



Abertay University

Evaluation of the use of Automation in Malware
Analysis

Stuart Rankin, 1701198

BSc Ethical Hacking, 2021

School of Design and Informatics
Abertay University

Table of Contents

Table of Figures	iii
Table of Tables	v
Acknowledgements	vi
Abstract.....	viii
Abbreviations, Symbols and Notation	x
1 Introduction.....	1
1.1 Background and Context	1
1.2 Research Question	2
1.3 Aims.....	3
1.4 Structure	3
2 Literature Review	4
2.1 Malware Analysis Issues.....	4
2.2 Existing Tools	5
2.3 State-of-the-Art Malware.....	7
2.4 Summary	9
3 Methodology.....	10
3.1 Research Stage	10
3.2 Setup	10
3.2.1 Requirements	10
3.3 Static Analysis	11
3.3.1 Basic Script.....	11
3.3.2 FLOSS.....	11
3.3.3 CAPA.....	12
3.3.4 PEFile.....	12
3.3.5 VirusTotal	13
3.3.6 Suspicious	13
3.4 Dynamic Analysis	15
3.5 VM Hardening.....	18

3.6	Testing.....	22
4	Results	23
4.1	Developed Tool.....	23
4.2	Cuckoo Sandbox	24
4.3	VirusTotal.....	25
5	Discussion.....	26
5.1	Samples.....	26
5.2	Tools.....	31
6	Conclusion.....	33
6.1	Future Work.....	35
	List of References	37
	Appendices	41
	Appendix A.....	41
	Hello World	41
	PAFish	42
	WannaCry.....	44
	Mass Logger.....	45
	Upatre	47
	Chthonic.....	49
	Unnamed Malware.....	50
	Appendix B.....	53
	Hello World	53
	PAFish	54
	WannaCry.....	55
	Mass Logger.....	56
	Upatre	57
	Chthonic.....	58
	Unnamed Malware.....	59

Table of Figures

Figure 1. Total Malware Infections 2009-2018 (PurpleSec, 2021).	1
Figure 2. retoolkit's tool folder (github, 2021a).	5
Figure 3. COVID-19 Phishing (McAfee, 2020)	9
Figure 4. FLOSS Suspicious Results of WannaCry.	14
Figure 5. CAPA Suspicious Results of WannaCry.	14
Figure 6. IPv4 Adapter Settings	16
Figure 7. PAFish Results.	19
Figure 8. Device Manager.	20
Figure 9. Regedit.	21
Figure 10. PAFish Results Post-Mitigation 1/2.	22
Figure 11. PAFish Results Post-Mitigation 2/2.	22
Figure 12. HelloWorld Cuckoo Sandbox Summary.	25
Figure 13. Unnamed Sample FLOSS & CAPA highlights.	30
Figure 14. Automa HelloWorld Highlights.	41
Figure 15. Automa PAFish Highlights.	42
Figure 16. PAFish INetSim.	43
Figure 17. Automa WannaCry Highlights.	44
Figure 18. Automa MassLogger Highlights.	45
Figure 19. MassLogger INetSim	46
Figure 20. Automa Upatre Highlights.	47
Figure 21. Upatre Rifle.pdb String.	48
Figure 22. Automa Chthonic Highlights.	49
Figure 23. Automa Unnamed Highlights 1/2.	50
Figure 24. Automa Unnamed Malware 2/2.	51
Figure 25. CAPA debugger.	51
Figure 26. CAPA Network.	51
Figure 27. Unnamed Malware INetSim.	52
Figure 28. HelloWorld Cuckoo Sandbox.	53
Figure 29. PAFish Cuckoo Sandbox.	54
Figure 30. WannaCry Cuckoo Sandbox.	55
Figure 31. MassLogger Cuckoo Sandbox.	56
Figure 32. Upatre Cuckoo Sandbox.	57

Figure 33. Chthonic Cuckoo Sandbox.	58
Figure 34. Unnamed Malware Cuckoo Sandbox.....	59

Table of Tables

Table 1. Automa Tool Results.....	23
Table 2. Automa Suspicious Items.....	24

Acknowledgements

Firstly, I would like to thank my supervisor, Ross Heenan for his guidance and support throughout this project.

I would also like to thank my friends and family for their help in making these the best years of my life so far.

Dedicated to the memory of my dad. If I had been half as hard-working, I am sure this would have been finished a lot sooner.

Abstract

Malware analysis has become more relevant as malware continues to become more advanced. It provides a useful role in the understanding of how malware works which can help to stop or prevent future attacks. However, malware analysis can be time-consuming and difficult, especially for the inexperienced. Therefore, analysis tools have been developed to help. Though these typically have limitations especially when dealing with malware that utilise anti-analysis techniques.

This project aims to develop an automation tool for malware analysis including the detection and mitigation of anti-analysis techniques employed by malware. The tool will also be tested with other available automated malware analysis tools and compared.

The tool was developed for Linux using Python 3 and implemented various malware analysis tools. The static analysis component utilised tools such as FLOSS, CAPA, PEFile and Unipacker. The dynamic and memory analysis functionality used VirtualBox, Volatility, PE-Sieve and INetSim. The resulting data from the tools was formatted and output to an HTML report. To evaluate the effectiveness, it was tested with 7 samples of varying functionality. To help compare, these samples were also tested on Cuckoo Sandbox and VirusTotal.

The tool was found to have varying effectiveness on the samples, with a number of tools having false positives. It did succeed in mitigating anti-analysis techniques with multiple samples. VirusTotal detected every sample as malicious despite the test set containing non-malicious samples. Cuckoo Sandbox would likely have performed better if the documentation had covered virtual machine hardening.

The project identified the issues with current malware analysis. It also highlighted the limitations with Cuckoo Sandbox, including its documentation, setup process and reporting. The testing also showed

how malware analysis requires human input with VirusTotal's 100% detection rate of the test set. With further development and improvements made the tool could greatly aid malware analysts.

Abbreviations, Symbols and Notation

Malware	Malicious Software
PE	Portable Executable
RWX	Read, Write, Execute
VAD	Virtual Address Descriptor
PID	Process ID
C2	Command & Control
VM	Virtual Machine

1 Introduction

1.1 Background and Context

Ever since the advent of the Creeper Worm in 1971, malware has continued to develop and become an ever-increasing, more sophisticated problem. It has evolved into a never-ending game of cat and mouse between the malware developer and those developing protection from these threats such as anti-virus developers. With the total malware infections being on the rise in the last ten years as can be seen in Figure 1, it is likely to continue growing. Malware is also extremely costly to the victims, for example, ransomware attacks alone are estimated to cost \$6 trillion annually by 2021 (PurpleSec, 2021). For example, the NotPetya attack in 2017 was estimated by a White House assessment to have caused \$10 billion dollars in damage. (InfoTransec, 2019)

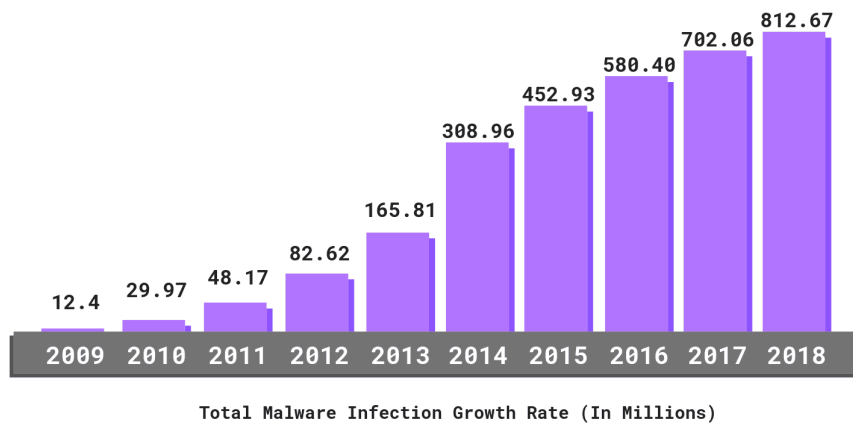


Figure 1. Total Malware Infections 2009-2018 (PurpleSec, 2021).

In response, software such as anti-virus tools and intrusion detection systems have been developed. These however do little to protect systems and networks once an attack has happened and malware has successfully infected a device. Organisations have therefore started creating incident response teams. One key element of this is malware analysis which can allow the team to gain an understanding of what the malware is doing and what it can do. Malware analysis can be incredibly useful, both in incident response and by discovering how a threat works mitigations can be developed and damage caused by an attack reduced.

One key example of this is that of WannaCry where an analyst was able to gain key insight into how WannaCry used a URL as a kill switch and register the domain to stop further spread (Wired, 2017).

Typically, malware analysis is split into two categories, static analysis and dynamic analysis. Static analysis pertains to analysis of the malware's binary, looking at the code and walking through it with the use of tools such as debuggers. Advanced static analysis includes reverse-engineering of the sample to discover exactly what it is doing. Dynamic analysis (or sometimes known as behavioural analysis) involves analysing how the malware behaves once it has been executed, typically by running it on a safe environment such as a virtual machine or a network-isolated PC.

A third type of analysis, memory analysis, has become more relevant in recent years with the increased prevalence of file less and memory malware. Memory analysis includes the examination of the test PC's memory for any malicious or suspicious elements. A computer's memory can often persist for a long time after an action. It also allows the possibility of extracting artefacts completely independently from the system, reducing the likelihood of malware interfering with the results (Ligh *et al.*, 2014)

Malware analysis typically requires a deep understanding of many areas such as malware, reverse-engineering, memory and networking. An understanding of many tools that may be used in the analysis process is also required. There has therefore been a demand for the development of tools that can help aid a malware analyst with the use of ideas such as automation.

1.2 Research Question

The research question for this was project was to investigate how automation of malware analysis tools can aid in the detection and analysis of malware samples.

1.3 Aims

The aims of this project were:

- To research and investigate current malware analysis methods and tools.
- To develop a tool that successfully allows for the automation of various malware analysis tools including static, dynamic and memory analysis tools.
- To test and compare with other available malware analysis tools on a test set of samples that also utilise anti-analysis techniques.

1.4 Structure

Chapter 2 explores previously completed research related to this project including malware analysis techniques and tools, as well as the issues involved with them and the current state of malware.

The methodology covered in Chapter 3 discusses the process that was followed throughout the development stage, including the implementation of the static analysis tools, the virtual machine set-up, dynamic analysis tools and the hardening of the virtual machine.

After the methodology the results of the testing will be provided and the discussion of the found results follows. Chapter 6 highlights any conclusions and any possible future work from this project.

2 Literature Review

The following chapter examines existing issues with malware analysis. As well as this, existing analysis tools were discussed with an emphasis on their limitations. Finally, a discussion on the current state of malware.

2.1 Malware Analysis Issues

Malware analysis has its limitations. The main one previously stated is that malware analysis typically requires a deep knowledge of techniques, effort and skill from the reviewer (Vasilescu, Gheorghe and Tapus, 2014). Static analysis specifically requires knowledge of assembly language and an understanding of the underlying operating system (Uppal, Mehra and Verma, 2014). Due to the time it takes to perform this type of analysis, with the rate of new malware being produced, the field requires automation in order to keep up (Yin and Song, 2012).

According to Gadhiya and Bhasvar one of the biggest limitations with static analysis is the fact the source code of malware samples are not readily available. This reduces the analysis techniques to those that retrieve information from the binary representation of the sample (Gadhiya and Bhavsar, 2013). Static Analysis at machine-code level can also be extremely difficult due to code-obfuscation techniques such as compression, encryption or self-modification (Willems, Holz and Freiling, 2007). Malware authors will know of the limitations of static analysis and will develop malware specifically designed to abuse these limitations.

One issue that dynamic analysis runs into is when malware utilises trigger-based behaviour. Trigger-based behaviour is what it sounds like, simply the malicious file will not perform any malicious behaviour until a trigger is activated. There are various techniques that can be used as a trigger for malware, anything from a date and time to a command received from a server. (Selçuk, Orhan and Batur, 2018). Dynamic analysis will typically fail to correctly analyse the file due to the fact that unless it by chance analyses the file whilst the trigger happens the

sample may seem non-malicious. The process of using static analysis to detect the trigger on the other hand, whilst theoretically possible, would be a massive effort for most samples particularly due to most malware implementing further defensive features that would protect it against static analysis.

2.2 Existing Tools

There exist tools not as widely used such as inhale (github, 2020a), which attempts to analyse and classify malware samples. Whilst a beta release, this tool has obvious limitations in that it only covers static analysis and fails to cover dynamic or memory analysis. There are many tools which fail to offer a hybrid analysis.

Toolkits are also popular by analysts, for example retoolkit (github, 2021a), which once installed creates the folder found in Figure 2. Whilst these tools can be useful it has the same issues as malware analysis in that it still requires knowledge and an understanding of malware analysis techniques and tools to effectively utilise them.

