Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Magnus Walmsnæss Refsnes

Exploring Trojanized Closed-Source Software Supply Chain Attacks Through Differential Malware Analysis

Master's thesis in MIS4900
Supervisor: Dr. Geir Olav Dyrkolbotn
Co-supervisor: Dr. Felix Leder
June 2023

NTNU
Norwegian University of
Science and Technology

Magnus Walmsnæss Refsnes

Exploring Trojanized Closed-Source Software Supply Chain Attacks Through Differential Malware Analysis

Master's thesis in MIS4900
Supervisor: Dr. Geir Olav Dyrkolbotn
Co-supervisor: Dr. Felix Leder
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Exploring Trojanized Closed-Source Software Supply Chain Attacks

Magnus Walmsnæss Refsnes

June 1, 2023

Abstract

In the last few years, there has been an increase in the amount of software supply chain attacks. The SolarWinds attack in 2020 was an insidious attack conducted by an Advanced Persistent Threat that managed to remain undetected within the Solarwinds networks for over a year, and while they backdoored their way through several victimised customers. This master's thesis sought to determine if basic analysis methods such as PE file analysis and embedded string analysis together with dynamic sandboxing leveraged through differential analysis of the benign and trojanized sample pairs would reveal indicators of compromise. The analysis of 10 sample pairs of trojanized and legitimate software was conducted and led to the findings that these static methods were well suited to finding both malicious indicators and indications of obfuscation, while the sandboxing techniques were less able.

Sammendrag

I de siste årene så har det vært en økning av angrep gjennom kompromittert forsyninskjede for programvare. SolarWinds angrepet i 2020 var et ondsinnet angrep som ble gjennomført av en avansert trussel aktør som hadde hold seg unna deteksjon i SolarWinds sitt nettverk i over et år mens dem tilegnet seg tilgang til kundenettverk. Denne masteroppgaven ønsket å avgjøre om grunnleggende statiske analysemetoder som PE fil analyse og string analyse sammen med dynamisk sandboksing sammen med differensial analyse kunne brukes til å detektere slike angrep. Analysen har blitt gjennomført på 10 par med trojaniserte og legitime filer, og ledet til funn at statiske metoder var vel fungerende til å oppdage ondsinnede indikatorer og indikatorer for obfuskering, mens sandboksing fungerte ikke like bra.

Peface

This thesis was written spring of 2023 at the Norwegian Institute of Technology in Gjøvik. It was written in collaboration with CrossPoint Labs.

I would first like to thank my supervisors, Dr Geir Olav Dyrkolbotn at the CCIS and Dr Felix Leder from Crosspoint Labs. Their guidance and input have provided valuable insight into the project and thesis.

Furthermore, I thank my friends and family for their support these last years—lastly, my gratitude to Anne Kine for all her help and support, especially the last few months.

Contents

Figures

Tables

Code Listings

Acronyms

APT	Advanced Persistent Threat. 1, 2
C2	Command and Control Traffic. 14
DGA	Domain Generation Algorithm. 14
DLL	Dynamic Linked Library. 8, 9, 18
ENISA	The European Union Agency for Cybersecurity. 2, 6
EXE	Windows Executable. 8
NIST	National Institute of Standards and Technology. 6
OEP	Original Entry Point. 13
OS	Operating System. 11, 15, 20
Pafish	Paranoid Fish. 21
PE	Portable Executable. 3, 5, 8, 10, 22
TA	Threat Actor. 2
VM	Virtual Machine. 11, 17, 18, 20, 29, 32, 33, 36
VPN	Virtual Private Network. 16
WMI	Windows Management Interface. 15

Chapter 1

Introduction

1.1 Topics Covered by the Project

Towards the end of 2020, it was uncovered that Solarwinds, a US-based software company with a massive customer list including the Norwegian Sovereign Wealth Fund[1], 425 of the US Fortune 500[2], and the Pentagon, had been hacked [3]. Moreover, the attacker, an Advanced Persistent Threat (APT)[3], had compromised their software build process and turned their proprietary closed-source software into a backdoor to the networks of Solarwinds' customers[3]. These attackers remained unnoticed for more than a year [4] until, eventually, the security company FireEye discovered that they themselves had been compromised, tracked it down to a Solarwinds server [5], tore it apart, and reported it to SolarWinds [6].

These types of attacks are named closed-source software supply chain attacks, and they differ from regular cyber-attacks in that an otherwise trusted software component has, somewhere along the delivery process, been turned into a type of malware [7]. This master thesis concerns the analysis of software that has been compromised in such a fashion by comparing its forensic artefacts to a benign version. This process is called a differential analysis, and for this thesis, the focus has been on testing if basic static analysis and sandboxing methods can be used to uncover a potential software supply chain attack.

1.2 Key Words

Supply Chain Security, Supply Chain Attack, Malware Detection, Malware Analysis, Closed-Source, Differential Analysis, Basic Analysis, Sandboxing

1.3 Problem Description

Traditionally the IT world considered something as trusted once it has passed existing checks such as authentication or having a valid signature from a trusted supplier[8]. Digital signatures were considered to mark that the software was legitimate, and one could update or install it when it passed the checks. However, therein lies the problem with a supply chain attack, as it seeks to exploit customers' inherent trust in the services their suppliers deliver [8]. Defending against an attacker exploiting this trust through compromised software is difficult, as modern software systems are large, complex, and use third-party dependencies[8]. Furthermore, organisations often have several software components and suppliers in their systems.

When SolarWinds was compromised, and their software was used to conduct attacks worldwide, there were three times in which a security company discovered and tracked down the suspicious activity to an Orion server [5]. The first two times, the issue was dropped, while the last time, the security company FireEye, having been compromised themselves, decompiled the server code and found it to be malicious [5]. FireEye is a professional security company with many security engineers and analysts on its payroll compared to regular companies. However, the attackers behind the supply chain attack compromised their networks and stole their penetration testing tools [9]. A question then arises, if an attacker can compromise a mature security organisation using these methods, how are we supposed to detect these attacks?

It is not unreasonable to conclude that most companies that are not security companies (and most likely not even these) do not have several dedicated expert reverse engineering malware analysts on hand, readily able to decompile every single update and installation before it is deployed to any system. So what happens when a trusted third-party supplier is compromised, and that business-critical software suite has become a backdoor for cyber criminals and nation-state actors into the company systems?

1.4 Justification, Motivation, and Benefits

It is important to look for possible methods of detecting a supply chain attack before the Threat Actor (TA) is able to exfiltrate any data, deploy ransomware, or cause any other damage. In European Union Agency for Cybersecurity [10], ENISA reports that state-backed TA and cyber criminals alike are increasingly focusing on Supply Chain Compromises. An example is that in 2021 supply chain attacks accounted for 17% of intrusions, up from just 1% in 2020 [10].

As this attack vector grows, it becomes more and more relevant to research techniques and methods that might be leveraged by non-expert personnel in order to detect these types of indicators of maliciousness in new versions of legitimate and signed software. Furthermore, basic analysis methods and sandboxing are possible to automate and scale up compared to manual analysis using a disassembler. However, the TA behind supply chain attacks is often APT groups [7] [11], capable of conducting evasive and highly skilled attacks.

It would be impossible for a master's thesis to solve the issue of detecting closed-source software supply chain attacks on its own. However, by looking specifically at the differences between the benign and trojanized versions, it aims to provide parts of a solution to a larger problem and contribute to a better understanding of the problem.

1.5 Research Questions

This research questions defined for this thesis are as follows:

RQ1. What are the current state-of-the-art methods for detecting a trojanized version of closed-source software?

RQ2. What indications of compromise can one detect in closed-source software supply chain attacks by comparing a previous benign file to a trojanized version?

RQ3. To what extent can basic static and dynamic differential analysis techniques be used in detecting malicious behaviour and static changes in trojanized software compared to the legitimate version?

RQ4. To what extent is looking for obfuscation and evasion techniques reliable in detecting trojanized closed-source software?

1.6 Scope & Contributions

This thesis aims to identify suspicious and malicious differences between compromised closed-source software and legitimate benign versions. The scope is analysing artefacts that can be extracted using automated tools rather than needing an expert-reverse engineer or proprietary high-cost software. This thesis aims to contribute towards future solutions to detecting software supply chain attacks by providing groundwork within a subject with very little written academic research. More specifically, it focuses on analysing the differences between the trojanized sample and the legitimate one in the Portable Executable (PE) file format, embedded strings, and runtime behavioural changes from sandbox runs. It then presents these findings so that they can be used for future development of solutions within the domain.

1.7 Ethical and Legal Considerations

The thesis's work has been analysing proprietary software trojanized alongside benign samples. Reverse engineering using a disassembler and presenting those findings could have some judicial issues, as it touches upon copyright law. However, the purpose and intent of this thesis have not been to disassemble the software nor to reveal any trade secrets. The functionality of the software revealed in this thesis is that which is available without disassembly. Furthermore, these samples were all uploaded to online software or malware repositories before the thesis.

1.8 Thesis Outline

This thesis is divided into five chapters. It starts with the introduction, then follows Chapter 2, which concerns the relevant theory, information about technologies, and a literary review of the state-of-the-art methods of detecting closed-source software supply chain attacks. Next comes Chapter 3, which details the dataset, experiment setup, and analysis method. Afterwards comes Chapter 4; this chapter contains the differential analysis of the different trojanized software, and the results are summarised at the end. Lastly, Chapter 5 this discusses the results, limitations and methods while containing the conclusion and future work.

Chapter 2

Background

This chapter seeks to present the required background reading for the thesis. First, it presents the theoretical foundations for the concepts within the paper, then the required overview of the technologies used. Lastly, it presents the state-of-the-art works within open-source and closed-source detection of software supply chain compromise.

2.1 Theory

This section presents the definitions used by the thesis for closed-source software supply chain attacks and malware. It then describes malware analysis and the general terms used for the types of malware analysis conducted before presenting a clear view of the PE file format. Furthermore, it describes the components of sandboxing and virtualisation. Lastly, it seeks to provide a grasp on obfuscation and evasion techniques.

2.1.1 Closed Source Software Supply Chain Attack

This section provides detail on what a closed-source software supply chain attack is. It does so by first defining the difference between open-source and closed-source, defining a supply chain, and defining a supply chain attack. Lastly, it closes by combining these three definitions.

Open and Closed Source Software

Open-Source Software is software readily available for the public to use, interact with, and inspect [12]. Such software development is often decentralised and collaborative [12]. In theory, this transparency could mean that any open-source project can have its code and dependencies inspected for malicious additions before the software is deployed to an environment.

On the other hand, *Closed-Source software*, or proprietary software, is software where the source code is closed to the public and not readily available outside of the organisation or company owning it [13][14]. This means that the software users do not have direct access to the source code of the software they are using, short of using other software to attempt to decompile it.

Supply Chain

This thesis uses the definition of The European Union Agency for Cybersecurity (ENISA) when defining what a supply chain is. ENISA defines a supply chain as an "*ecosystem of processes, people, organisations, and distributors involved in the creation and delivery of a final solution or product.*"[7]. Furthermore, in their report **ENISA THREAT LANDSCAPE FOR SUPPLY CHAIN ATTACKS** [7], they define the four main elements of a supply chain, those being:

- **Supplier**: This is the entity that supplies a product or a service to another entity.
- **Supplier Assets**: These are the valuable elements used by the supplier to produce the product or service.
- **Customer**: This is the entity that consumes the product or service produced by the supplier.
- **Customer Asset**: These are the valuable elements owned by the target.

Lastly, ENISA defines an entity as an individual, group of individuals, or organisation. Assets are defined as people, software, documents, finances, hardware, or others[7].

Supply Chain Attack

A supply chain attack builds on understanding the definition of a supply chain and its four main elements introduced in the section above. ENISA defines a supply chain attack as a combination of at least two attacks, the first being an attack on a supplier that is then leveraged to attack a target to gain access to its assets[7].

National Institute of Standards and Technology (NIST) defines a cybersecurity compromise in the supply chain as a cybersecurity incident within the supply chain "*whereby the confidentiality, integrity, or availability of a system or the information the system processes, stores, or transmits is jeopardised. A supply chain incident can occur anywhere during the life cycle of the system, product or service.*" in their publication **NIST SP 800-161r1** [15]

From this, we can draw that a supply chain attack is one where a malicious actor can attack their target by utilising access gained from successfully compromising or attacking a product or service delivered by a supplier

Closed-Source Software Supply Chain Attack

From the definitions above, we can draw the following conclusion, a closed-source software supply chain attack is where a malicious actor compromises a software provider's infrastructure, commercial software, or software deployment, which is then used to leverage an attack against a target further down the supply chain, e.g. the software provider's customers.

This thesis uses the terminology *Trojanized Software* when discussing and referencing legitimate software that has been weaponised by a malicious actor somewhere in the software process.

2.1.2 Malware

Once the previously legitimate and benign software has been *trojanized*, it can be considered malware. This thesis uses the definition of malware from Sikorski and Honig [16], their definition is as follows: "*Any software that does something that causes detriment to the user, computer, or network*" [16].

When discussing and analysing malware, it is relevant to discuss its capabilities, which most often means categorising it into different types of categories. The most relevant categories of malware for this thesis are the following:

- A Backdoor is malicious software or code that allows a malicious actor access to the machine it is installed on [16].
- A Downloader is a malicious software that has the single purpose of downloading other malicious code [16].
- A trojan is a piece of malware that disguises itself as wanted and desirable software [17], though with the ability to act maliciously on the system.
- A worm is a malware that can copy itself and infect additional machines on the network [16].
- Ransomware is malicious software or code that encrypts files on a target machine removing a person or organisation's ability to access their data until a ransom is paid [17].
- Wiper malware is malicious code or software built to delete data and ensure it cannot be recovered [18].
- Coin miner malware is malicious software or code intended to run complex calculations in order to collect cryptocurrency [19].

Malware Analysis

Malware analysis is the act of examining malware with the intention of learning how it works, what it does, how it might be detected, how it could have been prevented, and how it could be eliminated [20] [16].

At the most basic level, Malware Analysis can be divided into static and dynamic analysis. Static analysis is the method of examining the malware without running it and is an initial analysis method that allows one to gain preliminary information on the malware's functionality [20]. The more advanced version of static analysis is essentially to reverse-engineer the malware using a disassembler, which allows one to look at the program instructions to discover what it does [16].

Dynamic analysis, or behavioural analysis, is essentially to execute the malware in an isolated system and observe its behaviour and effect on the system. This can be furthermore complicated by utilising a debugger, which allows one to examine the behaviour more closely by looking at the internal state of the malware as it runs [16].

Lastly, differential analysis can be described as the process of using baseline information about the state of a system and comparing it to the system state after an event has occurred [21]. The essential part of the differential analysis is having the system's baseline or known good configuration [21]. Translating this for this thesis is to establish a known good baseline which is the static artefacts and behaviour of the legitimate benign software, and to compare this to the artefacts and behaviour of the trojanized software.

2.1.3 Portable Executable

This thesis focuses on malware analysis of Windows-based trojanized software, and due to this, it is important to have a clear view and description of the file format used by Windows executable files. This format is named Portable Executable (PE) and is used by Windows Executable (EXE), Object Code, and Dynamic Linked Library (DLL)s [16]. A PE file is essentially a sequence of structures and sub-components with information that the operating system needs to load it into memory [20]. This section contains relevant information and descriptions of the PE format as to this thesis. However, for a full in-depth description of the format, one should visit Microsoft specification on the format in [22] which is the source for the information in this section.

The first part of a Portable Executable (PE) file is the **DOS header** which is there for backward compatibility with MS-DOS. It contains a **DOS stub** and a **signature** that identifies the file as a PE file, with the signature being "PE\0\0" or "PE" [22]. The following section is the **COFF File Header** (from now on called File Header), which contains information about the Portable Executable (PE) file, such as which CPU architecture the executable is intended for, the number of sections, a UNIX timestamp for when it was created, size of the optional header, and flags that indicate the attributes of the file (system file, executable, and so on) [22]. Following the file header is the Optional Header. The optional header is generally not present in object files but is required for image files (Dynamic Linked Library (DLL) or .Windows Executable (EXE)) [22], and it consists of three parts:

- Standard Fields:
  - The standard fields contain the integer that identifies if the image is a 32-, 64-bit, or ROM image. Furthermore, it contains information related to the size of the code or the sum of all code sections if multiple. Size of the initialised and the uninitialised data sections (or sum if there are multiple sections). Entry point address and address that is relative to the image base of the beginning-of-code section [22].
- Windows-Specific Fields:
  - It contains information needed by the linker and loader in Windows,

such as the preferred address to load into memory, major and minor operating system requirements, required subsystem, and more [22].

- Data Directory Table:
    - The data directory table is a structure with two members, the first one being a pointer to the relative address of a data directory and the second being the size of the data directory [22]. A data directory, on the other hand, is a piece of data in the PE file. This data can, for example, be the export table which contains a list of exported functions that could be used by other programs [22].
        - During static malware analysis looking at the imported libraries and functions used is useful to guess what the malware is attempting to do, while with trojanized binaries, additions of previously unseen functions could indicate maliciousness, while a lack of functions could indicate packing or obfuscation. Meanwhile, looking at the changes in exports in a Dynamic Linked Library (DLL) file could reveal changes in capabilities, indicating trojanization.

Immediately following the optional headers is the section table, where each row of the table is, in function, a section header. A section header contains the name of the section (no longer than 8 characters), the VirtualSize or the size of the section when loaded into memory, the VirtualAddress, which is the first byte of the section relative to the image base when the section is loaded into memory [22], the SizeOfRawData which is the size of the initialised data on disk, and flags that describe the characteristics of the section (Executable, writeable, and so on) [22].

Lastly, following the section table comes the sections themselves. These contain very useful information when looking at it from a malware analysis perspective [16] as they contain the data of the executable file. The book M. Sikorski and A. Honig, *Practical malware analysis : The hands-on guide to dissecting malicious software*, eng, San Francisco, 2012 contains a list of the most common and interesting sections (from a malware analyst perspective) in a PE file:

- **.text** section is the executable code of the program and should usually be the only section that includes code [16].
- **.rdata** section typically has the export and import information but can also store read-only data used by the program. This can also be replaced by *.idata* and *.edata* sections which respectively would contain the import function information and the export function information [16].
- **.data** section should contain the data which is accessible anywhere in the program, also called the global data [16].
- **.rsrc** section the resources used by the program such as icons, images, strings and menus. Strings can be stored elsewhere, but they are often stored here due to multi-language support [16].
- **.reloc** section has information for the relocation of library files.

However, there are several other sections (and section names) also as listed by Microsoft in [22], some of those being *.bss* for uninitialised data, *tls* or Thread Local Storage that provides storage for executing threads of the program, or *.debug* that contains debug information. Sections become relevant in differential malware analysis in that changes such as an increase in entropy, changes in RawSize/VirtualSize ratio, changes in naming schemes, or packer indicators could all be signs of trojanization. A graphical illustration of the Portable Executable (PE) file format can be found in figure 2.1.

A graphical illustration of the PE file format can be found in figure 2.1.

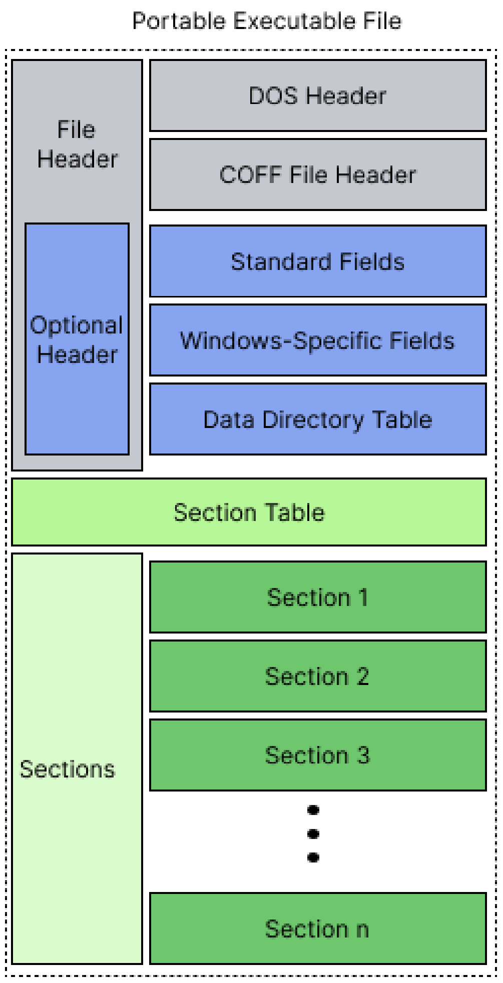


Figure 2.1: Portable Executable File Structure

2.1.4 Sandboxing

Sandboxing is a dynamic analysis technique to detect malicious or suspicious software behaviour. This is accomplished by using an isolated virtual environment to run untrusted programs, reducing or removing the risk of malicious software infecting the "real" system [16]. One such environment is a virtual machine, which is effectively a simulated computer (called the guest) within a computer (the host). A virtual machine uses software (called a hypervisor [23]) instead of hardware to run programs and applications [16], and there are several types of virtualisation software [1]. This VM can then function similarly to other computer systems with an Operating System (OS) running on the hypervisor's software and applications installed and running on the OS. An illustration of this relationship can be seen in figure 2.2, which shows two guest virtual machines being run by the hypervisor software on a single physical host machine.

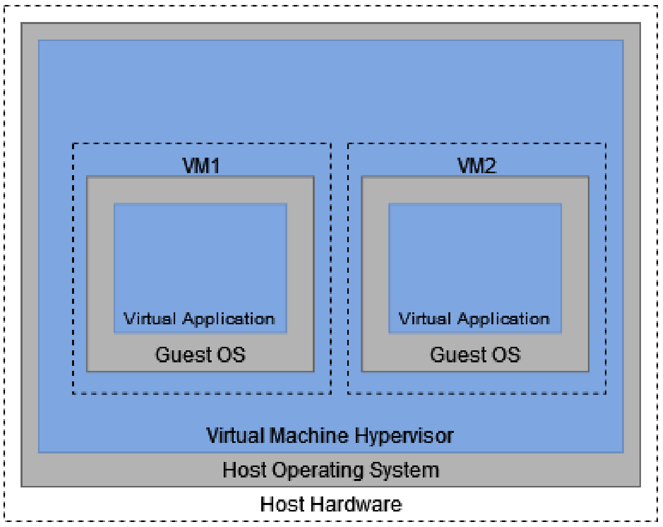


Figure 2.2: An example of a virtual machine running running through a hypervisor on a host system

A benefit of using Virtual Machine (VM) software is the ability to create snapshots which are essentially recordings of a system state at a certain point in time[24]. This can be used to recover the VM to a point before any malware is installed or executed. Furthermore, for differential malware analysis, it allows one to establish a baseline of the system and compare the baseline to the execution of the legitimate and malicious sample, which can allow us to observe behavioural differences between the two samples.

