

**MASTER**

## **An In-depth Analysis of the AZORult Infostealer Malware Capabilities**

van Rijn, H.W.J.

*Award date:*  
2021

[Link to publication](#)

### **Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

# An In-depth Analysis of the AZORult Infostealer Malware Capabilities

*Master Thesis*

H.W.J. van Rijn

Supervisors:

Dr. Luca Allodi

Michele Campobasso MSc

Dr. Tanir Ozcelebi

Eindhoven, September 2021

# An In-depth Analysis of the AZORult Infostealer Malware Capabilities

HUUB W.J. VAN RIJN, Eindhoven University of Technology, The Netherlands

The AZORult malware has been used in many criminal use cases. One particular use case is the stealing of information from its victims that can be used to impersonate the user and providing this stolen information to an underground marketplace, which, in turn, sells this information. In this paper we analyse four samples of the AZORult malware to find out what its information stealing methods are, what information it is capable of stealing and how this information is processed by the malware. Finally, we run an experiment that shows that, using the AZORult malware, we can successfully impersonate a user across a variety of experimental setups. Our results show that the malware is heavily targeting browser user data and has the means and customisability that allows a criminal to steal the relevant data for impersonation attacks and more.

Additional Key Words and Phrases: Malware, AZORult, Impersonation-as-a-Service

## 1 INTRODUCTION

In recent years, the cybercrime domain has been in a constant arms race against advancing antivirus software, smarter network monitoring systems and better corporate security training of employees. This arms race gives rise to a constant need of innovation on both the attacking as well as the defending side of the domain. A recent development in the threat landscape is a threat model called “Impersonation-as-a-Service” – IMPaaS for short [4]. In the IMPaaS model, the product sold is a bundle of information, namely ‘fingerprints’, that has been harvested from thousands of computer systems by means of a so-called ‘infostealer’ malware. This malware has the capability to steal various pieces of information about the infected device, such as credentials, cookies, browser and user behaviour metadata. An attacker having access to these products can inject this information into a browser to mimic the identity of a victim. By doing so, an attacker can circumvent state-of-the-art authentication mechanisms, such as Risk-Based Authentication (RBA) that may trigger further verification via Two Factor Authentication (2FA).

In the IMPaaS model, large amounts of machines compromised by infostealer malware provide the market with valuable information, while the market, effectively, sells the means to achieve victim impersonation. The AZORult malware has been suspected to be the leading supplier of so-called ‘fingerprints’ within the Impersonation-as-a-Service model[4] that are potentially created from the stolen data, supplying an underground market with products (the ‘fingerprints’) to sell[13]. This suspicion is partially confirmed by the report from KELA[6], who were able to “link over 300,000 AZORult infections” to such an underground market. These ‘fingerprints’ are essentially sets of identifiable pieces of information of a user that can be used in Risk Based Authentication (RBA) measures (used by websites such as Google, Steam and Facebook[17]) to evaluate the true identity of the user through information such as system locale, system time, IP-address and what browser the user is using, on top of the more traditional user and password combination. When an RBA mechanism finds an oddity amongst the identifying information of

a user, for example, a detected log-in location that does not match the user’s IP-address, it often forces the user to use their chosen 2-Factor-Authentication (2FA) method, such as a confirmation email or text message.

### 1.1 AZORult

The AZORult malware was first discovered in 2016/2017 [16][12] and is mainly used to steal information such as credentials to websites and programs like Skype, Discord and Steam, a user’s browser history and cookies, cryptocurrency wallet private keys and system files from a victim’s system. The malware sends this information as a ZIP-file to its affiliated Command-and-Control (C2) server. An important secondary behaviour of the malware is that it can receive a specific list of information to steal from the victim’s system, as well as a list of commands to execute or external executables to download and run. Based on this behaviour the AZORult malware can also be classified as a dropper, based on the fact it can download and execute extra files. An example of the AZORult 3.4.1 control panel used in the C2 server is shown in Figure 1. A predefined list of functionalities creates a clear picture of what information this malware is stealing. The *Files grabber* line allows the person controlling the malware to add specific file names or directories to be stolen. Finally, the *Loader* line allows the person controlling the malware to include a list of links to executables that should be downloaded and executed. Furthermore, AZORult is used by multiple malware families and exploit kits as the information-stealing component[1][16].

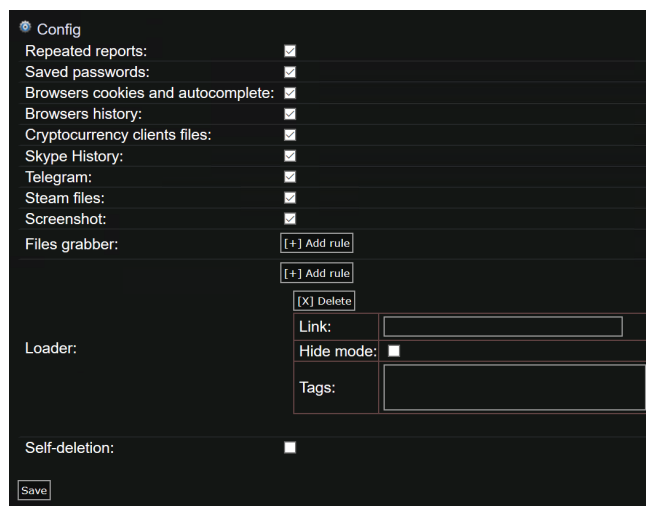


Fig. 1. The configuration panel used on the C2 server for AZORult version 3.4.1.

It is known that AZORult has been developed and updated over the years[1], as various versions have been identified. We also know

that the programming language, C, was used to create the initial program and in the versions, after version 3.3, C++ is used. The driving factor behind the malware's development is highly likely to be the successes it has booked within the Malware-as-a-Service<sup>1</sup> (MaaS) model[14], where developers create and sell customisable malware to suit the buyer's needs. One of the places AZORult has flourished is in the supply chain for underground markets[4][5][6].

## 1.2 Research Questions

A number of questions remain about what stolen information is required to successfully impersonate a user, how the processing of this stolen information leads to a fingerprint and how the fingerprints of the same victims are continuously updated. Also, it is unknown how exactly the malware steals information and what specific information this includes. Since this malware steals information that, combined or processed, creates fingerprints that are used to circumvent certain RBA measures, it can be of value to understand how certain information is gathered by the malware, or to find a distinction between legitimate sessions (where a regular user produces a fingerprint) and sessions using the stolen, injected fingerprints. When there is a better understanding of these intricate details of the malware and the stolen information in relation to the fingerprints, measures against the methods that AZORult and its affiliated IMPaaS platforms employ can be strengthened. Therefore the main research question (MRQ) is:

**What are the capabilities of AZORult with respect to information stealing as an enabling step to impersonate infected internet users?**

The following research questions are defined to aid in answering the main research question:

- RQ1** What information is stolen from a computer system that is infected with AZORult?
- RQ2** How does AZORult steal the specific information from the computer system it infects?
- RQ3** To what extent can the data stolen by AZORult be used to impersonate the victim when using online services?

Together with answering these research questions, two other contributions are made in this paper. During the analysis we (i) give an in-depth look into multiple AZORult samples and their capabilities and (ii) give an insight into the variety of roles AZORult can fulfil in the cybercrime landscape.

## 2 RELATED WORK

Multiple parties have analysed renditions of AZORult versions 3.2 and 3.3. Trustwave and BlackBerry have showed how the C2 server communication was set up and encrypted with a simple XOR key and what the C2 server looked like [8][9][15]. Trend Micro has information regarding the use of AZORult in multiple campaigns, where AZORult is used by different other exploit kits or malware kits to infect victims [16]. The team of Securelist found a change in the development process of AZORult where, in some cases, they switched from using the C programming language to C++. This can aid us in

determining what AZORult we are dealing with [1]. Securelist also found an underground market that uses the IMPaaS model and lists some of the information that is used to impersonate users by that market [13]. Campobasso and Allodi coined the term Impersonation-as-a-Service and their work indicates that AZORult is used as an infostealer malware in the scheme of an IMPaaS market[4]. The findings by the team at KELA also confirm the use of AZORult in IMPaaS and fingerprinting practices.[6] Wiefeling et al. showed how online services determine which login attempts are suspicious using an Risk-Based Authentication (RBA) system. They identified the most important properties an online service can check for to determine the trustworthiness of the login attempt. The fingerprints in the IMPaaS model are intended to circumvent these RBA systems by means of spoofing the properties that such an RBA system checks for such that it looks as if the victim is logging in based on these properties. [17][18]

## 3 METHODOLOGY

In this section we describe the strategy that we use to answer the research questions, and what techniques are implemented to derive those answers. After that, the environment used to analyse the malware is described. Next the rationale behind the static and dynamic analysis is described and to what end various tools aid in applying this rationale to the problem. Finally, an experiment is described that simulates the step-by-step scenario of an AZORult infection, followed by the information stealing and finally impersonation using this stolen information.

### 3.1 Strategy

In order to answer the research questions, first we should establish a number of behavioural and characterising malware features that can be used to compare and distinguish the different samples. These features can aid in determining the differences between the samples with respect to, for example, the information that is stolen or the way information is exfiltrated. The features to be compared are listed in Table 1 and discussed below. Then, a number of acquired AZORult samples are obtained and described. Once obtained, a strategy to find an answer to each individual research question is described. Next, the static and dynamic analysis that is to be performed on the samples with the goal of answering **RQ1** and **RQ2** is described. Finally, an experiment is detailed in Section 3.3 to answer **RQ3**. The mapping of research questions to the focus and applied methods is detailed in Figure 2.

**3.1.1 Features.** In order to compare and identify differences between samples of the AZORult malware, we define a set of features, listed in Table 1. This approach is based on the features of the ATT&CK matrix by MITRE [3], that uses the philosophy that among most adversarial cyber threats, a set of tactical and technical features can be combined to describe a threat as a whole. We select a subset of features based on the assumptions that (i) in order to steal the relevant information the malware has to access files and registry keys on Windows machines, as this is the place where the information is stored, and (ii) the malware has to exfiltrate this information via a network service in some shape or form. Furthermore, analyses by other instances [8][1] show that older strains of the

<sup>1</sup>This is the 'criminal equivalent' of the better known Software-as-a-Service (SaaS) model.

