# On the Reverse Engineering of the Citadel Botnet

Ashkan Rahimian[†‡], Raha Ziarati[†‡], Stere Preda[†‡], and Mourad Debbabi[†‡]
[†] National Cyber-Forensics and Training Alliance Canada
[‡] Computer Security Laboratory, Concordia University
Montreal, Quebec, Canada

***Abstract*—Citadel is an advanced information stealing malware that targets financial information. This malware poses a real threat against the confidentiality and integrity of personal and business data. Recently, a joint operation has been conducted by FBI and Microsoft Digital Crimes Unit in order to take down Citadel command-and-control servers. The operation caused some disruption in the botnet but has not stopped it completely. Due to the complex structure and advanced anti-reverse engineering techniques, the Citadel malware analysis process is challenging and time-consuming. This allows cyber criminals to carry on with their attacks while the analysis is still in progress. In this paper, we present the results of the Citadel reverse engineering and provide additional insights into the functionality, inner workings, and open source components of the malware. In order to accelerate the reverse engineering process, we propose a clone-based analysis methodology. Citadel is an offspring of a previously analyzed malware called Zeus. Thus, using the former as a reference, we can measure and quantify the similarities and differences of the new variant. Two types of code analysis techniques are provided in the methodology namely assembly to source code matching, and binary clone detection. The methodology can help reduce the number of functions that should be analyzed manually. The analysis results prove that the approach is promising in Citadel malware analysis. Furthermore, the same approach is applicable to similar malware analysis scenarios.**

***Index Terms*—Reverse Engineering, Malware Analysis, Clone Detection, Botnet Takedown, Incident Response, Zeus Botnet Variant, Static Analysis, Dynamic Analysis**

## I. Introduction

One of the offspring of Zeus malware that has been making headlines in recent months *(March 2013 - July 2013)* is called Citadel. Cyber criminals behind the Citadel malware have stolen more than 500 million dollars from online bank accounts [15]. Zeus was a prolific information stealing Trojan that has been around since 2007. In 2011, its source code was leaked on the internet and became available to the underground community. Since then, several malware have been developed based on the Zeus source code. Citadel has been employed by botnet operators to steal banking credentials and personal information [10], [17]. In addition, Citadel has features that extend beyond targeting financial institutions. Spying capabilities such as video capture is an example of such features that literally enables cyber criminals to collect anything from a victim's machine. The malware also acts as ransomware and scareware in order to extort money from victims. Reverse engineering is often considered as the primary step taken to gain an in-depth understanding of a piece of malware. However, it is a challenging and time-consuming process, which requires a great deal of manual intervention.

The major objectives of this paper are to reverse engineer the Citadel malware and gain more insights into its structure and functionality. In particular, the objectives can be summarized as follows:

1) Quantify the similarities and differences between Citadel and Zeus malware.
2) Get additional insights into online open source components used in Citadel.
3) Accelerate the reverse engineering process of similar malware variants.

To enhance and speed up the process, a new approach termed as clone-based analysis is employed in this study. Indeed, this paper illustrates the usefulness of the proposed approach in the analysis of new variants of a malware family. In this scenario, a preceding malware $P$ is supposed to be analyzed and understood. If a variant $V$ uses portions of the $P$ code, the approach will highlight the shared portions. Consequently, disregarding the clones could reduce the analysis time. In a more general case where $P$ is not known in advance, the approach can still provide insights into the components of $V$, in comparison to other sources.

The main contributions of this paper are three folds. First, a detailed reverse engineering analysis of the Citadel malware is presented and its functionality is described. Second, a new methodology for reverse engineering malware is proposed. This methodology significantly decreases malware analysts' efforts and reduces the analysis time. Third, the similarity between the Citadel malware and the Zeus malware is precisely quantified. Also, additional insights are provided into the open-source components used in the Citadel malware.

This case study has been chosen for a number of reasons. First, Citadel and Zeus are real threats against confidentiality, integrity and availability of information systems. Cyber criminals are constantly enhancing their tools for gaining access to personal and financial data. The profitability of such crimeware tools in the underground market depends on the timeliness and support for new vulnerabilities. Therefore, malicious developers often reuse all or parts of existing components during their incremental development process. As a result, it is quite probable that they leave fingerprints of previously analyzed malcode on the new releases. Clone-based analysis comes in handy in such situations due to its potential for producing quick results. Integrating a clone-based analysis in

This article is the final draft post-refereeing version of the paper appeared in FPS 2013. DOI: 10.1007/978-3-319-05302-8_25

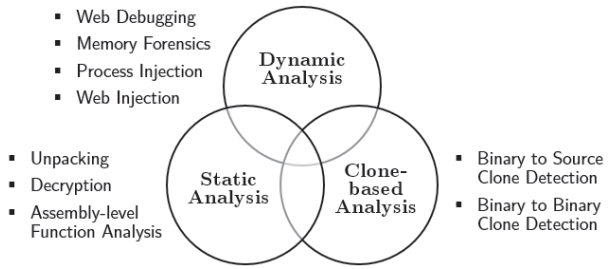


Fig. 1. The overlap in reverse engineering methodologies

the reverse engineering process will significantly reduce the overall analysis time. The second benefit of this case study is that it allows us to leverage our developed tools such as *RE-Source* [7] and *RE-Clone* [8] in reverse engineering sophisticated malware. The lessons learned during the analysis would bring new opportunities for future extensions of our tools. Third, the analysis provides us with practical solutions for mitigating future threats in a timely fashion. Once the analysis is performed on Zeus and Citadel, new Zeus-based malware variants with shared components can be analyzed faster.

The reminder of this paper is organized as follows. Section II, is dedicated to explaining our methodology in studying the malware. Section III details the dynamic analysis and explains the debugging process and memory forensic approaches. The main features of the Citadel malware are also described in this section. Section IV presents the static analysis and the steps led to the actual de-obfuscated code. Section V presents the clone-based analysis. The threat mitigation is briefly presented in Section VI and the conclusion is drawn in Section VIII.