An example of a sandbox that uses a VM environment would be running the sandbox software on a host computer that runs a hypervisor software running an isolated VM connected to the sandboxing software through a closed virtual network. The sandbox software can transfer the malware sample to the VM using a sandbox agent running on the guest VM. This malware sample can then be launched on the guest system while the sandbox software observes and records its behaviour. The generated behavioural data can then be transferred back to the

[1]Some of the options for virtual machines: `https://www.virtualbox.org/`, `https://www.vmware.com/products/workstation-player.html`, and `https://www.linux-kvm.org/page/Main_Page`

host, where the sandbox software analyses the generated data and produces a report based on the activity. An illustration of this simplified relationship can be seen in Figure 2.3.

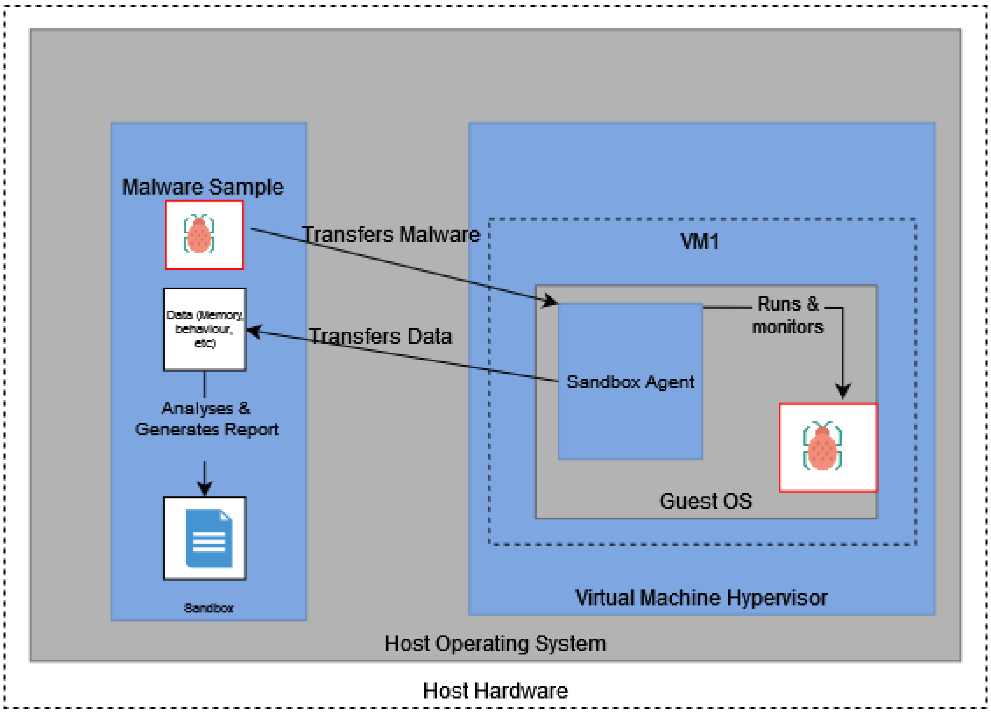


Figure 2.3: A simplified example of the relationship between a host running sandboxing software and guest VM.

2.1.5 Obfuscation

Obfuscation is a collection of techniques that makes a program or file harder to detect or more complicated to analyse [16]. It is worth mentioning that legitimate software also uses obfuscation techniques to prevent attackers from stealing intellectual property, discovering vulnerabilities, and making unauthorised modifications [25]. There are several types of obfuscation techniques, and the most relevant ones for this thesis are listed below.

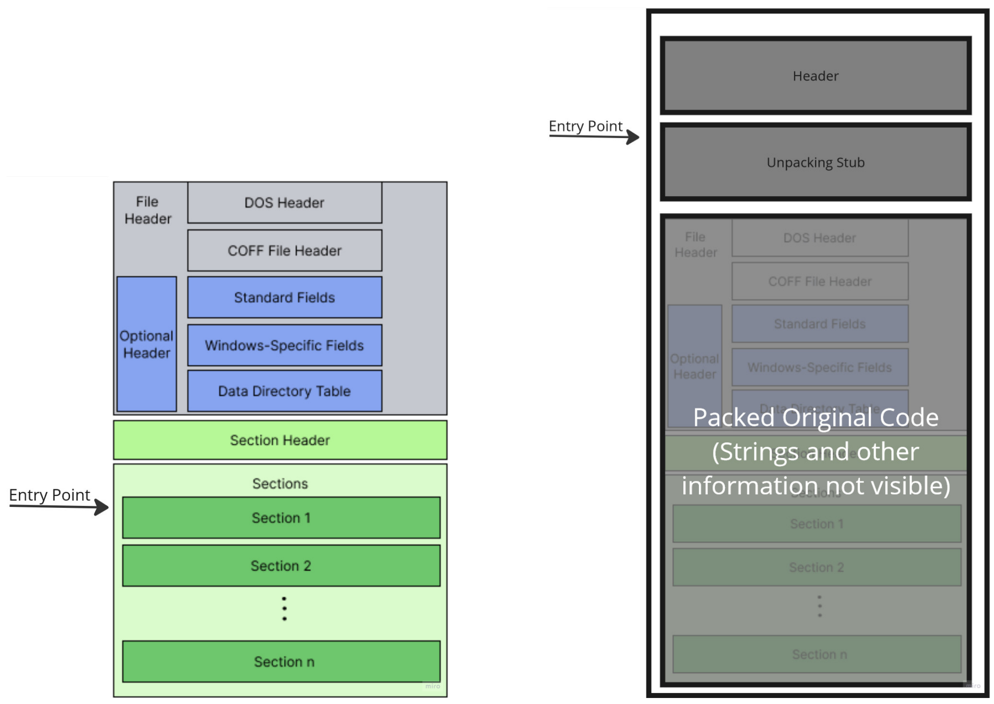
Packing

Packing is a technique used to compress the software or file in order to obfuscate it and outputs a new executable which is a packed program [16]. This new executable contains the previous one as compressed data, and upon execution, it runs a decompression routine that extracts the original file in memory and triggers the execution [16]. Some packers may only pack code and data sections, others might leverage encryption techniques to make it more challenging to analyse, and some may utilise special techniques to avoid analysis [16].

The decompression routine, or the unpacking snub, becomes the new entry point for the packaged executable, and it is often small and has a simple functionality [16]:

- Resolve all imports from the original executable.
- Transfer the execution to the Original Entry Point (OEP).
- Unpack the original executable into memory.

Should one attempt a static analysis of the packed software, then it is the functionality of the unpacking snub you would observe rather than the packaged software's functionality. An illustration of a unpackaged and packaged software can be seen Figure 2.4.



(a) Unpackaged Executable **(b)** Packaged Executable

Figure 2.4: The left figure is the unpackaged executable, while the right figure illustrates a packaged executable. The illustration is based on the illustrations from [16]

Simple Encoding

Simple encoding is the use of encoding algorithms such as base64 encoding or xor encryption to obscure the data[20]. Attackers use these simple algorithms because they are easy to implement, take fewer system resources, and can obscure the code's content from analysts and coders [20]. For example, base64 encoding is essentially just using a 64-character set, where every 3 bytes of binary data is translated into 4 characters from the character set [20]. XOR means exclusive OR, and it is a logical operator. XOR encoding uses a static byte value and modifies each byte by performing an XOR operation with the static value [16].

Encryption

Encryption is another method malware authors use to obfuscate the malware binary and command and control traffic [20]. Encrypting the malware, code, or strings within the malware can obfuscate it from analysis and detection [20]. Added cryptographic functionality to the trojanized software could serve the function of decrypting incoming command and control traffic [26], encrypting the data for exfiltration data [20], or in the case of some trojanized software, allowing it to act as ransomware or wipers.

Domain Generation Algorithm

Domain Generation Algorithm (DGA) is a method by which malware can avoid depending on a specific IP address, or domain [26]. Instead, based on a routine, it generates a new domain that the attacker can purchase and continue delivering Command and Control Traffic (C2) traffic or receive exfiltrated data. These routines must be predictable for the malware and the author behind it while still being unpredictable for security researchers [26]. It usually consists of a Seed or base element, a variable that changes with time, and one or more top-level domains [26].

2.1.6 Evasion

Virtual environments are different from regular host environments, often by a large amount of detectable and observable artefacts [27]. Virtual environments are often used to analyse suspicious and benign files both. Evasion techniques are attempts by malware to detect that it is running in a virtualised or debugging environment and, depending on if it is, either acts benign without arousing suspicion or carries out its malicious purpose [27]. However, it is worth mentioning that with the increased use of cloud services across organisations [28] and increased virtualised environments, this might no longer be synonymous with being analysed [29]. There are several types of evasion techniques, and this section lists examples relevant to this thesis.

Detecting Virtualisation

Detecting virtualisation consists of several sub-techniques that all look for indicators of the system being virtualised, and this thesis will describe several of them. For an expansive list, one can visit the collection of evasion techniques by Checkpoint and Ladutska [27]. Detecting virtualisation boils down to hunting for artefacts within the file system indicating virtualisation, such as names of processes, installed drivers, registry keys, hardware, Mac addresses and more [26]. One way to get OS and hardware info is to utilise Windows Management Interface (WMI) queries using COM interfaces and methods. Other detection methods can be timing-based, such as having long delays before performing any malicious activity to escape detection or detecting when a Sandbox attempts to skip such delays [30].

Detecting Debugging & Analysis Tools

Another method malware authors use to avoid being analysed is to detect the presence of a debugger, as one being present is a solid indication that the malware sample is actively being debugged [16]. Detecting debugging is as simple as using Windows API functions such as IsDebuggerPresent, CheckRemoteDebuggerPresent, NtQueryInbformationProcess, and OutputDebugString [16]. Another typical method is to perform the previous calls' functions manually rather than use the Windows API. Other methods include identifying debugger behaviour through checksum checks, timing checks, and INT scanning [16]. Other than strictly checking for debuggers or the presence of a virtual environment, it is also possible to check for malware analysis and security tools on the system [26].

User activity & Logic Bombs

It is also not uncommon to check the system for traces of activity that one would expect either from a live server environment or user usage. Checking if the window of the process is in focus, checking the number of monitors, browser history, mouse movement, and more [26]. Another method is using a logic bomb, which only activates the malware on a specific hardware, system, domain, specific event, or more. Should the trojanized software not be on the intended system then it does not detonate [26].

2.2 The Cuckoo Sandbox

This section concerns Cuckoo Sandbox, which is the primary sandboxing tool used during this thesis. It first presents a basic overview of the sandbox and its capabilities before describing more in-depth the components and modules Cuckoo is composed of, and lastly describes Cuckoo's networking capability and some of its available tools.

2.2.1 Cuckoo Basics

Cuckoo Sandbox is an open-source automated malware analysis tool. Initially, it started as a Google Summer of Code project in 2010, and from there went through several versions before the final version was published on June 19, 2019

[31]. Cuckoo is used to automatically run malicious and benign files, collect the behaviour and changes on the system, and provide an analysis of the collected data. Cuckoo was designed to be modular and work both as a standalone application and to be integrated into a framework [31].

Cuckoo is able to retrieve information from the guest system related to the creation, deletion, and downloaded files. It can collect memory dumps of both malware processes and full dumps of the machine memory. It can collect and monitor network traffic and trace the calls made by the malware and the processes it spawns. It is able to do this on most file formats, including Windows Executables, DLL, PDF, Microsoft Office, Zip, and more [31].

Files are uploaded to the guest system either through the command line or a web interface, and Cuckoo supports both simulating the network, routing it through Virtual Private Network (VPN) or tor, dropping the traffic, or simply just letting it through to the internet [31].

Cuckoo Sandbox was chosen for this thesis early during the project in collaboration with the project supervisors. However, during the project work timeline, it was found that the Cuckoo Sandbox Project was officially archived on April 26, 2021, [2], and since then, it has not seen active development. Due to the fact that Cuckoo was no longer developed, it was considered to switch to CAPEv2 [3], a community developer fork of the cuckoo. It was decided to continue with Cuckoo for the following reasons:

1. The project was already underway, and it would take time from the project and thesis in order to set up, configure, and run CAPEv2.
2. Cuckoo Sandbox has, in general, more and better documentation over several years and was more familiar to the author of the thesis.
3. Other online-hosted sandbox solutions are available that could be used to reference the results of the Cuckoo analysis and supplement should there have been any discrepancies.
4. Cuckoo was a contemporary solution during most attacks from which the trojanized software was deployed. Therefore it could also provide relevant insight into whether differential behaviour analysis could have detected or prevented some attacks.

2.2.2 The Cuckoo Architecture

Cuckoo Sandbox is a central management software that handles the execution and analysis of the samples with one or more distributed and isolated guest machines. At the start of each analysis, a fresh host is started by Cuckoo and the sample is transferred over a virtual network to the guest, where it is executed and analysed. An illustration of the process can be seen in figure 2.5.

[2]`https://github.com/cuckoosandbox/cuckoo`

[3]`https://capev2.readthedocs.io/en/latest/`

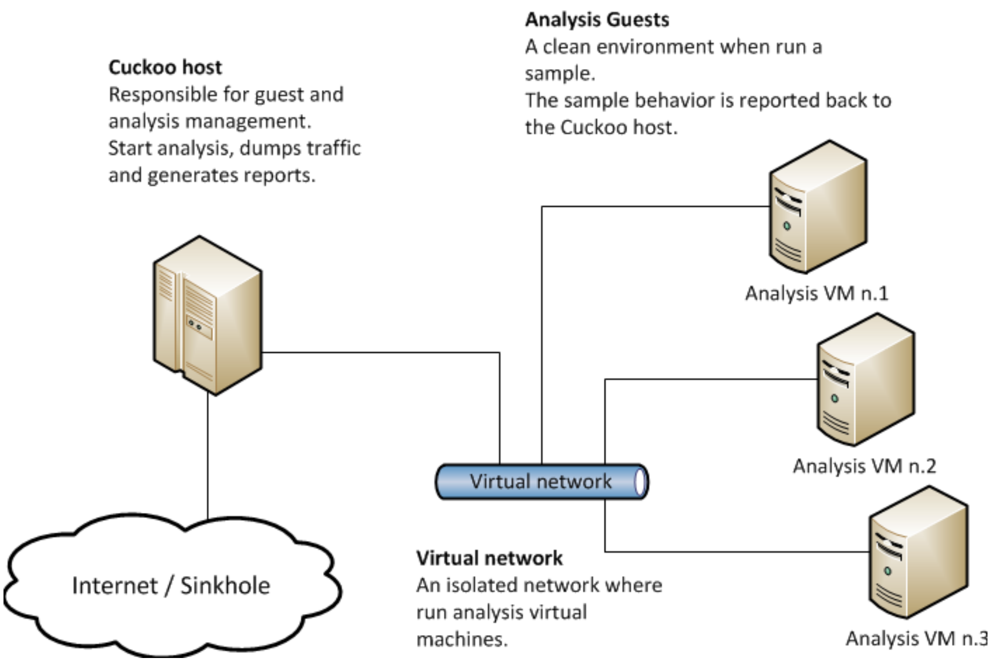

Cuckoo host
Responsible for guest and analysis management.
Start analysis, dumps traffic and generates reports.
Analysis Guests
A clean environment when run a sample.
The sample behavior is reported back to the Cuckoo host.
Analysis VM n.1
Analysis VM n.2
Analysis VM n.3
Virtual network
Internet / Sinkhole
Virtual network
An isolated network where run analysis virtual machines.

Figure 2.5: An Illustration of the Cuckoo Sandbox Architecture from the official Cuckoo documentation [31].

2.2.3 Components

This chapter is based on the information provided in the Hatching blog post by Zutphen [32], one of the backend developers of Cuckoo[4].

Machinery modules are the components that interact with the hypervisor (such as Virtualbox or VMware) or the physical machine (if the guest is a physical machine rather than a virtual one). These modules start, stop, and restore the VM to a clean slate Zutphen [32].

The Scheduler is continuously running while Cuckoo is. It handles initialising the configured machinery module, for example, a configured VirtualBox module, and starting new tasks if there are enough resources on the system Zutphen [32]. As the Scheduler is running, it checks for any available VMs; if there are, it checks for any pending tasks. Lastly, when the Scheduler is ready to start a task, it hands it over to the Analysis Manager Zutphen [32].

The Analysis Manager starts with the Scheduler when there is a new task and enough resources. When it starts, it will find a machine that matches the one requested by the task, as different tasks can require to run in different environments

[4]`https://cuckoosandbox.org/about`

such as Linux or Windows. Before it starts any machine, it will first start any required auxiliary modules, then it will start the machine, and from there on, the Guest Manager will handle the analysis. Finally, the Analysis Manager will stop the machine after a set time or a critical timeout.

Auxiliary Modules are responsible for all sorts of tasks and must be completed before the machine starts or as it runs. An example of these modules is Sniffer, which dumps all network traffic the machine generates.

The Cuckoo Agent is a simple Python HTTP server that can start processes and upload files. It sits on the guest VM and must be already started when the machine is brought up.

The Guest Manager continuously communicates with the Cuckoo agent on the guest VM. It checks if the machine is started, and when it is, it uploads everything needed for the analysis to the guest through the Cuckoo agent and starts the Analyzer. Then it keeps in contact with the agent and asks if the Analyzer is finished yet. Should a critical timeout be reached, then it stops the Analyzer.

The Analyzer is a Python file on the host machine that is uploaded to the guest machine by the Guest Manager through the Cuckoo Agent. There are different analyzers for each platform, and the configuration of the Analyzer depends on what parameters the analysis had when it was started. For example, the Analyzer will inject a DLL file named "Cuckoo Monitor" on Windows systems, which logs behaviour by hooking data and following processes.

The Result Server is located on the host machine and receives all the behavioural data extracted from the guest. It then stores this data in the correct format and under the correct directory.

Processing Modules are Python scripts that allow one to define how the raw results are supposed to be analysed [33]. It transforms the behavioural data from the guest machine into data that the signatures can use Zutphen [32]. It also allows the results to be easily presentable as a report Zutphen [32]. There is a total of 24 modules. The complete list can be found at [33]. However, below are listed a few of the processing modules.

- AnalysisInfo –generates some basic information about the analysis.
- Memory – Runs Volatility[5] on a full memory dump
- Dropped – Information on the files dropped by the malware and dumped by Cuckoo.

[5]`https://www.volatilityfoundation.org/`

- BehaviorAnalysis – Parses raw behaviour logs and provides the complete process tracing, a process tree, and a behavioural summary.

Signatures are ran against the data when it completes processing. Should any of the signatures have a match, then that match will be added to the results. Cuckoo supports creating signatures, and there is a community repository from which one can download signatures developed by others[6]

2.2.4 Routing & Tools

Routing

As previously mentioned in Section 2.2.1, there is several routing options available when using Cuckoo. These can be found in Table 2.1.

Table 2.1: Cuckoo Network Routing Options from [34]

Routing Option	**Description**
None Routing	Cuckoo does no routing.
Drop Routing	Drops all non-cuckoo traffic.
Internet routing	allows the guest full access to internet.
InetSim Routing	Uses the InetSim project[7] which is a suite designed to simulate different internet services [34].
Tor Routing	Routes the network traffic through the Tor network[8].
VPN Routing	Routes the traffic through a VPN as to simulate a different country location.

Volatility Framework

Cuckoo uses the Volatility Framework for memory forensics. The Volatility Framework is an open-source collection of tools written in Python[16] for analysing volatile memory (RAM). There are several Volatility plugins[9] available for Cuckoo, examples being plugins for detecting hooked API, hidden processes, and injected code.

YARA

YARA is a pattern-matching tool used to identify and classify malware samples through rule-making that matches textual or binary patterns [35]. There is a

[6] `https://github.com/cuckoosandbox/community/`

[9] The full list can be found in the memory.conf file: `https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/private/cwd/conf/memory.conf`

Cuckoo module for YARA which allows one to create YARA rules based on the behavioural information provided by Cuckoo Sandbox [36].

2.3 Other Technologies

This section introduces other technologies, such as online sandboxes, virtualisation tools, and analysis tools used for the thesis. The purpose of this section is to familiarise the reader with the name and functionality of the tools referenced in later chapters of the thesis.

2.3.1 Online Sandboxes

Malware sandbox services are also available online through websites and API interfaces; these services usually have some functionality available for free or signed-in users, with some functionality locked behind a payment service or license. This section provides a short introduction to three online sandbox services that were used either for dataset collection, comparison, or running samples.

VirusTotal

VirusTotal is an online sandboxing service that allows one to drop and test files and URLs against several antivirus engines, website scanners, and sandboxes through API or the site itself [37]. When a file has been run, it presents the user with several tabs from which one can inspect information, such as AV detections, static details, behavioural data, and relations to other files, domains and IP addresses [38]. Additionally, each file has a community section in which users can comment on the file. The site also has additional functionality that, alongside sample downloads, is locked behind a premium subscription intended for enterprises [38].

Any.Run

Any.Run is another service that allows one to test URLs and files for maliciousness [39]. Any.Run boots up a virtual machine according to user-selected criteria, from which the user can navigate the URL, file installation, or file use. As the file is run, its behavioural data is analyzed, and after the run is presented to the user. However, the most important feature for this thesis is that one can search public analysis tasks for hashes and download their samples using a free user [40].

Hatching Triage

Hatching Triage (Triage from now on) is a sandboxing solution similar to Any.Run in that the user can submit files and URLs according to user criteria (such as OS, web browser, and internet connection) [41]. It collects behaviour and static data from the sample and provides a user interface to interact with the VM as the malware runs. After the run, it presents the information in a report similar to Cuckoo

Sandbox with signatures, a process tree, networking information, and a MITRE ATT&CK matrix. It assigns a score between 1 and 10, calculating maliciousness from where 1 is no malicious behaviour detected, 2-5 is likely benign, 6-7 is suspicious behaviour, 8-9 is likely malicious, and 10 is a known bad file [10]. Triage is developed by some of the developers behind Cuckoo Sandbox [42], and the cloud version is free to use for individual persons and researchers.

2.3.2 Tools

This section briefly introduces the tools such as software, libraries, or scripts used for the VMs and the malware analysis. These introductions are intended to familiarise the reader with the capabilities and functions of the tools most relevant to the thesis. However, this section does not provide a complete overview of the tools nor their full functionality or configurations. However, links are provided to the documentation from which this can be found.