Feature	Technique	Why Relevant
Packed	Static analysis, Packing identification	Distinguishing samples
Networking	Network analysis, Dynamic analysis	Identifying means of exfiltration
Additional configuration	Network analysis, Static and dynamic code analysis	Insight in modularity and capability
Evasive behaviour	Dynamic analysis, API hooking	Insight in MO
Dropping behaviour	Dynamic analysis, static code analysis	Insight in MO
Persistence	Dynamic analysis, API hooking	Distinguishing samples, means of exfiltration
Process injection	Dynamic code analysis, API hooking	Identifying the data flow
# Registry interactions	API hooking, logging	Indication of amount of data stolen
# File system interactions	API hooking, logging	Indication of amount of data stolen
Executable size	Preliminary analysis	Footprint, IOC
Installation artefacts	Manual observation, API hooking, static code analysis	Footprint, data flow
Cryptography	Static and dynamic code analysis, API hooking	Data processing
Self deletion	API hooking, command logging, Manual observation, Static code analysis	Footprint, Distinguishing samples

Table 1. A list of features used in the comparison between malware samples, the techniques to find if these features are present in a sample and the software implementing these techniques

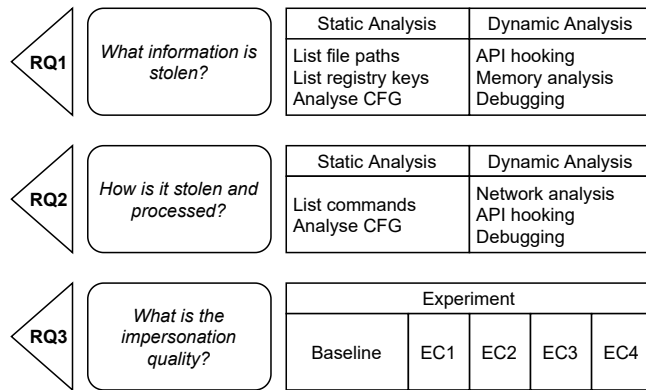


Fig. 2. Mapping from research questions to the focus and methods per research question.

AZORult malware gather extra actions to perform in the shape of commands from their affiliated C2 server, giving us another feature to check for and compare samples with (seen in Table 1 as ‘Additional configuration’). Additionally, AZORult has been seen in the wild as part of other exploit kits and is often not the sole piece of malware[16] involved in a campaign. Finally, AZORult also has the option to drop additional payloads. The samples that we have acquired for this analysis might therefore be separate stages of the infection or alternatively have kill chains that differ in length that eventually lead to the AZORult malware execution. Therefore we will analyse the features ‘evasive behaviour’, ‘dropping behaviour’ and ‘persistence’. Table 1 summarises these features.

**3.1.2 Samples.** A set of AZORult samples was obtained from an underground forum that trades and shares malware. The samples are given away freely, as the developer of the malware supposedly stopped support for those specific samples and the malware became outdated.

In this paper four samples of the AZORult information stealer malware are analysed. The first three are named ‘azorult 2019-03 (1).exe’, ‘azorult 2019-05 (2).exe’ and ‘azorult 2019-05 (3).exe’. For the remainder of this paper, they are referred to as ‘(1)’, ‘(2)’ and ‘(3)’, respectively. Each one of these consist of a single binary executable (.exe) file each.

Next to these three samples, a fourth sample of the malware is analysed. It consists of a C2 server (with instructions on how to install it on a webserver) and a builder executable that generates an AZORult payload upon giving the builder a URL. This URL should point to the C2 server setup such that the final payload uses that URL as its C2 server for exfiltrating information and receiving extra configuration instructions, as seen in Figure 1. The resulting payload of this builder is henceforth referred to as ‘(4)’.

Analysing multiple samples gives us an insight in the ‘average’ capability of the AZORult malware and it provides us with information on techniques and a possible modus operandi (MO) for the malware in general. For example, some samples might show persistent behaviour where others don’t, but perhaps all samples steal an identical (sub)set of information. This ‘core capability’ can then be identified by looking at a set of AZORult samples which will answer our research questions in a way that covers the AZORult malware in a more general sense and not just on a one-sample basis. The background on the main techniques used in this work (static and dynamic analysis) is given Appendix A.

**3.1.3 RQ1.** In order to find out what exact information is stolen by AZORult, the malware samples are statically and dynamically analysed. Static analysis is necessary because there might be useful information in the malware’s binary that does not require execution to find and can also aid in finding code that might be deliberately ignored while running the malware. This information could include literal filenames, directories or commands the malware might use and therefore give clues to what kind of information is stolen. Dynamic analysis tools can aid in capturing networking behaviour that

potentially includes the exfiltration of the stolen information we are after. Also, dynamic analysis using debugging tools can give clues with regards to the handling of this stolen information. For example, if any pre-processing is involved to send the data that is stolen in a certain format that cannot be deduced by capturing the networking behaviour alone or is too complex to statically analyse. The details of these analyses are described later. In order to answer **RQ1**, our attention during these analyses is mainly focused on observing the interaction that the malware has with the filesystem and Windows Registry, as these are the two main sources of information on a Windows machine. During dynamic analysis, a network capture can be made to see what data is sent to a potential C2 server and in what format it is sent. Furthermore, sample (4) supplies us with a unique look into what configurations that specific version of AZORult supports and can give insights in what information is stolen when the builder payload and C2 server are set up to work correctly.

Finally, a comparison between the analyses of the samples is in order to see if there exists a common set of information that is always stolen by AZORult and what additions to this set are made in the different samples.

**3.1.4 RQ2.** To answer this question, we aim to understand the malware's kill chain from the point of infection to the point of upload of information to the malware's C2 server. Using static analysis, we can firstly see if the malware already has any suspicious strings related to directories or registry keys in its data. Secondly, it can reveal any insights of how the malware saves, processes or packages the data it steals. Then, using dynamic analysis, the Windows API functions imported by the malware and flow of information from the system through the malware out towards a potential C2 server can be seen step-by-step.

Sample (4) can aid in understanding the final 'visible' step of the chain before creating and selling the fingerprints. As it gives us a look into the C2 infrastructure and possibly how the credentials are saved this might reveal valuable insights in what data is deemed more important than others and if any processing is done on the C2 side of the attack. Therefore, we simulate the attacker and victim environment at the same time by installing the C2 server on the analysis virtual environment as described below in Section 3.2. A payload can then be generated (sample (4)) that sends its stolen information to the locally hosted C2 server as if the machine were a victim. This allows us to observe any data storage or data processing that is performed on this stolen data by the C2 server.

By comparing the stealing and information processing techniques used by the different samples we come to a conclusion that shows the differences in processes and, more importantly, the similarities that lead to a proper description of how AZORult treats its stolen information before it is sold on an underground market, either as stolen data or as a fingerprint.

**3.1.5 RQ3.** Since we know that the information stolen by AZORult is in some way crafted into a fingerprint that, in turn, is used by an attacker to impersonate the victim, we can attempt to do the same in an experimental scenario. In a virtual environment a fake user is created for a number of online services: Google, Facebook and Twitter. These were selected because it is confirmed by Wiefling et

al.[18] that the services Google and Facebook employ RBA systems and an expected response is given. It was decided to use Twitter as a third service as there is not much known about its RBA system nor its RBA responses. Then, the browsing information of that fake user is stolen by AZORult using sample (4) and 'manually'<sup>2</sup> injected into a second environment. This environment should closely mimic the adversary environment suggested by the information available on the underground market. When logging into the accounts on this second environment, we monitor the email account of this user to see if any of the online services deem our login attempt suspicious or not. Depending on the success rate of these logins, we can attempt to gradually remove features or data that are similar between the victim and attacker environment to see what information is needed to successfully impersonate the victim.

## 3.2 Analysis setup

In order to securely analyse the samples for **RQ1** and **RQ2**, we have to set up a safe environment. We have decided to use FLARE VM<sup>3</sup> to perform most analytic tasks in. FLARE VM comes with a large amount of pre-installed tools that aid in the analyses of important aspects of binaries. There is no internet connection to and from the virtual machine other than a Remote Desktop connection. This was a choice, such that the analysis does not lead to flooding any malicious C2 servers with data about the virtual environment and to hide the fact that an analysis is being done. This leads to a problem where a malware sample fails to connect to its C2 server and therefore decides to change its behaviour, as it is often an indication of a sandboxing environment. To circumvent this shortcoming of the virtual environment, the internet can be partially mimicked by means of allowing DNS requests to resolve to the machine itself and a webserver can be installed to send responses to the requests. These might not be the responses the malware is expecting, but could lead to more information about what the 'correct' response may look like during dynamic analysis. Any URLs that are found during analysis which do not resolve can be added to the 'hosts' file<sup>4</sup> on the machine pointing towards 127.0.0.1 (also known as localhost, the machine itself) such that DNS requests to this previously unknown domain will be resolved from then on and point towards the (XAMPP) webserver running on the machine itself.

## 3.3 ImpaaS experiment setup

This experiment is done to answer **RQ3**. In order quantify the extent of the impersonation we need to determine the minimum amount of information that is required to *not* trigger a negative RBA response. We define a negative RBA response as follows: After logging in to the online service, the service either (i) denies entry altogether or (ii) the user needs to provide extra authentication or (iii) a security alert is sent to the account's owner. When the required information for a successful attack can be stolen by AZORult alone, it would mean the

<sup>2</sup>This entails simply copying the stolen data and pasting it into the target folder, overwriting existing files in that folder.

<sup>3</sup>a fully customisable, Windows-based security distribution for malware analysis, incident response, penetration testing, etc." <https://github.com/fireeye/flare-vm>

<sup>4</sup>This file, located in `C:\Windows\System32\drivers\etc\hosts` is the first list checked for DNS resolution. It contains an editable list of a mapping from domain names to IP-addresses.

stolen data is enough to perform such an attack and factors outside this information (such as IP address or ASN/ISP) are not necessarily needed to aid this attack. Then, the data stolen by AZORult can be deemed sufficient and would prove that AZORult alone is enough to support an ImpaaS attack with the necessary information.

**3.3.1 Experimental Conditions.** A baseline and a set of experimental conditions (EC) is defined that gradually change the attacker's set of properties in comparison to the victim's properties per different EC. The general approach of the experiment is as follows: we first simulate the victim machine as close as possible and call this the attacker machine. Then, we incrementally change the set of information that we inject into this attacker machine such that we can measure the responses from online services based on the changes we made. Every new setup we create when changing this set of stolen information is a new At first, we shall attempt to reproduce the underground market's product where an attacker possesses the fingerprint of a user. The market recommends its users to obtain a SOCKS5 proxy near the victim in order to impersonate the victim as closely as possible. We shall refer to this setup as the *baseline*. The only difference between the baseline and victim setup (in the eyes of an RBA system belonging to an online service) will be the IP address. The hypothesis is that the setup in the baseline is at best equal to the real one, therefore able to achieve a successful login on all online services.

Then, we create the ECs. The order of ECs is based on the work of Wiefeling et al. [17] where they provide a ranking of RBA features based on how important they are in deciding what login attempts are malicious and which are not. We make incremental changes using the properties 'IP-address', 'ISP', 'geolocation' and 'User Agent'. The victim machine and EC1 differ in IP address and ISP. For EC2, the set of differences is IP address, ISP and geolocation, where the location is about 70km away from the victim machine but within the same country. For EC3 we then have a foreign geolocation, different IP address and a different ISP. Finally, EC4 has differs in User Agent and IP address from the victim machine, as we think User Agent alone weighs more heavily than EC3's foreign geolocation, IP address and ISP change, according to Wiefeling et al. [17].

