE&)TrueCreateFileW, MyCreateFileW);
        DetourDetach(&(HANDLE&)TrueCreateFileA, MyCreateFileA);
        DetourDetach(&(PVOID&)TrueGetFileSize, MyGetFileSize);
        DetourDetach(&(PVOID&)TrueWriteFile, MyWriteFile);
        DetourDetach(&(PVOID&)TrueWriteFileEx, MyWriteFileEx);
        DetourDetach(&(PVOID&)TrueFlushFileBuffers, MyFlushFileBuffers);
        DetourDetach(&(PVOID&)TrueCloseHandle, MyCloseHandle);
        DetourDetach(&(PVOID&)TrueCopyFileA, MyCopyFileA);
        DetourDetach(&(PVOID&)TrueCopyFileW, MyCopyFileW);
        DetourDetach(&(PVOID&)TrueCopyFileExA, MyCopyFileExA);
        DetourDetach(&(PVOID&)TrueCopyFileExW, MyCopyFileExW);
        DetourDetach(&(PVOID&)TrueDeleteFileA, MyDeleteFileA);
        DetourDetach(&(PVOID&)TrueDeleteFileW, MyDeleteFileW);
        DetourDetach(&(LPVOID&)TrueVirtualAlloc, MyVirtualAlloc);
```

```
DetourDetach(&(LPVOID&)TrueVirtualAllocEx, MyVirtualAllocEx);
DetourDetach(&(PVOID&)TrueVirtualProtect, MyVirtualProtect);
DetourDetach(&(PVOID&)TrueVirtualProtectEx, MyVirtualProtectEx);
DetourDetach(&(PVOID&)TrueVirtualFree, MyVirtualFree);
DetourDetach(&(PVOID&)TrueVirtualFreeEx, MyVirtualFreeEx);
DetourDetach(&(PVOID&)TrueWinExec, MyWinExec);
DetourDetach(&(PVOID&)TrueShellExecuteW, MyShellExecuteW);
DetourDetach(&(PVOID&)TrueShellExecuteExA, MyShellExecuteExA);
DetourDetach(&(PVOID&)TrueShellExecuteExW, MyShellExecuteExW);
DetourDetach(&(PVOID&)TrueRegGetValueA, MyRegGetValueA);
DetourDetach(&(PVOID&)TrueRegGetValueW, MyRegGetValueW);
DetourDetach(&(PVOID&)TrueRegQueryValueEx, MyRegQueryValueEx);
DetourDetach(&(PVOID&)TrueRegSetValueA, MyRegSetValueA);
DetourDetach(&(PVOID&)TrueRegSetValueEx, MyRegSetValueEx);
DetourDetach(&(PVOID&)TrueRegSetValueExW, MyRegSetValueExW);
DetourDetach(&(PVOID&)TrueRegEnumKeyExA, MyRegEnumKeyExA);
DetourDetach(&(PVOID&)TrueRegEnumKeyExW, MyRegEnumKeyExW);
DetourDetach(&(PVOID&)TrueRegOpenKeyEx, MyRegOpenKeyEx);
DetourDetach(&(PVOID&)TrueRegCreateKeyEx, MyRegCreateKeyEx);
DetourDetach(&(PVOID&)TrueAdjustTokenPrivileges, MyAdjustTokenPrivileges);
DetourDetach(&(PVOID&)TrueAttachThreadInput, MyAttachThreadInput);
DetourDetach(&(PVOID&)TrueBitBlt, MyBitBlt);
DetourDetach(&(LPVOID&)TrueCertOpenSystemStore, MyCertOpenSystemStore);
DetourDetach(&(PVOID&)TrueControlService, MyControlService);
DetourDetach(&(HANDLE&)TrueCreateMutex, MyCreateMutex);
DetourDetach(&(HANDLE&)TrueCreateMutexEx, MyCreateMutexEx);
DetourDetach(&(PVOID&)TrueCreateProcess, MyCreateProcess);
DetourDetach(&(PVOID&)TrueCreateProcessW, MyCreateProcessW);
DetourDetach(&(PVOID&)TrueTerminateProcess, MyTerminateProcess);
DetourDetach(&(HANDLE&)TrueCreateRemoteThread, MyCreateRemoteThread);
DetourDetach(&(HANDLE&)TrueCreateRemoteThreadEx, MyCreateRemoteThreadEx);
DetourDetach(&(HANDLE&)TrueCreateService, MyCreateService);
DetourDetach(&(HANDLE&)TrueCreateToolhelp32Snapshot, MyCreateToolhelp32Snapshot);
DetourDetach(&(PVOID&)TrueCryptAcquireContextA, MyCryptAcquireContextA);
DetourDetach(&(PVOID&)TrueCryptAcquireContextW, MyCryptAcquireContextW);;
DetourDetach(&(PVOID&)TrueDeviceIoControl, MyDeviceIoControl);
DetourDetach(&(PVOID&)TrueEnumProcesses, MyEnumProcesses);
DetourDetach(&(PVOID&)TrueEnumProcessModules, MyEnumProcessModules);
DetourDetach(&(PVOID&)TrueEnumProcessModulesEx, MyEnumProcessModulesEx);
DetourDetach(&(HANDLE&)TrueFindFirstFile, MyFindFirstFile);
DetourDetach(&(HANDLE&)TrueFindFirstFileEx, MyFindFirstFileEx);
DetourDetach(&(PVOID&)TrueFindNextFile, MyFindNextFile);