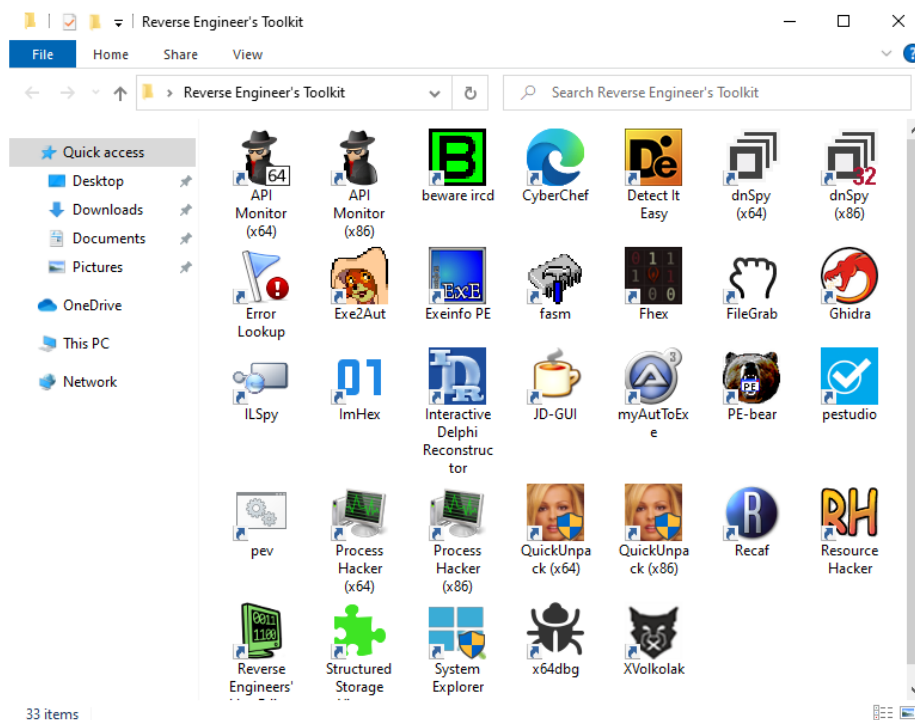


Figure 2. retoolkit's tool folder (github, 2021a).

Tools such as DRAKVUF (DRAKVUF, no date) can provide useful functionality, such as in-depth execution tracing of arbitrary binaries, which can be nearly undetectable from the perspective of the malware. The main problem with this tool is in its hardware limitations, it requires an Intel CPU with VT-x (Intel Virtualisation software) and Extended Page Tables.

One of the most popular automated malware analysis tools is that of Cuckoo Sandbox. Cuckoo Sandbox is an open source malware analysis system that began development in 2010. Malware can avoid analysis by successfully detecting that it is running within a Cuckoo Sandbox environment. One of the easiest ways is simply through the use of shared folder detection. Cuckoo uses a folder on the guest system to share information to the host. By default, on a Windows guest machine the default is C:\Cuckoo. Malware could simply search for a folder of that name containing any possible code and then pause any malicious activity if it does. Another method is pipe detection, due to the fact that Cuckoo's pipe name between the host and guest system is hard coded, malware could quickly check for its presence. Cuckoo also uses an agent python file to handle functionality on the guest machine. Python is a popular language however it is unlikely to find it running on an actual machine. Malware could look for python.exe or pythonw.exe in the running processes. (Ferrand, 2015).

A common technique by malware is to attempt to detect whether it is running in a virtual machine and if the malware finds that it is, it will act benignly. This technique can also be utilised against Cuckoo Sandbox due to its use of virtual machines for dynamic analysis. Typically, virtual machines are detected through hardware information. Malware checks for items such as MAC Addresses being the same as the default for various virtualisation software i.e VirtualBox or VMWare. (Lindorfer, Kolbitsch and Comparetti, 2011). It can also check for information such as unusual RAM sizes, storage sizes, odd number of cores etc. All of these can help

paint a picture of the environment for the malware to decide if it should act maliciously or not.

One of the unique issues with online public malware analysis tools is that malware developers could submit a decoy sample which retrieves the IP address of the analysis machines. This could then be utilised by malware to create a blacklist for which malware would not run if it was on one of these IPs. (Yoshioka, Hosobuchi, Orii and Matsumoto, 2010)

Whilst a lot of sandboxes and malware analysis tools utilise VMs that malware could detect, one unique possible issue with Cuckoo Sandbox is its open-source nature. The majority of tools are closed source and how they work is often kept close to the chest by anti-malware companies. This would allow an attacker to more easily design a malware to detect or escape Cuckoo Sandbox. It could also be used to develop a separate method of evading analysis. Rather than simply acting non-maliciously once it has detected it is in an analysis environment tools can be developed to crash the analysis environment such as anticuckoo (github, 2018). Whilst this might cause an analyst to look more closely at the sample, it could also cause the analyst to have to spend time and money testing Cuckoo Sandbox and implementing new environments in an attempt to find the problem. In that time the malware could have caused a lot more damage. It also is useful due to the fact that Cuckoo Sandbox is often used on samples already known to be malicious to attempt to get a further understanding and by crashing the environment it prevents this dynamic assessment.

2.3 State-of-the-Art Malware

Stuxnet was arguably the first malware to show the large damage that is possible to the world with specifically designed malware. It was the first discovered malware to spy on and subvert industrial systems. Discovered in 2010, believed to be developed by the USA with support from Israel. It was designed specifically to slow the Iranian nuclear program.

Researchers estimated the effort of Stuxnet to be around 5-10 developers

working for six months full-time with access to Scada systems. (Chen and Abu-Nimeh, 2011).

Unlike most common worms it utilised 4 Zero-Day exploits to spread, these methods included USB drives, shared printers and two other vulnerabilities regarding privilege escalation. It targeted PCs running Windows and once it had infected the PC, utilised stolen certificates to download the rootkit and tools for controlling Siemens Simatic WinC/Step-7 software. From here it could access the controllers for industrial devices and launch its attack. The attack was simply changing the speed of the nuclear centrifuges' rotors. This caused irreparable damage and delayed Iran's nuclear program significantly (Baezner and Robin, 2017).

A more recent example of a malware developed by state actors comes from the family of malware, "Foudre". An evolution of the previously known "Infy" malware, the infection vector is a simple spear-phishing email with a malicious Word or PowerPoint document. The malware is known to target dissidents and enemies of the Iranian state, hacking only a handful of targets (Baezner, 2019). It typically performs mostly common attacks such as keylogging, capturing the clipboard data and system information such as browser data. The most notable feature of this attack was when Palo Alto Networks took down the C2 domains using a DNS sinkhole, the Telecommunication Company of Iran blocked Palo Alto Networks through the use of DNS tampering and HTTP filtering (Checkpoint Research, 2021).

Even AstraZeneca has been the target of a nation state attack during the COVID-19 pandemic after suspected North Korean hackers targeted them. The attackers posed as recruiters and approached staff with job offers and documents that would run malicious code when opened. Whilst there was no specifically designed malware utilising zero-days, this case shows how prevalent malware is and how necessary analysis of any files received is. (Stubbs, 2020).

COVID-19 was exploited not just by state actors, with the move to remote working attackers began using it as an opportunity to launch themed attacks. As well as this, attackers targeted remote-workers due to the likelihood of files being stored insecurely at home that previously would have been stored in the company network. (Wang *et al*, 2020). One example of the type of COVID-19 themed phishing malware attacks can be seen below in Figure 3.

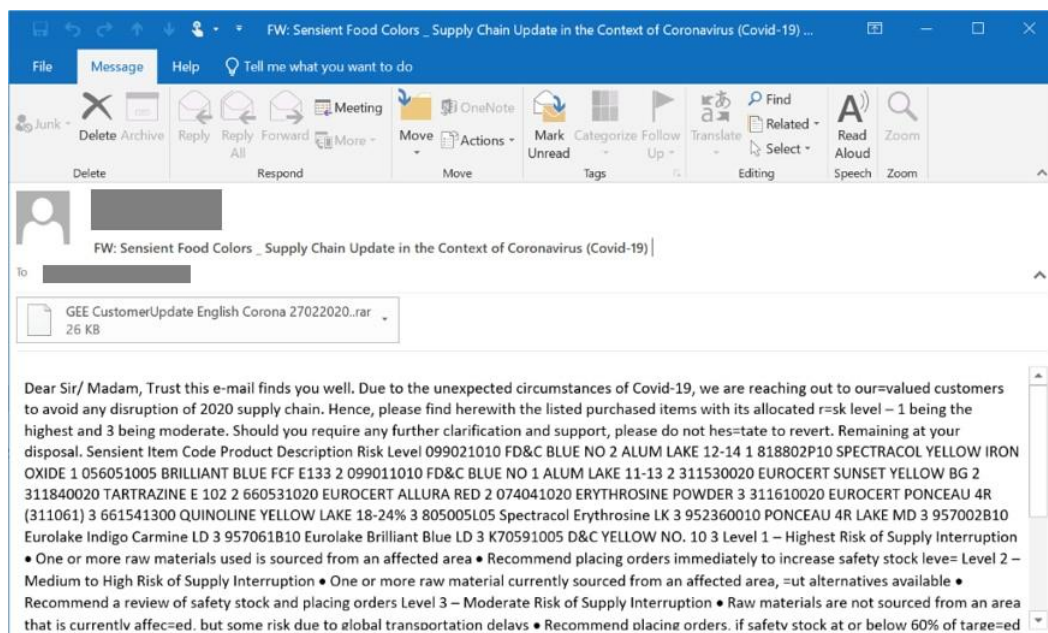


Figure 3. COVID-19 Phishing (McAfee, 2020)

2.4 Summary

The research covered within this chapter has focused on the common issues with malware analysis techniques, the current tools that have been developed as well as their failings. It also sets out the current state of malware and highlights the damage that it can cause and the necessity of automation tools. This project builds upon this necessity and failings of other tools.

3 Methodology

3.1 Research Stage

The initial phase of the methodology was focused on investigating and researching current available tools, including papers and journal articles. This included tools such as Cuckoo Sandbox, REMnux and FLARE VM.

3.2 Setup

3.2.1 Requirements

This project was developed using an iterative prototyping approach. The design for the tool was limited at the start to prevent the prototype of the tool being over developed and out of scope. It was limited to Windows Malware in a PE (Portable Executable) format. Therefore, the tool is not designed to work with any Linux or MacOS malware or any malware that comes in different forms such as PDFs or Microsoft Office file types.

The tool was developed in a Debian 10 Virtual Machine due to the high risk of damage when dealing with live malware samples. Debian was chosen because of its slow update policy leading to the unlikely chance that the tool's dependencies would be updated mid-development. It was also chosen due to the fact that a lot of malware analysis tools are developed with Linux in mind and therefore there are more available as well as more documentation.

The specifications of the PC were an AMD Ryzen 7 2700X, Gigabyte GeForce GTX 1080 WINDFORCE OC and 16GB of 3200MHZ RAM. The Debian VM was allocated 4 Cores and 8Gb of RAM. The ISO was downloaded from the official source and installed following documentation.

3.3 Static Analysis

3.3.1 Basic Script

Initially a basic script was created, “automa.py” (Automated Malware Analysis), this used Python’s argparse to allow for the user to pass in a filename for the sample. Python’s OS library was also used to check the file existed.

The tool was designed to have a class of type “Sample”, this allowed for multiple files to be passed into the script if the user had need for it. The class developed to have various variables such as name, md5, size and a list of suspicious items. This allowed for the analysis functions in the script to use objects of this class to easily analyse multiple files.

Automa.py also had a report functionality that used Python’s file functionality to save an HTML report to the “reports” directory. Most of this functionality was later separated into the file “formatter.py” to simplify the main file.

3.3.2 FLOSS

The first tool that was developed into the project was that of FireEye’s FLOSS (FireEye Labs Obfuscated String Solver). This is an expansion on the basic “strings” functionality that comes with any Linux operating system. It attempts to de-obfuscate any strings it finds in the sample binary. Whilst packing is a common technique by malware authors to hide malicious activity, it can often lead to easy detection due the fact that packing is not a very common technique for benign executables. More advanced techniques include encoding specific strings without encoding the entire file (FireEye, 2016). This was installed to the Debian VM using pip, this added floss to the Python’s path and could be run from command line. The script then ran FLOSS by using the OS library’s system method which provides the ability to run commands as if on the command line.

The first iteration simply printed the result of FLOSS on the sample but was later changed to saving the results as a json string in the sample object. This could then be used to output in the HTML report as a table.