**3.3.2 Measurements.** We measure the response of the RBA system per online service by seeing if the services respond with on-screen alerts, notifying the victim with emails or texts, or asking for additional authentication. For every EC and online service combination we then define the following 2 outcomes:

**Outcome 1:** The RBA system has a positive response: The attacker is allowed entry to the services without further (user-known) suspicion.

**Outcome 2:** The RBA system has a negative response: The user is (i) denied entry to the services OR (ii) needs to provide extra authentication OR (iii) the victim is alerted by means of email or text message.

## 4 ANALYSIS

This section contains the findings of the analysis following the methodology of Section 3 of the four acquired samples (1), (2), (3) and (4). An overview of the findings and comparison between the four samples can be found in Table 2.

### 4.1 Analysing AZORult 2019-03 (1)

**4.1.1 Overview.** When opening the file properties of sample (1), Windows identifies it as an Application (.exe file) with the description *LetsSee! Setup*. The sample behaves as an installer and installs a program named *YTLoader*. Upon executing sample (1), the program spawns the familiar Windows Installer Wizard window, as it is the same as when installing any other arbitrary program on Windows. The last step of this installer asks the user if it would like to launch the installed program: *YTLoader*. When answered with 'yes', the program *YTLoader* then opens up and presents a simple GUI consisting of an input field for a URL and one of seven included adverts below the input field. The URL input field expects a YouTube video link that it would then download as an MP4 file. The GUI can be seen in 3. The *YTLoader* program sends a suspicious request to a domain that is listed by online sources like AnyRun and VirusTotal as a malicious C2 domain. The request includes processor and operating system information and suggests that *YTLoader* sends this information as a means of identifying potential victims for further infection or additional payload drops via the *smpchost* or *YTLoader* program. This is the most suspicious and malicious interaction found in this sample by analysis. *VirusTotal* and *AnyRun* show other potential outcomes with analyses where the C2 servers were still responding. The complete infection chain can be seen in Figure 4. Three files are dropped and executed from the initial infection, namely *YTLload.exe*, which in turns drops *smpchost.exe* and *YTLoader.exe*. This information answers the following for our research questions:

**RQ1** *What information is stolen from a computer system that is infected by AZORult?* The malware steals the following set of information: the operating system and its version, whether the processor is 64 or 32 bits, the processor name, model and manufacturer and a unique hardware ID generated by the malware. Online analysis tools *VirusTotal* and *AnyRun* find more stolen information that correspond to the functionalities as seen in Figure 1, with the exception of the Skype, Telegram and Steam information. This adds 'saved passwords', 'browser cookies and autocomplete', 'cryptocurrency wallets' to the stolen information. The discrepancy between our analysis and the online analysis tools is deemed to be due to limitations in our setup discussed later in Section 4.5.

**RQ2** *How does AZORult steal the specific information from the computer system it infects?* This sample has an evasive execution path (see Figure 4) with multiple installers. Eventually the sample installs a program named 'YTLoader' which is a modified open source program that now contains a malicious function which steals the data listed above and sends it to its affiliated C2 server using a simple GET request. The online analysis tools *VirusTotal* and *AnyRun* lead

	Sample (1)	Sample (2)	Sample (3)	Sample (4)
Packed	Partially <sup>4</sup>	Partially <sup>4</sup>	No	No
Networking	Yes	Yes	Yes	Yes
Additional configuration	Possibly <sup>1</sup>	Possibly <sup>1</sup>	Highly probable <sup>1</sup>	Yes
Evasive behaviour	High	Medium	None	None
Dropping behaviour	Yes	Yes	Highly probable <sup>1</sup>	Possible
Persistence	Possibly <sup>1</sup>	Possibly <sup>1</sup>	Yes <sup>3</sup>	No
Process injection	No	No	No	No
# Registry interactions <sup>5</sup>	26204	19051	9856	3785
# File system interactions <sup>5</sup>	14574	11840	16102	3401
Executable size	3.5 MB	3.7 MB	80 kB	112 kB
Installation artefacts	Yes	Yes	No	No
Cryptography	Possibly, <sup>1</sup> Weak <sup>2</sup>	Possibly, <sup>1</sup> Weak <sup>2</sup>	Yes, Weak <sup>2</sup>	Yes, Weak <sup>2</sup>
Self Deletion	No	No	Yes	Yes, by config

<sup>1</sup> There are no live C2 servers for these samples so additional configuration cannot be tested. However, based on the analysis of (3), (4), and VirusTotal and AnyRun reports these seem highly probable.

<sup>2</sup> This cryptography consists of encoding data with a 3-byte XOR key.

<sup>3</sup> Only when the program runs on specific Windows Server versions, this is assumed to either support further infection on a network and having direct access instead of behind a firewall.

<sup>4</sup> Only the file `smphost.exe` was packed by the UPX packer. This file is dropped by both samples (1) and (2).

<sup>5</sup> Average amount of interactions measured over 3 executions for each sample.

Table 2. Comparison of the features used in Table 1 found during the analysis of the four samples. Sample (4) was always able to connect to its (locally hosted) C2 server and receive a configuration, the other samples merely received a HTTP 200 response in order to reduce unrealistic crashing.

us to believe the the sample is a dropper and a more sophisticated and targeted executable is loaded in realistic circumstances.

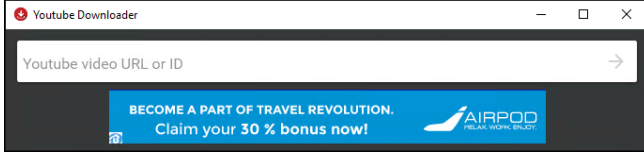


Fig. 3. The GUI of YTLloader, dropped by samples (1) and (2).

**4.1.2 Static analysis.** When analysing the executable with *PEiD*, there are no signs of a packer<sup>5</sup> being used to pack the executable.

The binary of sample (1) is scanned for potential clues using the *Strings* tool and a few unexpected values stand out. Sample (1) lists the string *Inno Setup* at a number of locations. *Inno Setup*<sup>6</sup> reveals itself to be open-source installer software that can be used to install any bundle of executable files on a Windows system. This is probably used such that the executable looks more generic as an installer, as Inno Setup is a common installer and so would avoid suspicion. This supports the idea that sample (1) is using evasive behaviour to disguise its actual intent.

The dropped files `YTLload.exe`, `smphost.exe` and `YTLloader.exe` show different behaviours. When statically analysing `YTLload.exe` using IDA, *Strings* and Ghidra, it shows imports of only Windows libraries that would assist in installation of the files `smphost.exe` and `YTLloader.exe`. Multiple strings referring to Registry keys and

file paths refer only to default installation information needed to install any program, therefore `YTLload.exe` is only as malicious as the files it installs. It is therefore a step in the evasive behaviour of sample (1).

Statically analysing `YTLloader.exe` using *Strings* and *DnSpyx86* (without debugging or running) shows references to a possible author of the program. Using *dnSpy*, we can analyse decompiled code that is syntactically equal to the source code. The *YTLloader* program still has specifically named namespaces and other symbols that can be used to identify the software. The program is an edited version of the open source project *YoutubeExplode*<sup>7</sup>. The parts that are different from the GitHub repository are 7 advertisements consisting of .png files and corresponding website URLs that were found. and the most significant line of code:

```
new StreamReader(
WebRequest.Create("https://istats.club/KosUS"
+ this.response()).GetResponse()
.GetResponseStream()).Close();
```

Where the function `this.response()` creates a URL-encoded string to add to this `istats.club/KosUS` URL as GET request data. The data consists of the machine's operating system, the amount of bits the processor supports (32 or 64), the exact processor model and a 'hwid'. After reverse engineering how this 'hwid' is constructed, it turns out to be an MD5 hash over the concatenation of the values contained in the Windows Registry keys `Software\Microsoft\Windows NT\CurrentVersion\ProductName` and `Software\Microsoft\Windows NT\CurrentVersion\ProductId` like so:

$$hwid = MD5(ProductName + ProductId)$$

<sup>5</sup>See Appendix A.1.1 for more details on packing.

<sup>6</sup><https://jrsoftware.org/isinfo.php>

<sup>7</sup><https://github.com/Tyrrrz/YoutubeExplode>



Altogether, the empty webrequest looks as follows:

`GET https://istats[.]club/kosUS?os=&bit=&processor=&model=&manufacturer=&hwid=`. What is interesting to note is that the program does not use the response of the server in any way, where one would expect the program to do more than just exfiltrate operating system and processor info. The only other information that would be gained is the IP address of the victim's machine, which, altogether, is a rather small amount data to exfiltrate by itself.

**4.1.3 Dynamic analysis.** Executing sample (1) leads to a multi-stage process tree found by Process Monitor that can be seen in Figure 4. The program drops multiple files, renames them and then repeats this a number of times. Often this behaviour is purposefully built into malware in order to evade anti-virus programs or to obfuscate malicious behaviour from the user. Furthermore the files and registry keys accessed can be listed to find a list of information that can be stolen by the malware. The list of files and Registry keys that are accessed by this main component are by no means malicious, as no information other than that required for installing an arbitrary Windows program is read.

When the process chain of sample (1) reaches the execution of the installed file `YTLoader.exe`, a number of registry keys are read that contain information about the hardware and operating system of the infected computer. The template GET request found in our static analysis in Section 4.1.2 is filled with the stolen information and sent to the `'istats[.]club/KosUS'` URL, as is confirmed by using Fiddler to analyse the traffic generated by the malware.

Dynamic analysis of the `smrchost.exe` binary leads to only more confusion about the goal of this piece of the malware, as it attempts no internet connections and crashes for unknown reason after a lot of fuzzy, unclear functionality. The executable is checked for Structured Exception Handler (SEH) chain manipulation, as the binary adds its own SEH to the chain and seems to crash on purpose. This technique uses the fact that the binary can add its own code as a SEH to the front of the SEH-chain and then deliberately crashing the program, executing their code as an error handler. This usually goes unnoticed in programs like *IDA* and *Ghidra* that analyse 'reachable' parts of the code and ignore SEH functions. In this case, using *OllyDbg*, a part of code that was labelled as 'data' by *Ghidra* during our static analysis, is actually identified as a function that is added to the SEH chain by `smrchost`. However, nothing malicious seems to happen within this function and the exception simply gets passed on to the next SEH in the chain. However, the VirusTotal report links to a report by Yomi Hunter (a sandboxing environment like *Any.Run*) suggests that `smrchost.exe` should be connecting to a C2 server of its own and attempting to open a backdoor using, among other things, the command `cmd.exe /c netsh firewall add portopening TCP 3389 "Remote Desktop"`. Unfortunately, we were not able to reproduce this connection in our testing environment. Since VirusTotal also lists process injection as one of the actions this binary performs, it is analysed with API Monitor to look for calls to API functions that are commonly used for this technique, but these are not used during the execution of `smrchost.exe` on the virtual machine. This leads us to believe that the crash of `smrchost.exe` was induced by a lack of environmental features, as no 'real' internet connection was available for the malware on our FLARE VM to download additional

configuration or payload(s) from a C2 server as VirusTotal describes was successful in their analysis.