## II. Methodology

Static and dynamic analysis are commonly used in studying malware [1], [5]. Static analysis focuses on malware code for inspecting its structure and functionality without execution. In contrast, dynamic analysis deals with behavior monitoring during the malware execution. In general, the process of malware reverse engineering is a combination of these two approaches, which is time-consuming and costly. The success of these approaches are tightly coupled with the functionalities of the tools and skills of the reverse engineer [3], [4].

To enhance and accelerate the process in analyzing the Citadel malware another dimension is considered in our study as shown in Figure 1. This new dimension is called clone-based analysis. In few words, the clone-based analysis identifies the pieces of code in Citadel malware that are originated from other malware and open-source applications. This step is performed automatically by leveraging the tools that are designed and developed in our security lab [7], [8]. There are two main advantages in considering this extra dimension into the static analysis. First, to avoid dealing with low-level assembly code in situations where the corresponding high-level code is available. Second, to prevent reverse engineering parts of the malware that has already been analyzed. This approach is very promising, especially in scenarios similar to Citadel that shares a significant portion of code with a previously reverse engineered malware like Zeus [6]. The process of assembly to source code matching is performed using the *RE-Source* framework [7]. Also, the binary clone matching is carried out using *RE-Clone* [8].

The proposed methodology is composed of three processes. Each process comprises several steps. We elaborate each step in the following sections:

1) *Static Analysis Process*
   - The disassembly is reviewed for finding obfuscated segments, decoder stubs, and embedded file images. The feasibility of static data decryption is assessed. It might be necessary to switch to dynamic analysis for code and data decryption.
   - A suitable circumvention strategy is adopted for bypassing the anti-static protection of the malware. Having a de-obfuscated/decrypted disassembly is a prerequisite for the clone-based analysis process.
   - Control flow analysis and data flow analysis are applied to gain an understanding of the crypto algorithms and encoding/decoding functionality.
2) *Dynamic Analysis Process*
   - A debugging environment is set to execute the binary, attach the debugger, set the breakpoints, control the unpacking, dump the process memory, generate an executable image, and save the process to file. The dumping process is repeated according to the analysis scenario.
   - System calls are monitored, malware activities are logged, network traffic is captured, downloaded files are backed up, and the communication protocol is observed. Also, the interesting artifacts are extracted.
3) *Clone-based Analysis Process*
   - Using the unpacked and de-obfuscated disassembly, a search is performed for standard and open-source components by applying the assembly to source-code matching technique of *RE-Source* [7]. The data matching technique encompasses two threads of *online* and *offline* analysis. This step is repeated for all process memory dumps and the set of matched projects are stored as *online* analysis results.
   - An *offline* analysis is performed for assigning the functionality tags according to API call classification in *RE-Source*. Function labels are updated, the proportion of assembly functions in each functionality group is calculated, and the functionality tags are reviewed based on the scenario.
   - Using the unpacked and de-obfuscated disassembly, a binary clone matching is done against the previously analyzed malware binaries in *RE-Clone*. Then, the occurrences of *inexact* and *exact* clones are recorded.

- The outputs of assembly to source-code matching and binary clone matching are combined for quantifying the similarities and difference of malware variants. The results draw a high-level picture of the code.
- The clones are selectively used to guide the static and dynamic analyses. In order to speed up the process, the clones are removed and the analysis focus is shifted to the original (non-clone) functions.

According to Figure 1, three connected processes are defined in the proposed methodology. In the Citadel case study, the dynamic analysis track focuses on web debugging, memory forensics, process injection and web injects. An important aspect in this process is the observation of malware's behavior in response to controlled inputs. On the other hand, the static analysis process focuses on assembly-level functions. De-obfuscation could occur in the overlapping area of these two methods. Unpacking and decryption are relevant examples that fall in this area. It is assumed that a database of previously analyzed code is available during the analysis. Code search engines provide an interface to online open source code repositories. Likewise, an offline code repository is maintained for storing the malware assembly code and the results of previous analysis sessions. One advantage of the clone-based analysis is that it can guide the dynamic and static steps. In other words, it highlights the important directions that the other two processes should follow by eliminating code clones, recognizing library functions, and providing additional comments. Therefore, the analysis focus is shifted to non-clone parts of the payload, resulting in a shorter analysis timeline.

## III. Dynamic Analysis

The purpose of the dynamic analysis process is to execute the malware and monitor its behavior in a controlled environment. Many tools and techniques are available for debugging malware [2], [3]. Sandboxing is a common technique in dynamic analysis and it is used for running untrusted code in a virtual setting. However, modern malware are well-equipped with anti-virtual machine protection against popular tools such as *Oracle VirtualBox* and *VMWare Workstation*. The malware can easily sense whether it is running on a virtual machine by checking certain artifacts in memory or on disk. As a result, the malware might change its normal behavior by taking an alternative execution path for hindering the analysis. Malware can even go one step further and try to exploit the virtual machine vulnerabilities in order to gain access to the host operating system. Thus, successful dynamic analysis may require caution and pre-processing steps. Debugging Citadel is challenging due to the built-in anti-debugging and injection capabilities but the protection can be circumvented by choosing the right strategy. As it will be discussed in Section V, *RE-Source* can provide informative tags such as `ADB`, `PSJ` or `AVM` for functions that potentially contain anti-debugging, process injection, or anti-virtual machine functionality. Upon the first execution, Citadel begins the infection process based on an embedded attack configuration.


Fig. 2. Citadel process injection and agent mode

### A. Citadel Infection Process

The Citadel bot operates in several modes. Upon the first execution, the dropper is in the *installation mode*. First, it unpacks and decrypts itself into memory. Then, it creates a copy of the binary file and stores it in the `%AppData%` folder under a randomly generated sub-folder and file name. The bot file is referred to as *Random.exe* in this context. As an example, the output path could be similar to: *...\AppData\Roaming\Random\Random.exe.* The bot also generates a batch file for removing the installation code. Looking for the existence of this path is a way of checking whether the system has been infected by the malware. Once the *Random.exe* is run from the new location, a sequence of similar steps are taken for unpacking and decrypting the bot. Afterwards, the bot switches to the *injection mode* and injects itself into the Explorer process and its child processes.