3.3.3 CAPA

The second tool that was added was another FireEye tool, CAPA. CAPA attempts to detect the capabilities and functionality of either a portable executable or shellcode. Unlike FLOSS the tool could not be installed using pip, instead the standalone binary was downloaded and moved to the tool location (github, 2021b). The tool was run by automa using a similar method of OS's system. It outputted to a temporary file, "capa.json" that would then be read and made an object variable. This was also output JSON so that it could be easily converted to the HTML report.

After initial testing it was found that CAPA struggled with binaries that were packed. To deal with these binaries the tool unipacker was used. This is a tool that attempts to unpack a given binary. It supports the most common packers such as UPX, ASPack, FSG (github, 2021c). This was also installed like FLOSS using pip to add it to the Python \$PATH. Automa was developed so that if CAPA found a sample to be packed, the sample would then attempt to be unpacked using OS module to run unipacker on the sample. Once this was done CAPA would run on the unpacked version of the sample the exact same way, saving to the sample object variable.

3.3.4 PEFile

For a more in-depth and advanced data dump which could be analysed PEFile was used. PEFile is a Python module that can parse and work with Portable Executable files. It requires some understanding of PE file formats but has functionality such as inspecting headers, packer detection, warnings for suspicious and malformed values (github, 2021d).

It was simply installed using pip and included in the automa file. The sample would be used to create a PE object using `pe = pefile.PE(sample)`. This could then utilise PEFiles functions such as `pe.dump_info()`. The relevant data was saved to the sample object's variables to later be used when outputting the HTML report.

3.3.5 VirusTotal

VirusTotal is an online tool that provides the ability to upload samples, which can then be scanned and provide information on whether various Anti-Virus software detected them as malicious or not. VirusTotal also offers an API for ease of use, as well as this there is an official library for VirusTotal, `vt-py` (github, 2021e). This was installed using pip and imported into the automa file. The MD5 of the sample was calculated using the tool `md5sum`. This was then used with the VirusTotal API to check whether the file had already been tested and if so, the data was received from VirusTotal. If not, the file was uploaded, scanned and then the data was received. VirusTotal like most APIs require an API Key, for this a personal VirusTotal account was used to generate one. However, this would not work if the tool was to be publicly released, one solution would be to ask the user for their own personal key or to use the private API which has a cost. The data returned from the API used JSON so could easily be implemented into the final report.

3.3.6 Suspicious

A key element of the report is the suspicious section. This is the section near the top of the file which contains the items the tools found that Automa has found to be of interest and that the user is likely to as well. Due to each analysis tool being different, the way Automa selected what was of interest varied. For FLOSS a wordlist was created. This file could easily be edited to add or change more words by the user. This was then used to find any strings found by FLOSS that contained words from the wordlist. The words in the wordlist were created to try and catch the most items and are therefore not extremely specific. For example, in the wordlist is "mal" this is to catch any string containing mal, so malware,

malicious, malloc (C function for memory allocation). This can lead to false positives as seen below in Figure 4 whilst malloc could be used by malicious application, it is also an extremely common function.

FLOSS

- j)LMAL
- MAIR"
- malloc
- }LMal
- UUPx((

Figure 4. FLOSS Suspicious Results of WannaCry.

For CAPA, it utilised specific YARA rules to determine the capabilities. In the resulting JSON file these were split based on the set of rules used. The main way in which serious functionality was determined was using MITRE's ATT&CK and was labelled based on that. Therefore, to report capabilities of note that were not just basic functionality any capabilities labelled ATT&CK were reported in the suspicious section. An example output can be seen in Figure 5.

CAPA

- check if file exists
- create service
- encode data using XOR
- encrypt data using RC4 KSA
- get file size
- get hostname
- link function at runtime
- link many functions at runtime
- parse PE header
- persist via Windows service
- query registry entry
- query registry value
- reference AES constants
- start service

Figure 5. CAPA Suspicious Results of WannaCry.

PEFile has a function simply named get_warnings. This ran when the sample was passed to PEFile and if there were any results these were simply reported in the suspicious section.

If any of the tools on VirusTotal find the sample malicious these are stated in the suspicious section.

3.4 Dynamic Analysis

For dynamic analysis a Windows 7 VM was used. This VM was sourced from Microsoft who offer a publicly available VM that expires after 90 days (Microsoft, 2021).

This was run using Oracle's VirtualBox within the Debian VM machine. The machine's network was initially set as NAT to allow internet access which was made use of to install any necessary tools such as Python. Python 3.9 does not support Windows 7 and therefore Python 3.8.9 was installed.

A NAT network adapter is not suitable for malware usage. This is due to the fact it can access the internet and be used to spread or escape from the VM. The recommended adapter type for malware analysis is Host-Only. This required the setup of a virtual network adapter. This was done by navigating to File > Host Network Manager and creating an adapter. The IP Address/Mask was set as 192.168.56.1/24 and from there the VM could be changed over a host-only type and connected to this virtual adapter. The VM network settings also have to be changed from within the machine. For Windows 7 this is done by changing the adapter settings this can be found by navigating Control Panel > Network and Internet > Network and Sharing Center > Change adapter settings. By right-clicking, clicking Properties and then in the "Internet Protocol Version 4 (TCP/IPv4)" properties, the IP address and DNS server can be manually set. These were set up as can be seen in Figure 6.

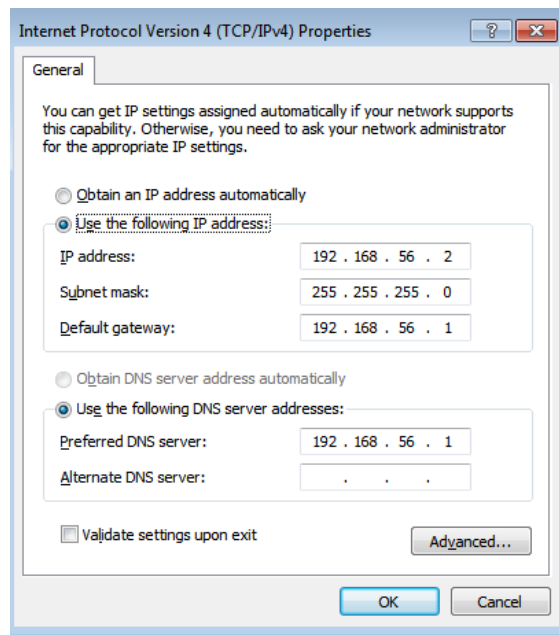


Figure 6. IPv4 Adapter Settings

This only allowed communication between the Host and the VM which was needed for functionality such as file transfer and analysis data transfer. To do this however an agent had to be developed to handle this on the VM side.

On the host machine sockets.py was developed which utilised Python's socket module. This had two main functions, send and receive. The send function took the parameter of a file which was to be sent to the VM and the receive function would write out the file it received from the VM to the host. Whilst it may be possible for the malware to escape by exploiting this method, it would have to be a state-of-the-art designed malware due to the understanding of how the communication works and because of the two different OSs. A similar file was developed with the same functionality on the VM. This allowed for samples to be sent from the host to the VM and for relevant files to be sent back. On the VM another file was developed analyser.py. This was used to run the received sample and track its PID for use in tools. This sockets.py file was run and then the snapshot was saved so that when the VM was reset for the next

sample, the user did not need to log in to the VM and run the sockets file each time to receive the sample.

The VM was temporarily set back to NAT to install procmon and pe-sieve before being set back up as Host Only. Initially procmon was installed on the VM for use in analysis but was not used after the introduction of volatility which was able to perform essentially the same role with more functionality. The first plan was to use moneta over pe-sieve but moneta failed to work on Windows 7 and so instead pe-sieve was employed. Pe-sieve can scan memory for a given process and attempt to detect any malicious implants such as replaced/injected PEs, shellcodes, hooks and in-memory patches (github, 2021f).

Analyser.py used the OS module to run pe-sieve passing the PID from the sample and setting it to dump JSON to the file "pe-sieve.json". Once this was done sockets.py would attempt to send pe-sieve.json to the host for inclusion in the report. Any detections by pe-sieve would be included in the suspicious section of the report.

One key feature of a lot of malware is network functionality, whether this is to reach a command & control server or to spread to more machines. This VM as previously discussed was Host Only and could not access the wider net. To analyse any network traffic INetSim was installed (INetSim, 2021). INetSim is a software suite for simulating common internet services. It has a lot of useful features, for example a sample requested a "jpeg" file rather than returning useless data, it would work and send a generic jpeg back in an attempt to keep the sample functioning. The functionality was included in the automa.py file using OS to run prior to sending the sample to the VM and then shutting it down once the pe-sieve data had been returned or errored out. Once it was shutdown INetSim would generate a report if there had been any traffic, to which Automa would add to the final report. Due to the virtual network adapter having the IP 192.168.56.1 the configuration file for INetSim had to be changed from 127.0.0.1.

As previously stated, Volatility Framework was added to provide more detailed memory analysis. Volatility is a collection of tools for the extraction of artefacts from volatile memory samples (github, 2020b). Volatility requires a memory dump to work on for this, VirtualBox provides a useful feature of being able to dump the memory of a live VM using the command “`VBoxManage debugvm VirtualMachineName dumpvmcore -filename=dump.elf`”. This outputs the memory dump to dump.elf for whatever virtual machine is given as *VirtualMachineName*. There have been many customised and user-built plugins that can be downloaded and implemented. For use in Automa, a set of plugins were downloaded, “Volatility-Plugins” (github, 2021g).

Using Volatility’s documentation these were imported. The first one used RAMSCAN, lists the running processes with their PID and Parent PID and then checks what the virtual address descriptor (VAD) is set to. If the VAD is set to Read, Write and Execute it is marked as suspicious by RAMSCAN. The second one used is CMDCHECK which scans the memory for cmd.exe and checks the standard handles. If cmd.exe is being used for malicious activity such as data exfiltration, it is likely the handles will change and so this can be a good way to check for backdoors or modifications.

3.5 VM Hardening

The virtual machine for dynamic analysis was initially set up without any hardening techniques employed. With some malware utilising VM detection techniques as a way to prevent analysis or to escape the VM and infect the host machine, it is necessary to harden the VM. There exist many tools that use the same techniques implemented by malware to try and detect VMs. One of these, pafish (github, 2019), was used to find how malware could detect the VM. Pafish is a tool that attempts to detect sandboxes and analysis environments whilst also displaying which techniques successfully found evidence of a VM. This was downloaded

onto the Windows 7 VM and run, the result of this can be seen in Figure 7.

```
* Pafish (Paranoid fish) *
Some anti(debugger/UM/sandbox) tricks
used by malware for the general public.

[*] Windows version: 6.1 build 7601
[*] CPU: AuthenticAMD
    Hypervisor: UBoxUBoxUBox
    CPU brand: AMD Ryzen 7 2700X Eight-Core Processor

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing UM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known UM vendors ... traced!

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... traced!
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceEx() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... traced!
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... traced!
[*] Checking if physical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... traced!
[*] Checking if operating system IsNativeUhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbidll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> @ identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... traced!
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\UBOX) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\UBOX) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\UBOX) ... traced!
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\UBox*) ... traced!
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... traced!
[*] Driver files in C:\WINDOWS\system32\drivers\UBox* ... traced!
[*] Additional system files ... traced!
[*] Looking for a MAC address starting with 08:00:27 ... traced!
[*] Looking for pseudo devices ... traced!
[*] Looking for UBoxTray windows ... traced!
[*] Looking for UBox network share ... traced!
[*] Looking for UBox processes (uboxservice.exe, uboxtray.exe) ... traced!
[*] Looking for UBox devices using WMI ... traced!

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> @ identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:5
0:56 ... OK
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VMware serial number ... OK

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> @ identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK

[-] Feel free to RE me, check log file for more information.
```

Figure 7. PAFish Results.

Each line with the red “traced” indicates that the tool detected a VM using that method. As can be seen there were many detections. The first step was to simply change some Virtual Machine settings. The MAC address

was changed from a standard VirtualBox address to a generated MAC address that would not be detected by a list of common VM MAC addresses. The amount of RAM allocated did not need to be adjusted as it was already more than 1GB. The CPU was allocated another core to prevent detection and another for better performance, for a total of 3 virtual cores. The virtualisation settings were also changed, Nested VT-x/AMD-V was disabled and the paravirtualization interface was set to none. The next step was to uninstall VirtualBox's guest tools, this was installed initially as it provides useful features that help the machine perform better and more like a PC, however it installs drivers that can be detected. This was uninstalled following VirtualBox's documentation, one driver was left behind that was still helping pafish detect, "Base System Device" which can be found in Figure 8 which was removed.