VirtualBox

VirtualBox is a hypervisor software capable of running on Windows, macOS, Linux, or Oracle Solaris operating systems [43]. In addition, VirtualBox supports hosting many different OS, such as several versions of Windows, a large amount of the Linux family, and others [43]. The complete list can be found at `https://www.virtualbox.org/wiki/Guest_OSes`.

VirtualBox has many options for the VM that it hosts. However, some of the most relevant ones for this thesis are listed below[11]

- Hardware emulation for a few devices, such as storage, networking, and USB.
- Record and restore the system state through snapshots.
- Modify motherboard options such as RAM allocation.
- Configure CPU utilisation, such as the number of processors, execution cap, and nested virtualisation.
- Hardware virtualisation extensions such as paravirtualisation interface to improve time-keeping and nested paging.

PaFish

Paranoid Fish (Pafish) is an open-source tool for detecting virtual machines and analysis environments [44]. It accomplishes this by leveraging several anti-debugger, -VM, and -sandbox checks within the Windows environment it is run [29]. When executed, it runs through these techniques, such as using the Windows API to check if it is being debugged, CPU-based timing checks, and searching for registry keys related to virtual machines [44].

[10]The complete scoring system with explanations `https://tria.ge/docs/scoring/`

[11]The complete list of configuration options can be found at: `https://www.virtualbox.org/manual/UserManual.html#BasicConcepts`

PeStudio

PeStudio[12] is a static malware analysis tool that runs in a Windows environment. It does not require installation and can be dropped onto the system and run from there. The purpose of PeStudio is to discover suspicious artefacts within an executable file [45]. It does so by presenting an overview of the different sections of the PE file, as described in Section 2.1.3, and listing the observed contents of the file, such as strings, imports, compile date, and more. Then, it flags and highlights different potential suspicious artefacts [46]. These suspicious artefacts could be things that have been seen previously in malware, imports that could be used maliciously, or other such suspicious artefacts [46].

Exeinfo PE

Exeinfo PE is a tool to detect packers, compilers, and cryptors used to build a PE file [20].

Stringshifter

Stringsifter is an analysis tool utilising ML to rank strings based on their relevancy for malware analysis [47]. It can take both binaries and strings as input and produce ranked results.

FLOSS

FLOSS obfuscated string solver is an open-source tool that automatically deobfuscates strings from malware samples [48]. It can also extract regular strings from samples, but the primary purpose is to decode and deobfuscate strings.

PeFile Library

Pefile is a library module for Python, and it is used to work with PE files [49]. It can be used to read and extract the information in a PE file through Python programming.

2.4 State of the Art

This section presents relevant academic literature for closed-source and open-source detection of software supply chain compromises. Additionally, it was decided to include some government and industry literature on detecting software supply chain attacks as they present solutions to the problem. The sections 2.4.1 and 2.4.3 presents the answer to **RQ1.** What are the current state-of-the-art methods for detecting a trojanized version of closed-source software?

[12]PeStudio homepage: `https://www.winitor.com/`

2.4.1 Closed-Source Detection

This first section presents the academic literature on closed-source software supply chain attacks. It is worth mentioning that there is little published research on the security and detection of closed-source supply chain attacks [11] [8].

In Barr-Smith *et al.* [11] the authors presented an approach to detecting maliciousness in closed-source software based on a differential analysis of binaries. This was accomplished by developing a system consisting of several modular components written in Python that they named Exorcist. Exorcist's two main components are an automatic static analysis using deobfuscation and automated reverse engineering methods and a detection method for abnormal activity during dynamic runs. Then, the system applies a weighted heuristic system to suspicious differences between the binaries, and if the weight of this system exceeds a threshold, it is identified as malicious. The project tested their system against 12 samples of supply chain attack binaries, with their findings being a presence of obfuscation in either a minor or major form in all the samples and a prevalence of additions of static indicators of maliciousness.

In Wang [8] the author proposed a signature-less detection approach for supply chain attacks based on tracking information flows from strategic locations. Wang identifies that the main problem of detecting a software supply chain attack lies in the fact that modern mission-critical systems, such as SolarWinds Orion, often consist of millions of lines of code and have complex dependencies on third-party software packages that it then becomes impossible for any cyber defence to know the inner workings of all the software components. The author then identifies that there is a need to develop novel detection capabilities that: "*1) is independent from various suppliers; 2) requires no signature or prior knowledge of the attack.*". The author's solution is essentially to use inter-packet timing-based flow watermarking technologies to tag outgoing data and detect anomalies in traffic indicative of command and control traffic or data exfiltration.

2.4.2 Open-Source Detection

Comparatively, there have been several recent ventures of academic research into open-source software supply chain attacks. One such paper is in Ohm *et al.* [50] where the authors researched how malicious functionality was injected into the supply chain through malicious repositories such as npm, PyPI, and RubyGems. The authors manually collected and analysed a dataset of 174 malicious software packets used as part of attacks on the software supply chain. The paper also presented two general attack trees; one provides a view of the techniques to inject malicious code into the open-source ecosystem, while the other provides an attack tree to execute malicious code. Most relevant of their findings on the malicious packages was that 55% had data exfiltration as the goal, 49% employed obfuscation tactics, and 56% triggered malicious behaviour on installation.

Another work by Marc Ohm et al. [51] presents a framework for dynamic analysis of software and its third-party dependencies named Buildwatch. This was accomplished through an analysis of compromised npm packages. They compared the compromised packages to a baseline of their legitimate benign versions using Cuckoo Sandbox. One of the key findings of the authors was that the malicious packages had an increase in STIX Cyber Observable Objects[52] compared to the benign versions, with these most often being related to operations on files or previous unseen processes used to run malicious additions. Limitations of the case study were the sample size of 6, only using samples that had malicious behaviour during installation, and the command and control servers were no longer operational.

Finally, in another work by Ohm *et al.* [53], the authors used the malicious samples from [50] together with the top 15 thousand npm packages (which were considered to be benign). They leveraged this combined dataset with a diverse set of commonly used supervised machine learning techniques to find the best-performing techniques to detect malicious packages. Their finding was that Kernel Support Vector Machines, Multi-Layer Perceptrons, and Random Forest produced the best results, and they then tested the results on real-world data. Combining the results of the three classifiers, they managed a True Positive Rate of 70% and identified 13 previously unknown malicious packages.

There have also been other recent academic research into using machine learning techniques to detect malicious packages, such as in Sejfia and Schäfer [54] where they presented AMALFI, which is lightweight, in that it requires only a few seconds to extract features and run the classifier against it. Furthermore, in Vu *et al.* [55], they presented LastPyMile a method of detecting discrepancies between source code and packages in PyPi by detecting differences between build artefacts of software packages and the source code repository itself, and in Scalco *et al.* [56] where they ported the LastPyMile [55] method to be used for JavaScript packages in the npm system. Another look into detecting malicious packages was in Ladisa *et al.* [57], where the focus was on open-source Java projects; they presented indicators of malicious behaviour in Java bytecode which was observable through static indicators.

2.4.3 Industry relevant detection

Looking at the previous two sections, the disparity between academic literature on detecting closed- and open-source software supply chain attacks in the form of detecting trojanized software or packages becomes apparent. Therefore, it is necessary to look at what the industry is doing to understand their best practices and the solutions they have developed for this problem. It is worth keeping in mind that industry papers do not go through the same peer review process that academic literature does, and when published on behalf of businesses, they can have the intention of marketing a solution.

The first industry white paper is one by Splunk (LaFerrera and Kovar [58]), and it is in some ways similar to Wang [8] in that it describes a method for detecting software supply chain attacks through outgoing traffic from critical servers and highly restricted and critical networks. They propose a method of anomaly detection through the use of the open-source methods JA3 and JA3s hashing, where JA3 is a method for generating an MD5 hash of a specific value found in the SSL/TLS handshake process, while JA3s is a method for calculating JA3 hash for server sessions. It is worth keeping in mind that this is more of a way of detecting anomalous behaviour from these critical networks and servers that do not generate large volumes of SSL/TLS events rather than detecting and preventing trojanized software installation.

The second industry white paper is by Intel (Zhang *et al.* [59]. It is a combination of work by Intel and Microsoft, and they developed a solution for utilising hardware-based control flow monitoring and anomaly detection by following the zero-trust principle and combining this with CPU telemetry and machine learning heuristics. The solution continuously monitors the behaviour of known programs and verifies that they behave as expected. Through testing, it detected process hijacking and software supply chain backdoors through behaviour deviation in otherwise benign processes while generating low amounts of false positives.

Chapter 3

Methodology

This chapter presents the methodology used within this thesis to answer the research questions from Section 1.5. It presents the method by which the dataset was collected and the included challenges therein, presents the experimental and hardware specifications used for the testing environment, before finally describing the techniques and tool usage for the malware analysis.

3.1 Dataset

The dataset is a collection of samples consisting of legitimate and benign software versions and the trojanized version of that application. Where possible, it was attempted to collect the benign version from either immediately before or between different versions of the trojanized software. However, this proved very difficult, leading to some samples being older by some degree or even newer software versions.

The malicious sample hashes were found by visiting reports and writeups by security researchers, experts, and companies. These hashes were then used as search criteria, where a small number of samples were downloaded from the sandbox platform Any.Run. Most of the malicious samples were found in VirusTotal, where the supervisor from CrossPoint Labs could download them, as this required an enterprise license.

The legitimate samples were downloaded using tools such as WayBackMachine[1] to visit a snapshot of a previous version of the suppliers' site through software repositories such as `http://www.oldversion[.]com/`, and in some cases, as samples uploaded to either VirusTotal or Any.Run.

Challenges

There have only been a few closed-source software supply chain attacks[60], so the sample set is small. In addition, it proved challenging to find and download most

[1]Link to the site by the internet archive: `https://archive.org/web/`

Table 3.1: Benign Samples

Software	Type	MD5	Compile Date	File Version
SmartPSS	exe	51ebe0db8fabace8ebc9d005b3c6cdec	2009-12-05 23:50:41	V2.002.0000009.0.R.190426
Swiss Ranger	exe	6120d14f8bb27b469724333947d5717e	2009-12-05 23:50:52	1.0.14.706
eGrabIt	exe	8a6783a0b5cff2932b35b8c58925f5ab	1999-04-08 22:24:47	3.1.0.85
Talk2M eCatcher	exe	877848de6f2135e2dbc7d036f6804528	1992-06-20 00:22:17	4.3.0.15531
CCleaner	exe	4d4f7f80a542a93d0d3c822153e2c254	2015-12-29 22:34:49	5.32.00.6129
MediaGet	exe	deb8a3ceadaa16500777aecb27d4b9bf	2018-12-13 13:05:01 UTC	2, 1, 0, 0
3CXDesktopApp	msi	20a680ee3826a8cb316a7bed58eb31c3	2023-01-23 08:30:50 UTC[a]	18.11.1213
Solarwinds Orion	dll	2d9b1245d42bb9f928da2528bb057de2	2020-08-11 15:40:55	2020.2.15300.12766
M.E.Doc	dll	23fdc5d07b0a7d743137cce040345ba2	2017-03-01 12:54:15 UTC	Unknown[b]

[a] This is the creation date, rather than compiling timestamp. [b] The version number is unknown.

Table 3.2: Trojanized Samples

Software	Type	MD5	Compile Date	File Version
SmartPSS	exe	1430291f2db13c3d94181ada91681408	2020-08-01 04:44:50	V2.002.0000007.0.R.181023-General-v1
Swiss Ranger	exe	e027d4395d9ac9cc980d6a91122d2d83	2011-05-28 18:04:38	1.0.14.706
eGrabIt	exe	1080e27b83c37dfeaa0daaa619bdf478	2007-03-31 17:09:46	3.0.0.82 (version 3.0 Build 82)
Talk2M eCatcher	exe	eb0dacdc8b346f44c8c370408bad4306	2007-03-31 17:09:46	4.0.0.13073
CCleaner	exe	75735db7291a19329190757437bdb847	2015-12-29 22:34:49	5.33.00.6162
MediaGet	exe	bc32bd0289e420add468315bc007a984	2018-02-06 23:19:12 UTC	2, 1, 0, 0
3CXDesktopApp	msi	0eeb1c0133eb4d571178b2d9d14ce3e9	2023-03-13 06:33:26 UTC[c]	18.12.0416
3CXDesktopApp	msi	f3d4144860ca10ba60f7ef4d176cc736	2023-03-03 12:21:46 UTC[d]	18.12.0407
Solarwinds Orion	dll	b91ce2fa41029f6955bff20079468448	2020-03-24 09:52:34	2019.4.5200.9083
M.E.Doc	dll	3efe62f6cb7285153114f888900a0962	2017-06-21 14:58:42 UTC	189

[c d] This is the creation date, rather than compiling timestamp.

of the malicious samples on free and open sources, which created a delay when having to request samples as they were found rather than being able to download them directly.

However, finding legitimate versions to compare them to was the most challenging part. This was because none of the suppliers had the old versions available on their sites. Therefore, WayBackMachine was used to find some of the samples by going to an old snapshot, but this was unreliable as, in most cases, the download link and binary were not saved. For some others, finding the samples through free software repositories was possible. However, this was also unreliable as the sample was bundled or packed with other software, even in cases where it was possible to find it.

The previous solutions were not applicable for samples part of enterprise solutions such as SolarWinds.Orion.Core.BusinessLayer.dll and ZvitPublishedObjects.dll. There was no way to download them using WayBackMachine, nor were they hosted in online software repositories. Eventually, hashes for legitimate versions were found on online forums, and they could then be downloaded from VirusTotal.

Contacting other researchers through email was attempted, but this proved ineffective. However, with time and effort, most legitimate versions were found with only two malicious samples having no pairs and therefore were excluded from the analysis.

3.2 Experimental Setup

This thesis's setup and experimentation environment were performed from a Windows 10 Desktop computer using a nested virtualisation scheme to conduct the experiments. The specifications and components for the host computer can be seen in Table 3.3. The main host used the hypervisor VirtualBox to host an Ubuntu Guest VM, which functioned as the Cuckoo host for the experiments. The Cuckoo Host specifications and software installations can be seen in Table 3.4. Lastly, the Cuckoo Host also had an installation of VirtualBox, which was used to host a Windows 7 VM. Ubuntu was chosen as the Cuckoo host and Windows 7 was selected for the guest because this was the recommended setup by the developers of the platform [34].

Table 3.3: Hardware and Software components of the Host Computer

Component	Specification
`CPU`	12th Gen Intel(R) Core(TM) i7-12700, 2100 Mhz, 12 Core, 20 Threads
`RAM`	32GB DDR5 4800MHz
`GPU`	NVIDIA GeForce RTX 3060 Ti
`C:\`	Kingston NV1 NVMe M.2 SSD 1TB 2100M MBps(R) / 1700 MBps (W)
`D:\`	WD Blue 3D 2.5" SSD 1TB 560 Bps(R) / 530 MBps (W)
`OS`	Microsoft Windows 10 Home Build 19044
`VirtualBox`	Version 6.1.40 r154048 (Qt5.6.2)

Cuckoo Host Setup

The Cuckoo host was installed according to the recommendations provided in the documentation for Cuckoo in [34]. This includes all the dependencies listed in their documentation. The Volatility module was used to handle memory analysis, while the Yara module and the Community signatures were used for signature detection. The Cuckoo was configured with just a single type of network available, INetSim, to simulate internet services. INetSim together with Tcpdump to then dump the network behaviour. INetSim was chosen to emulate network activity to keep the samples isolated for the safety of the host computer and network.

The Cuckoo Guest

The Cuckoo guest setup went through several iterations testing the VM detection capabilities. Initially, the machine was installed and configured as described in

Table 3.4: Guest VM Ubuntu Linux: Cuckoo Host

Component	Specification
`Processor`	8 Cores
`CPU Settings`	100% Execution Cap, Nested Virtualization
`RAM`	16GB
`Video Memory`	128 MB
`Disk`	200 GB VHD
`Kernel`	5.15.0-71-generic
`OS`	Ubuntu 20.04.6 LTS
Software	**Version**
`Cuckoo Sandbox`	2.0.7
`VirtualBox`	6.1.38_Ubuntu R153438
`Python2`	2.7.18
`Python3`	3.8.10
`Yarapython`	3.6.3
`TcpDump`	4.9.3
`INetSim`	1.3.1
`MongoDB`	v3.6.8
`Volatility`	2.6.1
`pydeep`	0.4
`openpyxl`	2.6.4
`ujson`	2.0.3
`libcap2-bin`	2.13.3-7ubuntu5.2

Cuckoo's documentation [34]. The Windows 7 machine was installed using an iso file on VirtualBox, the dependencies such as Python2 and pillow were installed, and the agent script was downloaded onto the machine. Then, the Windows firewall and automatic updates were turned off, and a Host-Only Adapter was set up using VirtualBox, which would isolate the machine allowing only communication to-and-from the host.

Initially, all the samples were run through that virtual machine, providing some starting results. However, as will be discussed in the results chapter, some of the samples did not behave any differently during behavioural analysis. It was reasoned that this could be due to the samples detecting that they were in a virtual environment, and Paranoid Fish[2] was used to harden the VM for anti-VM techniques. Pafish was executed through Cuckoo Sandbox as this best simulated the experiment environment. The Pafish report initially contained 25 detections, ranging from hypervisor detections, low processor count, and several detections on VirtualBox artefacts. The first changes made were made in the VirtualBox GUI. These changes can be seen in Table 3.5, where the top "GuestInitial" is the initial

[2]`https://github.com/a0rtega/pafish`

specification while "GuestFinal" is the final.

Table 3.5: Nested Virtualised Windows: Cuckoo Guest Ver1

GuestInitial	**Specification**
`Processor`	1 Core
`CPU Settings`	100% Execution Cap, Nested Virtualization
`RAM`	2 GB
`Video Memory`	30 MB
`Disk`	80 GB VHD
`OS`	Windows 7 SP1 Build 7601
`Paravirtualisation`	Hyper-V
Software	**Version**
`VirtualBox Guest Additions`	6.1.38
`Python2`	2.7.18
`Python2.5.3 Pillow`	Pillow-2.5.3
GuestFinal	**Specification**
`Processor`	2 Cores
`CPU Settings`	100% Execution Cap, Nested Virtualization
`RAM`	4 GB
`Video Memory`	128 MB
`Disk`	80 GB VHD
`OS`	Windows 7 SP1 Build 7601
`Paravirtualisation`	Legacy
Software	**Version**
`Python2`	2.7.8

In addition to adding more RAM, video memory, and an additional CPU core, the paravirtualisation interface was changed to legacy rather than Hyper-V. This was as the Hyper-V interface reveals that the system is running on a hypervisor (by default named VBoxVBoxVBox), while the legacy option did not. Two guides were followed while hardening the VM, those being [61] and [62]. Noisy services such as User Access Control and Security Center were turned off, and VirtualBox guest additions were uninstalled. From these two guides were also a total of three scripts which aided in hardening the VM:

- The first script is from [61], there was a script to fake hardware components inside the VM[3]. However, the script was intended for Windows, not Linux. Therefore, The relevant commands were, ported to a very simple shell script for this purpose. The VirtualBox commands allowed for faking several hardware components from inside the VM. The relevant commands can be seen

[3]`https://github.com/xyafes/VBoxAntiDetection/blob/main/statick.bat`

in Code listing A.1.
- The second script was also from [61], the script, named *dynamic.ps1*[4] was intended to run on the Cuckoo Guest. It renames several registry keys that can be used to recognise a virtual environment by an attacker.
- The last script is from [62]. It is a registry file and it changes a few registry keys related to the bios while deleting several related to VirtualBox. The script can be seen in Code listing A.2.

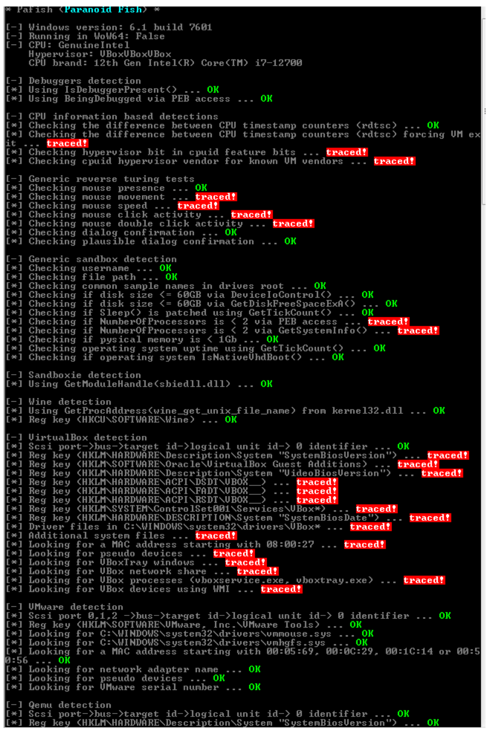
The results of the changes can be seen in Figure 3.1, where there was only a total of 2 detections after the configurations. The two detections are the "difference between CPU time-stamp counters and the difference between CPU time-stamp counters forcing VM exit." These two techniques use RDTSC instructions to reveal if it is running in a virtual machine. It does this by retrieving an in-CPU 64-bit counter that increases at a constant speed close to the CPU frequency [63]. This value is called the time-stamp counter (TSC). Measuring TSC before and after an operation execution allows one to find the time elapsed, which can detect a virtual environment [63]. The author of the thesis was able to find some possible workaround for this using some other virtualisation tools but was unable to find one for VirtualBox. The comparative results for the Pafish run can be found in Figure 3.1, while the individual figures can be found in the appendix under A.1 and A.2.

3.3 Experiments & Analysis

This section describes both the tools and methods used to conduct the differential analysis on the trojanized and legitimate samples. It first describes the methods and experiments used during the static differential analysis before doing the same for the dynamic differential analysis.