## 4.2 AZORult 2019-05 (2)

**4.2.1 Overview.** Upon execution of sample (2), nothing visual happens, but the program *YTLoader* is also installed in the same directory as when sample (1) is executed. Sample (2) has the same behaviour as the intermediate stage of sample (1), *YTLoad.exe*. This is illustrated in Figure 4.

**RQ1** *What information is stolen from a computer system that is infected by AZORult?* The malware steals the exact same information as we found in sample (1).

**RQ2** *How does AZORult steal the specific information from the computer system it infects?* As sample (2) behaves identical to the intermediate step of sample (1), the same methods of stealing and sending are found in sample (2).

**4.2.2 Static analysis.** When hashing the sample with the MD5 hashing tool, the hash is not equal to the hash of *YTLoad.exe*, the intermediate stage of sample (1). However, when hashing the dropped file *YTLoader.exe*, it turns out both versions are identical. For the `smrchost.exe` binary this is not the case, as their hashes differ. Upon further inspection with *PeID*, we find that both versions of `smrchost.exe` are packed using different versions of the UPX packer. Hashing these unpacked versions with the MD5 hashing tool still results in different hashes. However, the remaining metadata listed by the MD5 hashing tool is almost identical: both are compiled on the same date, *but at a different time*, and their PE versions, import hashes, compiler signatures and linker signatures are identical. Using *binwalk* we then find that some additional random bytes were added to the unpacked version of `smrchost.exe` that is dropped by sample (2). We assume it was done to change the size and hash of the binary by a minor amount. The contents are equal as found in sample (1), as confirmed by carving out the individual parts and hashing those parts one by one for each sample.

**4.2.3 Dynamic analysis.** As the files are overall unchanged, the same behaviour is found here as it was in sample (1).

## 4.3 AZORult 2019-05 (3)

**4.3.1 Overview.** This sample is a smaller – 80 kB, where (1) = 3.5 MB, (2) = 3.7 MB – and single-process executable that deletes itself after execution. It does not drop any files but opens up a large amount of registry keys and files and attempts to send this to a C2 server located at `ravor[.]ac[.]ug` in a single XOR-encoded POST request. The analysis shows that within the binary there is code that can open a RDP backdoor. It shows little resemblance to the samples (1) and (2) as it shows no evasive behaviour and the executable is small and straight forward in its execution. There is no obfuscation or protection of the intention of the malware and it is obvious from static analysis alone that this is a simple but thorough infostealer. The process chain for sample (3) is illustrated in Figure 5.

**RQ1** *What information is stolen from a computer system that is infected by AZORult?* Sample (3) steals 'system information' that consists of the PC name, username, system locale, system time(zone), OS version, processor information, currently running processes

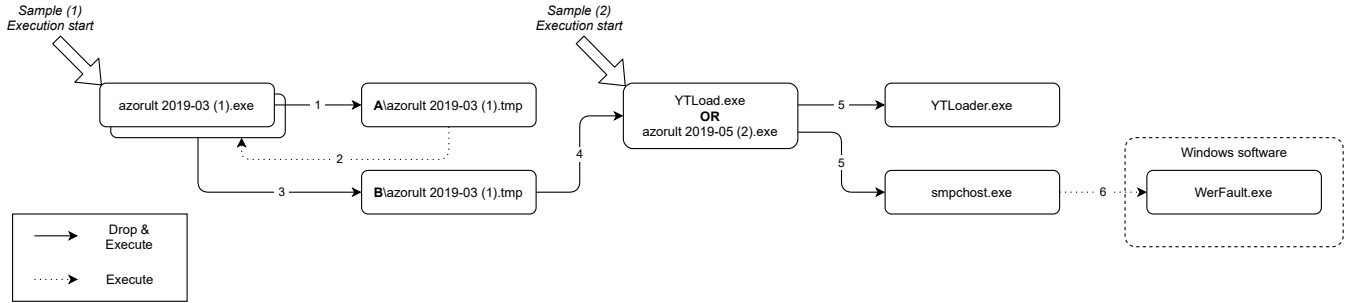


Fig. 4. Sample (1) and (2) combined process spawn chain. All interactions except for arrow 2 are drop-and-execute interactions. Arrow 2 executes a new instance of *azorult 2019-03(1).exe* that, in turn, continues with arrow 3 by dropping and executing another *.tmp* file. Arrow 6 executes *WerFault.exe*, a Windows error handling program. After arrow 4 the execution paths of sample (1)'s stage 'YLoad.exe' and the main stage of sample (2) are identical.

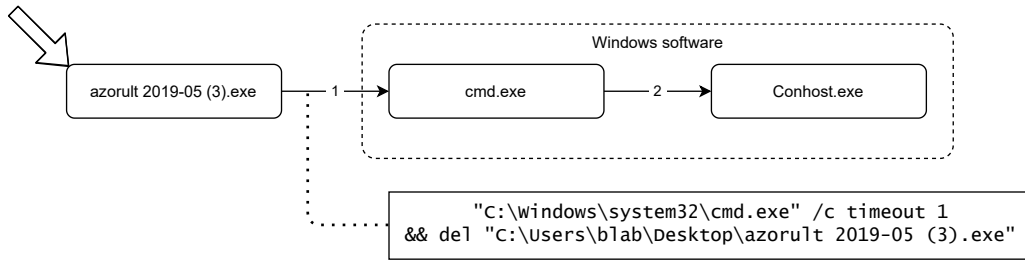


Fig. 5. Sample (3) process spawn chain. Most operations occur within *azorult 2019-05 (3).exe* and the command executed in *cmd.exe* is listed underneath, signalling to remove the original executable file after waiting one second.

and a list of installed software. Also, it steals the IP address, a list of browser cookies and finally a single screenshot taken during execution of the malware.

**RQ2** How does AZORult steal the specific information from the computer system it infects? Where in samples (1) and (2) there was a lot of evasive installing and execution behaviour, sample (3) is very straightforward. It executes, shows no signs of anything being executed except for a busy-cursor and proceeds to read the files and registry keys it wants to steal and finally makes a screenshot. The information is then neatly ordered into the files 'System.txt', 'ip.txt', 'scr.jpg' and 'CookieList.txt'. Next, the malware creates a ZIP-file of these files. Finally, this ZIP-file is encrypted using a simple XOR-key and sent to the sample's affiliated C2 server using a POST request.

**4.3.2 Static analysis.** Running *Strings* on this binary returns only 410 strings, due to the small size of the binary. Yet, there's a few that stand out. Firstly, the name 'azorult' is present in what seems a directory listing of the original development environment of this executable, namely 'F:\orders\cpp\azorult\_new\Release\azorult\_new.pdb'. The string 'cpp' is highly likely to relate to the programming language C++. When searching the internet for the MD5 hash of the file, it turns out that sample (3) is one of the samples discussed in [1]. The directory 'orders' could be considered as proof of this malware being built for a customer, as if it were one of more orders the author received to build malware. Then there's a single URL [http://ravor\[.\]ac\[.\]ug](http://ravor[.]ac[.]ug), possibly the C&C server of the malware. Furthermore, there are multiple strings hinting at the information the malware attempts to steal, like 'PasswordsList.txt', 'CookieList.txt',

'Coins\%s\wallet.dat'. Moreover, the malware lists multiple Windows API functions that allow the malware to connect to the internet. Also, a *cmd* command string stands out: '*/c netsh firewall add portopening TCP 3389 "Remote Desktop"*'. When this command would be run in *cmd*, it would attempt to add a firewall exception for port 3389, which would allow an external user to use the Remote Desktop Protocol to connect to the machine if they have credentials for a user on this machine and acting as a backdoor. Finally, there are a number of strings hinting at possible internet connectivity by the malware, namely 'POST' and 'Content-Type: application/x-www-form-urlencoded' which are strings normally found in a HTTP POST request, implying the malware at some point builds a POST request.

The overall CFG of the malware is relatively short and concise. Its *start* function consists of only 24 nodes. There is a section within this *start* function that is a list of *if* statements, where it checks a list of bytes one by one to see if they are zero or not. At every comparison, a decision is made to go into the following function or not. This is repeated 6 times. These 6 functions are attempting the following, respectively: Firstly, attempting to open up a program called Vault<sup>8</sup> that is used to save passwords in a secure way locally on a system and read the secrets the Vault software is safekeeping. Secondly, it attempts to steal browser cookies. Thirdly, a function that screenshots the entire display at the moment of execution. Then fourth, a function that searches for cryptocurrency wallets and attempts to steal the private keys of these wallets. Fifth, a list of stolen passwords is written to the file *PasswordsList.txt*. Finally

<sup>8</sup>For more information and documentation on Vault, see <https://www.vaultproject.io/>

sixth, a list of stolen cookies is created indicated by file writing to *CookieList.txt*. After all these function have either been executed or skipped, the malware continues with a function that is never skipped and where the system information is stolen. A file called *System.txt* is then created where a large number of information is written concerning the exact details of hardware of the machine, the current user, system time, locale, timezone, machine ID, *PATH* value, the Windows product version, graphics card and driver information and a list of programs installed on the machine listed in the Windows registry.

What is important to note is that the set of information stealing functions is very similar to the list of checkboxes seen in Figure 1.

After this large info stealing set of functions, the *.txt* files are zipped in memory – such that no files are ever created on the system and everything is done in memory – and added to a POST request to be sent to the malware’s C2 server. The malware then attempts to create a backdoor using RDP. It has a hardcoded username and password combination, namely *user=7J0nPFTYqvCO*, *password=GWsBUkp5rKGgK42R* which it attempts to add as a user to the network’s local group. This is followed by the malware changing the registry keys to allow for RDP connections, followed by running the command *cmd /c netsh firewall add portopening TCP 3389 "Remote Desktop"* which opens the port 3389 for TCP connections on the device, allowing for incoming RDP connections. This serves as a backdoor – a form of persistence – and would allow the attacker with the correct credentials to access the machine later on and might be an indication of how the AZORult malware was able to update its stolen information as mentioned in Section 1. This code only executes when the malware is on a specific Windows environment, as it checks the build and OS versions.