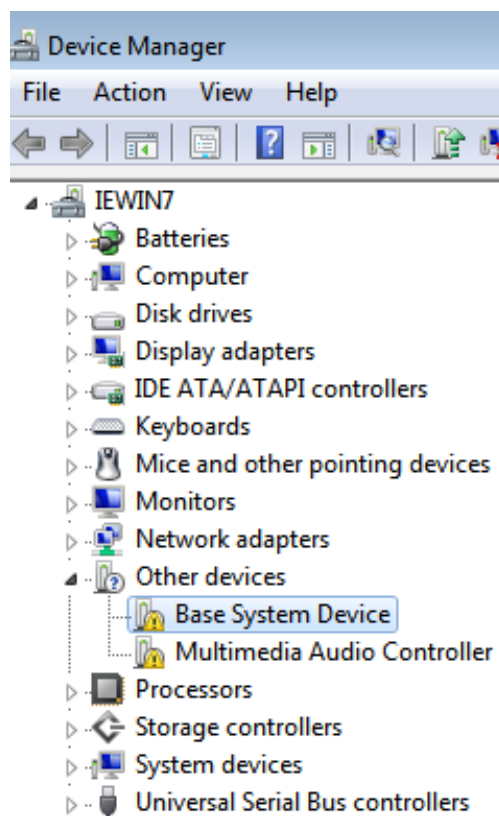


Figure 8. Device Manager.

The next stage was to edit some of the registry keys in the Windows 7 machine. The main ones changed were SystemBiosDate, SystemBiosVersion and VideoBiosVersion. This was done using regedit as can be seen in Figure 9. The Bios date was changed to a more

realistic date of 2019 rather than 1999. SystemBiosVersion and VideoBiosVersion removed any instances of VBox or VirtualBox and replaced with “Gigabyte” that the host machine’s keys contained. The keys in DSDT\VBOX_, FADT\VBOX_, RSDT\VBOX_ were simply removed. These reset on reboot and so the snapshot previously saved was updated to include these edits to the registry.

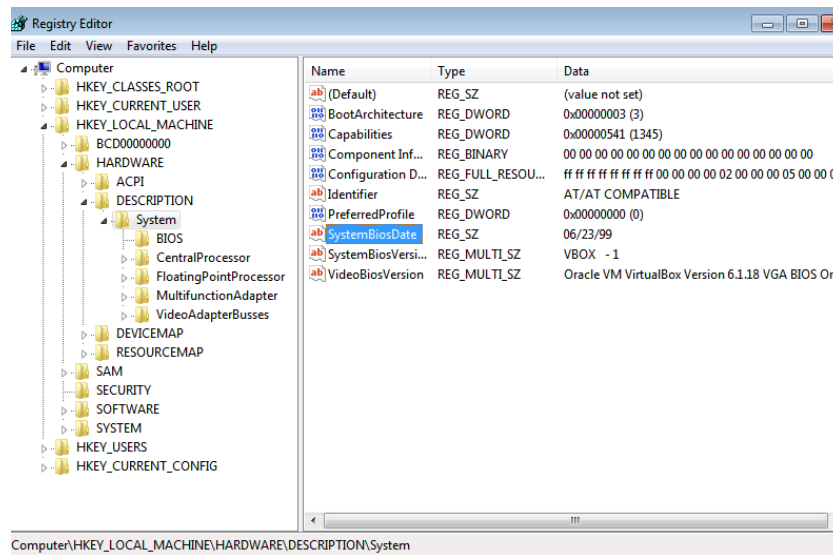


Figure 9. Regedit.

There were also some miscellaneous detections. One of these such as “Using mouse activity”, was solved by changing Automa to boot the snapshot for analysis in headless mode. There was also detection via GetTickCount(), this returns the uptime of the PC. This works by assuming that a virtual machine for analysis will likely be booted up and the sample ran instantly where a more realistic case would be a user on a PC for a certain amount of time before the PC is infected. This was mitigated by having the machine run for a period of time until pafish no longer detected it and then the snapshot for use in Automa was saved.

There were still some detections by pafish as can be seen in Figure 10 and 11. The disk size could not be easily changed due to the fact it was a premade VM from Microsoft. Previously researched malware that implemented similar checks did not check for as high as 60GB and were closer to 10GB. The difference between the CPU timestamp counters

(rdtsc) was also still used to successfully trace the VM. Based on research this could not be easily solved and so was left for future work.

```
Some anti(debugger/VM/sandbox) tricks
used by malware for the general public.

[*] Windows version: 6.1 build 7601
[*] CPU: AuthenticAMD
    CPU brand: AMD Ryzen 7 2700X Eight-Core Processor

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... traced!
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... OK
[*] Checking cpuid hypervisor vendor for known VM vendors ... OK

[-] Generic sandbox detection
[*] Using mouse activity ... traced!
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... traced!
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if physical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeUhdBoot() ... OK
```

Figure 10. PAFish Results Post-Mitigation 1/2.

```
[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\UBOX_) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\UBOX_) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\UBOX_) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\UBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\UBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for UBoxTray windows ... OK
[*] Looking for UBox network share ... OK
[*] Looking for UBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for UBox devices using WMI ... OK
```

Figure 11. PAFish Results Post-Mitigation 2/2.

3.6 Testing

To measure the effectiveness of the developed tool, seven samples were chosen that effectively show the type of samples a tool like this may receive and to show how well the tool deals with them. The first “helloworld.exe” was a simple hello world application written in .NET. The second sample, “pafish.exe” had previously been used to harden the VM but was a useful sample as whilst it uses similar functionality to malware to detect VMs, it is not malicious in and of itself. The first malicious sample is WannaCry.exe, the infamous ransomware previously discussed. The second malicious sample is MassLogger a .NET credential stealer, which implements simple anti-debugging techniques.

The third, Upatre, a downloader tool responsible for delivering additional trojans to the PC of the victim. It uses various anti-analysis techniques such as a 12-minute delay in activity. Chthonic, the next sample is a trojan aimed at banking with some obfuscation techniques. The final sample is an unnamed sample that contains various anti-analysis techniques such as checking for VBox, checking usernames, drive size.

These were all ran through the developed tool and the outputted report saved. They were then also submitted to other available tools, Cuckoo Sandbox and VirusTotal, to compare the effectiveness and the report quality.

4 Results

4.1 Developed Tool

The samples were submitted to the developed tool which produced a report for each. These were then examined and broken into more readable tables. To find if the tools automated by the developed tool successfully ran, the reports were examined for correct output. The results of these can be found in Table 1. As can be seen, PE-Sieve failed to run for the HelloWorld and MassLogger samples.

Sample	MD5 Hash	FLOSS	CAPA	PEFile	Volatility	PESieve	INetSim
HelloWorld	5194ae60ca8803e130f87e5a829d2ac3	✓	✓	✓	✓	x	✓
PaFish	9159edb64c4a21d888d088bf2db23f3	✓	✓	✓	✓	✓	✓
WannaCry	84c82835a5d21bbcf75a61706d8ab549	✓	✓	✓	✓	✓	✓
MassLogger	da06734f45a86b28e5f6e73cdde69ae3	✓	✓	✓	✓	x	✓
Upatre	a0e0a4d830b213ed381084312aef74a3	✓	✓	✓	✓	✓	✓
Chthonic	aba6f9b372254cf34879ddc5283927c9	✓	✓	✓	✓	✓	✓
Unnamed Malware	de1af0e97e94859d372be7fcf3a5daa5	✓	✓	✓	✓	✓	✓

Table 1. Automa Tool Results

The reports were then analysed to find if they found “malicious” items i.e. WannaCry’s network functionality detected by INetSim. This can be found in Table 2. Looking at Table 2, whilst tools may have correctly run, it does

not mean they successfully identified malicious actions. INetSim only found malicious items with 2 samples and PE-Sieve identified one sample with malicious items.

Sample	FLOSS	Capa	PEFile	Volatility	PESieve	INetSim	VirusTotal
HelloWorld	x	x	✓	✓	N/A	x	✓
PaFish	✓	✓	x	✓	x	x	✓
WannaCry	✓	✓	x	✓	✓	x	✓
MassLogger	x	x	✓	✓	N/A	✓	✓
Upatre	✓	✓	x	✓	x	x	✓
Chthonic	✓	✓	✓	✓	x	x	✓
Unnamed Malware	✓	✓	x	✓	x	✓	✓

Table 2. Automa Suspicious Items

The HTML reports provide a better idea of what the tool produced. These can be found in-Appendix A.

4.2 Cuckoo Sandbox

Cuckoo Sandbox was set up following the current documentation. The documentation is lacking when it comes to hardening the guest VM. The default settings within the VM were used without any hardening unless stated otherwise by the documentation. Cuckoo Sandbox uses a Django Web server to host the results of its analysis. However, it also produces an HTML report that contains a summary of the findings. These HTML reports can be found in Appendix B. The summary includes basic file info, detected signatures and network traffic. The summary report of the HelloWorld sample can be found below in Figure 12 as an example.

Summary - helloworld.exe

File info

name: helloworld.exe

type: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows

size: 3584 bytes

Checksums

SHA1 0ba843dfb1475aa36f7b7f239053aa91bcf3bfef

MD5 5194ae60ca8803e130f87e5a829d2ac3

Detected signatures

- i** Checks if process is being debugged by a debugger 2 events
- i** Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available 1 event
- !** Allocates read-write-execute memory (usually to unpack itself) 32 events
- !** One or more thread handles in other processes 1 event
- ⊗** PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 74 events
- ⊗** Malfind detects one or more injected processes 1 event
- ⊗** Stopped Firewall service 1 event
- ⊗** Stopped Application Layer Gateway service 1 event

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	ctldl.windowsupdate.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	www.msftncsi.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 12. HelloWorld Cuckoo Sandbox Summary.

4.3 VirusTotal

The samples were also submitted to VirusTotal. VirusTotal produces a large page of information. The sample reports can be found by following the VirusTotal link at the top of the developed tool's reports or by searching for the MD5 Hash of the sample (provided in Table 1) on the VirusTotal site. The report typically consists of three pages, detection, details and behaviour. The detection page contains a list of the anti-virus tools that VirusTotal submits the sample to as well as the result of their scan. The details page includes the specifics of the file such as the size

and hashes etc. The behaviour section provides further details including the registry keys that were set and deleted and the process and service actions from when the sample was ran in a sandbox environment.

5 Discussion

The chapter will discuss the results and typically follow the order of samples in the Tables above.

5.1 Samples

Sample 1, the Hello world application was a basic .NET application generated to be used in bug testing of the tool. It had no functionality other than printing "Hello World!". Three tools however found some suspicious items, PEFile, VirusTotal and Volatility's RAMSCAN plugin. PEfile found that the Byte 0x00 made up 62% of the file which would be suspicious in a normal application. The VirusTotal result was simply that of one anti-virus software, "MaxSecure" which detected it as a Trojan. It could be that this was due to the same item PEFile found or just a simple false positive. However, VirusTotal uses many anti-virus applications and having only one detect a sample as malicious that is truly malicious would be extremely unusual. The tool could therefore be improved by perhaps implementing a threshold before adding to the report or notifying the user that one detection is not enough to warrant it being a malicious activity. The Volatility items appear to have been universal throughout testing and could possibly be an issue with the plugin, in that it found many services to have a suspicious RWX virtual address descriptor (VAD) as an error. Alternatively, it could have been an issue with the VM and the RWX allocations are used by Microsoft in the specific VM instance. Interestingly, the Cuckoo Sandbox summary report also identified the sample as allocating RWX memory whereas in Automa it is not included in the Volatility. As can be seen in Table 1 PE-Sieve failed to run with this sample and another, MassLogger. Based on bug testing during development, it is likely this is due to the process ending and leaving the memory before PE-Sieve has a chance to scan.

Pafish.exe was the second sample tested. This was an ideal sample in that it had a lot of functionality that to a typical anti-virus software would look malicious. Pafish was used in Section 3.5 to test the VM for detection using malware techniques. Likewise, to HelloWorld Volatility RAMSCAN found several processes with suspicious RWX VAD. However, after analysing the report it appears to include a correct detection of pafish with CAPA also detecting that pafish allocating RWX memory. CAPA also detects a lot of possibly malicious activity such as “execute anti-VM instructions”. VirusTotal had 35 anti-virus tools that detected pafish as malicious whilst some also correctly identified it as “Paranoid Fish”. Using the wordlist, FLOSS identified several strings as suspicious, including obvious items such as “MALWARE” and “VIRUS” and other strings such as “Some anti(debugger/VM/sandbox) tricks” from the word “anti” in the wordlist. Unlike the HelloWorld sample, INetSim captured some network traffic, three DNS connections. After analysis of these requests, they appear to all relate to Windows services and therefore not related to pafish’s functionality. PE-Sieve also successfully scanned the memory for suspicious items unlike the HelloWorld and MassLogger samples but did not find anything of note. The Cuckoo Sandbox report did not contain significant difference compared to Automa in what it detected the sample performing.