DetourDetach(&(HANDLE&)TrueFindResourceA, MyFindResourceA);
DetourDetach(&(HANDLE&)TrueFindResourceExA, MyFindResourceExA);
DetourDetach(&(HANDLE&)TrueFindWindow, MyFindWindow);
DetourDetach(&(HANDLE&)TrueFindWindowEx, MyFindWindowEx);
DetourDetach(&(HANDLE&)TrueFtpOpenFileW, MyFtpOpenFileW);
```

```
DetourDetach(&(PVOID&)TrueFtpPutFile, MyFtpPutFile);
DetourDetach(&(PVOID&)TrueGetAdaptersInfo, MyGetAdaptersInfo);
DetourDetach(&(PVOID&)TrueGetAsyncKeyState, MyGetAsyncKeyState);
DetourDetach(&(PVOID&)TrueGetDC, MyGetDC);
DetourDetach(&(PVOID&)TrueGetForegroundWindow, MyGetForegroundWindow);
DetourDetach(&(PVOID&)TrueGetWindowText, MyGetWindowText);
DetourDetach(&(PVOID&)Truegethostbyname, Mygethostbyname);
DetourDetach(&(PVOID&)Truegetaddrinfo, Mygetaddrinfo);
DetourDetach(&(PVOID&)Truegethostname, Mygethostname);
DetourDetach(&(PVOID&)TrueGetKeyState, MyGetKeyState);
DetourDetach(&(PVOID&)TrueGetModuleFileName, MyGetModuleFileName);
DetourDetach(&(PVOID&)TrueGetModuleFileNameExA, MyGetModuleFileNameExA);
DetourDetach(&(PVOID&)TrueGetModuleFileNameExW, MyGetModuleFileNameExW);
DetourDetach(&(PVOID&)TrueGetModuleHandle, MyGetModuleHandle);
DetourDetach(&(PVOID&)TrueGetModuleHandleEx, MyGetModuleHandleEx);
DetourDetach(&(PVOID&)TrueGetProcAddress, MyGetProcAddress);
DetourDetach(&(PVOID&)TrueGetStartupInfoA, MyGetStartupInfoA);
DetourDetach(&(PVOID&)TrueGetSystemDefaultLangID, MyGetSystemDefaultLangID);
DetourDetach(&(PVOID&)TrueGetTempPathA, MyGetTempPathA);
DetourDetach(&(PVOID&)TrueGetThreadContext, MyGetThreadContext);
DetourDetach(&(PVOID&)TrueGetVersionEx, MyGetVersionEx);
DetourDetach(&(PVOID&)TrueGetWindowsDirectory, MyGetWindowsDirectory);
DetourDetach(&(PVOID&)Trueinet_addr, Myinet_addr);
DetourDetach(&(HANDLE&)TrueInternetOpen, MyInternetOpen);
DetourDetach(&(HANDLE&)TrueInternetOpenW, MyInternetOpenW);
DetourDetach(&(HANDLE&)TrueInternetConnectW, MyInternetConnectW);
DetourDetach(&(HANDLE&)TrueHttpOpenRequestW, MyHttpOpenRequestW);
DetourDetach(&(HANDLE&)TrueHttpSendRequestW, MyHttpSendRequestW);
DetourDetach(&(HANDLE&)TrueHttpSendRequestExW, MyHttpSendRequestExW);
DetourDetach(&(HANDLE&)TrueInternetOpenUrlA, MyInternetOpenUrlA);
DetourDetach(&(HANDLE&)TrueInternetOpenUrl, MyInternetOpenUrl);
DetourDetach(&(PVOID&)TrueInternetReadFile, MyInternetReadFile);
DetourDetach(&(PVOID&)TrueInternetWriteFile, MyInternetWriteFile);
DetourDetach(&(PVOID&)TrueIsWow64Process, MyIsWow64Process);
DetourDetach(&(PVOID&)TrueLdrLoadDll, MyLdrLoadDll);
DetourDetach(&(PVOID&)TrueRtlCreateRegistryKey, MyRtlCreateRegistryKey);
DetourDetach(&(PVOID&)TrueRtlWriteRegistryValue, MyRtlWriteRegistryValue);
DetourDetach(&(HANDLE&)LoadResource, LoadResource);
DetourDetach(&(PVOID&)TrueLsaEnumerateLogonSessions, MyLsaEnumerateLogonSessions);
DetourDetach(&(PVOID&)TrueMapViewOfFile, MyMapViewOfFile);
DetourDetach(&(PVOID&)TrueMapVirtualKeyA, MyMapVirtualKeyA);
DetourDetach(&(PVOID&)TrueMapVirtualKeyExA, MyMapVirtualKeyExA);
DetourDetach(&(PVOID&)TrueMapVirtualKeyW, MyMapVirtualKeyW);
DetourDetach(&(PVOID&)TrueMapVirtualKeyExW, MyMapVirtualKeyExW);
DetourDetach(&(PVOID&)TrueModule32First, MyModule32First);
DetourDetach(&(PVOID&)TrueModule32Next, MyModule32Next);
DetourDetach(&(HANDLE&)TrueOpenMutexA, MyOpenMutexA);
```

```
        DetourDetach(&(HANDLE&)TrueOpenProcess, MyOpenProcess);
        DetourDetach(&(PVOID&)TrueOutputDebugString, MyOutputDebugString);
        DetourDetach(&(PVOID&)TrueOutputDebugStringA, MyOutputDebugStringA);