Finally, the malware decides to either exit the process with a call to *ExitProcess* or run the final command */c timeout 1 && del \"%s\*, where *%s* will be filled in with the original executable’s name, therefor deleting itself after waiting 1 second. During this 1 second, the process exits with a call to *ExitProcess*, after which the command to delete the file will be executed and the original executable deleted.

**4.3.3 Dynamic analysis.** Executing the sample without any analysis tools running results in the Windows wait cursor appearing for a few seconds and then disappearing. After a few more seconds, the executable file disappears and therefore shows clear self-deletion. Then, using Process Monitor, the file and registry interactions are observed. As expected, a large amount of registry keys are accessed, mainly in relation to the strings found during static analysis. Browser data, cryptocurrency wallets and system information is stolen, zipped and sent to its C2 server located at *ravor[dot]ac[dot]ug*. We can perform a man-in-the-middle attack on this traffic with Fiddler and adding the C2 server to our hosts file such that it points at our XAMPP setup.

When analysing the network traffic with Fiddler during execution, the a large POST request is sent to the C2 domain. This POST request is not in clear text. After cutting out the data part of the request, the data can be decrypted by performing a block-wise XOR operation over the data using the key *‘0x03 0x55 0xAE’* as was used in earlier versions of AZORult and is documented in Trustwave’s blogpost [8].

This gives us a ZIP-file that contains the files *‘CookieList.txt’*, *‘ip.txt’*, *‘scr.jpg’* and *‘System.txt’*, containing the stolen cookies, the victim’s IP, a screenshot and stolen system information, respectively.

The behaviour shown by sample (3) has been listed in the VirusTotal reports of both sample (1) and (2), mainly the RDP backdoor and self-deletion commands discussed in Section 4.3.2. These commands are identical across all three of VirusTotal’s analyses.

The RDP connection seen in the code in Section 4.3.2 is not executed when running the sample in our environment.

## 4.4 Builder Payload (4)

**4.4.1 Overview.** This binary has a lot of resemblance to sample (3) in size – 112 kB, where (3) = 80 kB, – and behaviour. It performs all operations in memory and leaves no temporary files on the system. The information stolen also consists of almost the same data as sample (3), with some more ‘modern’ additions of Telegram, Skype and Steam data.

**4.4.2 Static analysis.** Sample (4) contains similar strings and has a similar CFG structure in comparison to sample (3). Of those strings, the most similar ones are the ones for the output files that are sent to the C2 server. Their names are equal to the ones found in Section 4.3.3. The CFG also contains very similar part where some of the 6 steps discussed in Section 4.3.2 are seen again. In the case of sample (4), these refer to the steps to perform or skip according to the configuration sent by the C2 server.

However, there are some notable differences. There is no notion of the RDP backdoor in this sample, as no part of the code supports this. Also, a number of functions that are present in sample (4) are not present in sample (3), such as functions that are used to steal Telegram, Skype or Steam data.

**4.4.3 Dynamic analysis.** Without any analysis tools running, the sample behaves similar to sample (3), as the only visible evidence of the program running is the Windows wait cursor popping up. The executable does only delete itself if the checkbox for ‘self-deletion’ is checked in the C2 configuration. Analysing the execution of the binary with Fiddler shows that it follows the exact same networking process as described by Trustwave [9], where the malware first subscribes to the C2 server, then receives a configuration from the server, performs the steps according to the configuration, creates a ZIP-file of the stolen information, XOR-encodes it using the same 3-byte key and sends it to the C2 server. Process Monitor lists significantly lower amounts of registry and file system interactions compared to sample (3). This in combination with the fact that self-deletion is not built-in to sample (4) leads us to believe that some ‘file grabber’ configuration is built into the payload in the AZORult version used in sample (3), as this is not the case in sample (4), where the configuration of the C2 server specifies what files should be stolen next to information that is stolen by the list of checkboxes in Figure 1.

**4.4.4 Command and Control.** For this sample we took the opportunity to look at the server side of the malware. The installation is very straightforward – thanks to the clear Russian manual – and controlling the malware is done by using the control panel seen in 1

as discussed in Section 1.1. The database storing the data is a MySQL database and stores everything in plain text. The incoming data is decrypted using the 3-byte XOR key mentioned before in Section 4.3.3 and stored as a unique ZIP-file on the server. A reference to this filename is stored in the MySQL database and shown in the ‘reports’ panel of the C2 web page. No data processing is done whatsoever, but the web page does support converting cookies from the Netscape format to JSON and also supports multiple output formats for exporting stolen user-password combinations as seen in Figure 6.

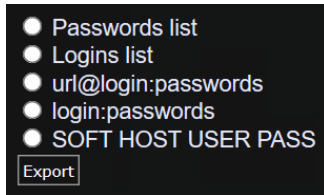


Fig. 6. The exporter menu in the C2 server of AZORult v3.4.1.

Overall, the C2 server is clearly designed with the idea of mass scale information stealing, as there are search and filter functionalities on the webpages that, for example, allow for quick grouping of victims by country.

A notable difference to what we see in the behaviour of sample (3) is that here in sample (4), the C2 configuration allows to either self-delete or not, while in sample (3) this was always done, with no exceptions. Also, there is no notion of installing a backdoor in either the code nor the C2 options of sample (4). Nonetheless, there is the option to make the malware ‘Load’ another executable file as seen in Figure 1.

#### 4.5 Discussion

Of the four samples analysed in this paper, samples (1) and (2) are identical in two of the files they drop. The two ‘installed’ components ‘YTLoader.exe’ and ‘smphost.exe’ revealed themselves to be identical files (based on MD5 hash) for both samples (1) and (2). Sample (1) uses a Windows Installer Wizard and (2) does not. Due to the installation process being slightly different, the analyses for the main components will differ. Overall the samples (1) and (2) are very similar in regards to the features we tested for as seen in Table 2. The differences in ‘# Registry interactions’ and ‘Evasive behaviour’ are attributed to the fact that sample (2) is essentially equal to half of the process chain of sample (1) as described in Section 4.2 and detailed in Figure 4.

Sample (3) showed no similarity to samples (1) and (2) during our analysis. However, based on the VirusTotal results of sample (1) that listed several commands executed that were identical to the ones of sample (3), we suspect that samples (1) and (2) did not reach their full potential in our analysis setup. This might be because of a variety of reasons, but the most important one is the fact that there is no internet connectivity from our analysis environment and therefore the samples’ affiliated C2 servers are not available in our setup. Another finding that supports this claim is that we were able to repair the YTLoader.exe crashing after a failed connection to

its C2 server. Here, we prevented the executable from crashing and the crash being handled by the Windows Error Reporting program (WER), but safely running and exiting. This was not the case for the smphost.exe program, which crashed unexpectedly and without clear reason. In the VirusTotal result of sample (1), there is no notion of WER being run, while in reports of this sample where the commands equal to that of sample (3) were *not* run, the WER program was listed as child process of smphost.exe. This leads to believe the program crashes when it cannot connect to its C2 server.

It is certain that sample (3) is the actual AZORult malware and it is suspected that samples (1) and (2) are droppers for sample (3) or an equivalent AZORult executable. This idea is supported by the fact that sample (4) closely resembles sample (3), as it also certainly is ‘pure’ AZORult without any dropper stages before the actual payload executes and both have the same method of exfiltrating data. Samples (3) and (4) both share similar sizes (see Table 2) and their analyses revealed very similar parts of code (flow) with similar steps and stealing functions per functionality. Their stolen data was formatted and sent to their respective C2 servers in the same fashion (zipped and encrypted with the same XOR key) as discussed in Sections 4.3.1 and 4.4.1. Finally, given that AZORult is often part of a malware kit or exploit kit, it is highly likely that a ‘pure’ payload like in samples (3) and (4) are less commonly found in the wild without any dropping mechanism before the payload execution.

It should also be noted that these raw executables are often delivered to a victim in ways that are not in scope of this paper. These methods include, but are not limited to, Microsoft Office macros and various layers of obfuscation.

Overall the behaviour is as was expected of an infostealer malware. It targets information that is important to the user and has a value on the underground market such as browser cookies, saved passwords and cryptocurrency wallets, which answers **RQ1**. However, it is interesting that samples (1) and (2) both send this GET request from ‘YTLoader.exe’ to a C2 server consisting of rather superficial (processor and operating system) information. Combined with the VirusTotal analysis where sample (1) drops and executes a binary that closely resembles the behaviour shown by sample (3), we find it likely that the information stolen with this GET request serves as some sort of preliminary screening to identify interesting targets for a larger C2 infrastructure where, based on what processor and OS the machine has, different payloads can be dropped by the samples (1) and (2) for different purposes. Furthermore, the data that is stolen is encrypted only lightly with a simple 3-byte XOR operation and is sent in one big POST request. This would make it easy for any network intrusion prevention system (IPS) to notice the malware and is not so stealthily as we expected from the fact the malware tends to update the information of its victims periodically.

#### 5 IMPERSONATION EXPERIMENT SETUP

The experiment is divided in separate ECs, as described in Section 3.3, and between these ECs, we shall wait a day before progressing to the next EC. This is done in order to not raise suspicion further than needed, as some changes might be too drastic, where a user logs in from The Netherlands and Finland on the same day, for instance. The experiment is illustrated in Figure 8, where a timeline

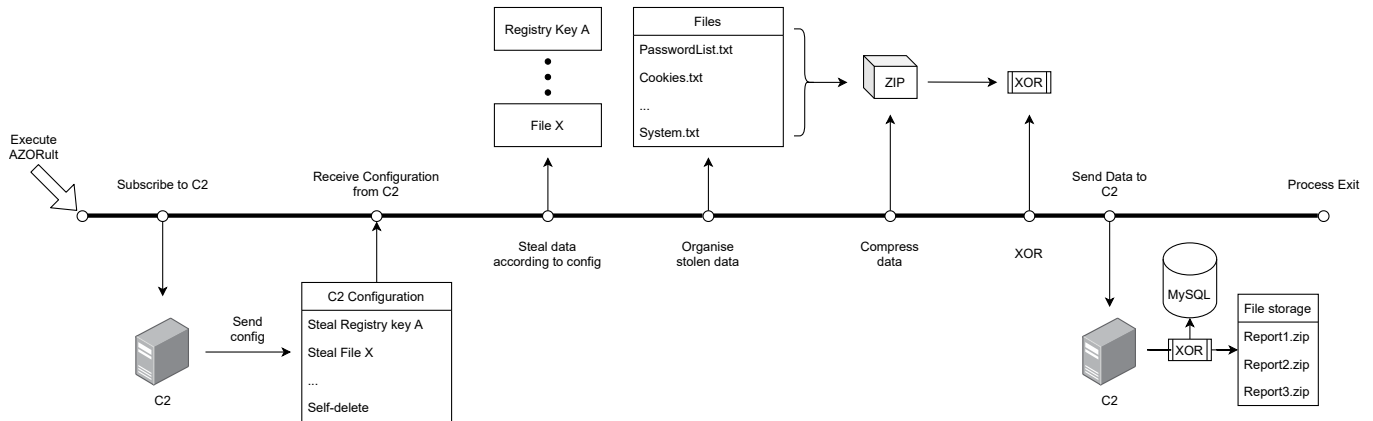


Fig. 7. A timeline showing the flow of data during an AZORult infection. The timeline itself depicts the execution of AZORult and shows the steps that AZORult takes while it is running. It eventually comes to an end at the 'Process Exit'.