The third sample, WannaCry, was the only sample in which PE-Sieve identified a suspicious item. It found an implanted PE in memory, showing the necessity for memory analysis. RAMSCAN found several RWX VADs again including in wannacry.exe, however CAPA did not identify an allocation of RWX VAD so more manual analysis would have to be performed. CAPA did however identify many suspicious features such as “persist via Windows service” which shows at least one of the ways in which WannaCry attempts to persist on a victim’s PC. It also found “reference AES constants” and “encrypt data using RC4 KSA”, presumably used by the ransomware to encrypt the victim’s files. FLOSS did not find much of relevance however it did find a string containing “UPx”, the common packer. Due to the nature of encryption it is likely it

could also just be part of an encryption key or randomly generated string because CAPA or PEFile did not detect any packing. Unsurprisingly the majority of tools on VirusTotal detected WannaCry with the API returning 61/75. The more surprising result is that some tools did not identify it. With further investigation it appears the majority of tools that did not detect the malware was due to the file type being unsupported or some type of failure similar to a timeout. There were a couple tools that failed to detect it such as Arcabit. The original WannaCry famously tried to make a connection to a domain and if it exists the infection stops. Based on both the Cuckoo and Automa report no network traffic was captured so therefore it can be concluded this was a more recent sample with the kill switch removed. WannaCry also utilised an SMB exploit to spread to other PCs, however there was no detection of this by Cuckoo or Automa which both use INetSim. Since INetSim does not currently support SMB functionality, it could perhaps be detected if Automa was developed to allow for a second VM to act as a victim PC to be infected from the original VM and the traffic between the two captured.

Automa struggled with the next sample, MassLogger. There was a lack of useful information detected by the static tools. As stated earlier, PE-Sieve failed with this sample. CAPA detected only three known functionalities that were not malicious. FLOSS did find many strings, but none contained any items from the wordlist. However, VirusTotal API returned a lot of detections with 55/76. Without VirusTotal the main suspicion from the sample comes from INetSim. The traffic captured by INetSim contained two odd requests. One to google.com and kbolias.gr both unlikely to be Windows Services. The traffic captured by Cuckoo Sandbox only contained one suspicious request, the google address. This is likely due to the lack of hardening of the Cuckoo VM. MassLogger utilises a couple anti-analysis techniques including a basic GetTimeCount() to get the uptime of the PC.

The next sample, Upatre had more detected by Automa than the previous Masslogger sample. FLOSS found many strings, but the only item found

by the wordlist appears to be a false positive, "GetCommandLineA" from the word "command" in the wordlist, which was added to try and detect strings referencing C2 servers. It did however manage to identify some decoded strings and stack strings as well as interesting strings that were not caught by the wordlist such as "E:\Data\My Projects\Troy Source Code\tcp1st\rifle\Release\rifle.pdb" which after some research appears to relate to Rifdoor. CAPA also found many useful items such as "persist via Run registry key". There was not much discovered by the dynamic aspect of Automa. Upatre's nature as a downloader would imply network functionality but INetSim found nothing. Some variations of Upatre implement a 12-minute delay before starting any downloading of malware. However, the Cuckoo Sandbox managed to identify communication with the IP "158.69.155.155". Therefore, Cuckoo Sandbox must have captured this simply due to its longer timeout. Automa, does not actually utilise a timeout feature but could greatly benefit from one, instead it only allows time for the malware, PE-Sieve and the VirtualBox memory dump to run.

The second last sample, Chthonic, also had a lack of results from static tools. It had the same FLOSS false positive that Upatre had, "GetCommandLineA". It did also identify references to email addresses and various websites including Symantec. PEFile found suspicious flags set, IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE which can indicate a packed executable. However, Automa due to the way it was programmed only runs Unipacker when CAPA detects a packed sample which it did not. Therefore, Automa could be improved by including PEFile's suspicions as a way to attempt unpacking. CAPA only found three basic items, "copy file", "link function at runtime" and "move file" which could also suggest a packed file due to the lack of functionality identified. RAMSCAN did identify chthonic as having a suspicious RWX VAD but because CAPA did not identify it could not be verified by Automa. However, using the Cuckoo Sandbox report it successfully identified that it allocated RWX memory. The Cuckoo Sandbox also detected a lot more features dynamically such as the sample allocating

execute permission and writing to the memory of another process which could suggest code injection.

Automa was a lot more successful at analysing the final sample - the unnamed malicious sample. FLOSS found a lot of strings that contained items from the wordlist including the obvious “MALWARE”, “VIRUS” and “MALTEST”. It also identified “Keylogger timeout -%i ms” and “keylogger” which clearly suggests the existence of keylogger functionality within the sample. CAPA also found many suspicious functionalities, including various anti-VM techniques. It found various references to anti-VM strings targeting Qemu, VMWare and VirtualBox and checking if the process was running under Wine (a Linux tool to run Windows software). The CAPA results also verify the keylogger functionality with it finding “log keystrokes” and “log keystrokes via polling”. The CAPA results also contained “read clipboard data” and “parse credit card information”. The FLOSS and CAPA highlights have been provided in Figure 13 below.

FLOSS	<ul style="list-style-type: none">• Small+Business+Srv• Keylogger timeout - %i ms• MALTEST• VIRUS• http:shellopen\command• VIRUS• Small+Business+Server+Premium• keylogger• SELECT * FROM AntiVirusProduct• Command received• MALWARE• wpespy.dll
CAPA	<ul style="list-style-type: none">• access PE header• acquire debug privileges• allocate RWX memory• capture screenshot• check if file exists• check if process is running under wine• encode data using Base64• encode data using XOR• enumerate files via kernel32 functions• enumerate processes• get OS version• get disk information• get disk size• get file size• get hostname• get routing table• get session user name• get system information• link function at runtime• log keystrokes• log keystrokes via polling• modify access privileges• parse PE header• parse credit card information• persist via Run registry key• query registry entry• query registry value• read clipboard data• reference anti-VM strings• reference anti-VM strings targeting Qemu• reference anti-VM strings targeting VMWare• reference anti-VM strings targeting VirtualBox• self delete via COMSPEC environment variable• timestamp file• write process memory

Figure 13. Unnamed Sample FLOSS & CAPA highlights.

CAPA also found useful items but were not tagged as ATT&CK and so were not placed in the highlights at the top of the report by Automa. These included various anti-debugger and anti-VM functions like “check for debugger via API” and checking for various sandbox items. There was also more functionality that clearly identified the goal of the malware to capture sensitive data like “capture screenshot” and “get keyboard layout”. It too contained functions that were found in the previous samples for example “persist via Run registry key” and “allocate RWX memory”. The allocation of RWX memory similarly verified the Volatility RAMSCAN result which identified the sample has having an RWX VAD. The CAPA results also contained various references to network functionality such as “resolve DNS”, “send HTTP request” and “send data”. These functions can be confirmed looking at the INetSim report. The INetSim report for this sample included the most data in the test samples with six DNS connections, two of which appeared to be related to Microsoft services. The other four look to be malicious with common malware domain names such as “websitesecurity” and “securitydomains1”. The Cuckoo Sandbox report was lacking in comparison to Automa’s whilst it identified common things such as the sample being packed and the four suspicious domains, it identified various anti-VM methods but due to the fact that the Cuckoo VM was not hardened it could not provide much more data.

5.2 Tools

As expected, the tools each had their pros and cons. As previously stated, Cuckoo Sandbox had a lack of documentation for the size of the tool. The hardening of a VM can be quite a difficult task for an inexperienced user or a truly new user to malware analysis might not consider it a factor. The tool itself is highly configurable with many conf files that can enable or disable tools and many options. These options which can be extremely useful to adjust for specific samples can also be a hindrance in the set-up, with a lot of the useful tools disabled by default.

A lot of the useful data and functionality found by Cuckoo Sandbox is in the Web Server but as far as documentation covers there is not a way to

export this neatly to a report. The only file report Cuckoo Sandbox provides is a summary report that consists (with the configurations made) of basic sample and analysis info, detected signatures and network processes. This is not enough data for an analyst to use. Cuckoo Sandbox is open-source which in the case of malware analysis tools can also be a negative by being easier for a malware author to exploit.

VirusTotal is an extremely useful tool but it still has limitations due to its nature. Due to the number of tools utilised, it can lead to samples being detected which are false positives. This was proven in the testing with the HelloWorld application. It can also have cases with samples like PAFish because of the speed in which analysis is done there is no human element deciding on whether the sample is malicious, leading to assumptions of malware when similar functionality is used. The VirusTotal report also does not provide much detailed information and an analyst would have to carry out further investigation themselves. Unlike Cuckoo Sandbox being free and open-source, VirusTotal can require a paid enterprise account depending on the situation. It also requires an online connection so can be unsuitable on occasion.

Automa whilst a proof-of-concept mitigates a lot of the issues with current malware analysis. Volatility's RAMSCAN as seen in testing had a lot of false positives nonetheless still successfully identified a couple samples with suspicious RWX VADs that were verified by CAPA. CAPA also failed to detect and unpack one sample. The tool has a lot of benefits to a beginner user in that it is a lot simpler than Cuckoo Sandbox for example. There is not a large amount of documentation for the user to read before getting the tool setup. All tools are enabled by default and do not have many configuration settings that have to be adjusted. The tool's output is also relatively easy to understand with perhaps the exception of PEFile's dump. In comparison to Cuckoo Sandboxes summary tool output, Automa had more of a focus on static and memory analysis whereas Cuckoo Sandbox's analysis of a sample's features appear to come from mainly dynamic analysis. This could be useful to implement into Automa

to identify functionality that CAPA missed, perhaps due to packed samples.

The virtual machine used is also hardened but importantly by simply using different methods to Cuckoo Sandbox, it makes the fight harder for malware authors. One of the most effective items in the fight against malware is the idea of the “swiss cheese” defence. No tool is ever going to be perfect but by implementing many different tools, malware must find security gaps in each layer (tool) to successfully spread/infect or avoid analysis. Automa also has similar limitations to other analysis tools, dynamically the strongest anti-analysis method is arguably still time. If a sample is run but does not perform any actions for a 24hr period it is unlikely that the sample will be analysed in the environment for that long, especially with the rate of new samples coming in.

6 Conclusion

Overall, the project was successful in meeting its aims. There were several papers identified that state the need and benefits of automation of malware analysis. The issues with current techniques and tools were also researched and identified. A tool was developed that allowed for the automation of various malware analysis tools. They covered static, dynamic and memory analysis tools. The tool outputted a formatted report of the results, as well as including a highlights section to attempt to bring any suspicious items to the forefront of the analyst reading the report.

The tool was also compared with other malware analysis automation tools, Cuckoo Sandbox and VirusTotal. They were tested with seven samples that had a range of functionality from basic hello world to complex malware with anti-analysis techniques. The testing revealed the effectiveness as well as the limitations of the proof-of-concept tool. It was rather effective at producing a readable report with relevant data for a malware analyst. It had more basic setup than Cuckoo Sandbox which has various configuration files and a large number of settings. The report

also provided a lot more data than Cuckoo Sandbox did which uses its Web Server functionality to display a lot of the found results. It also provided a lot more in-depth data when compared to VirusTotal which typically displays only a binary malicious or not result from anti-virus vendors.

The tools fought against various anti-analysis techniques. This includes basic anti-analysis like unipacker to unpack samples. FLOSS was used to improve upon the common usage of the “strings” command in analysis as FLOSS attempts to fight against common techniques such as obfuscation and encoding. The virtual machine used for dynamic analysis was hardened using Paranoid Fish in an attempt to prevent any detection by malware to detect its presence in a sandbox environment. This included the editing or removal of certain registry keys, removal of specific drivers and the MAC Address being changed. A lot of malware can avoid detection using malicious memory techniques so tools such as Volatility and PE-Sieve were implemented to detect any malware using memory. CAPA was used to detect various functionality of a given sample which could include anti-VM techniques as proven in testing. These had their limitations which included some false positives and a lack of useful output with certain samples. The fight against some anti-analysis techniques would be at the discretion of the analyst, for example a malware that waits to perform any malicious activity. It would be unrealistic to analyse the sample for the time required if it is a lot larger than that used by some samples such as Chthonic which waits 12 minutes.

Simply by being another tool that malware authors have to prevent detection by it can make the job a lot more challenging for them. Tools like Cuckoo use by default, standardised methods. These include items such as directory names or usernames. Automa being in its infancy and unknown state provide another variation. For example, Cuckoo Sandbox uses pipes as one of the ways to communicate with the sandbox environment and malware can attempt to detect the presence of specific

pipes. Automa on the other hand currently only utilises network sockets, whether that is sending the sample or receiving analysis data.

Overall, automation is a necessary area of malware analysis with the rate of new samples. With the limited number of tools available malware can try and defend against them so more tools are becoming more important. The need for a human element can also be seen with VirusTotal's false positives in testing. Automa's choice of tools covers a wide range of analysis areas. Automa also successfully improves upon aspects of tools like Cuckoo Sandbox with a more detailed report. It does have limitations as a proof-of-concept when compared to fully developed tools, but these could easily be mitigated with development time.