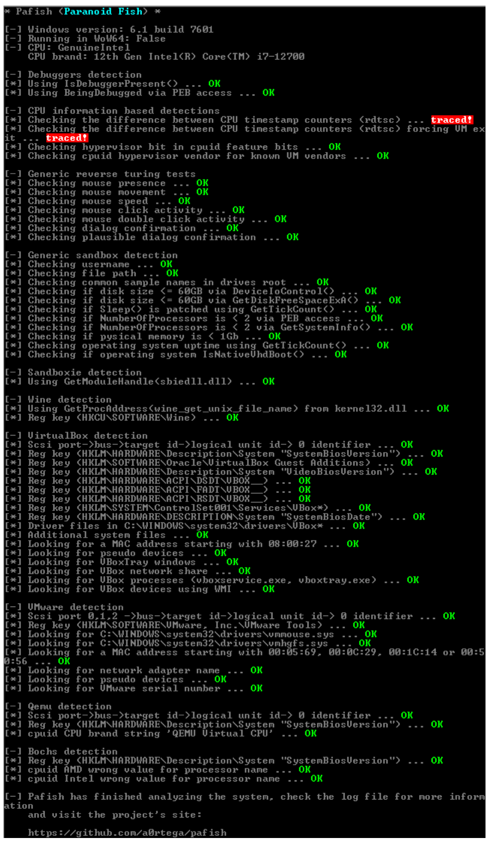
For this thesis, it was decided to focus on techniques that do not require a reverse engineering specialist or high-level expertise, so disassemblers such as IDA and hands-on debuggers were not included in the methods. This is as most organisations have several tens, if not hundreds, of different software components. When one considers that much of this software receives regular updates, it becomes apparent that it would require much time to review each sample and analyse it in disassemblers and debuggers. Therefore, it would be a more scalable solution to have a system in place that compares the previous version to the next and reports if there are suspicious changes. However, designing such a system and complicated work beyond the scope of a master's thesis. Therefore, this thesis seeks to find what indicators can be found with sandboxing and basic static analysis techniques that can be automated.

[4]`https://github.com/xyafes/VBoxAntiDetection/blob/main/dynamic.ps1`



```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: VBoxVBoxVBox
    CPU brand: 12th Gen Intel(R) Core(TM) i7-12700

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

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

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... traced!
[*] Checking mouse speed ... traced!
[*] Checking mouse click activity ... traced!
[*] Checking mouse double click activity ... traced!
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

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

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.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-> 0 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\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... traced!
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... traced!
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... traced!
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... traced!
[*] Additional system files ... traced!
[*] Looking for a MAC address starting with 08:00:27 ... traced!
[*] Looking for pseudo devices ... traced!
[*] Looking for VBoxTray windows ... traced!
[*] Looking for VBox network share ... traced!
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... traced!
[*] Looking for VBox devices using WMI ... traced!

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 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-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
```

(a) Initial PaFish Detections



```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    CPU brand: 12th Gen Intel(R) Core(TM) i7-12700

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... 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 reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... OK
[*] Checking mouse speed ... OK
[*] Checking mouse click activity ... OK
[*] Checking mouse double click activity ... OK
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.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-> 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\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 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-> 0 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

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

(b) Initial PaFish Detections

Figure 3.1: Figure 3.1a to the left shows the initial 25 detections, while Figure 3.1a to the right shows the final 2

3.3.1 Static Differential Analysis

The purpose of this section is to describe how the static differential analysis was performed on the samples. Two different environments were set up for the static differential analysis. The first was a separate snapshot of the Ubuntu environment used to run Cuckoo Sandbox, which had Stringshifter [5], Manalyze[6], PeFile [7], and FLOSS[8] installed. The second was a separate snapshot of the Windows 7 VM guest used for running the sandboxing malware, with Exeinfo PE[9] and PeStudio [10].

The first part of the analysis was undertaken from the Windows 7 VM, and the first step was loading each sample pair into Exeinfo PE. This can give an initial understanding of the software, allowing us to observe any changes in how

[5]`https://github.com/mandiant/stringsifter`
[6]`https://github.com/JusticeRage/Manalyze`
[7]`https://github.com/erocarrera/pefile`
[8]`https://github.com/mandiant/flare-floss`
[9]`https://github.com/ExeinfoASL/ASL`
[10]`https://www.winitor.com/`

the sample was compiled and if there are changes in the software packaging. In addition, should the sample be packaged, Exeinfo Pe can be used to attempt to unpackage it.

The next step is to load the sample into PeStudio. PeStudio provides a lot of static information on a PeFile and provides an initial analysis by providing indicators of maliciousness and anomalies within the file. The two files are both loaded into PeStudio, and the two outputs are compared against each other. Each of the tabs is manually compared to each other, focusing on the data highlighted by PeStudio. PeStudio is used to compare and note down the following information:

- General information:
    - Global Entropy, File Size, Signature, Version Info
- Sections
    - Changes in the names, names, entropy, file ratio, raw size, virtual size, and characteristics.
- Library/Imports/Exports
    - Changes in the number of imported functions/API, exported functions, or libraries.
- Strings
    - Changes in the number of flagged strings, which is are strings seen in relation to previous attacks.
    - PeStudio also tags strings with techniques from the MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework. This is a curated knowledge base for known adversarial behaviour through attributing tactics and technique abstraction [64]. Most relevant here is that it relates a string or event to a type of method of attack, such as process injection or data obfuscation.
    - Any addition of strings marked with under group or labels that indicate obfuscation or evasion.
- Overlay
    - Changes, addition, or removal of the overlay.

The next step was to use a combination of FLOSS and Stringsifter ranking functions. First, FlOSS was run against both the legitimate and trojanized binary. Then this output was piped through the ranking function of Stringsifter, which ranked the strings based on suspiciousness and outputted the suspicious strings from both samples into four different CSV files. The CSV files were in the interval top 10-, 100-, 1000-, and 10 000-suspicious strings according to the Stringsifter ML model. These four files then had their output compared to each other and generated four lists with the unique strings from the malicious sample for each top interval set.

The intended goal of using Stringsifter and Floss was to identify strings not found by PeStudio, de-obfuscate strings, remove those in the legitimate software, and then analyse the remainder for potential maliciousness and suspiciousness. Different amounts of strings were extracted to see if there was a cutoff point for useful indicators and if there was a similarity between the upper and lower information gain area. Compared to PeStudio, Floss and Stringsifter neither provide additional information to the strings nor flag them. Following below is an example of suspicious strings:

- New Suspicious Top-Level Domains:
    - cf, club, cn, co, ga, gq, icu, info, ml, pw, ru, tk, tokyo, top, work, xyz. [11]
- New Possible Code Access:
    - Examples: .php, .sh, .bat, .ps
- New URL or Domain
    - Domains or URLs not previously accessed in legitimate software. So if the software contacts google.com in the legitimate and in the malicious, it visits something.google.com, then it is not suspicious.
- New Encoded Strings:
    - Ex. Base64, XOR
- Possible Credential Access:
    - Presence of strings indicating possible theft of logins such as passwords and usernames.
- Possible C2:
    - Strings that hint at possible command and control traffic.

The last step was using a Python script that utilises the library Pefile to export these functions to a CSV file and perform general comparisons between the two samples. The script does the following actions:

- Calculates the global entropy of the two samples using the Shannon entropy formula.
- Fetches the entry point and file size, then compares them to see if there is a difference.
- Enumerates the sections in the file, gets the different values in each section, and calculates the entropy.
- The script reviews all the imported libraries, functions, and API calls in both samples and builds one list for the removed imports and one for the new ones in the malicious sample.
- Lastly, prints the extracted information to a CSV file.

The information gathered from the analysis was continuously tracked in a

[11] Based on the list from table1 in `https://unit42.paloaltonetworks.com/top-level-domains-cybercrime/`

Google Sheets sheet with a unique page for each binary. This Google Sheets sheet highlighted where there were changes, such as an increase or decrease of entropy, or if there was no change between the binaries. Eventually, when all samples were analysed, a single Google Sheets page was created to contain the observed static differences between the samples.

3.3.2 Dynamic Differential behaviour Analysis

The purpose of this section is to describe how the trojanized software and the legitimate software were dynamically analysed using sandboxing software and then how the resulting reports from the sandbox were used to perform a differential analysis of the results. The two environments used to run the samples were the Cuckoo sandbox with a few different configurations and the online sandbox Triage[12].

The samples were run locally using Cuckoo Sandbox. They were run both on a hardened and unhardened VM with INetSim simulating an internet connection and without INetSim doing so. First, the information in the static analysis part of Cuckoo from both files was compared to the results from the previous static analysis. Then, the detected Cuckoo and YARA signatures were noted in Google Sheets and compared to each other, where changes in any detections from benign to malicious samples were highlighted. Next, the process tree, network traffic, dropped files, dropped buffers, and process memory was compared between the binaries. This was to understand if the malicious sample had any behavioural differences from the benign sample that was not detected by any signature. A focus here was inspecting if the process behaved differently by starting a new or changed process, connecting outwards to new domains or IP addresses, or if there were changed contents or files in the memory or buffer of the sample.

Lastly, the samples were uploaded and run in the online sandbox Triage with an internet connection. Additionally, samples that would not run on the local VM due to dependency or version issues were run with and without an internet connection in the Triage sandbox. This allowed one to observe if there was any difference in behaviour when an internet connection was added, and it could support and add to the evidence collected from the Cuckoo sandbox runs.

[12]`https://tria.ge/`

Chapter 4

Results

This section is intended to summarise the different findings from the analysis together into a clear picture of the differences and indicators found across the samples. First, it presents the results from the static analysis before doing the same for the dynamic analysis. Lastly, it puts these results into the perspective of the research question and how these results contribute to a better understanding of the problem by providing perspective to the methods and places where one can look to detect trojanized closed-source software.

4.1 Analysis of Trojanized Software

This section provides the analysis of the individuals' samples and their findings. It is worth mentioning that if a step from the analysis method is not mentioned, then one can consider that the technique did not provide any findings, new information, and insight for the analysis. Also, the tables and contents only contain what differed from the legitimate sample. This means if they were both unsigned, imported 2 functions from Kernel32.dll, and triggered a signature for debugging detection in Cuckoo, then this is not mentioned. This is because we concern ourselves with the differences rather than the similarities in the files for the purpose of this analysis.

4.1.1 NotPetya

In June 2017, Ukraine was the target of a Wiper malware disguised as ransomware. However, the attack spread beyond Ukraine and hit several other regions, with estimates of the damages caused being more than 10 billion dollars [65]. It spread through a trojanized version of M.E.Docs containing a backdoor in the 'ZvitPublishedObjects.dll' module [66]. 'ZvitPublishedObjects.dll' had a few static changes from the benign sample, and we could not extract behavioural data from this sample even after attempting several analysis methods. Nevertheless, the differential analysis results can be found in Table 4.1:

As seen in Table 4.1 above, PeStudio provided most of the data on this sample pair. There were some minor changes in entropy and file size, some changes in

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	5,121	5,122	+0,001	PeStudio
File Size	5094400	5207040	+112640	PeStudio
PDB Path	`c:\branch\source\ZvitPublishedObjects\obj\Release\ZvitPublishedObjects.pdb`	`c:\branch2\source\ZvitPublishedObjects\obj\Release\ZvitPublishedObjects.pdb`	Very Slight change.	PeStudio
.text Entropy	5,112	5,123	+0,011	PeStudio
.text Raw-size	5092352	5204992	+112640	PeStudio
.text Virtual-size	5091988	5204724	+112736	PeStudio
.rsrc Entropy	2,835	2,831	-0,004	PeStudio
Flagged Strings	17	22	+5	PeStudio
MITRE Strings	12	17	+5	PeStudio

Table 4.1: NotPetya: Changes in entropy, size, PDB path, and flagged/MITRE strings.

two sections, an increased amount of flagged strings, and several possible MITRE techniques. Some of the most interesting observations are the addition of 6 possible obfuscation through encryption techniques and the two related to packing. When one references this finding against the analyses by ESET [66] and Talos [67], we can see that these strings were part of functions used by the attacker to send and receive command and control traffic.

In addition, the list's top lists generated through the use of FLOSS and Stringsifter did reveal some possible suspicious indicators. For example, there were no '.ru' domains in the benign file, while in the malicious one, there was a sudden addition of seven '.ru' domains, where 6 of them were email domains, and one was a search engine. While this would be very suspicious in most software, M.E.docs was (and is, at the time of writing) Ukrainian software at the time of the attack, so it might not be as questionable there as in other software. Lastly, some strings hinted at possible credential theft, network connections, and script download.

Type	**New Strings found**	**Method**
MITRE Data Obfuscation	TripleDESCryptoServiceProvider, CipherMode, CreateDecryptor, GZipStream, Encryption, Decryption	PeStudio
MITRE Software Packing	Compress, Decompress	PeStudio
MITRE Access Token Manipulation	OpenProcessToken, GetTokenInformation	PeStudio & FLOSS & Ranked_Strings
MITRE Execution through API	set_UseShellExecute	PeStudio
MITRE Modify Registry	CreateSubKey	PeStudio
Possible C2	`"AbsoluteUri:{0}Host:{1}Port:{2}UserName:{3}Password:{4}`	FLOSS & Ranked_Strings
Possible Credential Access	`"edropu:{0}name:{1}smtpServer:{2}smtpLogin:{3}smtpName:{4}smtpPass:{5}email:{6}"`	FLOSS & Ranked_Strings
Suspicious Top-Level Domains	Yandex.ru, mail.ru, ya.ru, list.ru, inbox.ru, bk.ru, and "mail.ru"	FLOSS & Ranked_Strings
Possible Code Access	`http://www.me-doc[.]com.ua/other_scripts/to_is_pro_execute_medoc.php`	FLOSS & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 433 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

Table 4.2: NotPetya: Static changes in strings

4.1.2 Solorigate

The backdoor in Solarwinds Orion Software suite was in the dll file SolarWinds.Orion.Core.BusinessLayer.dll, and the backdoor is composed of more than 4000 lines of code [68]. Microsoft conducted an in-depth analysis of the attack in [68], where using reverse-engineering forensic techniques, they were able to uncover that the backdoor's capabilities included 13 subclasses and 16 methods, as well as conducting an extensive check for the environment to ensure that it would only be running in an enterprise network rather than on an analysts machine. The analysis by Microsoft provides a good reference for what malicious artefacts we can discover using basic static methods compared to their advanced static and dynamic methods.

It is worth mentioning that just like the NotPetya backdoor, we could not extract dynamic behaviour from the sample, having the same problem that the sample is part of a larger software environment.

The Solorigate sample had one of the highest increased global entropies of all files while having an increased entropy in the executable section (*.text*) and a slightly lower entropy in the resources (*.rsrc*) section. There was also a slight increase in global file size due to the .text section being marginally larger by about 33kb, which is most likely due to the 4000 lines of code added in the malicious sample compared to the benign. Where the trojanized sample starts to show a real difference compared to the legitimate one is in the strings composing the

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	5,551	5,583	+0,032	PeStudio
File Size	977896	1011032	+33136	PeStudio
.text Entropy	5,538	5,569	+0,031	PeStudio
.text Raw-rize	968192	1001472	33280	PeStudio
.text Virtual-size	968112	1001340	33228	PeStudio
.text File-ratio	99,01%	99,05%	+0,04	PeStudio
.rsrc Entropy	3,034	3,016	-0,018	PeStudio
.rsrc Virtual-size	1324	1312	-12	PeStudio
.rsrc File-Ratio	0,16	0,15	-0,01	PeStudio
Flagged Strings	5	18	+13	PeStudio
MITRE Strings	7	12	5	PeStudio

Table 4.3: SolarWinds.Orion.Core.BusinessLayer.dll: changes in entropy, size, and amount of flagged/MITRE strings

program. PeStudio reported increased flagged strings and a general increase in MITRE-tagged strings. The table for these changes with the specific values can be seen in Table 4.3.

The extra flagged strings and MITRE techniques become apparent due to several strings related to the Windows API and systems function, such as those related to processes and token manipulation. Amongst those functions are the names of Windows utilities for Base64 encoding and decoding and functions for software packing and unpacking. Also present in the malicious sample is 162 compressed base64 encoded strings, which, when decoded, reveal several WMI-queries for uncovering the system environment, 2 new domains (*avsvmcloud[.]com* and *api.solarwinds[.]com*), and new possible access token manipulation functions.

When we reference these findings against the analysis by Microsoft, we find that the new domain *avsvmcloud[.]com*, we find that this was the location of the C2 server, while the *api.solarwinds[.]com* domain was used to test network connectivity after a blocklisted drivers test [3]. Additionally, several hardcoded IP addresses and subnet masks were encoded in the file. According to FireEye [3], these IPs were used for the C2 traffic. Therefore, when the backdoor communicates with the C2 server, it receives a DNS record that would resolve to one of these IP subnets, which would decide if it should continue beaconing, stop beaconing, or start stage two of the attack[3]. A table containing the interesting artefacts from the differential string analysis can be seen in Table 4.4

Type	**New Strings found**	**Method**
MITRE Data Obfuscation	Base64Decode, FromBase64String, ToBase64String	PeStudio
MITRE Software Packing	Compress, Decompress, Deflate, Inflate	PeStudio
MITRE Access Token Manipulation	LookupPrivilegeValue, OpenProcessToken, AdjustTokenPrivileges, SeRestorePrivilege**(O)**, SeTakeOwnershipPrivilege**(O)**	PeStudio & FLOSS & Ranked_Strings
MITRE Execution through API	set_UseShellExecute	PeStudio
MITRE Disabling Security Tools	Kill	PeStudio
MITRE Process Discovery	GetCurrentProcess	PeStudio
Encoded Strings(O)	162 Base64 and deflated strings(O). Amongst these strings are appsync-api, eu-west-1, us-west-2, us-east-1, us-east-2.	FLOSS & Ranked_Strings
IP Addresses(O)	13 new IP addresses (11 unique), 17 network masks. **IP Addresses:** 74.114.24.0, 71.152.53.0, 41.84.159.0, 41.84.159.0, 217.163.7.0, 20.140.0.0, 154.118.140.0, 131.228.12.0, 172.16.0.0, 144.86.226.0, 10.0.0.0, 99.79.0.0, 99.79.0.0 **Network Masks:** 255.0.0.0, 255.255.0.0, 255.255.0.0, 255.255.248.0, 255.255.254.0, 255.255.254.0, 255.255.254.0, 255.255.252.0, 255.255.252.0,255.240.0.0, 255.240.0.0, 255.254.0.0, 255.254.0.0, 255.255.255.0, 240.0.0.0, 240.0.0.0, 224.0.0.0	FLOSS & Ranked_Strings
New URL or Domain	2 Domains in Base64 Encoding (avsvmcloud.com, api.solarwinds.com)**(O)**, 9 New URLs	FLOSS & Ranked_Strings
WMI Queries**(O)**	Select * From Win32_OperatingSystem, Select From Win32_Process, Select From Win32_SystemDriver, Select From Win32_NetworkAdapterConfiguration where IPEnabled̄true	FLOSS & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 838 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

The **(O)** represents that the string was encoded/obfuscated.

Table 4.4: SolarWinds.Orion.Core.BusinessLayer.dll: Static changes in strings

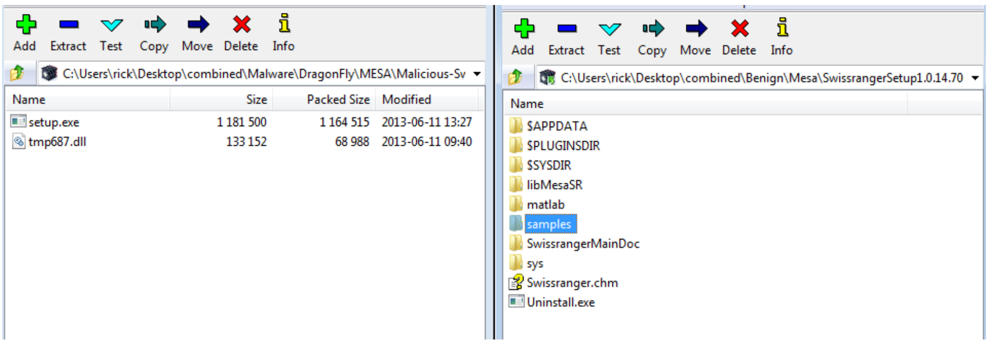
4.1.3 Dragonfly Campaign

In 2013 and 2014 the Dragonfly (Energetic Bear) Group[1] successfully compromised the support sites of three ICS vendors, MESA Imaging, eWon, and MB Connect Line [69]. From there, the attackers managed to trojanize five different legitimate software on the site. The attackers trojanized the software with either the Sysmain RAT or the Havex Rat [70]. An in-depth analysis was conducted by Langill [69] on behalf of Belden Concluded that the target of the attack was the pharmaceutical industry.

Swiss Ranger

The Swiss Ranger software was made by MESA imaging and is the driver software for their industrial cameras Swiss Ranger [70]. It was the first trojanized software in 2013, and it was implanted with the Sysmain RAT. It remained for 6 weeks until it was detected and removed. We were able to run both static and dynamic differential analyses on the sample.

The trojanized sample had several differences in the static analysis of the PE file structure and sections. The trojanized Swiss Ranger was 130KB larger than the benign, had no version info, and had its packer changed from Nullsoft to Zip archive. Attempting to unzip the archives reveals that the two files are quite different inside, with the malicious having the setup file for Swiss Ranger and the RAT dll. In contrast, the legitimate one had a directory containing several installation files. This can be seen in Figure 4.1.



C:\Users\rick\Desktop\combined\Malware\DragonFly\MESA\Malicious-Sv

Name	Size	Packed Size	Modified
setup.exe	1 181 500	1 164 515	2013-06-11 13:27
tmp687.dll	133 152	68 988	2013-06-11 09:40

C:\Users\rick\Desktop\combined\Benign\Mesa\SwissrangerSetup1.0.14.70

Name
$APPDATA
$PLUGINSDIR
$SYSDIR
libMesaSR
matlab
samples
SwissrangerMainDoc
sys
Swissranger.chm
Uninstall.exe

Figure 4.1: The left side is the malicious sample, while the right is the benign.

There were several changes in the PE sections, with the section '.ndata' missing and the section '.CRT' added to the trojanized version. There were also several changes across the sections, with both decreases in entropy in some sections, such as '.data', and increases in the '.text' and '.rsrc' sections. The complete list of these changes can be found in Table A.1.

Looking at the files' resources and imports, we find that the malicious sample has more resources, a changed manifest, and imports more libraries. For example,

[1]MITRE has a subpage on this group in `https://attack.mitre.org/groups/G0035/`

in Table 4.5, we can see that there has been a change in imported libraries, with one removed and two added, and some differences in the number of imports from the different libraries.

Type	**Benign**	**Malicious**
Resources Amount	12	15
Manifest	XML 1.0 document, ASCII text, with very long lines, with no line terminators	XML 1.0 document, ASCII text, with CRLF line terminators
Libraries	8	9
Library Imports	USER32.dll: **62**, KERNEL32.dll: **59**, ADVAPI32.dll: **9**, GDI32.dll: 8, SHELL32.dll: **6**, COMCTL32.dll: **4**, ole32.dll: **4**, **VERSION.dll: 3**	USER32.dll: **55**, KERNEL32.dll: **70**, ADVAPI32.dll: **5**, GDI32.dll: 8, SHELL32.dll: **8**, COMCTL32.dll: **2**, ole32.dll: **5**, **OLEAUT32.dll: 1**, **SHLWAPI.dll: 1**

Table 4.5: Havex-SwissRanger: Static Differential Analysis Resources & Library

Type	**New Strings found**	**Method**
MITRE Data Obfuscation	extract	PeStudio
MITRE Access Token Manipulation	runas	PeStudio
MITRE Execution through API	ShellExecuteEx	PeStudio
MITRE Process Injection	SetDllDirectory	PeStudio
MITRE Windows Discovery	GetWindowText	PeStudio
Possible LotL	`Setup=cmd/c%temp%\setup.exe&c:\windows\system32\rundll32.exe%temp%\tmp687.dll,RunDllEntry p.exe&c:\windows\system32\rundll32.exe%temp%\tmp687.dll,RunDllEntry`	Floss & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 3915 different strings than the benign out of the top 10 000 strings.	FLOSS & String-sifter

Table 4.6: Havex-SwissRanger: Static Differential Analysis Strings

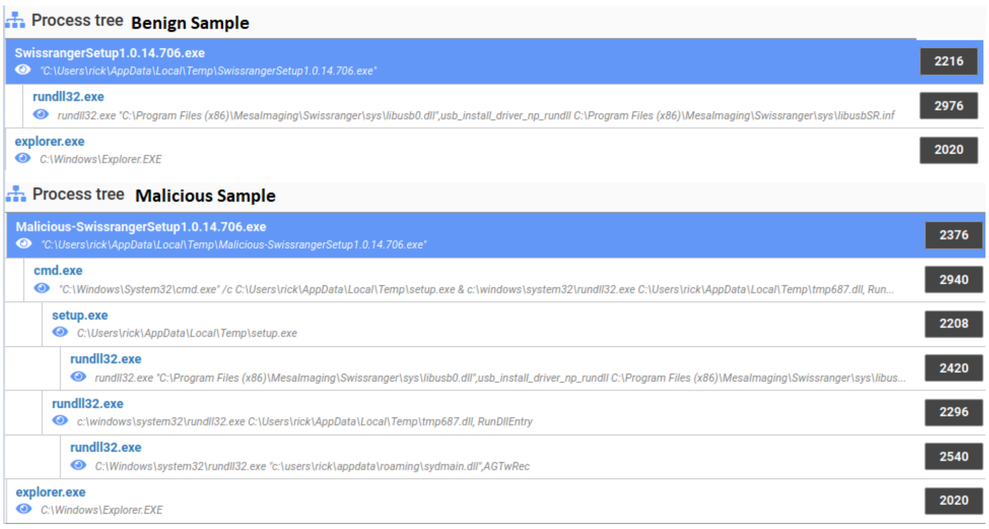
Looking at the strings of the PE file, we can see indications of unpacking of obfuscated files with the ’extract’ being referenced. We also see some techniques related to token manipulation, execution through API, process injection, and Windows directory. There was a general reduction in flags and tagged MITRE techniques in the malicious version compared to the benign. However, this can be attributed to the malicious sample being essentially "double-packed" with the legitimate setup being bundled with the 'dll' inside the PKZIP executable file. The most interesting artefact from the string analysis is the presence of a Living of the Land (LotL) technique with the two commands:

- `Setup=cmd/c%temp%\setup.exe&c:\windows\system32\rundll32.exe%temp%\tmp687.dll,RunDllEntry`
- `p.exe&c:\windows\system32\rundll32.exe%temp%\tmp687.dll,RunDllEntry`

Here we can see the setup and the RAT *'tmp687.dll'* being ran by the PKZIP executable file. The table for this can be seen in Table 4.6.

Lastly comes the dynamic differential behavioural analysis using Cuckoo Sandbox. The malicious version deviated heavily from the legitimate sample and triggered several more signatures. Amongst the signature detections was an attempt at using sleep to delay analysis, increased read-write-execute memory events, possible keylogging, dropping a binary and executing it, and more.

As observed in the static string analysis, tmp687.dll is run with the parameter RunDllEntry by Rundll32.dll. This process triggers several detections for both process discovery and injection. The process tree for the sandbox run can be seen in Figure 4.2. At the same time, the complete list of different dynamic behaviour in the trojanized sample can be found in the appendix Table A.2.