        DetourDetach(&(PVOID&)TrueOutputDebugStringW, MyOutputDebugStringW);
        DetourDetach(&(PVOID&)TruePeekNamedPipe, MyPeekNamedPipe);
        DetourDetach(&(PVOID&)TrueProcess32First, MyProcess32First);
        DetourDetach(&(PVOID&)TrueProcess32FirstW, MyProcess32FirstW);
        DetourDetach(&(PVOID&)TrueProcess32Next, MyProcess32Next);
        DetourDetach(&(PVOID&)TrueProcess32NextW, MyProcess32NextW);
        DetourDetach(&(PVOID&)TrueQueueUserAPC, MyQueueUserAPC);
        DetourDetach(&(PVOID&)TrueReadProcessMemory, MyReadProcessMemory);
        DetourDetach(&(PVOID&)TrueRegisterHotKey, MyRegisterHotKey);
        DetourDetach(&(PVOID&)TrueRegOpenKeyA, MyRegOpenKeyA);
        DetourDetach(&(PVOID&)TrueRegOpenKeyExA, MyRegOpenKeyExA);
        DetourDetach(&(PVOID&)TrueRegOpenKeyExW, MyRegOpenKeyExW);
        DetourDetach(&(PVOID&)TrueResumeThread, MyResumeThread);
//        DetourDetach(&(PVOID&)TrueSamIConnect, MySamIConnect);
        DetourDetach(&(PVOID&)TrueSetFileTime, MySetFileTime);
        DetourDetach(&(PVOID&)TrueSetThreadContext, MySetThreadContext);
        DetourDetach(&(PVOID&)TrueSetWindowsHookEx, MySetWindowsHookEx);
        DetourDetach(&(PVOID&)TrueSfcTerminateWatcherThread, MySfcTerminateWatcherThread);
        DetourDetach(&(PVOID&)TrueStartServiceCtrlDispatcherA, MyStartServiceCtrlDispatcherA);
        DetourDetach(&(PVOID&)TrueSuspendThread, MySuspendThread);
        DetourDetach(&(PVOID&)True_wsystem, My_wsystem);
        DetourDetach(&(PVOID&)TrueThread32First, MyThread32First);
        DetourDetach(&(PVOID&)TrueThread32Next, MyThread32Next);
        DetourDetach(&(PVOID&)TrueToolhelp32ReadProcessMemory, MyToolhelp32ReadProcessMemory);
        DetourDetach(&(PVOID&)TrueURLDownloadToFile, MyURLDownloadToFile);
        DetourDetach(&(PVOID&)TrueURLDownloadToFileA, MyURLDownloadToFileA);
        DetourDetach(&(PVOID&)TrueWideCharToMultiByte, MyWideCharToMultiByte);
        DetourDetach(&(PVOID&)TrueWriteProcessMemory, MyWriteProcessMemory);
        DetourDetach(&(PVOID&)Trueaccept, Myaccept);
        DetourDetach(&(PVOID&)Truebind, Mybind);
        DetourDetach(&(PVOID&)Trueconnect, Myconnect);
        DetourDetach(&(PVOID&)TrueConnectNamedPipe, MyConnectNamedPipe);
        DetourDetach(&(PVOID&)Truerecv, Myrecv);
        DetourDetach(&(PVOID&)Truesend, Mysend);
        DetourDetach(&(PVOID&)TrueWSAStartup, MyWSAStartup);
        DetourDetach(&(HANDLE&)TrueCreateFileMappingA, MyCreateFileMappingA);
        DetourDetach(&(PVOID&)TrueIsNTAdmin, MyIsNTAdmin);
        DetourDetach(&(PVOID&)TrueIsUserAnAdmin, MyIsUserAnAdmin);
        DetourDetach(&(HANDLE&)TrueLoadLibrary, MyLoadLibrary);
        DetourDetach(&(HANDLE&)TrueLoadLibraryExA, MyLoadLibraryExA);
        DetourDetach(&(HANDLE&)TrueGetConsoleWindow, MyGetConsoleWindow);
        DetourDetach(&(HANDLE&)TrueSetProcessDEPPolicy, MySetProcessDEPPolicy);
        DetourDetach(&(PVOID&)TrueWSASend, MyWSASend);
        DetourDetach(&(HANDLE&)TrueHeapCreate, MyHeapCreate);
```

```
        DetourDetach(&(PVOID&)TrueCoTaskMemAlloc, MyCoTaskMemAlloc);
        DetourDetach(&(PVOID&)TrueCoTaskMemFree, MyCoTaskMemFree);

        printf("All detached!\n");
        error = DetourTransactionCommit();
        fflush(stdout);
        printf("Flushed.\n");

    }
    return TRUE;
}
```

Appendix F Malware Samples

Sample	Classification
361B481084C41D0DF4C4EB763F268043	PUA
5CB07299CEDD69F096B09358754831E0	Trojan
F54D48B019436E28C1AF5BABF247748B	PUA