6.1 Future Work

Many mitigations were identified during development and testing that could be developed upon to improve the tool. There is possibly issues yet to be identified so more testing could always be done, especially on newer samples with more advanced features. Many of the issues are relatively simple including improving the formatting of the report. Currently the report is data driven and can sometimes be too long depending on the output tools. The output of tools like FLOSS could be added to a collapsible list depending on the length.

The usability could also be improved, currently the documentation is relatively short whilst this is useful a new user may run into an issue due to lack of experience. The tool could be improved by adding some configuration but unlike Cuckoo Sandbox the tools would be better enabled by default however settings such as timeout length could be extremely beneficial. On that note, the tool could benefit from a set timeout for analysis purposes. Instead of relying on the length it takes to run the sample and take a memory dump which would allow for consistency. Similarly, a lot of the code base uses poor methods i.e it powers on the virtual machine and waits 5 seconds for boot but if a user was on a slower PC the virtual machine might not be ready to receive the

sample after 5 seconds. Instead, it would be beneficial to wait for communication from the analysis VM to confirm it is ready.

A lot of features could be added that are inspired by Cuckoo Sandbox as it is a well-developed and feature full tool. The Web Server functionality could be extremely useful to setup a separate dedicated analysis machine that could receive samples through the network. One other feature of Cuckoo Sandbox which was useful which was stated previously, was the dynamic approach to finding a samples functionality. Whilst CAPA is extremely useful and effective it can sometimes fail, as was seen in testing when dealing with packed samples and by also using dynamic approaches it could verify any malicious functionality found. The tools already in Automa could also be improved upon. One main way would be dealing with the RAMSCAN issue that testing found. This could be fixed by dealing with the RWX memory within the VM, implementing a filter on these specific processes or it could also just be an issue with the plugin which could be fixed or replaced.

There is always going to be improvements in anti-analysis techniques that Automa will have to deal with. Malware has already been found that waits on a user's mouse movement before beginning malicious activity. Since Automa is programmed to boot the virtual machine in headless mode those samples would avoid dynamic analysis. There were also some methods of detection found by Paranoid Fish such as the size of disk being less than 60GB. These could be identified and mitigated as well.

List of References

Baezner, M. and Robin, P. (2017) 'Stuxnet' *ETH Zurich* doi: 10.3929/ethz-b-000200661.

Baezner, M. (2019) 'Iranian Cyber-activities in the Context of Regional Rivalries and International Tensions'. *ETH Zurich*. doi: 10.3929/ethz-b-000344841

Checkpoint Research (2021) *After Lightning Comes Thunder*. Available at: <https://research.checkpoint.com/2021/after-lightning-comes-thunder/> (Accessed: 19 April 2021).

Chen, T. and Abu-Nimeh, S. (2011) 'Lessons from Stuxnet' *Computer*, vol. 44, no. 4, pp. 91-93 doi: 10.1109/MC.2011.115.

DRAKVUF (no date). *DRAKVUF*. Available at: <https://drakvuf.com/> (Accessed: 22 April 2021).

FireEye (2016). *Automatically Extracting Obfuscated Strings From Malware*. Available at: <https://www.fireeye.com/blog/threat-research/2016/06/automatically-extracting-obfuscated-strings.html> (Accessed: 20 April 2021).

Ferrand, O. (2015). 'How to detect the Cuckoo Sandbox and to Strengthen it?', *J Comput Virol Hack Tech* 11, pp. 51–58. doi: 10.1007/s11416-014-0224-9.

Gadhiya, S., Bhavsar, K. (2013) 'Techniques for malware analysis', *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4), pp.2277-128.

Github (2018). *anticuckoo, David-Reguera-Garcia-Dreg*. Available at: <https://github.com/mentebinaria/retoolkit> (Accessed: 21 April 2021).

Github (2019) *pafish*, *aOrtega*. Available at:
<https://github.com/aOrtega/pafish> (Accessed: 26 April 2021).

Github (2020a). *Inhale*, *netspooky*. Available at:
<https://github.com/netspooky/inhale> (Accessed: 20 April 2021).

Github (2020b) *volatility*, *volatilityfoundation*. Available at:
<https://github.com/volatilityfoundation/volatility> (Accessed: 25 April 2021).

Github (2021a). *retoolkit*, *mentebinaria*. Available at: <https://github.com/mentebinaria/retoolkit> (Accessed: 21 April 2021).

Github (2021b). *capa*, *fireeye*. Available at: <https://github.com/fireeye/capa>
(Accessed: 21 April 2021).

Github (2021c). *unipacker*, *unipacker*. Available at:
<https://github.com/unipacker/unipacker> (Accessed: 21 April 2021).

Github (2021d). *pefile*, *erocarrera*. Available at:
<https://github.com/erocarrera/pefile> (Accessed: 22 April 2021).

Github (2021e). *vt-py*, *VirusTotal*. Available at:
<https://github.com/VirusTotal/vt-py> (Accessed: 22 April 2021).

Github (2021f). *pe-sieve*, *hasherezade*. Available at:
<https://github.com/hasherezade/pe-sieve> (Accessed: 24 April 2021).

Github (2021g) *volatility-plugins*, *TazWake*. Available at:
<https://github.com/TazWake/volatility-plugins> (Accessed: 25 April 2021).

INetSim (2020). *INetSim*. Available at: <https://www.inetsim.org/>
(Accessed: 24 April 2021).

InfoTransec (2019) *The Impacts of NotPetya Ransomware: What You Need to Know*. Available: <https://infotransec.com/news/the-impacts-of-notpetya-ransomware-what-you-need-to-know/> (Accessed: 29 April 2021).

Ligh, M. et al. (2014) *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux and Mac Memory*. Germany: Wiley.

Lindorfer M., Kolbitsch C. and Comparetti P. (2011) 'Detecting Environment-Sensitive Malware', *Recent Advances in Intrusion Detection. RAID 2011*. vol 6961. pp. 338-358 doi: 10.1007/978-3-642-23644-0_18.

McAfee (2020). *COVID-19 – Malware Makes Hay During a Pandemic*. Available at: <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/covid-19-malware-makes-hay-during-a-pandemic/> (Accessed: 2 May 2021).

Microsoft (2021) *Virtual Machines*. Available at: <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/> (Accessed: 23 April 2021).

PurpleSec (2021) *2021 Cyber Security Statistics*. Available at: <https://purplesec.us/resources/cyber-security-statistics/> (Accessed: 20 April 2021).

Selçuk, A. Orhan, F. and Batur, B. (2018) 'Intractable Problems in Malware Analysis and Practical Solutions'. *Journal of Internet Technology and Secured Transactions*, 6(2), pp. 588-595. doi: 10.20533/jitst.2046.3723.2018.0072.

Stubbs, J. (2020) 'Exclusive: Suspected North Korean hackers targeted COVID vaccine maker AstraZeneca', *Reuters*, 27 November. Available at: <https://www.reuters.com/article/us-healthcare-coronavirus-astrazeneca-no-idUSKBN2871A2> (Accessed: 30 April 2021)

Uppal, D., Mehra, V and Verma, V. (2014) 'Basic survey on Malware Analysis, Tools and Techniques', *International Journal on Computational Sciences & Applications*, Vol 4. pp. 103-112. doi:10.5121/ijcsa.2014.4110

Vasilescu, M., Gheorghe, L. and Tapus, N. (2014) 'Practical malware analysis based on sandboxing', *2014 RoEduNet Conference 13th Edition: Networking in Education and Research Join Event RENAM 8th Conference*, pp. 1-6, doi: 10.1109/RoEduNet-RENAM.2014.6955304.

Wang, L. *et al.* (2020). 'Beyond the virus: A first look at coronavirus-themed mobile malware'. *arXiv e-prints*.

Willems, C., Holz, T. and Freiling, F. (2007) 'Toward Automated Dynamic Malware Analysis Using CWSandbox', *IEEE Security & Privacy*, Vol. 5, no. 2, pp. 32-39. doi: 10.1109/MSP.2007.45.

Wired (2017). *How an Accidental 'Kill Switch' slowed Friday's Massive Ransomware Attack*. Available at: <https://www.wired.com/2017/05/accidental-kill-switch-slowed-fridays-massive-ransomware-attack/> (Accessed: 14 April 2021).

Yin, H., Song, D. (2012). *Automatic Malware Analysis*. New York: Springer.

Yoshioka, K., Hosobuchi, Y., Orii, T., and Matsumoto, T. (2010) 'Vulnerability in Public Malware Sandbox Analysis Systems' *10th IEEE/IPSJ International Symposium on Applications and the Internet*, pp. 265-268, doi: 10.1109/SAINT.2010.16.

Appendices

Appendix A

Due to the length of the reports they could not easily be included in their entirety. Instead any items mentioned in section 4 and 5 have been included and the full Automa report can be found in the tool's github repository under the reports directory which can be found:

<https://github.com/Rankin2000/automa>

Hello World

helloworld.exe

Filename helloworld.exe
MD5 5194ae60ca8803e130f87e5a829d2ac3
ImpHash f34d5f2d4577ed6d9ceec516c1f5a744
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">• The process svchost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process cmd.exe was also found to have Suspicious RWX VAD• The process conhost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process python.exe was also found to have Suspicious RWX VAD• The process SearchProtocol was also found to have Suspicious RWX VAD• The process SearchFilterHo was also found to have Suspicious RWX VAD
VirusTotal	<ul style="list-style-type: none">• MaxSecure
pefile	<ul style="list-style-type: none">• Byte 0x00 makes up 61.8862% of the file's contents. This may indicate truncation / malformation.

Figure 14. Automa HelloWorld Highlights.

PAFish

pafish.exe

Filename pafish.exe
MD5 9159edb64c4a21d8888d088bf2db23f3
ImpHash 5fd4caa76ea3c961f2d530674634f64d
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">• The process svchost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process cmd.exe was also found to have Suspicious RWX VAD• The process conhost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process python.exe was also found to have Suspicious RWX VAD• The process SearchProtocol was also found to have Suspicious RWX VAD• The process pafish.exe was also found to have Suspicious RWX VAD• The process SearchFilterHo was also found to have Suspicious RWX VAD
VirusTotal	<ul style="list-style-type: none">• Bkav• FireEye• Cylance• Zillya• SUPERAntiSpyware• Sangfor• K7AntiVirus• Alibaba• K7GW• Cyren• ESET-NOD32• APEX• Avast• Kaspersky• NANO-Antivirus• Paloalto• ViRobot• Rising• Comodo• VIPRE• Sophos• Jiangmin• Webroot• MAX• Kingsoft• AegisLab• Cynet• AhnLab-V3• ALYac• TACHYON• Malwarebytes• Ikarus• Fortinet• AVG• Qihoo-360
FLOSS	<ul style="list-style-type: none">• Some anti(debugger/VM/sandbox) tricks• C:\WINDOWS\system32\vbboxglpackspu.dll• used by malware for the general public.• VIRUS• malloc• VIRUS• The result is too small to be represented (UNDERFLOW)• %smalware.exe• MALWARE
CAPA	<ul style="list-style-type: none">• allocate RWX memory• check for unmoving mouse cursor• check if process is running under wine• enumerate processes• execute anti-VM instructions• find graphical window• get OS version• get disk information• get disk size• get local IPv4 addresses• get memory capacity• get process heap force flags• get session user name• get system information• link function at runtime• query registry entry• query registry value• reference anti-VM strings• reference anti-VM strings targeting Parallels• reference anti-VM strings targeting Qemu• reference anti-VM strings targeting VMWare• reference anti-VM strings targeting VirtualBox• reference anti-VM strings targeting Xen

Figure 15. Automa PAFish Highlights.

INetSim

=== Report for session '23662' ===

Real start date : 2021-04-26 15:24:13
Simulated start date : 2021-04-26 15:24:13
Time difference on startup : none

2021-04-26 15:24:14 First simulated date in log file
2021-04-26 15:24:14 DNS connection, type: A, class: IN, requested name: teredo.ipv6.microsoft.com
2021-04-26 15:24:15 DNS connection, type: A, class: IN, requested name: www.msftncsi.com
2021-04-26 15:24:15 DNS connection, type: A, class: IN, requested name: ipv6.msftncsi.com
2021-04-26 15:24:15 Last simulated date in log file

===

Figure 16. PAFish INetSim.