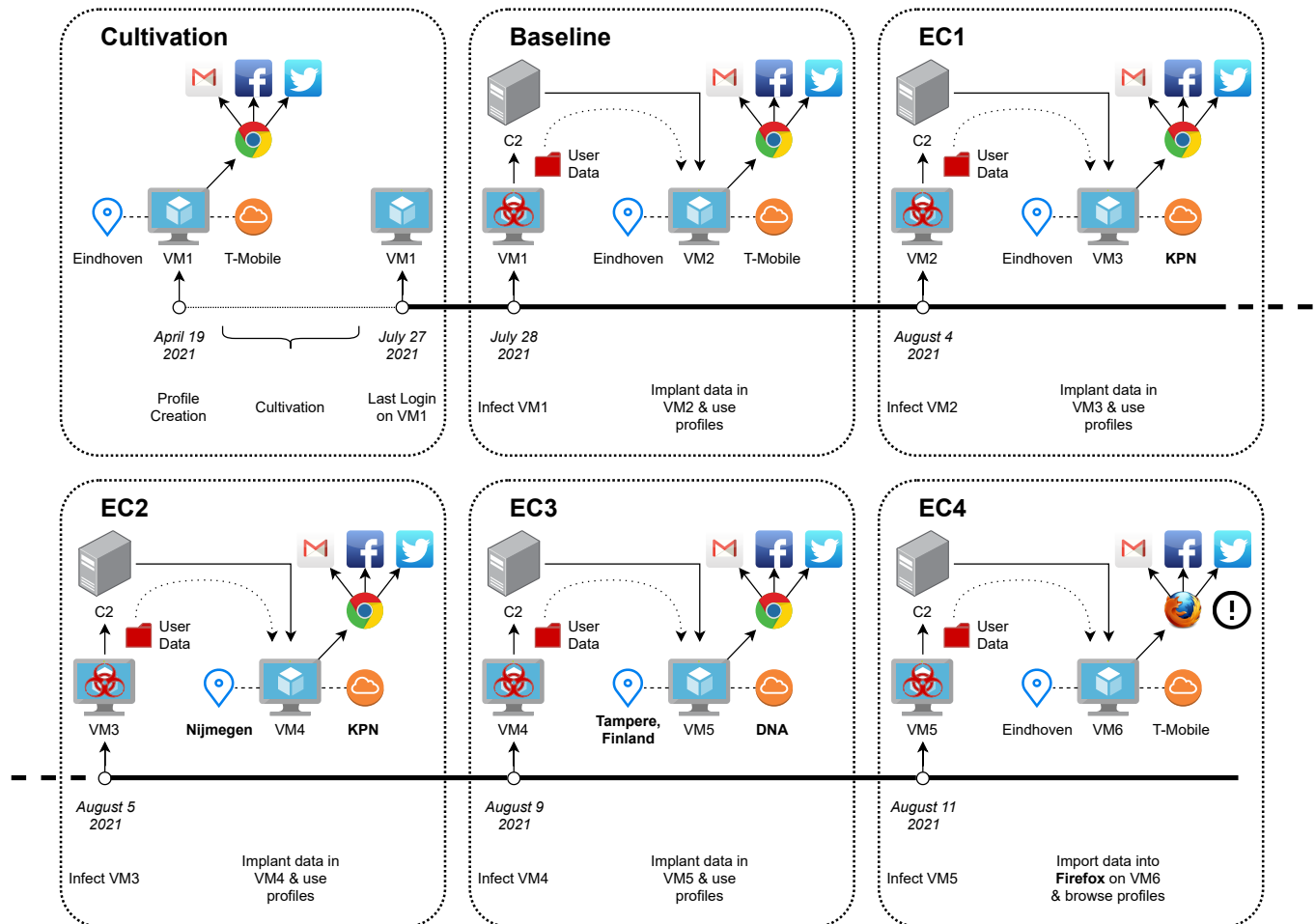


Fig. 8. An overview of the timeline of the experimental conditions and the differences between them.

is provided starting at the creation of the fake profile until the end of the experiment.

### 5.1 Setting up the experiment

**Overview** The experiment is designed to test for the ‘strength’ of the information stolen by AZORult with regards to impersonation. We start by creating a profile consisting of three accounts for Google, Facebook and Twitter and use these over a period of time (in this case three months) to make the profile credible in the eyes of these online services. This is done over such a long time to create a set of trustworthy logins as described by Wiefeling et al. [17] that can be regarded as a non-suspicious login. When the online service’s RBA systems notice a new login with a difference in some properties (e.g. IP address or ISP) when compared to the set of older, trusted logins, it makes a decision about how suspicious this new login attempt is. The goal of the experiment is then to see what set of information is needed to evade this RBA system such that it returns a positive RBA response and not a negative response. This is done by attempting new logins from a different systems, but where as many properties of the system are identical to the original victim system where the ‘trusted logins’ were performed during the cultivation period. These different systems are the baseline and various experimental conditions discussed before in Section 3.3 and will be discussed in detail below.

Firstly, there is VM1. On VM1, the accounts for Google, Facebook and Twitter were created and cultivated over a period of 3 months. The cultivation consisted of using the accounts at least once a week over a period of four months. During this cultivation the various online services were opened and used, by opening emails on Gmail, watching videos on Youtube (using the Google account), and browsing, liking and sharing posts (on Facebook and Twitter). VM1 is setup to appear similar to a generic environment, using Windows 10 and Google Chrome as operating system and web browser, respectively. VM1 can be seen as the ‘victim’ environment.

The first attacker environment (the baseline environment) will be simulated by another virtual machine that we denote as VM2. From what we know of the impersonation attacks and as emerges from our previous analysis of the malware, an attacker is able to mimic the victim’s browser (by means of stolen cookies, history, passwords etc.) and attempts to mimic the geolocation by buying a SOCKS5 proxy near the victim. Therefore VM2 will be setup such that it mimics the victim’s machine as close as possible to create a baseline of impersonation that can be tested against in the experimental conditions EC1 to EC4. In the experimental conditions EC1, EC2, EC3 and EC4 the VMs per EC are named VM3, VM4, VM5 and VM6, respectively, as seen in Figure 8.

All VMs in this experiment (the victim, the baseline and the four

ECs) are created from the same Windows 10 Enterprise Evaluation image (WinDev2102Eval, Version 10.0.19042 Build 19042) and are running the latest version of Google Chrome (starting at version 91.0.4472.114 64 bit and allowed to automatically update).

**Limitations** The research is limited by the amount of VMs that can be run. Since the instances should have different IP addresses and, generally, VPS solutions are IP-banned by online services such as Google.com, we are using different and limited residential locations to conduct the experiment. Therefore the amount of independent tests we can do is also lowered. Furthermore, given we have a single cultivated profile to use, the RBA response will be impacted

by every test we do, as most RBA systems are expected to take the history of trusted login attempts to create an average baseline login, as described by Wiefeling et al. [17]. This ‘average login’ can then be used to determine suspicious behaviour. When logging in from places that are not equal to the average login, but close enough to trust, the login added to this history, therefore altering the properties and the range of values that is considered to be an ‘average login’. Another limitation is the discrepancy between our baseline setup and the market product. The market product is most likely better than this setup, as there exists a process that is unknown to us between receiving the AZORult data arriving on its affiliated C2 server and the fingerprints being sold on the market that might raise the fidelity of the fingerprint product to a higher level than injecting the bare stolen AZORult data as done in this experiment.

Between EC infections, we do not return to login on the original victim machine (VM1). This is illustrated in Figure 8 where the data stolen is implanted, used and then stolen again for the next EC. This means that every EC is slightly influenced by the past ECs and the baseline environment. Essentially, this establishes a new baseline after every EC and is tested against in the next EC that introduces a new property change like ISP or geolocation.

### 5.2 Experiment preparation

**Victim accounts cultivation** The victim machine in this case is a Windows 10 Evaluation machine, where an account was created using the Google Chrome browser for the services of Google, Facebook and Twitter. The profile (consisting of the accounts for multiple services) was cultivated over a period of 3 months where (at least once a week) browsing behaviour was simulated by browsing to the pages of Google, Facebook and Twitter and minimal user interaction was performed such as retweeting tweets on Twitter, liking posts on Facebook and opening Gmail and watching YouTube videos using the Google account. The VM is located in the city of Eindhoven and uses the ISP T-Mobile.

In order to steal the data, the AZORult C2 server will be installed on another VM (C2-VM) that is running on the same host machine as the victim’s machine. Then the victim machine will be infected by a payload of the malware that is pointed at the C2-VM. The C2 will be configured such that the Google Chrome ‘User Data’-folder is stolen in its entirety and sent to the C2 server when the target is infected. The data will be downloaded from the C2 server, and finally this will be the set of data used in the following impersonation attacks.

**Establishing a baseline** To establish a baseline, the attacker VM is created in a different location within the city of Eindhoven with the same ISP as the victim machine. This is done to mimic the situation where an attacker acquires a SOCKS5 proxy close to the victim – as recommended by the market – such that the attacker has the same ISP and a roughly identical geolocation when compared to the victim. VM properties like Display resolution, number of CPU cores and amount of RAM memory are configured to be identical to the victim machine and the VM image is identical to that of the victim’s, therefore it is assumed that the only difference between victim and attacker machine in the eyes of the online services is the IP address, as detailed in Figure 8. This is as close as experimentally

possible to the market product and is done to evaluate that the stolen data is, in principle, enough to impersonate the victim.

The ‘User Data’ folder will be copy-pasted into the same location as where it was found on the victim machine. After that, we log in to Google, Facebook and Twitter and see if there is any blocking behaviour or messages regarding the suspicious login attempt sent to the fake profile’s email address or associated phone number.

We expect **outcome 1** for all services, as this environment is so similar to the original victim environment that the amount of suspicion the RBA system has should be below the ‘suspicion threshold’ that would trigger an alert or RBA response. Also, given that the market thrives with essentially the equivalent setup used in this baseline setup, it supports the hypothesis of this being the most likely outcome. Observing **outcome 2** means that the stolen information does not form a baseline for impersonation.

### 5.3 Experimental conditions

There are 4 different ECs that test different changes to the attacker setup and therefore will give us an indication of what is the minimum viable product required for an attacker to successfully perform the impersonation attack.