830400121A557B0668AE9374CD847C8B	Worm
2F54B592E62E66FBFE7F7DB24F70F36A	Ransomware
78B661723E5A4DE6446EE585A030E5C1	PUA
7D436F8B7C72498CD5738812D9E0EC61	Backdoor
0EC2EF48ECCC4E25FA35C59D3CB4A56E	Trojan
8F7AB3C56E3FF4A6D23C57D0E07A4D1E	Virus
44B5A3AF895F31E22F6BC4EB66BD3EB7	Worm

Appendix G MACT Commands

MACT Commands			
Verb	Noun	Parameters	Description
C	<none>	<none>	Continue executing application.
C	E	<none>	Continue to end of API.
D	A	<none>	Display artifact construct.
D	C	<none>	Display valid commands.
D	T	<none>	Display thread construct.
D	M	<none>	Display memory construct.
D	P	<none>	Display parameters of current function.
D	R	<none>	Display return values of current function.
G	M	Address, Length	Get and write memory from location.
M	A	Address, Length	Initiate the serialization of an artifact using the address and length specified.
M	O	Address, Length	Initiate the creation of a memory object using the address and length. This will be of type "C".
M	S	File Name, Address, Length	Initiate the substitution of the an artifact from the specified File Name to the Address and of the Length specified.
S	A	Address, Length	Substitute artifact at memory location.
S	P	Parameters as required by current function.	Substitute parameters for function call.
S	R	Return values as defined for current function.	Substitute return values from function call.

Table 15 Valid MACT Commands

Appendix H Definition of Terms

API – Application Programming Interface.

Backdoor – A method of accessing a program or system bypassing normal security mechanisms.

Binary State – The machine code representation of a program once it has been compiled.

Debugger – A tool used to test and debug computer programs.

Decompiler – A tool that takes a program in its binary state and converts it to a high level language.

Disassembler – A tool that takes a program in its binary state and coverts it to a low level language.

Dynamic Link Library (DLL) – A collection of programs that are loaded and used by other programs.

Graphical User Interface (GUI) – A visual way of interacting with a computer program. This is used in most modern operating systems.

Import Address Table (IAT)- Contains the of addresses of functions when the DLLs are loaded.

Keylogger – A program that records and logs each keystroke performed.

Obfuscation – The act of purposefully making something difficult to understand.

Malware - malicious software intentionally designed to harm data, computers and other devices or people.

Machine Code – A specific representation of a program that is specific to a Central Processing Unit (CPU). Each type of CPU has its own machine language.

MACT – Malware Analysis and Artifact Capture Tool

Ransomware – A program that blocks access to a computer or files until money is paid.

Reverse Engineering – The act of deconstructing or analyzing a device or software to determine how it functions.

Rootkit – A program designed to be hidden and provide privileged system access.

System Events – Events that occur within an operating system that affect the state of the system. These include events such as the creation of files, deletion of files, update of system registries.

Trojan – Malicious software disguised as legitimate benign software.

Virtual Machine (VM) – A program that simulates a computer. Often used in malware analysis. Typically used to protect the physical host system and its operating system while running malware samples on the simulated computer.