WannaCry

wannacry.exe

Filename wannacry.exe
MD5 84c82835a5d21bbcf75a61706d8ab549
ImpHash 68f013d7437aa653a8a98a05807afeb1
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">The process svchost.exe was also found to have Suspicious RWX VADThe process WmiPrvSE.exe was also found to have Suspicious RWX VADThe process cmd.exe was also found to have Suspicious RWX VADThe process conhost.exe was also found to have Suspicious RWX VADThe process WmiPrvSE.exe was also found to have Suspicious RWX VADThe process python.exe was also found to have Suspicious RWX VADThe process SearchProtocol was also found to have Suspicious RWX VADThe process wannacry.exe was also found to have Suspicious RWX VADThe process SearchFilterHo was also found to have Suspicious RWX VAD
PE-Sieve	<ul style="list-style-type: none">Found 1 modules that were implanted_pe
VirusTotal	<ul style="list-style-type: none">BkavElasticDrWebMicroWorld-eScanFireEyeCAT-QuickHealMcAfeeMalwarebytesZillyaSangforK7AntiVirusAlibabaK7GWCybereasonBitDefenderThetaCyrenSymantecESET-NOD32APEXAvastClamAVKasperskyBitDefenderNANO-AntivirusPaloaltoViRobotRisingAd-AwareEmsisoftComodoBaiduVIPRETrendMicroMcAfee-GW-EditionSophosIkarusJiangminWebrootAviraMAXKingsoftGridinsoftMicrosoftAegisLabZoneAlarmGDataCynetAhnLab-V3AcronisVBA32TACHYONZonerTrendMicro-HouseCallTencentYandexSentinelOneMaxSecureFortinetAVGPandaCrowdStrike
FLOSS	<ul style="list-style-type: none">j)LMALMAIR*malloclJlMalUUPx((
CAPA	<ul style="list-style-type: none">check if file existscreate serviceencode data using XORencrypt data using RC4 KSAget file sizeget hostnamelink function at runtimelink many functions at runtimeparse PE headerpersist via Windows servicequery registry entryquery registry valuereference AES constantsstart service

Figure 17. Automa WannaCry Highlights.

Mass Logger

masslogger.exe

Filename masslogger.exe
MD5 da06734f45a86b28e5f6e73cdde69ae3
ImpHash f34d5f2d4577ed6d9ceec516c1f5a744
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">• The process svchost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process cmd.exe was also found to have Suspicious RWX VAD• The process conhost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process python.exe was also found to have Suspicious RWX VAD• The process SearchProtocol was also found to have Suspicious RWX VAD• The process SearchFilterHo was also found to have Suspicious RWX VAD• The process MpCmdRun.exe was also found to have Suspicious RWX VAD
VirusTotal	<ul style="list-style-type: none">• Elastic• MicroWorld-eScan• CAT-QuickHeal• McAfee• Cylance• Zillya• Sangfor• CrowdStrike• Alibaba• K7GW• K7AntiVirus• TrendMicro• Cyren• Symantec• APEX• Avast• Kaspersky• BitDefender• NANO-Antivirus• Paloalto• ViRobot• Ad-Aware• Sophos• Comodo• F-Secure• VIPRE• Invincea• McAfee-GW-Edition• FireEye• Emsisoft• Ikarus• Jiangmin• Webroot• Avira• Antiy-AVL• Microsoft• Gridinsoft• Arcabit• AegisLab• ZoneAlarm• GData• VBA32• ALYac• MAX• Malwarebytes• Zoner• ESET-NOD32• TrendMicro-HouseCall• Rising• Fortinet• BitDefenderTheta• AVG• Cybereason• Panda• Qihoo-360

Figure 18. Automa MassLogger Highlights.

INetSim

=== Report for session '25605' ===

Real start date : 2021-04-26 15:37:31
Simulated start date : 2021-04-26 15:37:31
Time difference on startup : none

2021-04-26 15:37:48 First simulated date in log file
2021-04-26 15:37:48 DNS connection, type: A, class: IN, requested name: google.com
2021-04-26 15:37:49 DNS connection, type: A, class: IN, requested name: kbolias.gr
2021-04-26 15:37:49 DNS connection, type: A, class: IN, requested name: teredo.ipv6.microsoft.com
2021-04-26 15:37:55 DNS connection, type: A, class: IN, requested name: spynt2.microsoft.com
2021-04-26 15:37:55 Last simulated date in log file

===

Figure 19. MassLogger INetSim

Upatre

upatre.exe

Filename upatre.exe
MD5 a0e0a4d830b213ed381084312aef74a3
ImpHash 7c74db2070ce18d4e7700897ccc24743
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">• The process svchost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process cmd.exe was also found to have Suspicious RWX VAD• The process conhost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process python.exe was also found to have Suspicious RWX VAD• The process SearchProtocol was also found to have Suspicious RWX VAD• The process SearchFilterHo was also found to have Suspicious RWX VAD• The process upatre.exe was also found to have Suspicious RWX VAD• The process wuaucnt.exe was also found to have Suspicious RWX VAD• The process MpCmdRun.exe was also found to have Suspicious RWX VAD
VirusTotal	<ul style="list-style-type: none">• Bkav• Elastic• DrWeb• MicroWorld-eScan• FireEye• CAT-QuickHeal• ALYac• Cylance• Zillya• SUPERAntiSpyware• Sangfor• K7AntiVirus• Allbaba• K7GW• CrowdStrike• Arcabit• BitDefenderTheta• Cyren• Symantec• ESET-NOD32• APEX• Avast• ClamAV• Kaspersky• BitDefender• NANO-Antivirus• Paloalto• ViRobot• Tencent• Ad-Aware• Emsisoft• Comodo• VIPRE• TrendMicro• McAfee-GW-Edition• Sophos• Ikarus• Jiangmin• Avira• Kingsoft• Gridinsoft• Microsoft• ZoneAlarm• GData• Cynet• AhnLab-V3• Acronis• McAfee• MAX• VBA32• Malwarebytes• TrendMicro-HouseCall• Rising• Yandex• SentinelOne• eGambit• Fortinet• AVG• Panda• Qihoo-360
FLOSS	<ul style="list-style-type: none">• GetCommandLineA
CAPA	<ul style="list-style-type: none">• encode data using XOR• get OS version• get file size• get hostname• get session user name• link function at runtime• link many functions at runtime• parse PE header• persist via Run registry key• query registry entry• query registry value• write and execute a file

Figure 20. Automa Upatre Highlights.

- 190,11E
- E:\Data\My Projects\Troy Source Code\tcp1st\rifle\Release\rifle.pdb
- WS2_32.dll

Figure 21. Upatre Rifle.pdb String.

Chthonic

chthonic.exe

Filename chthonic.exe
MD5 aba6f9b372254cf34879ddc5283927c9
ImpHash 240ec44ac4fd9c1359c77209719a1838
VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

- The process svchost.exe was also found to have Suspicious RWX VAD
 - The process WmiPrvSE.exe was also found to have Suspicious RWX VAD
 - The process cmd.exe was also found to have Suspicious RWX VAD
 - The process conhost.exe was also found to have Suspicious RWX VAD
 - The process WmiPrvSE.exe was also found to have Suspicious RWX VAD
 - The process python.exe was also found to have Suspicious RWX VAD
 - The process SearchProtocol was also found to have Suspicious RWX VAD
 - The process SearchFilterHo was also found to have Suspicious RWX VAD
 - The process chthonic.exe was also found to have Suspicious RWX VAD
 - The process msixexec.exe was also found to have Suspicious RWX VAD
 - The process MpCmdRun.exe was also found to have Suspicious RWX VAD
- Volatility
RAMScan
- Elastic
 - MicroWorld-eScan
 - McAfee
 - Cylance
 - VIPRE
 - K7AntiVirus
 - Alibaba
 - K7GW
 - Cybereason
 - Invincea
 - BitDefenderTheta
 - Symantec
 - ESET-NOD32
 - TrendMicro-HouseCall
 - Paloalto
 - ClamAV
 - Kaspersky
 - BitDefender
 - NANO-Antivirus
 - Avast
 - Rising
 - Ad-Aware
 - Comodo
 - F-Secure
 - DrWeb
 - Zillya
- VirusTotal
- TrendMicro
 - FireEye
 - Sophos
 - APEX
 - GData
 - Jiangmin
 - Avira
 - MAX
 - Arcabit
 - AegisLab
 - ZoneAlarm
 - Microsoft
 - Cynet
 - AhnLab-V3
 - Acronis
 - ALYac
 - VBA32
 - Ikarus
 - Tencent
 - SentinelOne
 - Fortinet
 - Webroot
 - AVG
 - Panda
 - CrowdStrike
 - Qihoo-360
- pefile
- Suspicious flags set for section 0. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed executable.
- FLOSS
- GetCommandLineA
 - jamal@eneroexpress.com0
- CAPA
- link function at runtime

Figure 22. Automa Chthonic Highlights.

Unnamed Malware

Backdoor.Win32.Agent.dkbp.de1af0e97e94859c

Filename Backdoor.Win32.Agent.dkbp.de1af0e97e94859d372be7fc3a5daa5.exe

MD5 de1af0e97e94859d372be7fc3a5daa5

ImpHash aac2c65404c24ea498a73f89af6cba62

VirusTotal [Detection](#)

Here are some items that Automa found to be suspicious:

Volatility RAMScan	<ul style="list-style-type: none">• The process svchost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process cmd.exe was also found to have Suspicious RWX VAD• The process conhost.exe was also found to have Suspicious RWX VAD• The process WmiPrvSE.exe was also found to have Suspicious RWX VAD• The process python.exe was also found to have Suspicious RWX VAD• The process Backdoor.Win32 was also found to have Suspicious RWX VAD• The process SearchProtocol was also found to have Suspicious RWX VAD• The process SearchFilterHo was also found to have Suspicious RWX VAD• The process MpCmdRun.exe was also found to have Suspicious RWX VAD
VirusTotal	<ul style="list-style-type: none">• Bkav• Elastic• MicroWorld-eScan• FireEye• CAT-QuickHeal• McAfee• Cylance• Zillya• Sangfor• K7AntiVirus• Alibaba• K7GW• Cybereason• Cyren• Symantec• TotalDefense• APEX• Avast• Kaspersky• BitDefender• NANO-Antivirus• Paloalto• Tencent• Ad-Aware• Emsisoft• Comodo• DrWeb• VIPRE• TrendMicro• McAfee-GW-Edition• Sophos• Ikarus• GData• ESET-NOD32• eGambit• Avira• Kingsoft• Arcabit• AegisLab• Microsoft

Figure 23. Automa Unnamed Highlights 1/2.

- Cynet
 - AhnLab-V3
 - Acronis
 - BitDefenderTheta
 - ALYac
 - MAX
 - VBA32
 - Malwarebytes
 - TrendMicro-HouseCall
 - Rising
 - Yandex
 - SentinelOne
 - MaxSecure
 - Fortinet
 - Webroot
 - AVG
 - Panda
 - CrowdStrike
 - Qihoo-360
- FLOSS
- Small+Business+Srv
 - Keylogger timeout - %i ms
 - MALTEST
 - \VIRUS
 - http\shell\open\command
 - VIRUS
 - Small+Business+Server+Premium
 - keylogger
 - SELECT * FROM AntiVirusProduct
 - Command recived
 - MALWARE
 - wpspy.dll
- CAPA
- access PE header
 - acquire debug privileges
 - allocate RWX memory
 - capture screenshot
 - check if file exists
 - check if process is running under wine
 - encode data using Base64
 - encode data using XOR
 - enumerate files via kernel32 functions
 - enumerate processes
 - get OS version
 - get disk information
 - get disk size
 - get file size
 - get hostname
 - get routing table
 - get session user name
 - get system information
 - link function at runtime
 - log keystrokes
 - log keystrokes via polling
 - modify access privileges
 - parse PE header
 - parse credit card information
 - persist via Run registry key
 - query registry entry
 - query registry value
 - read clipboard data
 - reference anti-VM strings
 - reference anti-VM strings targeting Qemu
 - reference anti-VM strings targeting VMWare
 - reference anti-VM strings targeting VirtualBox
 - self delete via COMSPEC environment variable
 - timestamp file
 - write process memory

Figure 24. Automa Unnamed Malware 2/2.

- check for debugger via API
- check for sandbox and av modules
- check for sandbox username
- check for software breakpoints

Figure 25. CAPA debugger.

- resolve DNS
- self delete via COMSPEC environment variable
- send HTTP request
- send HTTP request with Host header

Figure 26. CAPA Network.

INetSim

=== Report for session '26462' ===

Real start date : 2021-04-26 15:44:51
Simulated start date : 2021-04-26 15:44:51
Time difference on startup : none

2021-04-26 15:44:56 First simulated date in log file
2021-04-26 15:44:56 DNS connection, type: A, class: IN, requested name: www.alfaomeagaaa.ws
2021-04-26 15:44:58 DNS connection, type: A, class: IN, requested name: www.securitycheckpointsdomain.com
2021-04-26 15:44:58 DNS connection, type: A, class: IN, requested name: spynet2.microsoft.com
2021-04-26 15:44:59 DNS connection, type: A, class: IN, requested name: www.securitydomains1.com
2021-04-26 15:45:00 DNS connection, type: A, class: IN, requested name: www.websitesecurity.ws
2021-04-26 15:45:12 DNS connection, type: A, class: IN, requested name: teredo.ipv6.microsoft.com
2021-04-26 15:45:12 Last simulated date in log file

===

Figure 27. Unnamed Malware INetSim.