The injection step is dependent upon the privileges of the user who runs the bot and the version of the operating system. Following the injection, the bot process is terminated and the installation files are removed. Also, the bot updates the registry and adds an entity so that it will execute each time the operating system boots up. The registry path would appear as: *HKU\...\Software\Microsoft\Windows\CurrentVersion\Run\Rad.* The *Random.exe* is almost identical to the dropper except for the flag bytes located at the end of the file. This portion is encrypted and is used for controlling the bot mode. Therefore, even though the two executables are very similar, their behavior is different as the *Random.exe* operates in *agent and injection modes* only. Upon each system startup, the bot initiates the intelligence gathering process as it has been demonstrated in Figure 2.

### B. Debugging and Memory Forensics

After setting up the analysis environment and infecting it with the malware, the bot execution can be monitored and controlled using a scriptable debugger [18], [19]. Several techniques are available for hiding the debugger process from the bot and gaining more control over the debugger [2]. A web debugger or a network protocol analyzer is used for monitoring the HTTP network communications of the malware. Citadel encrypts the command-and-control (C&C) network traffic with RC4. Therefore, the crypto keys are required to intercept the commands, and view the stolen data. One way to find the keys is through debugging and setting hardware breakpoints on functions that precede network communication.

As it will be discussed in Section V, such network-related functionality can be identified through the `NET`, `WNT` and `CRY` tags assigned in the offline analysis. Upon suc-

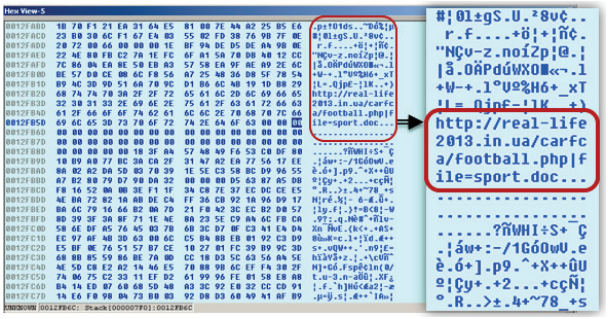


Fig. 3. Decoded Citadel config file name and location

cessful installation, the bot checks for Internet connectivity and tries to connect to embedded C&C addresses in order to announce its availability. The bot sends requests such as `POST /carfca/basket.php HTTP/1.1` or `POST /carfca/file.php HTTP/1.1` to the server. The server then replies and sends the encrypted config file to the bot. One major difference between Zeus and Citadel is in the way they handle the transmission of the configuration file. It was possible to find the location of Zeus config file and download it with minimal effort. Whereas in Citadel, it is more difficult to obtain the config file during the analysis. Citadel uses dynamic APIs and it decrypts strings in memory during the execution. This can be considered as an extra layer of protection that prevents the config file from being detected easily. Figure 3 shows one of the decrypted links to a Citadel C&C server which hosts the encrypted "*sport.doc*" config file. During the debug, the bot allocates memory for new segments and overwrites the memory space with decrypted code and data. The zero values in Figure 3 show the bytes that are yet to be overwritten by data. Blocking the malware's access to the requested C&C and modifying its timing mechanism will force the malware to enumerate the list of alternative embedded C&C servers.

Several tools and plug-ins are available for dumping memory, reconstructing import tables, and fixing PE headers. *OllyDump* and *ImpRec* are examples of such tools for unpacking Citadel [1], [3]. *Volatility* [20] was the most versatile and straightforward tool for memory forensics that was used in this project. It automatically builds the import tables and generates the executable versions of the unpacked binary. *Volatility* was utilized for creating executable process dumps and retrieving decrypted strings from memory. Figure 4 lists the utilized tools, and shows the number of detected functions, extracted strings, and identified function imports during different stages of the unpacking process.

### C. Citadel Attack Configuration

The configuration file is where the bot options are set. This file contains two sections for static and dynamic configuration as depicted in Figure 5. The bot builder reads this file and embeds the settings in the generated *bot.exe*. The bot encryption key is also defined in this file. The static config section is where the options for the initial attack are set. The setting

| | IDA Pro | PE Unpack | PE Explorer | OllyDump | ImpREC | x86 EMU | Volatility |
|---|---|---|---|---|---|---|---|
| Detected Functions | 13 | 155 | 162 | 160 | 161 | 160 | 796 |
| Extracted Strings | 337 | 3 | 3805 | 120 | 67 | 6628 | 917 |
| Function Imports | 11 | 16 | 107 | - | 107 | 107 | 386 |
| Executable Dump | Yes | Yes | Yes | No | No | No | Yes |

Fig. 4. Unpacking stages of Citadel binary

for web injects are defined in the dynamic config section. Web injects are used for tricking the users into revealing confidential information such as additional passwords or PINs. Since the man-in-the-middle attack (DLL hooking) occurs in the libraries such as *wininet.dll* or *nspr4.dll*, the victim user might not be able to distinguish the injected data from the genuine page. The result of injection could be in forms of extra fields, text boxes or warning messages. In comparison to Zeus, Citadel has a few extra features such as the anti-virus and security software evasion mechanism. Also, the DNS filter enables the bot to block the victim from accessing security-related websites and downloading new updates and patches. Consequently, this makes the machine more vulnerable to future attacks. A DNS redirection technique is used for implementing this feature. The config file includes a list of blocked websites and the corresponding redirected IP addresses. The report in [17] provides a lists of Citadel DNS filter domains. This type of DNS poisoning attack does not modify the Windows *Hosts* file. The settings related to the dynamic configuration can be updated by the C&C server according to predefined rules set by the botmaster. For instance, new modules can be remotely installed for country-specific web inject which target online banking accounts, automatic money transfer, and ransom [13], [17]. The encrypted configuration file can be obtained by capturing the bot traffic and replaying a crafted request in debugging.