**EC1: A change in ISP** In this case, the attacker VM will also be situated in Eindhoven, but having a different ISP and IP address. The remainder of the VMs properties will be equal to the setup on the victim’s machine and the machine used for the baseline result.

The expected outcome of this EC is **outcome 2** for Twitter, as Twitter might have a less flexible RBA system compared to Facebook’s and Google’s RBA systems [17]. It might also be the case that a change of ISP is a clear indication of something being wrong according to the RBA systems of all online services, but we expect **outcome 1** for Google and Facebook, since they appear to be more flexible<sup>9</sup>.

**EC2: A change in geolocation within the same country** For EC2, the attacker VM will be almost identical in setup to the VM in EC1 – where the IP and ISP will be different from the victim machine – with one additional difference: the geolocation. The attacker VM will be hosted close to the city of Nijmegen in The Netherlands, such that the rough geolocation is 70km further away than Eindhoven.

We expect **outcome 2** for the Twitter service, as we suspect that a change of IP address would be enough for the RBA system to have a negative response as stated in EC1. For Google and Facebook we also expect **outcome 2**, as the combination of changing IP, ISP and geolocation is a group of features that are all individually rated of high importance to the RBA models described by Wiefeling et al. [17], and – given that all of them change here – are therefore expected to trigger a negative RBA response.

**EC3: A Change in user geolocation to a foreign country** For EC3, an attacker VM will be hosted in Finland, which makes the following list of differences when compared to the victim’s machine: IP, ISP and geolocation. The important feature that we will be testing

will be a change in response from EC2, where the same differences are present, but now the geolocation is much further away and in a different country.

Wiefeling et al. [17] showed that the ISP plays a more important role than the sole IP geolocation, but combining the change of ISP and border-crossing change in region should suffice to trigger an RBA response from at least some of the online services, therefore **outcome 2** is expected for all services. We think that it is possible for users of these services to have such dramatic movement when, for example, travelling and using a mobile device or laptop to login. This might contribute to an RBA system that relaxes the requirement of identical geolocation as previous logins, but is stricter regarding features that should remain the same when the same device is used to login in two different places. For example, screen size, browser (User Agent), operating system and others.

**EC4: A change in User Agent** For EC4, the attacker VM will be hosted in Eindhoven again, with only a different IP from the victim machine. On this machine we will attempt to change the User Agent variable of the request by switching from a Google browser to a Firefox browser to see a change in RBA behaviour. This experimental condition removes the differences used in EC1 to EC3, as the only difference will be IP address and User Agent. We chose to do so as, according to Wiefeling et al. [17], the User Agent property weighs a lot heavier than the properties we tested for before. Therefore, we decided to test the User Agent separately from the other properties.

To switch browsers as cleanly as possible, we import the User Data from Google Chrome using Mozilla Firefox, which copies the relevant bookmarks, history, cookies and saved credentials to Firefox. This is done by first installing Google Chrome and Mozilla Firefox on the attacker machine, then opening Google Chrome once to make sure it sets up a fresh ‘User Data’-folder. We close Chrome after that. Then, we copy-paste the stolen ‘User Data’-folder onto this fresh ‘User Data’-folder. Now we open Firefox and use Firefox’s built-in ‘Import Wizard’ to import all the user data from Chrome.

Now we perform the same test as before, where we log in to the services Google, Facebook and Twitter one by one and observe the potential RBA response.

We suspect **Outcome 2** for all services, as the User Agent String is identified as one of the strongest identifying features by Wiefeling et al. [17].

### 5.4 Results

The baseline was successful and showed **outcome 1** across all services as expected. No login was needed as the cookies for the services were still valid and therefore browsing to the websites of Gmail, Facebook and Twitter resulted in continuing the previous session. The results for each of the four EC are discussed below.

**EC1** The results gained from EC1 showed **outcome 1** across all services Google, Facebook and Twitter. This is identical to the results seen in the baseline.

**EC2** On EC2, the results also showed **outcome 1** across all services, with identical behaviour to the baseline and EC1.

<sup>9</sup>This was tested with uncultivated accounts in similar setups by logging into these as a means to probe the RBA systems.



**EC3** Once more, **outcome 1** was found across all services with yet again the identical behaviour we have seen in the baseline, EC1 and EC2.

**EC4** For EC4, **outcome 2** was found for the services Google and Twitter and **outcome 1** for Facebook. For EC4, the cookies seemed to be invalidated as they did not allow for instant access to the services without logging in as seen in the behaviour of the other ECs and the baseline. This is likely due to us using another browser for EC4. In this case, Google allowed us to login but did send a security alert to the victim's email notifying the victim of a login using a 'new Windows device', but did not block the login. After logging in to Google, it showed a notification on the webpage saying 'Welcome to your new Windows' and recommended a 'privacy check' and a 'security check' in order to improve the security of the account. Twitter allowed us to login, yet also sends a security notification to the victim's email address notifying them of a new login using an unknown device. For Facebook the login was successful and did not notify the user of the new login attempt.

## 6 DISCUSSION

### 6.1 RQ1: The stolen data

Our analysis showed that the set of information that AZORult can steal is in theory infinite, as long as the attacker can specify what exact information to look for. Analysis of sample (3) and the C2 server of sample (4) showed that criminals that use AZORult can use it to steal any data as long as they specify it within the 'file grabber'-configuration. This 'file grabber' begins a search from the given directory and steals all files within this sub-directory that are matched to a regex provided in this configuration. That being said, system information is always stolen and there are simple checkboxes for stealing – among other information – 'saved passwords', 'browser cookies and autocomplete' and 'browser history'. Depending on the goals of the AZORult user, other files might be stolen in some cases compared to others. For example, in our impersonation experiment we showed that AZORult is able to steal Google Chrome's 'User Data' folder successfully.

The system information that is stolen by both samples (3) and (4) consists of OS information, the computer name and user name, the screen resolution, the language packs installed, the local time and timezone information, the CPU model, the amount of RAM installed, video driver information, the processes running when executing the malware and finally a list of installed software on the system.

Remaining information that is stolen by means of the preconfigured checkboxes consists of crypto wallets, Skype history, Telegram information and Steam files.

### 6.2 RQ2: How the data is stolen

During the analysis of samples (1) and (2) it became clear that there can be some extensive evasive behaviour that also potentially serves as a 'reconnaissance' by checking the OS and processor information and exfiltrating this information only. From samples (3) and (4) we have learned that the data is stolen from a programmed list of

Registry keys and files on a Windows system when it concerns the checkbox-configured stealing options. This information is then organised into separate files with a clear structure by the malware, then compressed, encrypted and sent to its affiliated C2 server. The C2 server then decrypts the ZIP-file and saves the information as-is without further processing. Finally, some of the data (user-password combinations and cookies) can be exported using some built-in utilities of the C2 web page. The data flow is illustrated in Figure 7.

### 6.3 RQ3: Impersonation effectiveness

Overall, the extent of the impersonation capability is large, as our results show that the stolen information allows us to clone the Google Chrome installation and User Data in such a way that the cookies stolen by AZORult remain valid when implanted. This leads to our results where **outcome 1** was observed in EC1, EC2 and EC3 where no login was even needed and we could still use the session stored in the cookies. Only EC4, where a different browser was used, showed **outcome 2** for two out of three services (Google and Twitter), which means there is still impersonation potential in this scenario as **outcome 1** was observed for the Facebook service. However, we think the import process of the user data from Google Chrome to the Firefox browser might have rendered the cookies invalid, as Facebook did allow us to login without a negative RBA response, but did not automatically open the stored session as we observed in the results of EC1, EC2 and EC3. All in all, this tells us that in the scenario that the stolen cookies are valid, the RBA systems of Google, Twitter and Facebook do not respond negatively (**outcome 2**) no matter the IP address, ISP or geolocation involved.

It should be noted that the Google Chrome version used in the experiments is the latest and not a version that AZORult was designed for. This prevents AZORult from actually decrypting and stealing the passwords in plaintext as it was designed to do [19]. However, using what Picazo-Sanchez et al. [11] found regarding how Google uses the Windows user's SID in its HMAC hashing to verify its installation and User Data, we found out that these passwords can still be accessed after stealing and implanting the data on a new VM that is a clone of the original one, since Google Chrome is successfully able to hash its files given the fact that the SID is identical on all VMs we use. If the SID is not the same, Google Chrome does not automatically login to the Google profile (embedded in the browser, which allows the user to view stored passwords) and blocks access to stored passwords. However, this does not play a role in the transfer and workings of the user cookies, as these are successfully transplanted and their sessions remained valid.

The results of EC1, EC2 and EC3 also seem to imply that, in our impersonation attacks, the RBA mechanisms used by Google, Facebook and Twitter are evaded altogether by using the session cookies. The fact that we do not have to actually log in to the services and can keep using the stolen (still valid) session cookies might mean we do not actually pass through any RBA system in the first place and are therefore always observe **outcome 1** as it does not seem to matter what properties the attacker has with respect to IP, ISP or geolocation as long as the stolen session cookies contain a valid session.



## 7 CONCLUSION

In this paper we have analysed and compared the capabilities of four samples of the AZORult malware to show that the AZORult malware is capable of stealing a multitude of information and is easily customised by an attacker to serve impersonation purposes. We have also discovered and presented the methods and techniques used by the AZORult malware to steal, process, send, collect and store the data. Finally, we conducted an experiment showing that the infostealing capabilities of AZORult allow for strong impersonation attacks under the right circumstances.

## 8 FUTURE WORK

It remains largely unclear to us how in the analysis of samples (1) and (2) the binaries ‘smpchost.exe’ and ‘YTLoader.exe’ and their internet connectivity fit in the bigger picture of the infection and could therefore be further analysed using a more realistic setup and more experienced malware researchers.

The impersonation experiment we conducted has a lot of room for improvement. Firstly, there’s the fact that we used clones of the same Windows VM in our impersonation attacks. This is highly unrealistic as it is virtually impossible to clone the victim machine without creating a complete image of the victim’s hard disk, which would be a difficult task to perform given the massive size of files involved. Secondly, we experimented with a single profile consisting of three accounts and had limited locations to attack from. This renders the data marginal and can be scaled up in the number of accounts, the number of services and the number of changes to the implanted data by whoever has the resources to do so, in order to further define the minimum viable product that the underground market requires to successfully impersonate a user.

## ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors Dr. Luca Allodi and Michele Campobasso MSc for their helpful feedback and guidance during this project as well as dealing with my occasional clumsiness around the experiments. I wish to thank Ganduulga Gankhuyag MSc for providing me with the virtual analysis environment that I have used almost every day for a number of months. I wish to thank my friend Simon for proofreading this paper and providing me with feedback on the text. Finally, I would like to express my sincere gratitude to my friends Abel, Eetu, Fernando, Jim, Lars, Loek, Mauk and my brother Andries for helping me host the virtual machines for the experiment, I could not have done this experiment without your help.