```
Process tree  Benign Sample
SwissrangerSetup1.0.14.706.exe                                                    2216
  "C:\Users\rick\AppData\Local\Temp\SwissrangerSetup1.0.14.706.exe"
  rundll32.exe                                                                    2976
    rundll32.exe "C:\Program Files (x86)\Mesalmaging\Swissranger\sys\libusb0.dll",usb_install_driver_np_rundll C:\Program Files (x86)\Mesalmaging\Swissranger\sys\libusbSR.inf
explorer.exe                                                                      2020
  C:\Windows\Explorer.EXE

Process tree  Malicious Sample
Malicious-SwissrangerSetup1.0.14.706.exe                                          2376
  "C:\Users\rick\AppData\Local\Temp\Malicious-SwissrangerSetup1.0.14.706.exe"
  cmd.exe                                                                         2940
    "C:\Windows\System32\cmd.exe" /c C:\Users\rick\AppData\Local\Temp\setup.exe & c:\windows\system32\rundll32.exe C:\Users\rick\AppData\Local\Temp\tmp687.dll, Run...
    setup.exe                                                                     2208
      C:\Users\rick\AppData\Local\Temp\setup.exe
      rundll32.exe                                                                2420
        rundll32.exe "C:\Program Files (x86)\Mesalmaging\Swissranger\sys\libusb0.dll",usb_install_driver_np_rundll C:\Program Files (x86)\Mesalmaging\Swissranger\sys\libus...
    rundll32.exe                                                                  2296
      c:\windows\system32\rundll32.exe C:\Users\rick\AppData\Local\Temp\tmp687.dll, RunDllEntry
      rundll32.exe                                                                2540
        C:\Windows\system32\rundll32.exe "c:\users\rick\appdata\roaming\sydmain.dll",AGTwRec
explorer.exe                                                                      2020
  C:\Windows\Explorer.EXE
```

Figure 4.2: The benign process tree is at top, with the malicious at the bottom

Talk2M eCatcher

Talk2m eCatcher is a remote access software developed by eWon, and in 2014 it was trojanized by the Dragonfly group as well. It differs from the Swiss Ranger compromise in that it was implanted with a newer RAT called Haxex rather than Sysmain [70]. The trojanized version remained at the Ewon support site for 10 days until it was taken down [70].

File and Sections:
There is no difference in the entropy of the file. However, the malicious sample is not signed and is missing version info. Additionally, the packer has changed from Inno/Borland Delph to Nullsoft. The section names have also completely changed from descriptive names in legitimate such as CODE, DATA, and BSS, to generic such as '.text' and '.data'. Additionally, there is a significant entropy increase in the remaining '.rdata' section, some, but not all, of which can be attributed to the benign one having a '.idata' and '.rdata' sections. This sample also had the setup and malicious dll file within the Nullsoft executable compared to a folder containing installation files in the legitimate one. The list of differences can be seen in the appendix Table A.3.

Resources and Imports:
The malicious sample had increased resources up to 14 from 6, and the executable's manifest was also changed to have CRLF line terminators rather than very long lines with no line terminators. There was also an increase of +70 functions imported, up to 152 from 82. Furthermore, the trojanized sample imports more libraries and has changes in the libraries imported. The table containing the differences between the samples can be found in appendix Table A.4.

String Differences:

As can be seen in Table 4.7, the sample has several new strings related to modifying, deleting, and enumerating registry keys and values. Furthermore, it has the ShellExecute and LoadLibraryEx. The malicious sample also has 5 additional URLs within its strings, those being for OCSP domains belonging to Symantec and Tawte. This addition may be due to the Nullsoft installer or the malware authors' expanded capability to check the validity and revocation status certificates. While some Tawte domains are present in the legitimate ones, the Symantec domains are unique to the malicious sample.

Behavioural Differences:
The behaviour of the trojanized version is also quite different from the legitimate benign sample. However, it behaves similarly to the Sysmain RAT from the Swiss Ranger sample. Like the Sysmain RAT, the Havex RAT also has behaviours indicative of process discovery, injection, and enumeration. Furthermore, it also injects into processes. However, it differs from Sysmain (and the legitimate eCather) because it checks the adapter address, attempts to hide the malicious files, and has HTTP traffic towards a C2 domain through a POST Request containing system information. Lastly, it configured WPAD proxy configurations possibly to attempt to intercept traffic, and this sample also has more 'Allocates Read-Write-Execute Memory' events

Type	**New Strings found**	**Method**
New URL or Domain	Two domains (Symantec & tawte) both OCSP links. Total of 5 URLs. All towards OCSP domain	FLOSS & Ranked_Strings
MITRE Execution through API	ShellExecute, LoadLibraryEx	PeStudio & Floss & Ranked_Strings
MITRE Modify Registry	RegCreateKeyEx, RegSetValueEx	PeStudio & Floss & Ranked_Strings
MITRE Data Destruction	RegDeleteKey, RegDeleteValue	PeStudio & Floss & Ranked_Strings
MITRE Query Registry	RegEnumKey	PeStudio & Floss & Ranked_Strings
MITRE File and Directory Discovery	FindFirstFile, FindNextFile	PeStudio & Floss & Ranked_Strings
MITRE Remote File Copy	MoveFile, MoveFileEx, CopyFile	PeStudio & Floss & Ranked_Strings
MITRE System Time Discovery	GetTickCount	PeStudio & Floss & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 1335 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

PeStudio could not open the file, though the unique strings extracted with A.3 from the malicious sample could be put into PeStudio.

Table 4.7: Havex-Talk2M eCatcher: Static Differential Analysis Strings

than the legitimate version. The table of differences from the behavioural data can be seen in the appendix Table A.5.

eGrabit

eGrabit is another eWON software trojanized with the Havex RAT by the Dragonfly group. It is not public knowledge how long this malicious version was up.

File and Sections:
The eGrabit sample pairs are very similar to the eCatcher pair, which is reasonable as it is the same developer for the software, the same attacker, and it was conducted simultaneously. In this sample, we also observe a change in that the trojanized version is neither signed nor has version info. Furthermore, the packer (or installer) has changed from 'Wise Installer' to Nullsoft, and a new section named 'ndata' has been added to the malicious. There is a very slight increase in global entropy but several significant changes in entropy for the sections. Especially the executable section '.text' is increased by 0,826. The rest of the differences can be seen in table Table A.6. However, worth mentioning is an increase of flagged and MITRE-tagged strings by PeStudio.

Resources and Imports:
The malicious version has an increase in the number of resources, amount of

libraries, and imported functions. For example, the number of functions in the malicious library is 151 compared to the 15 from the benign. The trojanized eGrabit also has no manifest compared to the benign one that has one. The details of the changes can be found in the appendix under Table A.7.

String Differences:
There was a large difference in the number of imported libraries, which is matched in the differences between the flagged and mitre-technique tagged strings by PeStudio. In addition, the malicious sample has several new strings related to the Windows API, indicating that it has the added functionality (compared to the benign sample) to manipulate clipboard data, interact with the registry, process injections, and more. It is also worth mentioning that when the script using Floss & Stringsifter's ranking method was run against the malicious and benign sample, it found that in the top 10 000 ranked strings, only 69 of them were equal. This was the most significant difference across all the sample pairs and was likely caused due to the change in packers. This table can be found in the appendix Table A.8.

Behavioural Differences:
The observed dynamic differences can be found in the appendix under Table A.9, which look almost identical to the behaviour of eCatcher. This can be explained by the fact that both trojanized samples carry the same version of the Havex RAT; however, the two samples have different C2 domains.

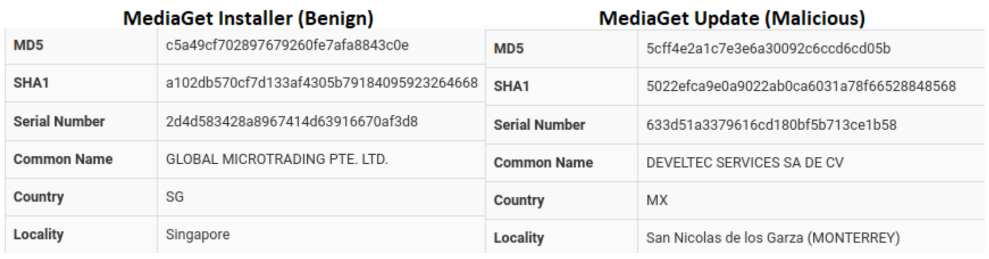
4.1.4 DoFoil

MediaGet is a Russian BitTorrent client, and in early March 2018, it was used as part of a large-scale coin mining attack [71]. The update server used for the application had been compromised, leading to the legitimate software downloading an update containing a new 'mediaget.exe' almost identical to the legitimate version but with additional backdoor capability [71]. After an hour, this malicious MediaGet binary would attempt to connect to its C2 server, where it would receive the command to download the DoFoil[2].

We had to make some accommodations to run this binary. We found both the update and a benign version of the installer for MediaGet. However, the malicious update would not install or run on a system without MediaGet preinstalled. We attempted to compare the behaviour of the software installation to the update, but this was not comparable due to the amount and difference in behaviour between a complete installation and a single file drop. We were unable to uncover a legitimate update for the software, and installing an old version and installing a modern update would have a difference of more than 5 years. Therefore, instead of comparing the full update's behaviour, we compared the behaviour from running the exe files on a system with MediaGet preinstalled.

[2]DoFoil is a malware dropper, also called Smoke Loader. `https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fdofoil`

File and Sections:
The trojanized version of MediaGet.exe has a lower entropy than the benign one, both globally and across all sections. However, the file and most sections are bigger in the malicious sample. The most notable difference between the pair is that the trojanized version is unsigned, while the legitimate one is signed. Though the malicious update was not compared to the legitimate installer, we noted that it was signed by a different company than the legitimate files from MediaGet. This can be seen in Figure 4.3, where the two signatures are pictured beside each other. The signature on the left matches the signature on the benign version of the MediaGet executable file, while the signature on the right belongs to an unrelated third-party company. The table for these observed differences can be found in the appendix under Table A.10.


MediaGet Installer (Benign)

MD5	c5a49cf702897679260fe7afa8843c0e
SHA1	a102db570cf7d133af4305b79184095923264668
Serial Number	2d4d583428a8967414d63916670af3d8
Common Name	GLOBAL MICROTRADING PTE. LTD.
Country	SG
Locality	Singapore

MediaGet Update (Malicious)

MD5	5cff4e2a1c7e3e6a30092c6ccd6cd05b
SHA1	5022efca9e0a9022ab0ca6031a78f66528848568
Serial Number	633d51a3379616cd180bf5b713ce1b58
Common Name	DEVELTEC SERVICES SA DE CV
Country	MX
Locality	San Nicolas de los Garza (MONTERREY)

Figure 4.3: The left side of the image contains the signature for the legitimate installer, while the right contains the signature for the malicious.

Resources and Imports:
Compared to several other sample pairs, the resources and manifest of the sample pair are identical. The library imports from the malicious sample are slightly lower, though these two samples load far more functions than all of the other samples, with them having over 3500 imported functions each. This list can be found under Table A.11 in the appendix.

String Differences:
The trojanized version of MediaGet has four hardcoded C2 domains that can be found in the strings of the program. Two of the domains are '.bit' domains. The '.bit' domain is a decentralized TLD that runs on an older fork of bitcoin, named Namecoin, it cannot be connected unless the client is running a Namecoin client or it queries a DNS server that supports the .bit domain like a Namecoin DNS. It then becomes apparent why the sample has 71 hardcoded Namecoin DNS servers. The two other domains are common TLDs (.online and .com). It is worth mentioning that one of the .bit domains and the .com domain both try to impersonate the legitimate MediaGet site.

Behavioural Differences:
As seen in Table 4.9, the sample had a sparse amount of additional dynamic

Type	**New Strings found**	**Method**
Possible Code Access	`http://goshan[.]bit/start.php` `http://goshan[.]online/start.php` `http://medla-get[.]com/start.php` `http://media-get[.]bit/start.php`	FLOSS & Ranked_Strings
IP Addresses	71 New addresses, resolves to NameCoin DNS servers [71]	FLOSS & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 536 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

Table 4.8: DoFoil-MediaGet: Static Differential Analysis Strings

Signature	**Instruction**	**Comment**
This executable has a PDB path	**Malicious**: `X:\MediaGet\src\Desktop.3745\build-ide\release\mediaget.pdb` **Legitimate**: `E:\mediaget-adframes-release\release\mediaget.pdb` There is an observed change in PDB path.	
Searches running processes potentiall to identify processes for sandbox evasion, code injection or memory dumping	Process32NextW	Searches 39 times compared to 8
Encryption Keys have been identified in this analysis	1 detection in the malicious sample, none in the benign.	There was little information on this within the report and signature

Table 4.9: DoFoil-MediaGet: Cuckoo Dynamic Analysis

indications of maliciousness. While both files triggered a few signatures, the difference was minute. It is worth mentioning that Cuckoo detected a difference in the PDB paths between the two samples, and it detected encryption keys in the malicious sample. However, the report on what or where these encryption keys were detected was lacklustre and did not provide much insight.

4.1.5 Floxif

CCleaner Using basic static and dynamic sandboxing, We were not able to uncover any significant differences or findings from the trojanized CCleaner installer. The only differences found were:

- a slight increase in file size and overlay,
- the virtual size of '.ndata', and
- a slight increase in entropy for the '.rsrc' section.

This can be seen in table Table 4.10. In order to verify that it was not simply the local installation or configuration of Cuckoo Sandbox, this sample was also run in the Triage sandbox and an online Cuckoo Sandbox hosted by an Estonian

Cert. However, none of these produced any different or new behavioural data. The reason for this may simply be that the legitimate benign version of the CCleaner installer triggered far more detections and signatures across sandboxes than any other sample (malicious or benign). In total, both the legitimate and malicious triggered 39 signatures in Cuckoo, which ranged from several signatures intended to detect evasion techniques, anti-sandboxing, anti-analysis, enumeration, and more.

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File-size	9747512	9791816	**+44304**	PeStudio
.ndata Virtual-Size	3477504	3510272	+32768	PeStudio
.rsrc Entropy	4,96	4,961	PeStudio**+0,001**	
Overlay Size	9655648	9699952	PeStudio**+44304**	

Table 4.10: Floxif-CCleaner: Static Differential Analysis General File & Sections

4.1.6 Darkside

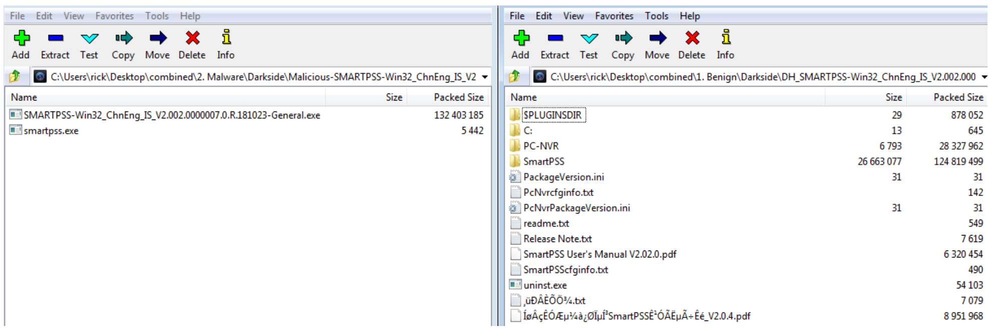
In 2021 (Likely, May or earlier), the Darkside affiliate UNC2465 was able to trojanize Dahua's video surveillance software named SmartPSS through their website [72]. The trojanized software was removed in June when Mandiant notified the company after one of Mandiants client's users downloaded the now malicious installer [72]. It was discovered as the installer executed several scripts and had a chain of downloads leading to SMOKEDHAM and NGROK being dropped and installed on the computer[72].

SmartPSS

File and Sections:
Similar to the trojanized binaries in the Dragonfly campaign, the malicious SmartPSS installer is "double-packed", with an additional Nullsoft installer wrapping a Nullsoft setup file and containing an additional file. This can be seen in the Figure 4.4, where instead of a dll file (as with the Dragonfly Campaign), it contains a file named smartpss.exe.

However, the version info from this file is rather different than the name suggests. The version info describes that this is MSHTA.exe, which is a legitimate executable file from Microsoft. The version info can be seen in Figure 4.5. At the time of the attack, the hash value of this executable was unknown, but it has since been tagged as related to the Darkside campaign. A complete summary of the file and section differences can be found in the appendix Table A.12.

Resources and Imports:
There is a considerable drop in the number of resources in the malicious binary


File Edit View Favorites Tools Help
Add Extract Test Copy Move Delete Info

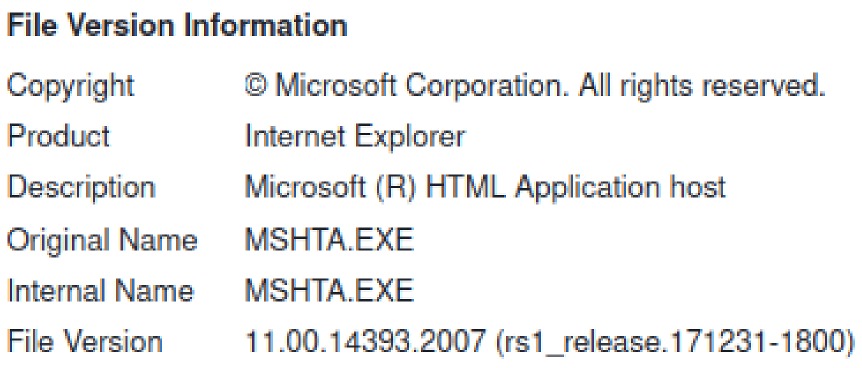
C:\Users\rick\Desktop\combined\2. Malware\Darkside\Malicious-SMARTPSS-Win32_ChnEng_IS_V2

Name	Size	Packed Size
SMARTPSS-Win32_ChnEng_IS_V2.002.0000007.0.R.181023-General.exe		132 403 185
smartpss.exe		5 442

C:\Users\rick\Desktop\combined\1. Benign\Darkside\DH_SMARTPSS-Win32_ChnEng_IS_V2.002.000

Name	Size	Packed Size
$PLUGINSDIR	29	878 052
C:	13	645
PC-NVR	6 793	28 327 962
SmartPSS	26 663 077	124 819 499
PackageVersion.ini	31	31
PcNvrcfginfo.txt		142
PcNvrPackageVersion.ini	31	31
readme.txt		549
Release Note.txt		7 619
SmartPSS User's Manual V2.02.0.pdf		6 320 454
SmartPSScfginfo.txt		490
uninst.exe		54 103
¸üÐÂÈÕÖ¾.txt		7 079
ÍøÂçÊÓÆµ¼à¿ØÏµÍ³SmartPSSÊ¹ÓÃËµÃ÷Êé_V2.0.4.pdf		8 951 968

Figure 4.4: The left side is the malicious sample, while the right is the benign.



File Version Information

Copyright	© Microsoft Corporation. All rights reserved.
Product	Internet Explorer
Description	Microsoft (R) HTML Application host
Original Name	MSHTA.EXE
Internal Name	MSHTA.EXE
File Version	11.00.14393.2007 (rs1_release.171231-1800)

Figure 4.5: The renamed binary's version info, screenshot taken from VirusTotal page

compared to the legitimate one. Furthermore, the malicious sample utilises one less library, with five more total function imports. The dropped library is the 'VERSION.dll'. This can be seen in Table 4.11

Type	Benign	Malicious
Resources Amount	84	21
Libraries	8	7
Library Imports	155	160
KERNEL32.dll	59	62
ADVAPI32.dll:	9	13
ole32.dll:	4	5
Version.dll	3	0

Table 4.11: Darkside-SmartPss: Static Differential Analysis Resources & Library

Behavioural Differences:
The malicious and benign SmartPSS installers produced almost identical results in Cuckoo Sandbox regarding signature detections and behavioural data. However, where they differed was in that the trojanized installer had a suspicious URL within the memory dump ('http://sdoc[.]xyz/ID-508260156241'), with this URL being a '.xyz' domain not present in the legitimate binary. However, Cuckoo Sandbox did not record any attempt by the installer to connect outwards towards this domain. Running the sample in Triage Sandbox revealed that the malware would leverage the renamed MSHTA.EXE ('smartpss.exe') to run the URL. These two entries for this can be seen in Table 4.12.

Signature	Instruction	Comment
Suspicious Top-Level Domains	http://sdoc[.]xyz/ID-508260156241	This was found in process memory of the malicious process
Triage: Executes Dropped EXE	Runs `C:\PROGRAMDATA\SMARTPSS-Win32_ChnEng_IS\smartpss.exehttp://sdoc[.]xyz/ID-508260156241` and`C:\SMARTPSS-Win32_ChnEng_IS_V2.002.0000007.0.R.181023-General.exe`	This is as explained by [72], and the file itself can be unzipped. This reveals two files inside, the legitimate installer and the malware dropper.

Table 4.12: Darkside-SmartPss: Triage & Cuckoo Dynamic Analysis

The trojanized software and the legitimate one did not produce human-readable strings other than the link for the Nullsoft installer. The reason for this was likely the packer. Furthermore, upon opening the sample with 7zip, inspecting it dynamically, and reading the online analysis conducted by Mandiant [72], it was decided that the other artefacts within (the installer for the legitimate SmartPSS) were not interesting for the thesis. This was, as reported by Mandiant [72] and observed in the Sandbox runs, that the malicious functionality of the sample was limited to a downloader.

4.1.7 3CX Supply Chain Attack

In March 2023, it was publicized that 3CX, the business communications supplier, had been the target of an attack and that its Voice over IP (VOIP) software had been trojanized by a suspected North Korean TA (UNC4376) [73].

This case is special because 3CX was compromised through another supply chain attack, where a trading software named 'X_trader' had been trojanized [74]. An employee (at 3CX) had downloaded X_trader onto his personal computer, and from there, the TA had pivoted onto the corporate networks using the VPN and the employee's stolen credentials [74]. Eventually, the TA compromised the build environments for the 3CX Desktop application and, from there, trojanized the application [74].