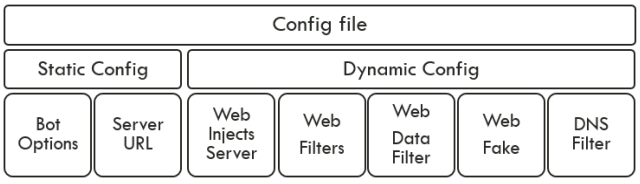


Fig. 5. Structure of Citadel configuration file

## IV. Static Analysis

In this section, we describe the main steps taken during the static analysis of the Citadel. The static malware analysis process normally starts by disassembling the malware binary. However, the initial disassembled code may not draw a complete picture of the original code due to different layers of obfuscation. Disassembling the Citadel malware using *IDA Pro* [19] results in a packed binary containing merely 13 functions, 11 imports, and 337 strings. The binary was

compressed, encrypted, and employed anti-reverse engineering techniques. Therefore, our static analysis started by de-obfuscating the malware. According to the first process of the proposed methodology, static and dynamic techniques should be interleaved for advancing the analysis.

### A. Unpacking Step

Not surprisingly, the malware was packed with a non-standard packing scheme. Therefore, automatic unpacking tools such as *UPX* could not be used and manual unpacking was necessary. To unpack the malware, a combination of static and dynamic techniques was used. The packed binary was executed in *Immunity debugger* [18] until the unpacking stub decompressed the binary in memory. Once the unpacking procedure was completed, the unpacking stub transferred the execution to the original entry point of the binary by making a jump from one segment to another segment. At this moment, *Volatility* [20] was used to dump the unpacked version of the binary's process out of memory and generate an executable unpacked version of the binary. The generated binary contained about 800 functions, 386 imports, and more than 900 decrypted strings.

### B. Code Decryption Step

The unpacking allowed the static analysis to be resumed. After this step, there were still some encrypted portions in the binary code. One of the interesting points was located at the address of `0x0040336` in our sample. An in-depth examination of the function, which cross-referenced this portion revealed the structure of encrypted data and the decryption mechanism. As shown in Figure 6, the structure size is 8 bytes and it consists of 4 chunks. Also, the key for string decryption is embedded in the binary file. Algorithm 1, presents the decryption procedure used for decrypting the data. It helped us recover more than 300 strings and 45 C&C commands from the packed data in the binary.

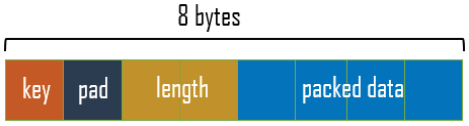


Fig. 6. Structure of the encrypted data

**Algorithm 1:** String Decryption Procedure

```
/* The command for decrypting embedded
   strings                                  */
for j in range length do
    UNPACKED_DATA =
    join(char(PACKED_DATA[j]) ^ j ^ key)
```

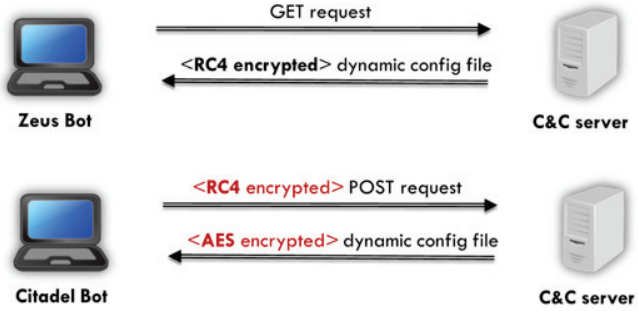


Fig. 7. Communication messages for retrieving the configuration file

### C. Crypto Algorithms

Receiving an RC4 encrypted configuration file from a C&C server in response to a plain `GET` request, and reusing of non-random values for encrypted messages were two main weaknesses of the Zeus malware. To overcome these weaknesses, significant improvements have been taken place concerning crypto algorithms in Citadel. As shown in Figure 7, Citadel C&C server requires a specially crafted RC4-encrypted `POST` message to send the configuration file. In addition, in order to provide better security, the configuration file is encrypted using AES. The Citadel authors have used a composition of different ciphers as shown in Figures 8 and 9. The RC4 encryption (Figure 8) starts by an customized encoding (obfuscation) mechanism known as *Visual Encrypt (VE)*.

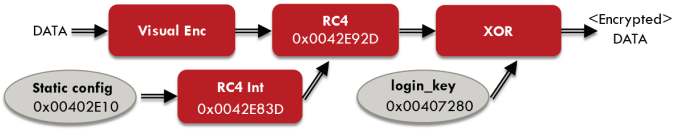


Fig. 8. Citadel RC4 encryption process

**Algorithm 2:** Visual Encrypt Procedure

```
void Crypt::VisualEncrypt(void *buffer, DWORD
size) {
for (DWORD i = 1; i < size; i++) do
    ((LPBYTE)buffer)[i] ^ = ((LPBYTE)buffer)[i - 1];
}
```

The input to the algorithm is an encoded buffer. The *VE* code is provided in Algorithm 2. This function was used in Zeus for crypto purposes as well. After the XOR operation, the non-standard RC4 initialization routine generates a 0x100 bytes key based on the static configuration data embedded in the binary. The output of the routine is a new RC4 key that is used in RC4 encryption function along with the customized XOR-ed data. Finally, performing an XOR on the RC4 output and the login key embedded in the binary, results in the RC4 encrypted data. Given *login_key=lkey* and *VE=encode*, the functionality can be stated as: *out* =

*lkey* XOR $RC4_{rkey}$*(encode(in))*. Therefore, *out*=Enc(*in*). The AES decryption is depicted in Figure 9. The configuration decryption routine takes the embedded static configuration data as input, and outputs the RC4 key. The MD5 hashed login key and the embedded RC4 key are fed to the RC4 routine. Next, the AES key is generated by performing an XOR on the output of the RC4 routine, and the login key. This key is used by the AES decryption function. Finally, the *Visual Decrypt (VD)* function (Algorithm 3) takes the result of the AES routine and decodes the decrypted data. The process can be formulated as: $AES_{key}$ = MD5(*lkey*) XOR $RC4_{rkey}$. Given *VD=decode*, the output can be stated as: *out* = *decode*($AES_{AES\_key}$(*in*)). The weakest point in the crypto process is that it is based on static config data, which shows that the authors lack competency in security algorithms and cipher composition.