Appendix B

Hello World

cuckoo Analysis report summary © 2021/05/02 01:05

Summary - helloworld.exe

File info	Checksums
name: helloworld.exe	SHA1 0ba843dfb1475aa36f7b7f239053aa91bcf3bfeef
type: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows	MD5 5194ae60ca8803e130f87e5a829d2ac3
size: 3584 bytes	

Detected signatures

- Checks if process is being debugged by a debugger 2 events
- Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available 1 event
- Allocates read-write-execute memory (usually to unpack itself) 32 events
- One or more thread handles in other processes 1 event
- PEB modified to hide loaded modules. DLL very likely not loaded by LoadLibrary 74 events
- Malfind detects one or more injected processes 1 event
- Stopped Firewall service 1 event
- Stopped Application Layer Gateway service 1 event

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	ctldl.windowsupdate.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	www.msftncsi.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 28. HelloWorld Cuckoo Sandbox.

PAFish

 Analysis report summary
2021/05/02 01:33

Summary - pafish.exe

File info	Checksums
name: pafish.exe	SHA1 124f46228d1e220d88ae5e9a24d6e713939a64f9
type: PE32 executable (console) Intel 80386 (stripped to external PDB), for MS Windows	MD5 9159edb64c4a21d888d088bf2db23f3
size: 76800 bytes	

Detected signatures

- i Queries for the computername 2 events
- i Checks if process is being debugged by a debugger 1 event
- i Command line console output was observed 128 events
- i Collects information to fingerprint the system (MachineGuid, DigitalProductId, SystemBiosDate) 1 event
- i Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available 1 event
- ! Queries the disk size which could be used to detect virtual machine with small fixed size or dynamic allocation 2 events
- ! Executes one or more WMI queries 2 events
- ! Searches running processes potentially to identify processes for sandbox evasion, code injection or memory dumping 36 events
- ! Checks adapter addresses which can be used to detect virtual network interfaces 1 event
- ! The binary likely contains encrypted or compressed data indicative of a packer 2 events
- ! One or more thread handles in other processes 1 event
- ! Executes one or more WMI queries which can be used to identify virtual machines 2 events
- o Looks for known filepaths where sandboxes execute samples 2 events
- o Checks the version of Bios, possibly for anti-virtualization 2 events
- o Detects virtualization software with SCSI Disk Identifier trick(s) 1 event
- o Attempts to detect a virtual machine by the use of a pseudo device 2 events
- o Detects Joe or Anubis Sandboxes through the presence of a file 1 event
- o Detects VirtualBox through the presence of a device 4 events
- o Detects VirtualBox through the presence of a file 16 events
- o Detects VirtualBox through the presence of a registry key 4 events
- o Detects VirtualBox using WNetGetProviderName trick 1 event
- o Detects VirtualBox through the presence of a window 2 events
- o Detects VMWare through the presence of various files 4 events
- o Detects VMWare through the presence of a registry key 1 event
- o PEB modified to hide loaded modules. DLL very likely not loaded by LoadLibrary 103 events
- o Malfind detects one or more injected processes 1 event
- o Stopped Firewall service 1 event
- o Stopped Application Layer Gateway service 1 event
- o Detects the presence of Wine emulator 2 events

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	ctldi.windowsupdate.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	www.msftncsi.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 29. PAFish Cuckoo Sandbox.

WannaCry

CUCKOO Analysis report summary 2021/05/02 02:07

Summary - wannacry.exe

File info	Checksums
name: wannacry.exe	SHA1 5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
type: PE32 executable (GUI) Intel 80386, for MS Windows	MD5 84c82835a5d21bbc775a61706d8ab549
size: 3514368 bytes	

Detected signatures

- Queries for the computername 2 events
- Command line console output was observed 4 events
- Uses Windows APIs to generate a cryptographic key 4 events
- Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available 1 event
- The executable uses a known packer 1 event
- The file contains an unknown PE resource name possibly indicative of a packer 1 event
- Checks whether any human activity is being performed by constantly checking whether the foreground window changed 0 events
- A process attempted to delay the analysis task. 1 event
- Queries the disk size which could be used to detect virtual machine with small fixed size or dynamic allocation 1 event
- Creates (office) documents on the filesystem 5 events
- Creates executable files on the filesystem 5 events
- Creates hidden or system file 681 events
- Creates a shortcut to an executable file 1 event
- Drops an executable to the user AppData folder 1 event
- A process created a hidden window 8 events
- Changes read-write memory protection to read-execute (probably to avoid detection when setting all RWX flags at the same time) 1 event
- The binary likely contains encrypted or compressed data indicative of a packer 2 events
- Uses Windows utilities for basic Windows functionality 1 event
- One or more thread handles in other processes 1 event
- Appends a known WannaCry ransomware file extension to files that have been encrypted 1762 events
- Deletes a large number of files from the system indicative of ransomware, wiper malware or system destruction 2391 events
- Writes a potential ransom message to disk 1 event
- Uses suspicious command line tools or Windows utilities 1 event
- PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 68 events
- Malfind detects one or more injected processes 1 event
- Kernel module without a name 1 event
- Stopped Firewall service 1 event
- Stopped Application Layer Gateway service 1 event
- Drops 239 unknown file mime types indicative of ransomware writing encrypted files back to disk 237 events
- Performs 2357 file moves indicative of a ransomware file encryption process 2357 events
- Appends a new file extension or content to 2357 files indicative of a ransomware file encryption process 2357 events

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	time.windows.com	Empty	-
A	ctldl.windowsupdate.com	Empty	-
A	sqm.microsoft.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	www.msftncsi.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 30. WannaCry Cuckoo Sandbox.

Mass Logger

 Analysis report summary
© 2021/05/02 01:21

Summary - masslogger.exe

File info

name: masslogger.exe

type: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows

size: 12288 bytes

Checksums

SHA1 8110d13efd40b251f3ec964f0f279c3129b6d124

MD5 da06734f45a86b28e5f6e73cde69ae3

Detected signatures

- i Queries for the computername 49 events
- i Checks if process is being debugged by a debugger 5 events
- i Command line console output was observed 6 events
- i Uses Windows APIs to generate a cryptographic key 57 events
- i This executable has a PDB path 1 event
- i Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available 1 event
- ! Allocates read-write-execute memory (usually to unpack itself) 187 events
- ! Creates executable files on the filesystem 1 event >
- ! Creates a shortcut to an executable file 1 event
- ! Creates a suspicious process 2 events >
- ! Drops an executable to the user AppData folder 1 event
- ! Executes one or more WMI queries 1 event
- ! A process created a hidden window 2 events
- ! Checks for the Locally Unique Identifier on the system for a suspicious privilege 1 event
- ! Uses Windows utilities for basic Windows functionality 1 event
- ! One or more thread handles in other processes 1 event
- ⊗ Looks for the Windows Idle Time to determine the uptime 1 event
- ⊗ Deletes executed files from disk 2 events
- ⊗ The process powershell.exe wrote an executable file to disk 15 events
- ⊗ PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 158 events
- ⊗ Malfind detects one or more injected processes 1 event
- ⊗ Stopped Firewall service 1 event
- ⊗ Stopped Application Layer Gateway service 1 event

Network


DNS (3)

Type	Name	Response	Post-analysis lookup
A	www.msftncsi.com	Empty	-
A	ctldl.windowsupdate.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	google.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 31. MassLogger Cuckoo Sandbox.

Upatre


Analysis report summary
© 2021/05/02 01:53

Summary - upatre.exe

File info

name: upatre.exe
type: PE32 executable (GUI) Intel 80386, for MS Windows
size: 95748 bytes

Checksums

SHA1	49cbbc3f58eb08635dbc2f71c9269580be8fcf75
MD5	a0e0a4d830b213ed381084312aef74a3

Detected signatures

- i This executable has a PDB path 1 event
- ! Creates executable files on the filesystem 1 event >
- ! Creates a suspicious process 2 events >
- ! Drops an executable to the user AppData folder 1 event
- ! A process created a hidden window 1 event
- ! Uses Windows utilities for basic Windows functionality 2 events
- ! One or more thread handles in other processes 1 event
- o Communicates with host for which no DNS query was performed 1 event
- o Installs itself for autorun at Windows startup 1 event >
- o Deletes executed files from disk 1 event
- o PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 111 events
- o Malfind detects one or more injected processes 1 event
- o Stopped Firewall service 1 event
- o Stopped Application Layer Gateway service 1 event
- o Connects to an IP address that is no longer responding to requests (legitimate services will remain up-and-running usually) 1 event

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	www.msftncsi.com	Empty	-
A	time.windows.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-
A	ctldl.windowsupdate.com	Empty	-
A	ipv6.msftncsi.com	Empty	-

Hosts (1)

IP Address

158.69.115.115

© 2010 - 2017, Cuckoo Sandbox

Figure 32. Upatre Cuckoo Sandbox.

Chthonic



Summary - cthonic.exe

File info

name: cthonic.exe
type: PE32 executable (GUI) Intel 80386, for MS Windows
size: 154816 bytes

Checksums

SHA1 f5724a63620621be8930972897da28c088547706
MD5 aba6f9b372254cf34879ddc5283927c9

Detected signatures

- Queries for the computername 110100 events
- One or more processes crashed 1 event
- Allocates read-write-execute memory (usually to unpack itself) 12 events
- Drops an executable to the user AppData folder 1 event
- Searches running processes potentially to identify processes for sandbox evasion, code injection or memory dumping 12 events
- Expresses interest in specific running processes 1 event
- One or more thread handles in other processes 1 event
- Allocates execute permission to another process indicative of possible code injection 1 event
- Potential code injection by writing to the memory of another process 1 event
- Resumed a suspended thread in a remote process potentially indicative of process injection 2 events
- PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 119 events
- Malfind detects one or more injected processes 1 event
- Stopped Firewall service 1 event
- Stopped Application Layer Gateway service 1 event
- Executed a process and injected code into it, probably while unpacking 4 events

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	www.msftncsi.com	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	ctldl.windowsupdate.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

Figure 33. Chthonic Cuckoo Sandbox.

Unnamed Malware


Analysis report summary
© 2021/05/02 00:46

Summary - Backdoor.Win32.Agent.dkbp.de1af0e97e94859d372be7fcf3a5daa5.exe

File info

name:
Backdoor.Win32.Agent.dkbp.de1af0e97e94859d372be7fcf3a5daa5.exe

type: PE32 executable (GUI) Intel 80386, for MS Windows

size: 64512 bytes

Checksums

SHA1 6b0c9722abf07f2ebdeb5363021593ba9bde7a17

MD5 de1af0e97e94859d372be7fcf3a5daa5

Detected signatures

- i Queries for the computername 2 events
- i Checks if process is being debugged by a debugger 1 event
- i Collects information to fingerprint the system (MachineGuid, DigitalProductId, SystemBiosDate) 2 events
- i The executable uses a known packer 1 event
- ! A process attempted to delay the analysis task. 1 event
- ! Queries the disk size which could be used to detect virtual machine with small fixed size or dynamic allocation 1 event
- ! Checks for the Locally Unique Identifier on the system for a suspicious privilege 1 event
- ! One or more thread handles in other processes 1 event
- ⊗ Checks the version of Bios, possibly for anti-virtualization 2 events
- ⊗ Executes one or more WMI queries 1 event
- ⊗ Detects VirtualBox through the presence of a registry key 1 event
- ⊗ Detects VMWare through the presence of a registry key 1 event
- ⊗ PEB modified to hide loaded modules. Dll very likely not loaded by LoadLibrary 115 events
- ⊗ Malfind detects one or more injected processes 1 event
- ⊗ Stopped Firewall service 1 event
- ⊗ Stopped Application Layer Gateway service 1 event
- ⊗ Detects the presence of Wine emulator 2 events

Network

DNS (3)

Type	Name	Response	Post-analysis lookup
A	www.securitydomains1.com	Empty	-
A	dns.msftncsi.com	Empty	-
A	www.alfaomeagaaa.ws	Empty	-
A	ctldl.windowsupdate.com	Empty	-
A	www.websitesecurity.ws	Empty	-
A	ipv6.msftncsi.com	Empty	-
A	www.securitycheckpointsdomain.com	Empty	-
A	www.msftncsi.com	Empty	-
A	teredo.ipv6.microsoft.com	Empty	-

© 2010 - 2017, Cuckoo Sandbox

Figure 34. Unnamed Malware Cuckoo Sandbox.