The trojanized 3CX software contained malicious code that, when started, would run the malware downloader SUDDENICON [73]. This would receive C2 traffic from encrypted files on a GitHub page. The last stage of the attack would be the download of ICONICSTEALER, an information stealer [73].

File and Sections:
The analysis of the 3CXDesktop app focused on the two trojanized files, '*ffmpeg.dll*' and '*d3dcompiler_47.dll*', as these were the known malicious files from the attack. The first file, 'ffmpeg.dll', had an increased global entropy, filesize, with changes in both size and entropy in most sections. Furthermore, the trojanized sample had one less section, missing the '*.voltbl*' from the benign file. Meanwhile, the second file '*d3dcompiler_47.dll*' had only a few noteworthy changes. The largest of these was the global filesize increase of 277KB, where there was no increase in section sizes but rather a reduction in their file-size ratio. However, this increase in filesize can be explained by the certificate being 554KB larger in the malicious sample compared to the benign. This could indicate that the certificate has added code between the versions, which according to the writeup by Blackberry [75], there was an encrypted shellcode in here [75].

The table for the differences in the d3dcompiler_47.dll file sizes and sections can be found in Table 4.13, while the complete list of differences in 'ffmpeg.dll' can be found in the appendix under Table A.13

String Differences:

There was no noticeable difference between the two samples' imports, resources, or manifest. However, there was a slight difference in the 'ffmpeg.dll'. Compared to the benign version, the malicious version contains a reference to two

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	6,392	6,535	0,143	Pefile Script
File-size	4891080	5168344	+277264	Pefile Script
.text File-Ratio	75,79%	71,72%	-4,07%	Pefile Script
.rdata File-Ratio	19,09%	18,07%	-1,02%	Pefile Script
.data File-Ratio	1,34%	1,27%	-0,07%	Pefile Script
.pdata File-Ratio	2,68%	2,54%	-0,14%	Pefile Script
Certificate Size	8648	563176	+554528	Pefile Script
Flags	41	43	+2	Pefile Script

Table 4.13: VEILEDSIGNAL-3CX-d3dcompiler_47.dll: Static Differential Analysis General File & Sections

libraries, 'ekernel32.dll' and 'd3dcompiler_47.dll'. The first could be a typo, or the section was interpreted slightly wrong. However, the second reference is to the '.dll' library with the drastically increased certificate. From the same analysis by Blackberry [75], we now know this was due to 'ffmpeg.dll' locating and reading the other dll to decrypt the shellcode stored in the certificate. This leads to the other two deviations within the file: a new import of the function VirtualProtect and a repeating string with a unique pattern.

From an analysis of the event by Zanki and ReversingLabs [76], we can see that this string an RC4 key used to decrypt the shellcode, meanwhile the Blackberry analysis concluded that the VirtualProtect function was used to mark the shellcode as executable [75].

The table containing the differences for the strings can be found in Table 4.14

Type	**New Strings found**	**Method**
New executable Reference	ekernel32.dll & d3dcompiler_47.dll	FLOSS & Ranked_Strings
MITRE Process Injection	VirtualProtect	PeStudio
Possibly Encoded Strings	The string "`3jB(2bsG#\spacefactor\@m{}c7`" is repeated twice. Once on it's own, and once several times over in a long string.	FLOSS & Ranked_Strings
Top 10k Ranked Strings difference	The malicious sample had 1573 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

Table 4.14: VEILEDSIGNAL-3CX-ffmpeg.dll: Static Differential Analysis Strings

Behavioural Differences:
There was not much difference between the two installers being run in the Cuckoo Sandbox nor within the Triage Sandbox from a differential analysis perspective. The two samples triggered the same detections within both sandboxes over multiple configurations to the run.

4.2 Summarised Results and Research Questions

Static Analysis Summarised

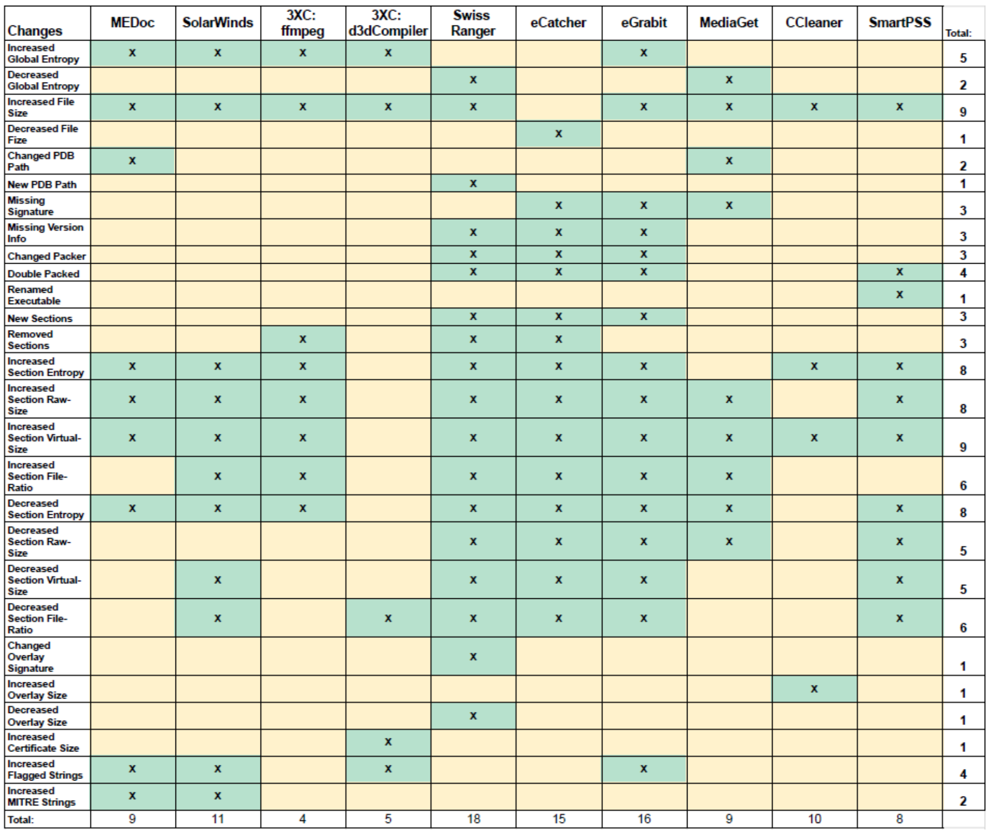
From the analysis, we can see that nine out of the ten files analysed had an increase in file size and had sections with higher virtual size than their benign counterparts. The file size increase is unsurprising as all of the samples had some variation of added code or bundled malware that would cause it to increase. In addition, an increase in the virtual size of one or more sections was observed across nine samples. This observation can be explained by the use of packers and functionality intended to load payloads or malware into memory.

Furthermore, when the malicious samples were compared to the benign versions, either an increase or decrease in section entropy was observed in eight samples. Furthermore, there was also a high amount of entropy increase and decrease across the samples. Lastly, we found that most of the samples had an increased raw size of one or more sections. On average, the samples had 10,5 (Median: 9,5) of the differences noted in the Figure 4.6.

There were a few findings that were only present in a small subset of the sample pairs, such as missing signatures between versions, the significant increase in certificate size in one sample, changes in packers, or changes in section names or numbers, that could provide an indicator that the sample should be inspected further for malicious behaviour. A summary of changes in sections, general file changes such as size or signature, and PeStudio flagged or tagged MITRE string amount can be found in Figure 4.6

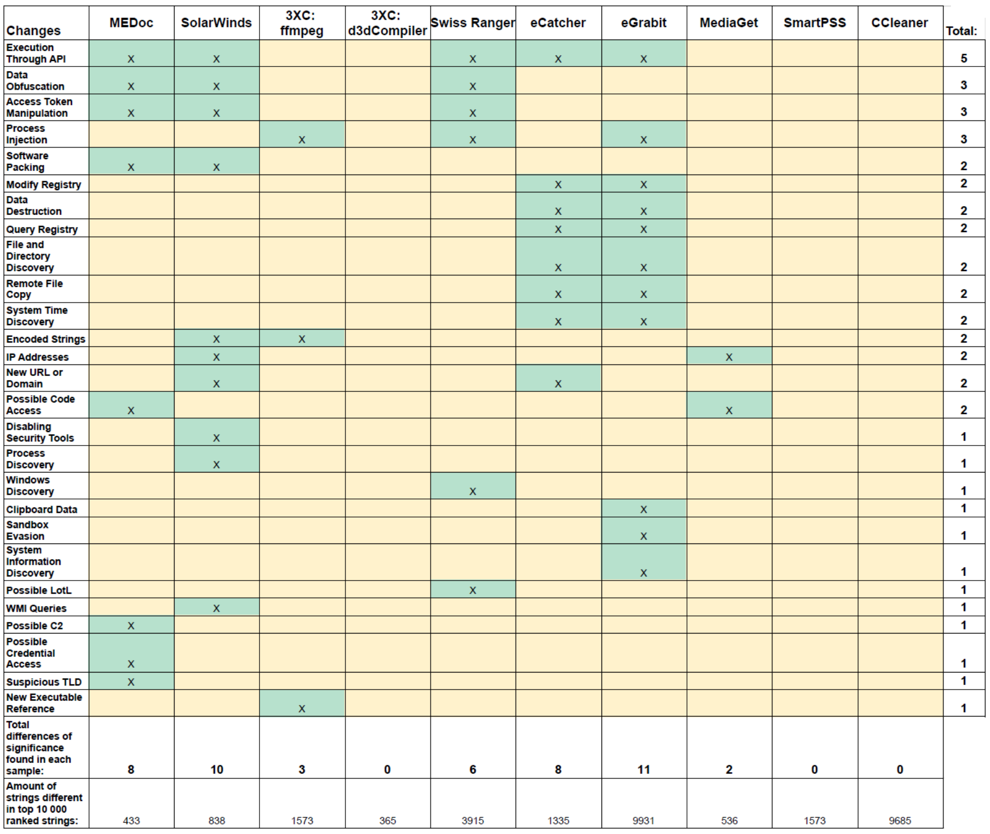
The second most frequent occurrence was the addition of strings that could be used for obfuscation and access token manipulation. However, these only occurred in three samples each. As with the SolarWinds supply chain attack, the sudden addition of these strings within a binary could be a reasonable indicator for further investigation. This could be seen in the Solorigate sample, where there were more than 100 compressed and base64 encoded strings that contained suspicious strings related to both system reconnaissance C2 domains. Figure 4.7 contains the summarised occurrence of suspicious or malicious indicators within the string section of the samples. At the bottom are two numerical values, the first of which is the total number of indicators in each file, while the second is the number of different strings in the top 10 000 strings ranked by Stringsifter. Furthermore, the value on the right is the total number of times this indicator appeared across all samples. It is worth mentioning that the 'X' marks the presence of one or more strings marked to the MITRE technique on the left and further down (starting at Possible LotL) indicators not assigned MITRE techniques.

Imports of libraries and functions naturally tie into this as expanded imports could hint at changes in functionality and purpose of the software just as strings or section differences can. The library and function imports from most samples could be inspected using Pestudio or the Pefile library. However, some samples, such as the dlls from the NotPetya and Solorigate campaigns, did not readily reveal what functions and libraries it imported when inspecting them using the tools



Changes	MEDoc	SolarWinds	3XC: ffmpeg	3XC: d3dCompiler	Swiss Ranger	eCatcher	eGrabit	MediaGet	CCleaner	SmartPSS	Total:
Increased Global Entropy	x	x	x	x			x				5
Decreased Global Entropy					x			x			2
Increased File Size	x	x	x	x	x		x	x	x	x	9
Decreased File Fize						x					1
Changed PDB Path	x							x			2
New PDB Path					x						1
Missing Signature						x	x	x			3
Missing Version Info					x	x	x				3
Changed Packer					x	x	x				3
Double Packed					x	x	x			x	4
Renamed Executable										x	1
New Sections					x	x	x				3
Removed Sections			x		x	x					3
Increased Section Entropy	x	x	x		x	x	x		x	x	8
Increased Section Raw-Size	x	x	x		x	x	x	x		x	8
Increased Section Virtual-Size	x	x	x		x	x	x	x	x	x	9
Increased Section File-Ratio		x	x		x	x	x	x			6
Decreased Section Entropy	x	x	x		x	x	x	x		x	8
Decreased Section Raw-Size					x	x	x	x		x	5
Decreased Section Virtual-Size		x			x	x	x			x	5
Decreased Section File-Ratio		x		x	x	x	x			x	6
Changed Overlay Signature					x						1
Increased Overlay Size									x		1
Decreased Overlay Size					x						1
Increased Certificate Size				x							1
Increased Flagged Strings	x	x		x			x				4
Increased MITRE Strings	x	x									2
Total:	9	11	4	5	18	15	16	9	10	8	

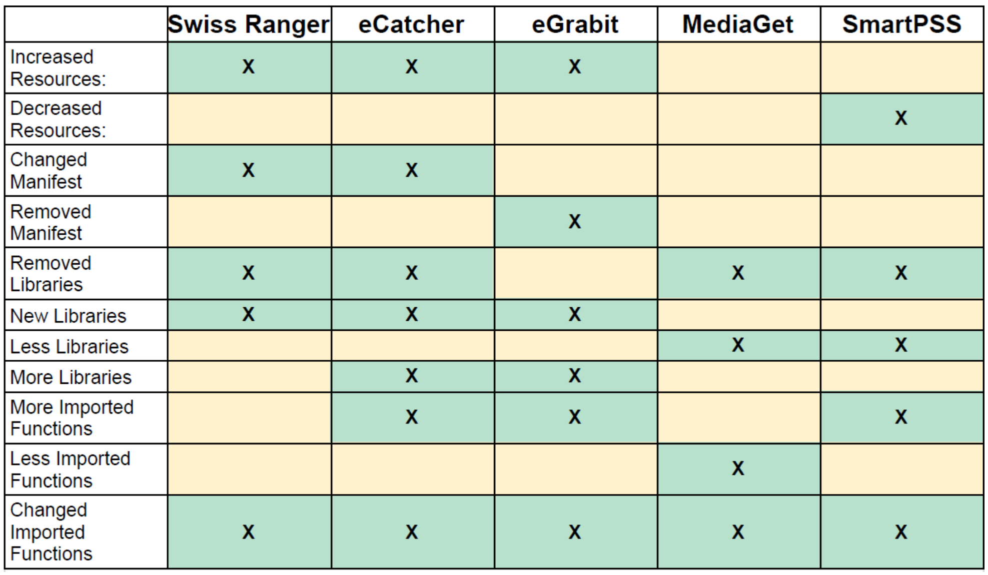
Figure 4.6: This table summarises the differences observed in the File and Section tables. The X represents that the sample file had one or more observed changes. On the bottom is a number for the total change, while on the side is the number for how many times this difference was found.



Changes	MEDoc	SolarWinds	3XC: ffmpeg	3XC: d3dCompiler	Swiss Ranger	eCatcher	eGrabit	MediaGet	SmartPSS	CCleaner	Total:
Execution Through API	X	X			X	X	X				5
Data Obfuscation	X	X			X						3
Access Token Manipulation	X	X			X						3
Process Injection			X		X		X				3
Software Packing	X	X									2
Modify Registry						X	X				2
Data Destruction						X	X				2
Query Registry						X	X				2
File and Directory Discovery						X	X				2
Remote File Copy						X	X				2
System Time Discovery						X	X				2
Encoded Strings		X	X								2
IP Addresses		X						X			2
New URL or Domain		X				X					2
Possible Code Access	X							X			2
Disabling Security Tools		X									1
Process Discovery		X									1
Windows Discovery					X						1
Clipboard Data							X				1
Sandbox Evasion							X				1
System Information Discovery							X				1
Possible LotL					X						1
WMI Queries		X									1
Possible C2	X										1
Possible Credential Access	X										1
Suspicious TLD	X										1
New Executable Reference			X								1
Total differences of significance found in each sample:	8	10	3	0	6	8	11	2	0	0	
Amount of strings different in top 10 000 ranked strings:	433	838	1573	365	3915	1335	9931	536	1573	9685	

Figure 4.7: This table shows the a summary of the differences observed in the Strings of the PE file. The X represents that the sample file had one or more of the observed difference from the benign file. On the bottom is a number for the total change, while on the side if the number for how many times this difference was found.

and methods from this thesis. Out of the 10 samples, only five of these showed differences. Drastic reductions in imports could be an indication that the software was packaged. However, only the malicious eCatcher and eGrabit samples had significant increases in the number of functions when compared to the benign ones. An overview of these changes can be seen in Figure 4.8, where an X indicates that the change to the left has occurred.



	Swiss Ranger	eCatcher	eGrabit	MediaGet	SmartPSS
Increased Resources:	X	X	X		
Decreased Resources:					X
Changed Manifest	X	X			
Removed Manifest			X		
Removed Libraries	X	X		X	X
New Libraries	X	X	X		
Less Libraries				X	X
More Libraries		X	X		
More Imported Functions		X	X		X
Less Imported Functions				X	
Changed Imported Functions	X	X	X	X	X

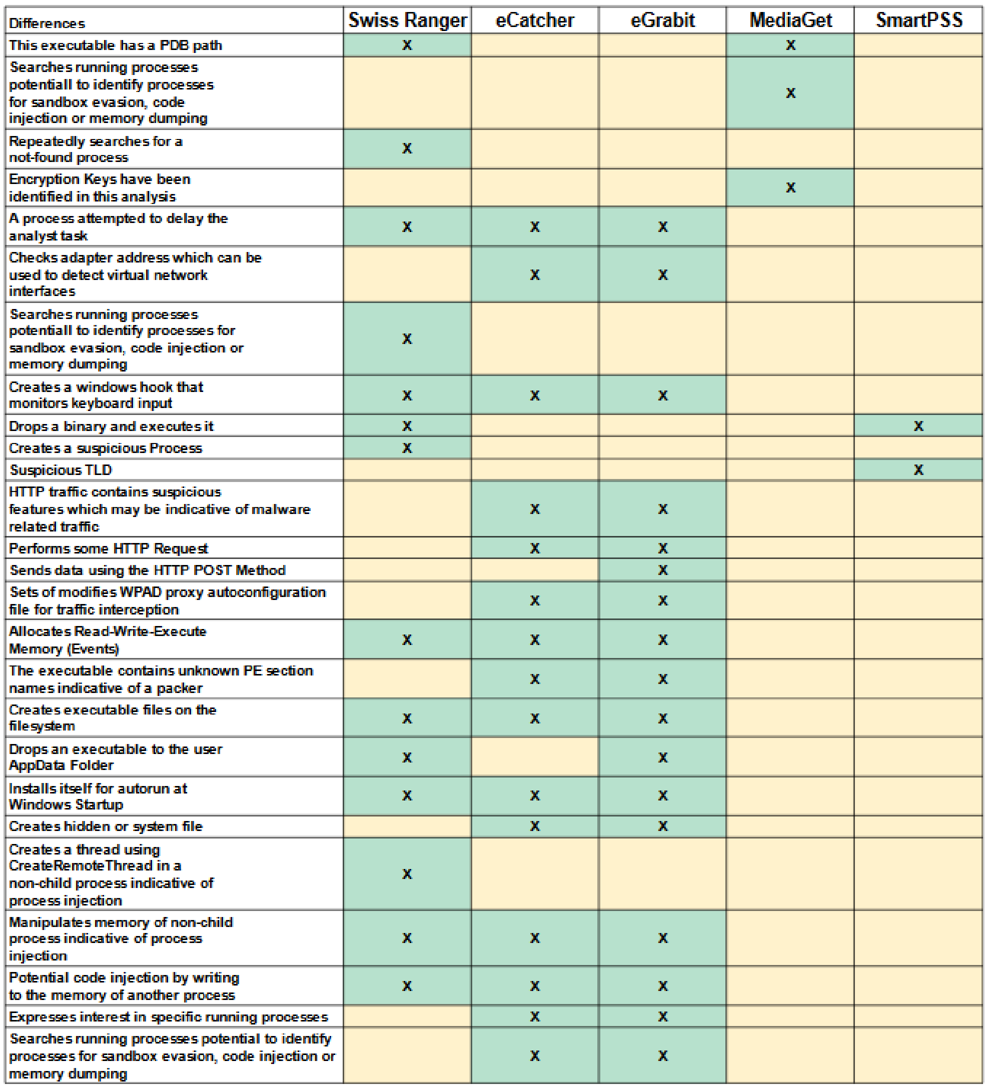
Figure 4.8: The left side is the malicious sample, while the right is the benign.

Dynamic Analysis Summarised

Sandboxing technology combined with differential analysis can provide some insight into the behaviour of a trojanized file and could reveal the presence of an otherwise undetected malicious sample. However, Sandboxing provided the least indicators of potential maliciousness during the analyses of the malicious and benign samples. An example is the seven sample pairs we were able to run; Only five trojanized versions showed a difference in the behaviour compared to the benign sample.

Of these five, the trojanized MediaGet and SmartPSS had a low variation to the legitimate binary, with the first only having encryption keys and an increased amount of Process32NextW API calls compared to the benign. The malicious Smart-PSS, on the other hand, only differed from the benign sample with a suspicious TLD in memory and its access through the execution of the renamed MSHTA.

Comparatively, the three Dragonfly RATs triggered several more signatures in all categories and had a very different behaviour than their legitimate counterpart. In addition, these samples triggered several detections, especially those related to process injection, persistence, and network. However, this could be due to the age of these three samples, as they are several years older than the other samples. So they could use deprecated and well-known methods that are no longer considered advanced. An overview of the different signatures triggered by the samples can be seen in Figure 4.9



Differences	Swiss Ranger	eCatcher	eGrabit	MediaGet	SmartPSS
This executable has a PDB path	x			x	
Searches running processes potentiall to identify processes for sandbox evasion, code injection or memory dumping				x	
Repeatedly searches for a not-found process	x				
Encryption Keys have been identified in this analysis				x	
A process attempted to delay the analyst task	x	x	x		
Checks adapter address which can be used to detect virtual network interfaces		x	x		
Searches running processes potentiall to identify processes for sandbox evasion, code injection or memory dumping	x				
Creates a windows hook that monitors keyboard input	x	x	x		
Drops a binary and executes it	x				x
Creates a suspicious Process	x				
Suspicious TLD					x
HTTP traffic contains suspicious features which may be indicative of malware related traffic		x	x		
Performs some HTTP Request		x	x		
Sends data using the HTTP POST Method			x		
Sets of modifies WPAD proxy autoconfiguration file for traffic interception		x	x		
Allocates Read-Write-Execute Memory (Events)	x	x	x		
The executable contains unknown PE section names indicative of a packer		x	x		
Creates executable files on the filesystem	x	x	x		
Drops an executable to the user AppData Folder	x		x		
Installs itself for autorun at Windows Startup	x	x	x		
Creates hidden or system file		x	x		
Creates a thread using CreateRemoteThread in a non-child process indicative of process injection	x				
Manipulates memory of non-child process indicative of process injection	x	x	x		
Potential code injection by writing to the memory of another process	x	x	x		
Expresses interest in specific running processes		x	x		
Searches running processes potential to identify processes for sandbox evasion, code injection or memory dumping		x	x		

Figure 4.9: The table above summarised the signatures detections only present in the malicious samples.

It is also possible that the cause of this is their purpose compared to the other trojanized software. The other samples analysed were primarily stage 1 downloaders (MediaGet, SmartPSS, and 3CX) or a highly targeted backdoor (CCleaner). It is also possible that the old samples could not detect the sandbox process, while the benign ones could.

4.2.1 Research Questions

RQ2. What indications of compromise can one detect in closed-source software supply chain attacks by comparing a previous benign file to a trojanized version?

There are several indicators which one can use to detect a potential closed-source software supply chain attack. Some highlighted static examples from the analysis are the presence of obfuscated strings, changes in the file sections, or new and unexplained imports or functionality of libraries related to execution through API. Meanwhile, for dynamic differential analysis, network traffic towards unrelated third-party domains, new system calls to sleep, or potential code injection behaviour could be indications.

RQ3. To what extent can basic static and dynamic-sandboxing differential analysis techniques be used in detecting malicious behaviour and static changes in trojanized software compared to the legitimate version?

Basic static analysis performed far better than initially expected in uncovering potential malicious behaviour. In the 10 sample pairs analysed, we observed suspicious differences indicative of a compromise in all of them except for CCleaner. In addition, several malicious artefacts uncovered using basic static techniques could be observed in the sandbox runs. Furthermore, for other samples that either could not run or did not behave differently in the sandbox run, such as M.E.Doc or 3CX, indicators were readily available that there had been changes. For example, in M.E.Doc's 'ZvitPublishedObjects.dll', strings related to encryption and SMTP communication were added. At the same time, for 3CX, there was a significant increase in the certificate size for 'd3dcompiler_47.dll' and the many changes to the different sections of "ffmpeg.dll'.

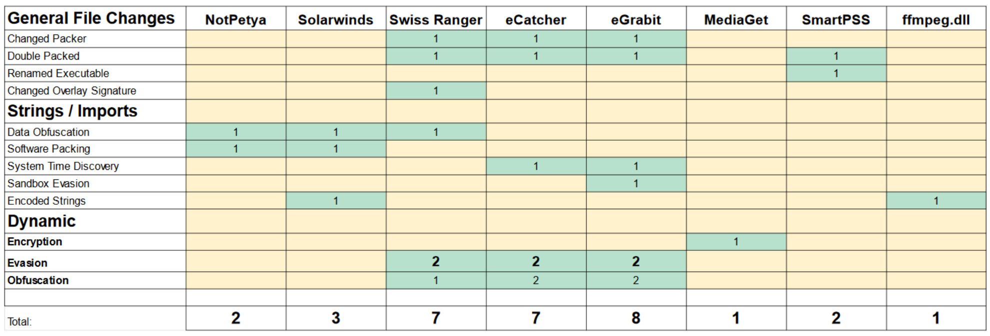
The Cuckoo Sandbox and Hatching's Triage revealed malicious behaviour different from the software's usual behaviour in three samples while providing some suspicious indicators in two more. However, they could not detect malicious behaviour differences in the CCleaner and 3CX samples. This leads to the conclusion that sandboxing can be useful at identifying some malicious behaviour and should be part of a larger solution. However, on its own, it can be avoided or not provide sufficient insight. Therefore, it can be used as part of a more comprehensive solution, especially if the malware samples are part of an active attack, as then the C2 domains would be online rather than defunct, which could lead to several differences should C2 traffic occur or second stage malware be dropped onto the system.

RQ4. To what extent is looking for obfuscation and evasion techniques reliable in detecting trojanized closed-source software?

In the 10 samples analysed, there were indications of obfuscation or evasion techniques within seven samples. This ranged changes in packers, renaming of executables, presence of encoded strings, imported functions intended for encryption, or dynamic behaviour such as calling a sleep system call function. However, it should be noted that several benign samples triggered signatures in Cuckoo and online sandboxes indicative of sandbox evasion, system fingerprinting, system sleep, and more. An example of this is CCleaner which enumerates the system for everything from installed AV to virtualisation software, and several of the samples checked for whether they were being debugged, checking disk and memory, WMI queries and more.

One consideration is if the build process is compromised, as in Solarwinds and CCleaner, could the attackers use these pre-existing checks rather than adding new functionality?

Table 4.10 illustrates which samples had indicators of obfuscation or evasion techniques identified during the analysis.



General File Changes	NotPetya	Solarwinds	Swiss Ranger	eCatcher	eGrabit	MediaGet	SmartPSS	ffmpeg.dll
Changed Packer			1	1	1			
Double Packed			1	1	1		1	
Renamed Executable							1	
Changed Overlay Signature			1					
Strings / Imports								
Data Obfuscation	1	1	1					
Software Packing	1	1						
System Time Discovery				1	1			
Sandbox Evasion					1			
Encoded Strings		1						1
Dynamic								
Encryption						1		
Evasion			**2**	**2**	**2**			
Obfuscation			1	2	2			
Total:	**2**	**3**	**7**	**7**	**8**	**1**	**2**	**1**

Figure 4.10: This summarises which samples had obfuscation or evasion IoCs.

Chapter 5

Discussion, Conclusion, and Future Work

This chapter discusses the methods, results, and limitations of the thesis. Furthermore, it includes the conclusion of the thesis and the future work section.

5.1 Discussion

In Section 3.1, the challenges of finding legitimate and malicious samples were described. To summarise, there has only been a few closed-source software supply chain attacks throughout the years, so there is a small sample size to collect.

Furthermore, finding the legitimate benign version within a similar timeframe was also challenging as they were no longer available for download and hashes for these files are not discussed and analysed the same way the attacks are. This mirrors the experience of the authors of Barr-Smith *et al.* [11], in which this lack of data and the need for a known benign binary added difficulty to the differential analysis task. However, the authors in Barr-Smith *et al.* [11] also theorised that this could explain the scarcity of previously published research. This is a reasonable and likely conclusion after conducting this master's thesis in the same domain.

This issue led to some of the samples being newer rather than older such as MediaGet and the Solarwinds Orion dll. Furthermore, the initial goal was to look at behaviour changes over successive benign samples compared to the Trojanized samples. However, this idea was dropped due to the difficulty of finding the samples, the time consumption of finding them, and the time constraints of the thesis.

A point that needs to be discussed is whether these samples and, by extension, this thesis applies to future supply chain attacks. Unfortunately, this is a difficult question to answer, as these types of attacks are very few compared to conventional malware attacks, so there is no large sample base for one to draw conclusions from or see trends within. A consequence of this is that solutions developed to detect closed-source software supply chain attacks lack the data to validate their solution, as was the case in Wang [8]. It is also worth mentioning that the actors behind most of these supply chain attacks were APT groups with high capability and expertise.