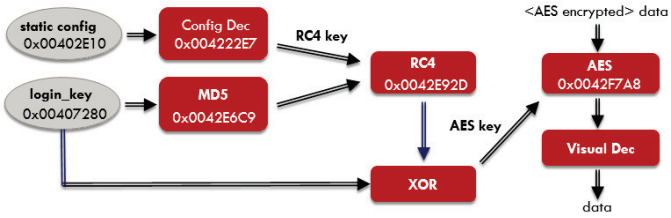


Fig. 9. Citadel AES decryption process

**Algorithm 3:** Visual Decrypt Procedure

```
void Crypt::VisualDecrypt(void *buffer, DWORD
size) {
if size > 0 then
    for (DWORD i = size − 1; i > 0; i−−) do
        ((LPBYTE)buffer)[i] ^= ((LPBYTE)buffer)[i - 1];
}
```

## V. Clone-based Analysis

The third process of the proposed methodology focuses on clone-based analysis, which can be applied for complementing the process of reverse engineering. Particularly, it could be helpful in reducing the required time for the static analysis phase. In this context, two techniques are taken into account for quantifying the similarities between Citadel and Zeus samples. The first approach uses *RE-Source* in order to reveal the open-source building blocks of the malware. The second approach utilizes *RE-Clone* for binary code matching. The major steps in the clone-based methodology can be enumerated as follows: (1) identification of standard algorithms and open-source library code in the malware disassembly, (2) assigning meaningful labels to assembly-level functions based on API classification, (3) commenting the assembly code based on a predefined dictionary of malware functions, (4) applying a window-based search and comparison mechanism for finding the pre-analyzed code components.

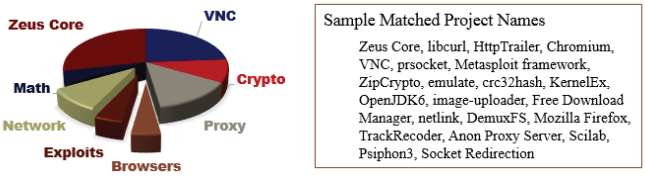


Fig. 10. Matched features with open-source projects

### A. Assembly to Open-Source Code Matching

The *RE-Source* framework [7] has been used for extracting assembly-level features from Citadel. This framework examines assembly functions in two phases of online and offline in order to find source files that share features with the disassembly. The key steps of the framework are: (1) extraction of interesting features, (2) feature-based query encoding, (3) query refinement for online code search engines, (4) request/response processing, (5) data extraction and parsing, (6) reporting results and updating comments, (7) feature-based offline analysis. Different features are considered for online and offline analysis. During the online analysis phase, *RE-Source* revealed the correlation between function-level features of Citadel and several open-source projects. The video capture capability of the malware was unleashed through the links to source files such as: *MHRecordContol.h*, *stopRecord.c*, *trackerRecorder.h*, *signalRecorder.h*, *waitRecord.c*, etc. (See Figure 11). This observation was further supported by occurrences of strings such as *"_startRecord16"* during the dynamic API de-obfuscation. Moreover, a "`video_start`" C&C command was also found in this process. Even though screen capture is a common feature in modern malware, live video capture capability is a new feature, which is only seen in complex and progressive samples. It should be noted that the online analysis results of *RE-Source* that suggested video-related capability were the outcome of an approximate code matching process. Although the matching process was not perfect, it was accurate enough to reveal the functionality context in this case. Similarly, *RE-Source* had commented the code with references to other open-source projects such as the ones listed in Figure 10. The number of matched projects in each category determines the size of each pie slice. Many Zeus-based malware variants have appeared online since the release of Zeus source code in 2011. Having access to Zeus source code enabled us to match Citadel binary against Zeus source code. The pie chart in Figure 10 shows the general categories of open-source projects that are used in Citadel. Apart from the detached slices, Citadel and Zeus share a considerable amount of code related to the core, VNC, crypto, and proxy functionality. However, the differences can be summarized in network communication code, new exploits and browser-specific code for web injects.

### B. Offline Analysis and Functionality Tags

*RE-Source* can also be used for tagging assembly functions based on API calls and classifying functions according to their

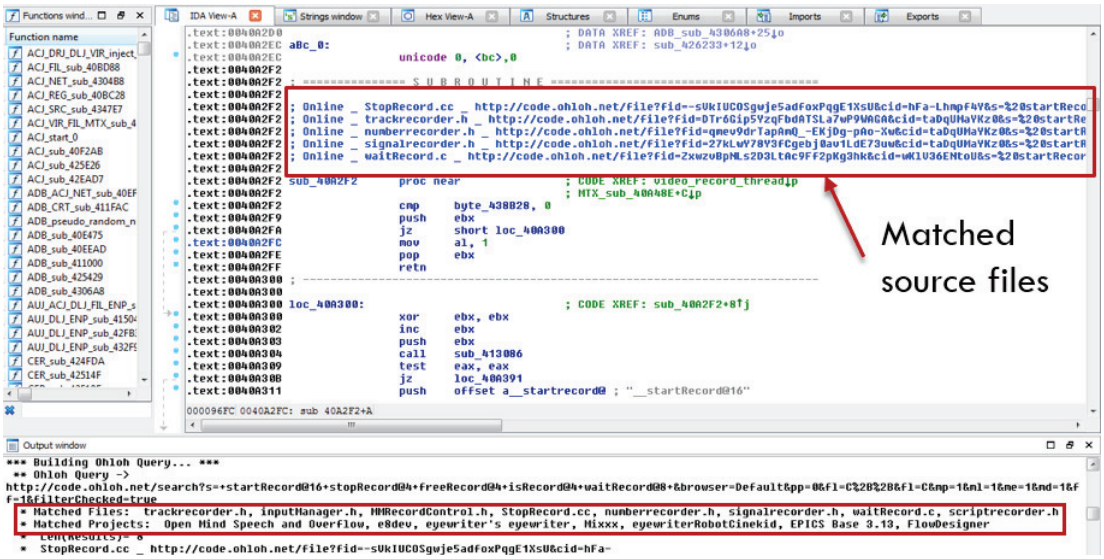


Fig. 11. The output of RE-Source pointing to video capture source code

potential functionality. When applied to the unpacked version of Citadel, 652 functionality tags were detected by the offline analyzer. A function is assigned several tags if it contains more than one system call. Accurate functionality tags could convey meaningful hints to the reverse engineer during the static analysis phase. In conjunction with the code and data cross-referencing, functionality tags can enrich the disassembly by highlighting the final system calls in a multi-level function call hierarchy. Since system calls serve as interaction points with the operating system, having a high-level view of them could draw a more organized view of the code.