## REFERENCES

- [1] Alexander Eremin. 2019. AZORult++: Rewriting History. <https://securelist.com/azorult-analysis-history/89922/>.
- [2] U. Bayer, A. Moser, and C. et al. Kruegel. 2006. Dynamic Analysis of Malicious Code. In *J Comput Virol*, Vol. 2. 67–77. <https://doi.org/10.1007/s11416-006-0012-2>
- [3] Andy Applebaum et al. Blake E. Strom. 2018. MITRE ATT&CK: Design and Philosophy. [https://attack.mitre.org/docs/ATTACK\\_Design\\_and\\_Philosophy\\_March\\_2020.pdf](https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf).
- [4] Michele Campobasso and Luca Allodi. 2020. Impersonation-as-a-Service: Characterizing the Emerging Criminal Infrastructure for User Impersonation at Scale. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, USA) (CCS ’20)*. Association for Computing Machinery, New York, NY, USA, 1665–1680. <https://doi.org/10.1145/3372297.3417892>
- [5] Kaspersky. 2019. Meet Genesis – the underground e-shop with tens of thousands of digital doppelgangers for sale to bypass financial anti-fraud solutions. [https://www.kaspersky.com/about/press-releases/2019\\_meet-genesis-the-underground-e-shop-with-tens-of-thousands-of-digital-doppelgangers](https://www.kaspersky.com/about/press-releases/2019_meet-genesis-the-underground-e-shop-with-tens-of-thousands-of-digital-doppelgangers)
- [6] Raveed Laeb. 2020. Exploring the Genesis Supply Chain for Fun and Profit. <https://ke-la.com/exploring-the-genesis-supply-chain-for-fun-and-profit/>
- [7] Kaiping Liu, Hee Beng Kuan Tan, and Xu Chen. 2013. Binary Code Analysis. *Computer* 46, 8 (2013), 60–68. <https://doi.org/10.1109/MC.2013.268>
- [8] Rodel Mendrez. 2019. Messing with Azorult Part 1: Malware Breakdown. <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/messing-with-azorult-part-1-malware-breakdown/>. Accessed: 22-12-2020.
- [9] Rodel Mendrez. 2019. Messing with Azorult Part 2: Command and Control. <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/messing-with-azorult-part-2-command-and-control/>. Accessed: 22-12-2020.
- [10] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of Static Analysis for Malware Detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. 421–430. <https://doi.org/10.1109/ACSAC.2007.21>
- [11] Pablo Picazo-Sanchez, Gerardo Schneider, and Andrei Sabelfeld. 2020. HMAC and “Secure Preferences”: Revisiting Chromium-Based Browsers Security. In *Cryptology and Network Security*, Stephan Krenn, Haya Shulman, and Serge Vaudenay (Eds.). Springer International Publishing, Cham, 107–126.
- [12] Darrel Rendell. 2019. Understanding the evolution of malware. *Computer Fraud & Security* 2019, 1 (2019), 17–19. [https://doi.org/10.1016/S1361-3723\(19\)30010-7](https://doi.org/10.1016/S1361-3723(19)30010-7)
- [13] SecureList. [n.d.]. Digital Doppelgangers. <https://securelist.com/digital-doppelgangers/90378/>. Accessed: 22-12-2020.
- [14] Seqrite. 2019. MaaS Moving Towards APT as a Service? <https://www.seqrite.com/blog/maas-moving-towards-apt-as-a-service/>. Accessed: 05-05-2021.
- [15] The BlackBerry Cylance Threat Research Team. [n.d.]. Threat Spotlight: Analyzing AZORult Infostealer Malware. <https://blogs.blackberry.com/en/2019/06/threat-spotlight-analyzing-azorult-infostealer-malware>. Accessed: 26-1-2020.
- [16] Trend Micro. 2019. AZORULT Malware Information. <https://success.trendmicro.com/solution/000146108-azorult-malware-information-kAJ4P000000kEK2WAM>.
- [17] Stephan Wiefeling, Luigi Lo Iacono, and Markus Dürmuth. 2019. Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In *ICT Systems Security and Privacy Protection*, Gurpreet Dhillon, Fredrik Karlsson, Karin Hedström, and André Zúquete (Eds.). Springer International Publishing, Cham, 134–148.
- [18] Stephan Wiefeling, Luigi Lo Iacono, and Markus Dürmuth. 2019. Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In *ICT Systems Security and Privacy Protection*, Gurpreet Dhillon, Fredrik Karlsson, Karin Hedström, and André Zúquete (Eds.). Springer International Publishing, Cham, 134–148.
- [19] ZDNet. 2020. Chrome 80 update cripples top cybercrime marketplace. <https://www.zdnet.com/article/chrome-80-update-cripples-top-cybercrime-marketplace/>.

## A ANALYSIS IN DETAIL

### A.1 Static analysis

The goal of static analysis is to find clues regarding the intent of the malware without executing it. This means that before running the malware, it gives insight into what capabilities the malware can have based on what DLL files are loaded (often needed for specific access to networking infrastructure such as TCP connections, WebSockets but also functions that can read Windows Registry keys), what URLs are affiliated with the malware and therefore potential C2 servers and other identifying string values that can be found within the data section of some executables.

Furthermore, malware authors are in a constant arms race with antivirus software authors and malware analysts and will attempt to hide the behaviour and make it harder to detect the malware in ways that would prevent an analyst from seeing the ‘real’ behaviour. An example of this is that the malware checks if it is in a debug environment or if other programs are running that could potentially be analysing the malware and based on this information the malware may alter its behaviour or even completely shut down. Using static analysis, this behaviour can be spotted beforehand and can be dealt with by, for example, patching the binary to jump over the code that is responsible for checking the environment or by altering the environment to suit the malware’s ‘expectation’ of a ‘victim machine’ by not using certain analysis programs.

However, malware can make use of different obfuscation methods to hinder static analysis, some of which make it very hard for programs that generate control flow graphs (CFG) to analyse the instruction flow of the program, as Moser et al.[10] has proven. This also renders static analysis hard to perform, as it can become quite difficult to comprehend the code and its potential behaviour.

Another technique malware can use to evade static analysis is using indirect jump tables (IJT), where the malware generates a table of addresses during run-time, therefore making it impossible to know some steps within the control flow of the program since the specific place in the code to be jumped to is unknown. Multiple other caveats are listed by Liu et al. [7].

Due to caveats like these, dynamic analysis is the next step of analysing the binary after exhausting the static analysis options.

**A.1.1 The process.** Firstly, clues can be found by hashing the sample with an MD5 hashing tool<sup>10</sup> and seeing additional information using the tool like resource imports. After that, the sample is checked to see if it was packed. Packing is a form of compression of code that often obfuscates the code. This makes it harder for static analysis to be performed as instructions are placed in a different order, the file hash will be different than the unpacked version of the program and might attempt to evade anti-virus software this way. The samples are analysed with the tool *PEiD*<sup>11</sup>, which looks for packing artefacts. When a packer is identified by *PEiD*, a corresponding unpacker can often be found online. When *PEiD* identifies the executable as a .NET application, the program *dnSpy*<sup>12</sup> is applicable. This tool allows the analyst to use a GUI similar to Visual Studio to debug

and analyse the source code of the executable and often serves the user with a large list of .NET libraries used by the executable to aid in analysis. Another tool that is used is IDA Free<sup>13</sup>, a disassembler serving the user by generating assembly language from the binary and is used to analyse the execution flow of the binary and give a first look at which Windows API calls it will attempt to make when run. It produces a list of strings and allocated variables within the binary, a list of imported functions from the Windows API and gives a nice overview of the code flow using graphs. The program Ghidra<sup>14</sup> is a reverse engineering tool that also disassembles the binary into assembly code like IDA, but goes one step further and make a (decompiled) C code rendition of that same code, which is more readable than assembly and thereby speeds up the analysis process. Therefore, Ghidra is used in the analysis to clear up parts found in the IDA analysis that are large and hard to understand. Both programs have search functions that aid in finding strings and variables. Finally, both have cross-referencing search functions that make it easy to find which parts of the code are used and where they are located in the program.

### A.2 Dynamic analysis

After a static analysis is performed, the dynamic analysis follows. Here the malware is being run in a safe environment to see how it behaves, what files and registry keys it attempts to access and to clear up certain parts of the assembly code that can be debugged to see the internal state of the program at any point during run-time. Also, dynamic analysis unveils what specific parts of code do that remained more opaque in the static analysis. Furthermore, networking attempts can be captured and their contents analysed to reveal more about any processing of information before it is exfiltrated.

As was shortly mentioned earlier in Section A.1, malware can have anti-virtualmachine (anti-VM) techniques that can be used to identify the analysis environment the malware is in. Bayer et al.[2] lists a few of these and they are kept in mind during the dynamic analysis of the malware.

**A.2.1 The process.** We start off by seeing if the samples are known by ‘VirusTotal.com’<sup>15</sup> or ‘AnyRun’<sup>16</sup>. These websites potentially already have reports describing the behaviour and dropped files of the malware samples that can be matched to the ones analysed in this paper by their file hash.

The behaviour is then analysed by firstly running the malware without any analysis programs and observing its behaviour. This can be used as a baseline to see if the behaviour changes, which would indicate the malware is capable of detecting potential debugging or analysis programs analysing it and therefore changing its behaviour. After this dry run the Process Monitor program is used to list the complete set of events that occur on the machine related

<sup>13</sup><https://www.hex-rays.com/ida-free/>

<sup>14</sup><https://ghidra-sre.org/>

<sup>15</sup>VirusTotal is an online database of known malware samples and their behaviour. <https://virustotal.com>

<sup>16</sup>AnyRun is an online automatic malware reporting environment. Public reports of analyses that were run before are usually quite extensive in checking for known malware indicators-of-compromise. [any.run](https://any.run)

<sup>10</sup>FLAREVM comes with an MD5 hashing tool as part of the Malcode Analyst Pack: <http://sandsprite.com/iDef/MAP/>

<sup>11</sup><https://www.aldeid.com/wiki/PEiD>

<sup>12</sup><https://github.com/dnSpy/dnSpy>

to interaction with files, the Windows registry, UDP/TCP connections and process control. After having started Process Monitor, the malware is executed such that the interactive behaviour is logged by Process Monitor for further analysis. Furthermore the tool 'API Monitor' can be used to render a complete list of all Windows API calls the program makes, potentially revealing its use of methods like process injection, privilege escalation or achieving persistence.

Combining the information gathered from looking at the execution flow in the static analysis and the interaction with the operating system monitored by Process Monitor, several interactions can be tracked down as to where and when they happen within the execution of the program by means of debugging.