There were other limitations within this paper, the first being the usage of Cuckoo Sandbox as the sandboxing platform. As explained in Section 2.2.1, this sandbox was chosen early during the project, and it was only discovered underway that this project was no longer developed. The reasons for remaining with Cuckoo were also described within Section 2.2.1, so we will not touch upon that in this section. However, it is worth mentioning some of the drawbacks of this approach.

The first of these is that there is a possibility that some of the trojanized samples could detect the environment, which caused there to be no difference to the benign sample or simply that Cuckoo could not detect the malware. This was compensated by running the samples on the online sandbox platform Triage. However, by doing so, one loses the customizability and control over the parameters for the run. An improvement would have been to conduct more in-depth research on the available options for open-source sandboxes in the pre-planning phase of the thesis. However, the Cuckoo Sandbox was period relevant for most of the attacks except for 3CX, and there was only a single sample where Triage managed to detect additional functionality compared to Cuckoo.

Another intentional limitation of the thesis was keeping to basic static analysis techniques and dynamic analysis through sandboxing. There is a limit to the number of indicators one can find through observing the behaviour of the file in a sandbox and keeping to analysing the file's metadata. However, this was an intentional limitation, as the problem demands solutions that can be scaled upwards. This thesis serves as a first step into first seeing if it is possible and what the extent is.

The findings within this thesis have shown that even attacks that were highly sophisticated at the time and led to extensive financial damages could have been uncovered through a differential analysis process using basic static analysis techniques, sandboxing and no budget. Furthermore, the methods used within this thesis do not require a reverse engineer to dig through every line of code in a disassembler or debugger in every single third-party closed-source software. Instead, the indicators of compromise found detailed in this thesis could also have been found by using libraries and software to automate and alert on anomalies or indicators.

At the time of writing, only two other academic papers were found after an extensive literature search. These two papers by Wang [8] and Barr-Smith *et al.* [11] likewise noted the lack of research in detecting closed-source software supply chain attacks.

5.1.1 Conclusion and Future Work

Closed-Source Software supply chain attacks are complicated to defend against, and the potential for damage is massive. This thesis aimed to take early steps into a domain of research where there is little published to see if there are simple solutions or methods to address a complex problem. Furthermore, the findings within this thesis serve as an early contribution to this domain, to possible solutions, and to understand the problem.

The findings and results of the analysis indicate that it is possible to uncover suspicious changes through a static differential analysis of the PE file and the embedded strings of trojanized software. It was found that most of the samples had several changes from the benign version to the trojanized version, and trails of obfuscation, C2 artefacts, and expanded potential malicious capabilities that could be found through analysis of the embedded strings. It was also found that most of the trojanized samples had several changes in the sections ranging from increased entropy, added sections, or changes in file ratio. Meanwhile, dynamic sandboxing of the samples provided fewer differences, with two samples having no differences during the sandbox analysis.

5.1.2 Future Work

There is a need to conduct more research into this domain. For example, the indicators provided in this thesis could be as comparative data to similar analyses of future software supply chain attacks. It would also be relevant to observe the changes between several generations of software before the trojanization occurred to establish a baseline for the change between versions. Furthermore, working towards an open-source solution or proof of concept for detecting these malicious changes between software versions would be relevant and valuable for future research. Finally, it could also be an interesting approach to test and compare several sandbox solutions available and analyse differential changes through this method.

Bibliography

[1] NTB. 'Norske kraftselskaper berørt av solarwinds-hacking.' Visited: 31.05.2023. (), [Online]. Available: https://e24.no/teknologi/i/906P7l/norske-kraftselskaper-beroert-av-solarwinds-hacking.

[2] J. Lemon. 'Solarwinds hides list of its high-profile corporate clients after hack.' Visited: 31.05.2023. (), [Online]. Available: https://www.newsweek.com/solarwinds-hides-list-its-high-profile-corporate-clients-after-hack-1554943.

[3] FireEye. 'Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with sunburst backdoor.' Visited: 29.05.2023. (), [Online]. Available: https://www.mandiant.com/resources/blog/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.

[4] S. Ramakrishna. 'New findings from our investigation of sunburst.' Visited: 31.05.2023. (2021), [Online]. Available: https://orangematter.solarwinds.com/2021/01/11/new-findings-from-our-investigation-of-sunburst/.

[5] D. Temple-Raston. 'A 'worst nightmare' cyberattack: The untold story of the solarwinds hack.' Visited: 31.05.2023. (2021), [Online]. Available: https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack.

[6] S. Corporation. 'Solarwinds corporation (swi) - form 8-k | current report.' Visited: 31.05.2023. (2020), [Online]. Available: https://seekingalpha.com/filing/5276758.

[7] European-Union-Agency-for-Cybersecurity, 'Enisa threat landscape for supply chain attacks,' 2021. DOI: https://www.doi.org/10.2824/168593.

[8] X. Wang, 'On the feasibility of detecting software supply chain attacks,' in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 458–463. DOI: 10.1109/MILCOM52596.2021.9652901.

[9] G. Myre and S. Bond. 'Top cyber firm, fireeye, says it's been hacked by a foreign govt.' Visited: 31.05.2023. (2020), [Online]. Available: https://www.npr.org/2020/12/08/944416183/top-cyber-firm-fireeye-says-its-been-hacked-by-a-foreign-govt.

[10] and European Union Agency for Cybersecurity, *ENISA threat landscape 2022 : July 2021 to July 2022*, A. Malatras, M. Theocharidou, I. Lella, E. Tsekmezoglou, C. Ciobanu and R. Naydenov, Eds. European Network and Information Security Agency, 2022. DOI: `doi/10.2824/764318`.

[11] F. Barr-Smith, T. Blazytko, R. Baker and I. Martinovic, 'Exorcist: Automated differential analysis to detect compromises in closed-source software supply chains,' in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED'22, Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 51–61, ISBN: 9781450398855. DOI: `10.1145/3560835.3564550`. [Online]. Available: `https://doi.org/10.1145/3560835.3564550`.

[12] I. Red Hat. 'What is open source?' Visited: 15.05.2023. (), [Online]. Available: `https://www.redhat.com/en/topics/open-source/what-is-open-source`.

[13] I. T. College. 'Open source vs. closed source software.' Visited: 15.05.2023. (), [Online]. Available: `https://iticollege.edu/blog/open-source-vs-closed-source-software/`.

[14] K. I. Encyclopedia. 'Closed-source software (proprietary software).' Visited: 15.05.2023. (), [Online]. Available: `https://encyclopedia.kaspersky.com/glossary/closed-source/`.

[15] J. Boyens, A. Smith, N. Bartol, K. Winkler, A. Holbrook and M. Fallon, 'Cybersecurity supply chain risk management practices for systems and organizations,' National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP) SP 800-161, Rev 1, Includes updates as of May 5,2022, 2022. DOI: `10.6028/NIST.SP.800-161r1`.

[16] M. Sikorski and A. Honig, *Practical malware analysis : The hands-on guide to dissecting malicious software*, eng, San Francisco, 2012.

[17] N. Pachhala, S. Jothilakshmi and B. P. Battula, 'A comprehensive survey on identification of malware types and malware classification using machine learning techniques,' in *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, 2021, pp. 1207–1214. DOI: `10.1109/ICOSEC51865.2021.9591763`.

[18] Iacob and I. M. Ioan Ionita. 'The anatomy of wiper malware, part 1: Common techniques.' Visited: 16.05.2023. (), [Online]. Available: `https://www.crowdstrike.com/blog/the-anatomy-of-wiper-malware-part-1/`.

[19] D. Simpson, A. Lobo, A. Jupudi, D. Vangel and C. Davis. 'Coin miners.' Visited: 16.05.2023. (), [Online]. Available: `https://learn.microsoft.com/en-us/microsoft-365/security/intelligence/coinminer-malware?view=o365-worldwide`.

[20] K. A. Monnappa, *Learning Malware Analysis*, eng. Packt Publishing, 2018, ISBN: 1788392507.

[21] J. Wright. 'Month of powershell: Threat hunting with powershell differential analysis.' Visited: 16.05.2023. (), [Online]. Available: https://www.sans.org/blog/threat-hunting-with-powershell-differential-analysis/.

[22] Microsoft. 'Pe format.' Visited: 18.05.2023. (), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[23] I. VMware. 'What is a hypervisor?' Visited: 19.05.2023. (), [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor.html.

[24] Oracle. 'Snapshots.' Visited: 19.05.2023. (), [Online]. Available: https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/snapshots.html.

[25] C. Linn and S. Debray, 'Obfuscation of executable code to improve resistance to static disassembly,' in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03, Washington D.C., USA: Association for Computing Machinery, 2003, pp. 290–299, ISBN: 1581137389. DOI: 10.1145/948109.948149. [Online]. Available: https://doi.org/10.1145/948109.948149.

[26] O. Or-Meir, N. Nissim, Y. Elovici and L. Rokach, 'Dynamic malware analysis in the modern era—a state of the art survey,' *ACM Comput. Surv.*, vol. 52, no. 5, Sep. 2019, ISSN: 0360-0300. DOI: 10.1145/3329786. [Online]. Available: https://doi.org/10.1145/3329786.

[27] Checkpoint and R. Ladutska. 'About evasion techniques.' Visited: 19.05.2023. (), [Online]. Available: https://evasions.checkpoint.com/about/.

[28] Eurostat. 'Cloud computing - statistics on the use by enterprises.' Visited: 24.05.2023. (), [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises#Use_of_cloud_computing:_highlights.

[29] VMRay. 'Vm detection – passing the pafish test.' Visited: 24.05.2023. (), [Online]. Available: https://www.vmray.com/cyber-security-blog/a-pafish-primer/.

[30] Checkpoint. 'Evasions: Timing.' Visited: 19.05.2023. (), [Online]. Available: https://evasions.checkpoint.com/techniques/timing.html.

[31] C. Foundation. 'What is cuckoo?' Visited: 19.05.2023. (), [Online]. Available: https://cuckoo.readthedocs.io/en/latest/introduction/what/.

[32] R. v. Zutphen. 'Cuckoo sandbox architecture.' Visited: 19.05.2023. (), [Online]. Available: https://hatching.io/blog/cuckoo-sandbox-architecture/.

[33] C. Foundation. 'Processing modules.' Visited: 19.05.2023. (), [Online]. Available: https://cuckoo.readthedocs.io/en/latest/customization/processing/.

[34] C. Foundation. 'Processing modules.' Visited: 19.05.2023. (), [Online]. Available: `https://cuckoo.readthedocs.io/en/latest/installation/host/routing/`.

[35] V. M. Alvarez. 'Yara: The pattern matching swiss knife for malware researchers (and everyone else).' Visited: 19.05.2023. (), [Online]. Available: `https://virustotal.github.io/yara/`.

[36] V. M. Alvarez. 'Cuckoo module.' Visited: 19.05.2023. (), [Online]. Available: `https://yara.readthedocs.io/en/stable/modules/cuckoo.html`.

[37] Microsoft. 'Virus total (preview).' Visited: 27.05.2023. (), [Online]. Available: `https://learn.microsoft.com/en-us/connectors/virustotal/`.

[38] VirusTotal. 'Reports.' Visited: 27.05.2023. (), [Online]. Available: `https://support.virustotal.com/hc/en-us/articles/115002719069-Reports`.

[39] Any.Run. 'Any.run documentation.' Visited: 27.05.2023. (), [Online]. Available: `https://app.any.run/docs/`.

[40] Any.Run. 'Any.run plans.' Visited: 27.05.2023. (), [Online]. Available: `https://app.any.run/plans/`.

[41] Hatching. 'Hatching triage.' Visited: 27.05.2023. (), [Online]. Available: `https://hatching.io/triage/`.

[42] Hatching. 'Hatching about page.' Visited: 27.05.2023. (), [Online]. Available: `https://hatching.io/about/`.

[43] O. and/or its affiliates. 'Virtualbox user manual: Chapter 1. first steps.' Visited: 24.05.2023. (), [Online]. Available: `https://www.virtualbox.org/manual/ch01.html`.

[44] A. Ortega. 'Pafish.' Visited: 25.05.2023. (), [Online]. Available: `https://github.com/a0rtega/pafish`.

[45] M. Ochsenmeier. 'Pestudio: Malware initial assessment.' Visited: 25.05.2023. (), [Online]. Available: `https://www.winitor.com/`.

[46] N. Fox. 'Pestudio overview: Setup, tutorial and tips.' Visited: 25.05.2023. (), [Online]. Available: `https://www.varonis.com/blog/pestudio`.

[47] Mandiant. 'Github stringsifter.' Visited: 25.05.2023. (), [Online]. Available: `https://github.com/mandiant/stringsifter`.

[48] Mandiant. 'Github stringsifter.' Visited: 25.05.2023. (), [Online]. Available: `https://github.com/mandiant/flare-floss`.

[49] E. Carrera Ventura, *pefile*, version 2023.2.7, Feb. 2023. [Online]. Available: `https://github.com/erocarrera/pefile`.

[50] M. Ohm, H. Plate, A. Sykosch and M. Meier, 'Backstabber's knife collection: A review of open source software supply chain attacks,' in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Maurice, L. Bilge, G. Stringhini and N. Neves, Eds., Cham: Springer International Publishing, 2020, pp. 23–43, ISBN: 978-3-030-52683-2.

[51] M. Ohm, A. Sykosch and M. Meier, 'Towards detection of software supply chain attacks by forensic artifacts,' in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20, Virtual Event, Ireland: Association for Computing Machinery, 2020, ISBN: 9781450388337. DOI: 10.1145/3407023.3409183. [Online]. Available: https://doi.org/10.1145/3407023.3409183.

[52] OASIS. 'Stix™ version 2.0. part 3: Cyber observable core concepts.' Visited: 17.05.2023. (), [Online]. Available: https://docs.oasis-open.org/cti/stix/v2.0/cs01/part3-cyber-observable-core/stix-v2.0-cs01-part3-cyber-observable-core.html.

[53] M. Ohm, F. Boes, C. Bungartz and M. Meier, 'On the feasibility of supervised machine learning for the detection of malicious software packages,' in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ser. ARES '22, Vienna, Austria: Association for Computing Machinery, 2022, ISBN: 9781450396707. DOI: 10.1145/3538969.3544415. [Online]. Available: https://doi.org/10.1145/3538969.3544415.

[54] A. Sejfia and M. Schäfer, 'Practical automated detection of malicious npm packages,' in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1681–1692, ISBN: 9781450392211. DOI: 10.1145/3510003.3510104. [Online]. Available: https://doi.org/10.1145/3510003.3510104.

[55] D.-L. Vu, F. Massacci, I. Pashchenko, H. Plate and A. Sabetta, 'Lastpymile: Identifying the discrepancy between sources and packages,' in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021, Athens, Greece: Association for Computing Machinery, 2021, pp. 780–792, ISBN: 9781450385626. DOI: 10.1145/3468264.3468592. [Online]. Available: https://doi.org/10.1145/3468264.3468592.

[56] S. Scalco, R. Paramitha, D.-L. Vu and F. Massacci, 'On the feasibility of detecting injections in malicious npm packages,' in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ser. ARES '22, Vienna, Austria: Association for Computing Machinery, 2022, ISBN: 9781450396707. DOI: 10.1145/3538969.3543815. [Online]. Available: https://doi.org/10.1145/3538969.3543815.

[57] P. Ladisa, H. Plate, M. Martinez, O. Barais and S. E. Ponta, 'Towards the detection of malicious java packages,' in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED'22, Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 63–72, ISBN: 9781450398855. DOI: 10.1145/3560835.3564548. [Online]. Available: https://doi.org/10.1145/3560835.3564548.

[58] M. LaFerrera and R. Kovar, 'Detecting supply chain attacks: Using splunk and ja3/s hashes to detect malicious activity on critical servers,' Splunk, San Francisco, CA, Tech. Rep. 21-21294-Splunk-Detecting Supply Chain Attacks-101-WP, 2021. [Online]. Available: https://www.splunk.com/en_us/pdfs/resources/whitepaper/detecting-supply-chain-attacks.pdf.

[59] Z. Zhang, S. Natarajan and A. Banerjee, 'Detecting process hijacking and software supply chain attacks using intel threat detection technology,' Intel, Santa Clara, CA, Tech. Rep. 0222/DCC/MZ/PDF, 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/white-paper-inteltdt-abd.pdf.

[60] S. Cordey, 'Software supply chain attacks. an illustrated typological review,' en, Zurich, Report, 2023-01. DOI: 10.3929/ethz-b-000584947.

[61] B. Bingöl. 'Virtualbox detection, anti-detection.' Visited: 20.05.2023. (), [Online]. Available: https://berhanbingol.medium.com/virtualbox-detection-anti-detection-eng-54a4cde1b509.

[62] R. Taissun. 'Installing and running cuckoo malware analysis platform – part 2.' Visited: 20.05.2023. (), [Online]. Available: https://secvision22.wordpress.com/2017/01/19/installing-and-running-cuckoo-malware-analysis-platform-part-2/.

[63] Y. Oyama, 'How does malware use rdtsc? a study on operations executed by malware with cpu cycle measurement,' in *Detection of Intrusions and Malware, and Vulnerability Assessment*, R. Perdisci, C. Maurice, G. Giacinto and M. Almgren, Eds., Cham: Springer International Publishing, 2019, pp. 197–218, ISBN: 978-3-030-22038-9.

[64] MITRE. 'Mitre att&ck home page.' Visited: 01.06.2023. (), [Online]. Available: https://attack.mitre.org/.

[65] A. Greenberg. 'The untold story of notpetya, the most devastating cyberattack in history.' Visited: 31.05.2023. (2020), [Online]. Available: https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/.

[66] A. Cherepanov. 'Analysis of telebots' cunning backdoor.' Visited: 28.05.2023. (), [Online]. Available: https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/.

[67] D. Maynor, M. Olney and Y. Younan. 'The medoc connection.' Visited: 28.05.2023. (), [Online]. Available: https://blog.talosintelligence.com/the-medoc-connection/.

[68] Microsoft. 'Analyzing solorigate, the compromised dll file that started a sophisticated cyberattack, and how microsoft defender helps protect customers.' Visited: 29.05.2023. (), [Online]. Available: https://www.microsoft.com/en-us/security/blog/2020/12/18/analyzing-solorigate-the-

compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/.

[69] J. T. Langill, 'Defending against the dragonfly cyber security attacks,' Belden, San Francisco, CA, Tech. Rep. Version 2.0, 2014. [Online]. Available: https://www.belden.com/hubfs/resources/knowledge/white-papers/Belden-White-Paper-Dragonfly-Cyber-Security-Attacks-AB_Original_68751.pdf?hsLang=en.

[70] E. Hjelmvik. 'Full disclosure of havex trojans.' Visited: 29.05.2023. (), [Online]. Available: https://www.netresec.com/?page=Blog&month=2014-10&post=Full-Disclosure-of-Havex-Trojans.

[71] Microsoft. 'Poisoned peer-to-peer app kicked off dofoil coin miner outbreak by.' Visited: 28.05.2023. (), [Online]. Available: https://www.microsoft.com/en-us/security/blog/2018/03/13/poisoned-peer-to-peer-app-kicked-off-dofoil-coin-miner-outbreak/.

[72] T. McLellan, R. Dean, J. Moore, N. Harbour, M. Hunhoff, J. Wilson and J. Nuce. 'Smoking out a darkside affiliate's supply chain software compromise.' Visited: 28.05.2023. (), [Online]. Available: https://www.mandiant.com/resources/blog/darkside-affiliate-supply-chain-software-compromise.

[73] J. Johnson, F. Plan, A. Sanchez, R. Fontana, J. Nicastro, D. Andonov, M. Fodoreanu and S. Daniel. '3cx software supply chain compromise initiated by a prior software supply chain compromise; suspected north korean actor responsible.' Visited: 31.05.2023. (), [Online]. Available: https://www.mandiant.com/resources/blog/3cx-software-supply-chain-compromise.

[74] A. Prodromou and 3CX. 'Security update thursday 20 april 2023 – initial intrusion vector found.' Visited: 31.05.2023. (), [Online]. Available: https://www.3cx.com/blog/news/mandiant-security-update2/.

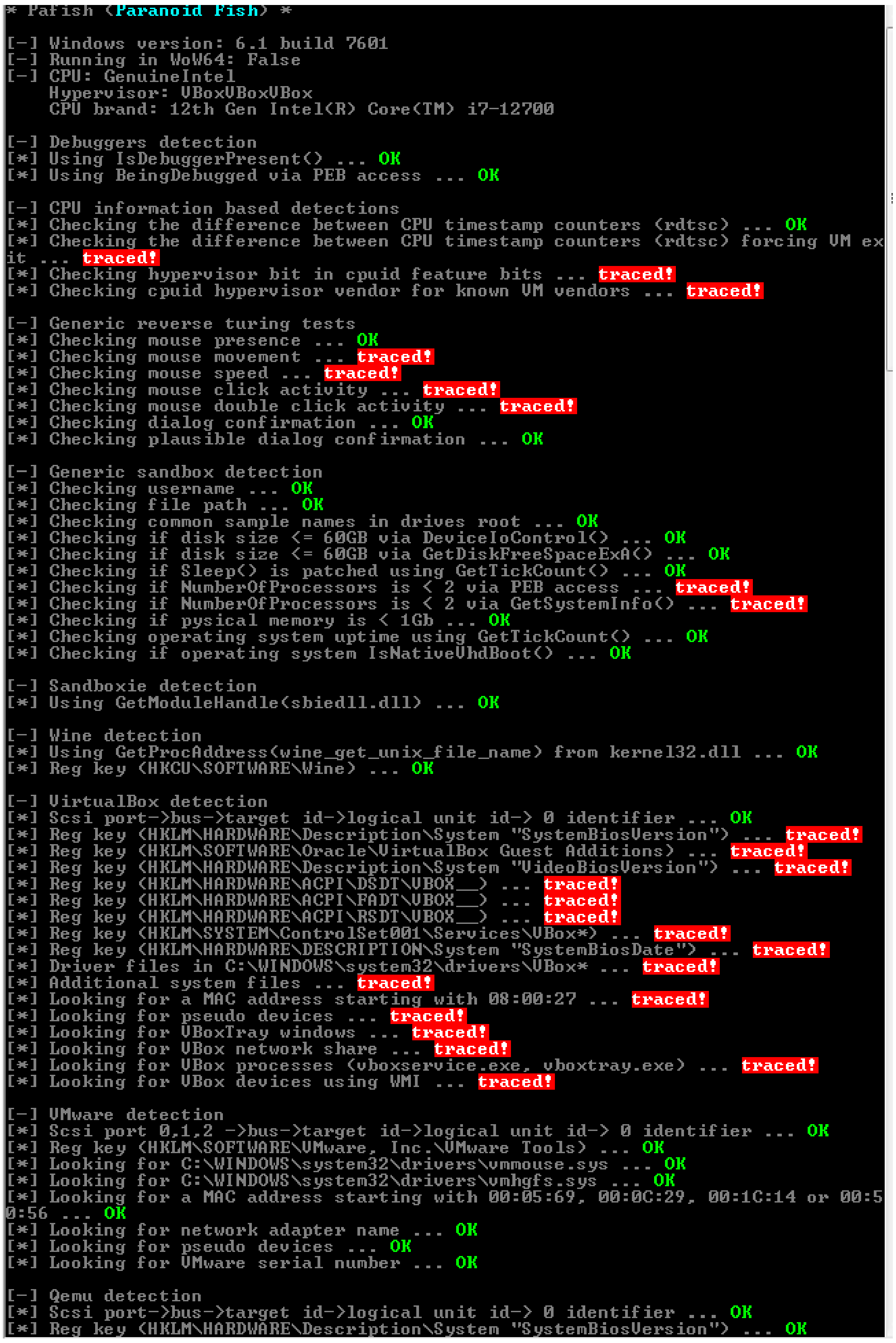
[75] Blackberry. 'Initial implants and network analysis suggest the 3cx supply chain operation goes back to fall 2022.' Visited: 30.05.2023. (), [Online]. Available: https://blogs.blackberry.com/en/2023/03/initial-implants-and-network-analysis-suggest-the-3cx-supply-chain-operation-goes-back-to-fall-2022.

[76] K. Zanki and ReversingLabs. 'Red flags flew over software supply chain-compromised 3cx update.' Visited: 30.05.2023. (), [Online]. Available: https://www.reversinglabs.com/blog/red-flags-fly-over-supply-chain-compromised-3cx-update.

Appendix A

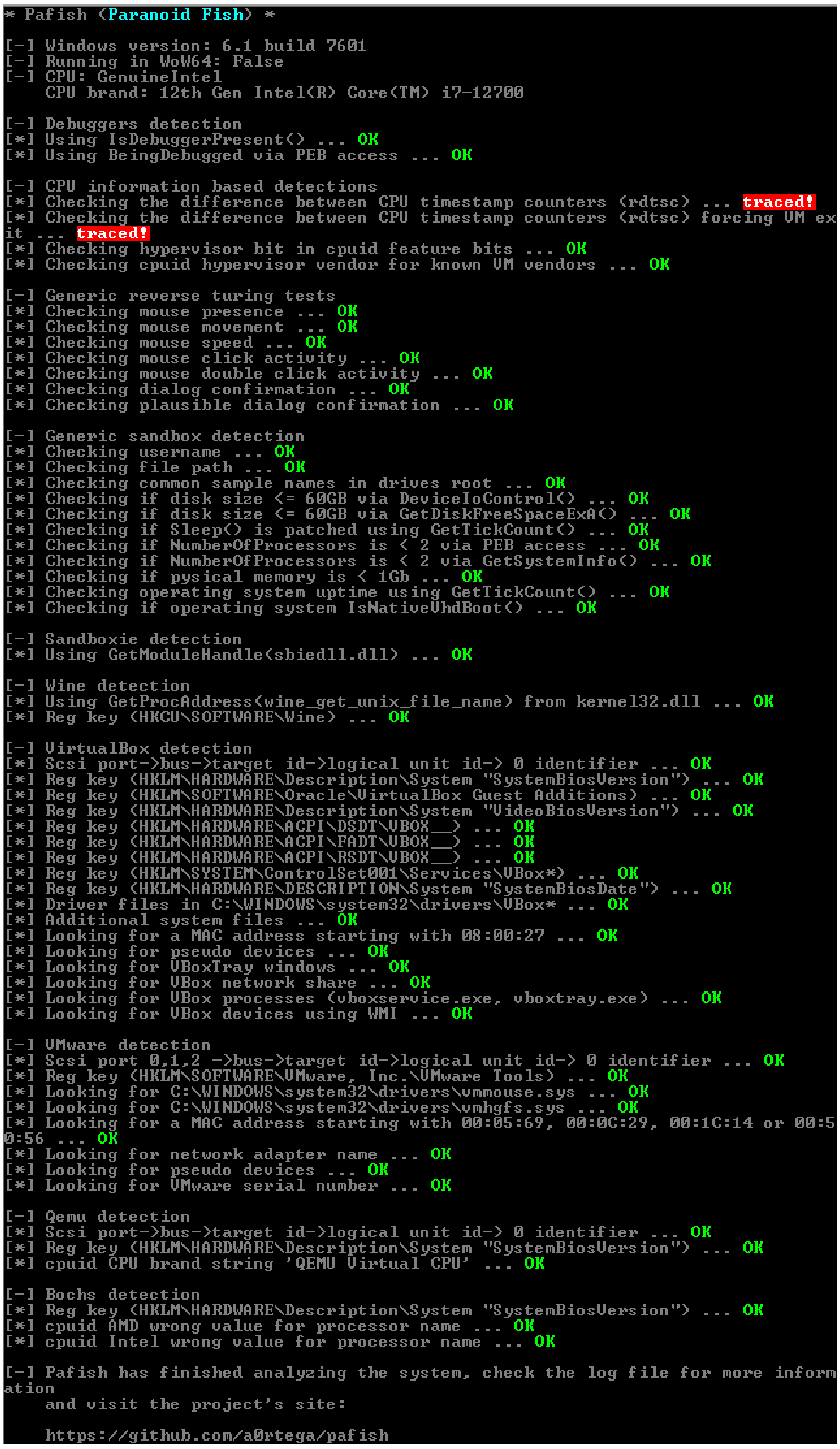
Additional Material

A.1 Screenshots