| *TAG* | *Functionality* | *TAG* | *Functionality* | *TAG* | *Functionality* |
|---|---|---|---|---|---|
| ADB | Anti-debugging | REG | Registry update | CER | Certificates |
| PSJ | Process injection | DIR | Directory | OSI | OS Information |
| DLJ | DLL injection | MTX | Mutex | SRC | Search |
| DRJ | Direct injection | PIP | Pipe | VIR | Virtual memory |
| HKJ | Hook injection | HTP | HTTP, Web | CRT | Critical section |
| ACJ | APC injection | URL | URI links | MOD | Modification |
| AUJ | APC userspace | ENP | Enumeration | SRV | Service |
| AKJ | APC kernerlspace | HAS | Hashing | LCH | Launcher |
| WNT | Win networking | CRY | Cryptography | AVM | Anti-VM |
| NET | Low-level socks | FIL | File processing | CSH | Cache |

Fig. 12. Functionality tags for offline analysis

Functionality tags are not limited to simple system calls merely for file processing or registry modifications. They can be composed of several operations related to common malware behavior. New patterns can be defined for highlighting common malicious code in downloaders, launchers, reverse shells, remote calls and keyloggers based on the combination of several simple system operations. In this context, process memory modification and code injection points are of great interest to the reverse engineer. *RE-Source* includes tagging

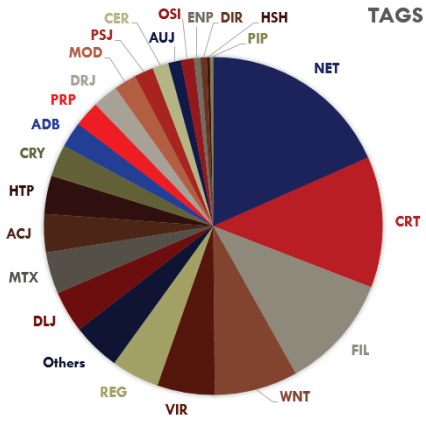


Fig. 13. Functionality tags assigned by offline analysis

categories such as *process injection*, *launcher*, *DLL injection*, *process replacement*, *hook injection*, *APC injection* and *resource segment manipulation* in the offline analysis. Figure 12 lists some of the available functionality tags in the prototype. A practical application of functionality tags is in disassembly comparison/synchronization of two malware variants. Instead of comparing the files by address, the code can be analyzed offline and the generated tags can be used as association criteria/sync points. In this process, the functions are sorted based on the assigned tags and the ones with similar tags are analyzed side by side. This technique was specifically helpful in synchronizing the disassembly of Citadel versus Zeus. Figure 13 depicts the detected functionality tags. The pie chart sectors are proportional to the number of assembly functions categorized under the same functionality group. The NET tag was assigned to 60 functions related to low-level socks. Also, 41 functions were tagged with CRT (critical section objects) for mutual exclusion synchronization. Similarly,

36 FIL tags were assigned to file manipulating functions. The other tags such as crypto, hashing, search and code injection were also identified during the analysis. The CRY (crypto) and HSH (hashing) tags provided an easy way of disassembly synchronization between Citadel and Zeus as the slight differences between the assembly files had no effect on the overall functionality group.

Translated into quantifiable terms, Figure 14 shows the output of *RE-Source* for Citadel vs. Zeus comparison. The numbers are reported in accordance with occurrence of certain features such as the number of assembly functions, API and functionality tags, common API in malware, number of matched opens source components, imported function calls, and Unicode strings. The results imply that the framework has been successful in revealing the internal components of the malware. The final outcome of assembly to source code matching is a list of source files along side the description from the malware dictionary. These information provide valuable insights into the potential functionality of the malicious code and facilitate the analysis.

| | Assembly Functions | Functionality tags | Malware API | Tagged Functions | Source URLs | Matched Projects | Imported Functions | Unicode Strings |
|---|---|---|---|---|---|---|---|---|
| Citadel 1.3.5.1 | 788 | 652 | 173 | 318 | 293 | 81 | 386 | 917 |
| Zeus 2.1.0.1 | 565 | 461 | 149 | 250 | 185 | 56 | 350 | 955 |

Fig. 14. RE-Source analysis results

### C. Binary Clone Analysis

The malware analysis process can be accelerated by identifying and removing the previously analyzed code fragments. The aim of binary clone analysis is to compare the assembly file of a new binary sample with a repository of analyzed code. The result of this analysis is a set of matched *clones*. In this context, we rely on the *RE-Clone* binary clone detector tool [8] that implements an improved version of the clone detector framework proposed in [9]. *RE-Clone* considers the same problem definition, that is the *exact* and *inexact* clone detection, as stated in [9]. Exact clones share the same assembly features, i.e., mnemonics, operands and registers. The only difference is in memory addresses. Inexact clones can be regarded as equal up to a certain level of abstraction, which means the number of common features must be greater than a certain threshold.

| *Malware Bot.exe* | *Functions* | *Window Size* | *Exact Clones* | *Inexact Clones* |
|---|---|---|---|---|
| Citadel 1.3.5.1 | 788 | 15 | 526 | 1876 |
| Zeus 2.1.0.1 | 565 | | | |

Fig. 15. Binary clone detection results

The analysis parameters such as search window size, normalization level and detection algorithm play a significant role on the analysis results. These parameters are set according


Fig. 17. Code analysis after clone elimination

to each analysis scenario. After marking the detected code fragments as clones, the analysis focus is shifted to non-analyzed and new code segments. The core components of the Zeus malware has been thoroughly studied in [6]. Also, the source and binary files are available online. Therefore, a new Zeus variant can be compared against the existing files in order to measure the similarity and detect the potential exact and inexact clones. This analysis is also applicable to finding the additional functions of the new malware variant. Figure 15 shows the results of the binary clone matching process. The samples share 526 exact binary clones with a window size of 15 instructions. In other words, almost %93 of Zeus assembly code also appears in Citadel. These clones form approximately %67 of the Citadel binary. This analysis highlights the remaining %33 of the Citadel assembly to be analyzed. Thus, a significant amount of time is saved by disregarding the clones. *RE-Clone* shows the address of each clone in the disassembly. Furthermore, the remaining functions can be examined in *RE-Source* before the manual analysis process is begun by the reverse engineer. This approach is depicted in Figure 17. The 1876 inexact clones reported by the tool include multiple combinations of regions that also contain the exact clones.

An interesting example of crypt-related clones is the detection of an inexact clone in the RC4 function that is used for encrypting the C&C network traffic. There are a few extra assembly instructions in the Citadel version of the RC4 function. This clone was found with a threshold of 0.8 and a two-combination inexact clone search method. In this approach, each two-combination of features are considered as a cluster. If more than %80 of regions appear in the same clusters, then they are treated as inexact clones. The red segments in Figure 16 highlight the detected inexact clones.

## VI. Threat Mitigation by Sinkholing

In June 2013, Microsoft Digital Crimes Unit reported on an operation known as *Operation b54*, in collaboration with FBI to shut down Citadel C&C servers [14]. As a result of this operation, 1400 Citadel botnets around the world were interrupted and redirected to sinkhole servers controlled by Microsoft. A comprehensive list of the domain names is available in [16]. Although, the operation has significantly disrupted Citadel botnets and has reduced the threat levels, it has also affected the honeypot systems that were used for identifying and locating the malware creators and distributors. Even though the threat counter-measurement has been successful, cyber criminals can still operate by infecting new machines and controlling their bots using alternative servers.

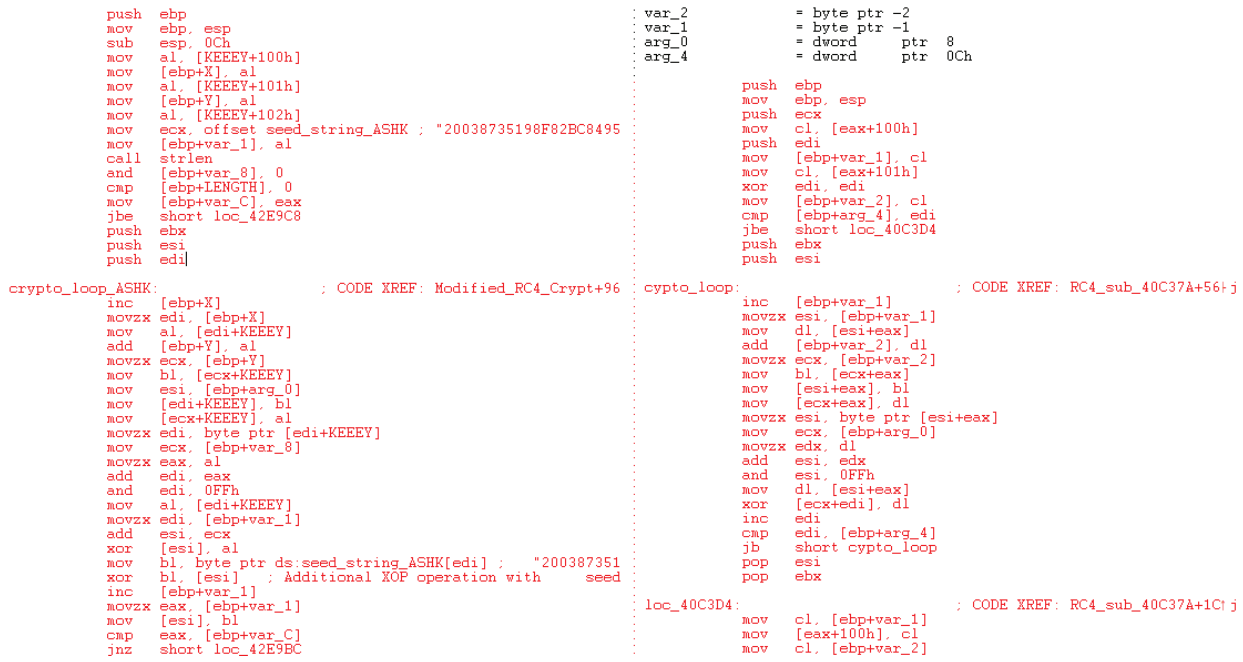


Fig. 16. An inexact clone detected in the RC4 encryption function (Citadel vs. Zeus)

## VII. Related Work

AnhLab [11], presented a comprehensive static analysis of Citadel malware. To the authors' knowledge, this report is the most complete analysis on Citadel malware which has been released so far. The process of infection, the structure of the malware binary, and malware's main functionalities and features are explained in details in this technical report. The report gives valuable insights on the malware and its capabilities, however, the methodology and steps that were taken for reaching the outcomes were not discussed. Also, although it is mentioned in the report that Citadel is remarkably similar to Zeus, the precise quantification of their similarity is not provided. Only approximate resemblance percentages is given without any details. To compare our analysis to this work, we provided a new methodology for reverse engineering malware by adopting clone-based analysis. Following our methodology, we concisely explained the steps we took in reverse engineering Citadel and insights that we obtained through our study.

SophosLabs [12], provided a brief report on Citadel malware. The major enhancements occurred in Citadel comparing to Zeus is explained in high-level and very briefly in this report. No explanation was provided about the process of reverse engineering the malware and how the authors gained those insights. Indeed, this report gave a decent overview about the Citadel malware without digging into the details. CERT Polska [13], also provided a technical report on Citadel malware. Similar to the previously mentioned report, this report was high-level and goes through the main features of Citadel without providing details. The reports mainly provided statistics focusing on the impact of the malware and its geographical distribution. The statistics were gathered based on the traffic to the sinkhole server after the domain had been taken down.

By leveraging the tools developed in our security lab, we quantified the similarity between Zeus and Citadel malware. These results could be further refined by integrating other existing techniques designed to automate malware analysis. For instance, our binary clone detector could be extended with a CFG-like dimension. For this purpose, we could benefit from the model proposed in [21] which aims to identify the common code fragments between two executable files and analyze the CFG subgraphs containing these fragments.

## VIII. Conclusion

The Citadel malware targets confidential data and financial transactions. It is an emergent threat against the online privacy and security. Citadel reverse engineering is challenging as it is equipped with anti-reverse engineering techniques for hindering the malware analysis process. As the number of incidents entailing new malware attacks are increasing, agile approaches are required for obtaining the analysis results in a timely fashion. The malware reverse engineering process consists of two major stages of static and dynamic analysis. This process can be accelerated and enhanced by adding a new dimension for clone analysis. Instead of initiating the process from the scratch, a quick clone-based analysis can easily highlight the similarities and differences between two samples of the same family. The analysis focus is then shifted to the differing sections. We have presented a methodology along with the tools and techniques for analyzing the Citadel malware. Also, we have compared Citadel with its predecessor, Zeus. The similarities have been quantified as the result of

two code matching techniques namely assembly to source, and binary code matching. The same methodology can be applied to other malware samples for providing insights into the potential malware functionality. The results of the malware analysis process can be added to a local code repository and used as a reference for measuring the similarities between future samples. They can also be used for improving the accuracy of the results. Overall, the successful completion of our objectives has led to underline best practices for supporting real-world malware analysis scenarios.

## ACKNOWLEDGMENT

The authors would like to thank ESET Canada for their collaboration and acknowledge the support of Mr. Pierre-Marc Bureau and the guidance provided by Mr. Marc-Etienne Leveille on de-obfuscation.

## REFERENCES

[1] M. Sikorski and A. Honig, Practical Malware Analysis, The Hands-On Guide to Dissecting Malicious Software, San Francisco: No Starch Press, 2012.

[2] J. Seitz, Gray Hat Python: Python Programming for Hackers and Reverse Engineers, San Francisco: No Starch Press, 2009.

[3] Malware Forensics Field Guide for Windows Systems: Digital Forensics Field Guides, Waltham: Syngress, 2012.

[4] C. Eagle, The IDA Pro book : The Unofficial Guide to the World's Most Popular Disassembler, San Francisco: No Starch Press, 2011.

[5] A. Singh, Identifying Malicious Code Through Reverse Engineering (Advances in Information Security), New York: Springer, 2009.

[6] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi and L. Wang, "On the Analysis of the Zeus Botnet Crimeware Toolkit," in *International Conference on Privacy Security and Trust (PST)*, Ottawa, 2010.

[7] A. Rahimian, P. Charland, S. Preda and M. Debbabi, "RESource: A Framework for Online Matching of Assembly with Open Source Code," in Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Miri, A., Tawbi, N. (eds.) FPS 2012. LNCS, vol. 7743, pp. 211-226. Springer, Heidelberg, 2013.

[8] P. Charland and B. C. M. Fung and M. R. Farhadi, "Clone Search for Malicious Code Correlation," in *NATO RTO Symposium on Information Assurance and Cyber Defense (IST-111)*, Koblenz, 2012.

[9] A. Saebjornsen, J. Willcock, T. Panas, D. Quinlan and Z. Su, "Detecting Code Clones in Binary Executables", in *Int'l Symposium on Software Testing and Analysis (ISSTA)*, Chicago, 2009.

[10] R. Sherstobitoff, "Inside the World of the Citadel Trojan," McAfee, 2013.

[11] AnhLab ASEC, "Malware Analysis: Citadel," December 2012. [Online]. Available: http://seifreed.es/docs/Citadel Troja Report_eng.pdf. [Accessed May 2013].

[12] J. Wyke, "The Citadel Crimeware Kit - Under the Microscope," December 2012. [Online]. Available: http://nakedsecurity.sophos.com/2012/12/05/the-citadel-crimeware-kit-under-the-microscope/. [Accessed May 2013].

[13] CERT Polska, "Takedown of the plitfi Citadel botnet," April 2013. [Online]. Available: www.cert.pl/PDF/Report_Citadel_plitfi_EN.pdf. [Accessed May 2013].

[14] Microsoft Digital Crimes Unit, "Microsoft, financial services and others join forces to combat massive cybercrime ring," June 2013. [Online]. Available: http://www.microsoft.com/en-us/news/Press/2013/Jun13/06-05DCUPR.aspx. [Accessed June 2013].

[15] J. Vincent, "$500 million botnet Citadel attacked by Microsoft and the FBI: Joint operation identified more than 1000 botnets, but operations continue," June 2013. [Online]. Available: http://www.independent.co.uk/life-style/gadgets-and-tech/news/500-million-botnet-citadel-attacked-by-microsoft-and-the-fbi-8647594.html. [Accessed June 2013].

[16] "List of Domain Names by Registry (Citadel)," June 2013. [Online]. Available: http://botnetlegalnotice.com/citadel/files/Compl_App_A.pdf.

[17] J. Milletary, "Citadel Trojan Malware Analysis," Dell SecureWorks, 2012.

[18] "Immunity Debugger: The Best of Both Worlds," Immunity, 2013. [Online]. Available: http://www.immunityinc.com/products-immdbg.shtml.

[19] "IDA Pro: Multi-processor Disassembler and Debugger," Hex-Rays, 2013. [Online]. Available: https://www.hex-rays.com/products/ida/debugger/index.shtml.

[20] "The Volatility Framework: Volatile Memory (RAM) Artifact Extraction Utility Framework," Volatile Systems, 2013. [Online]. Available: https://www.volatilesystems.com/default/volatility.

[21] G. Bonfante, J. Marion, F. Sabatier and A. Thierry, "Code Synchronization by Morphological Analysis", in *International Conference on Malicious and Unwanted Software (MALWARE)*, Washington, 2012.