```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: VBoxVBoxVBox
    CPU brand: 12th Gen Intel(R) Core(TM) i7-12700

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

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

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... traced!
[*] Checking mouse speed ... traced!
[*] Checking mouse click activity ... traced!
[*] Checking mouse double click activity ... traced!
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

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

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.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-> 0 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\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... traced!
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... traced!
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... traced!
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... traced!
[*] Additional system files ... traced!
[*] Looking for a MAC address starting with 08:00:27 ... traced!
[*] Looking for pseudo devices ... traced!
[*] Looking for VBoxTray windows ... traced!
[*] Looking for VBox network share ... traced!
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... traced!
[*] Looking for VBox devices using WMI ... traced!

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 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-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
```

Figure A.1: A screenshot of the initial detections in Paranoidfish for the Cuckoo Guest VM. The part of the bottom that is missing belongs to Qemu and Boch and is not relevant as it was running in VirtualBox



```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    CPU brand: 12th Gen Intel(R) Core(TM) i7-12700

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... 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 reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... OK
[*] Checking mouse speed ... OK
[*] Checking mouse click activity ... OK
[*] Checking mouse double click activity ... OK
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.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-> 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\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 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-> 0 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

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

Figure A.2: A screenshot of the final detection in Paranoidfish for the Cuckoo guest VM.

A.2 Code

A.2.1 VirtualBox Shellscript

Code listing A.1: VirtualBox Shell Script for editing VM. Based on the script from [61]

```
#!/bin/sh
vboxmanage modifyvm "cuckoo1" --paravirtprovider legacy
vboxmanage modifyvm "cuckoo1" --macaddress1 6CF0491A6E12
vboxmanage modifyvm "cuckoo1" --bioslogoimagepath C:\aqr.bmp
vboxmanage modifyvm "cuckoo1" --hwvirtex on
vboxmanage modifyvm "cuckoo1" --vtxvpid on
vboxmanage modifyvm "cuckoo1" --vtxux on
vboxmanage modifyvm "cuckoo1" --apic on
vboxmanage modifyvm "cuckoo1" --pae on
vboxmanage modifyvm "cuckoo1" --longmode on
vboxmanage modifyvm "cuckoo1" --hpet on
vboxmanage modifyvm "cuckoo1" --nestedpaging on
vboxmanage modifyvm "cuckoo1" --largepages on
vboxmanage modifyvm "cuckoo1" --mouse ps2


vboxmanage setextradata "cuckoo1" "VBoxInternal/CPUM/EnableHVP" 0
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemVendor" "ASUS"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemProduct" "ASUS␣ZenBook␣Pro"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemVersion" "1.0"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemSerial" "1A2B3C4D5E6F"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemSKU" "UX580GD-AB1234"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemFamily" "Ultrabook"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiSystemUuid" "9852bf98-b83c-49db-a8de-182c42c7226b"

vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSVendor" "ASUS␣Inc."
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSVersion" "1.10.3"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSReleaseDate" "10/15/2022"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSReleaseMajor" "5"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSReleaseMinor" "9"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSFirmwareMinor" "0"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBIOSFirmwareMajor" "1"

vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardVendor" "ASUSTek␣Computer␣Inc."
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardProduct" "Zenbook␣Pro"
```

```
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardVersion" "A01"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardSerial" "IMB456721"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardAssetTag" "ASUS-ZEN-3417"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardLocInChass" "Board␣Loc␣In"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiBoardBoardType" "10"

vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiChassisVendor" "Asus␣Inc."
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiChassisType" 10
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiChassisVersion" "Mac-F22788AA"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiChassisSerial" "CSN12345678901234567"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiChassisAssetTag" "WhiteHouse"

vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiOEMVBoxVer" "Extended␣version␣info:␣1.00.00"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/pcbios/0/Config/
    DmiOEMVBoxRev" "Extended␣revision␣info:␣1A"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port0/
    ModelNumber" "Hitachi␣HTS543230AAA384"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port0/
    FirmwareRevision" "ES2OA60W"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port0/
    SerialNumber" "2E3024L1T2V9KA"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    ModelNumber" "Slimtype␣DVD␣A␣␣DS8A8SH"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    FirmwareRevision" "KAA2"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    SerialNumber" "ABCDEF0123456789"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    ATAPIVendorId" "Slimtype"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    ATAPIProductId" "DVD␣A␣␣DS8A8SH"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/ahci/0/Config/Port1/
    ATAPIRevision" "KAA2"
vboxmanage setextradata "cuckoo1" "VBoxInternal/Devices/acpi/0/Config/AcpiOemId" "
    ASUS"
```

A.2.2 VirtualBox Registry File

Code listing A.2: Registry file from [62]

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\SYSTEM]
"SystemBiosDate"="06/12/10"
"SystemBiosVersion"="BC1.05"
"VideoBiosVersion"="VC1.20"
```

```
[-HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\VBOX__]
[-HKEY_LOCAL_MACHINE\HARDWARE\ACPI\FADT\VBOX__]
[-HKEY_LOCAL_MACHINE\HARDWARE\ACPI\RSDT\VBOX__]
[-HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\Virtual Box Guest Additions]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\VBox*]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\CriticalDeviceDatabase\pci#
    ven_80ee&dev_cafe]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\{4D36E97D-E325-11CE-BFC1
    -08002BE10318}\0020]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Enum\PCI\VEN_80EE&DEV_CAFE&
    SUBSYS_00000000&REV_00]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\services\VBoxGuest\Enum]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Enum\PCI\VEN_80EE&DEV_CAFE&
    SUBSYS_00000000&REV_00]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Control\Class\{4D36E97D-E325-11CE-BFC1
    -08002BE10318}\0020]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Control\CriticalDeviceDatabase\pci#
    ven_80ee&dev_cafe]
[-HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Enum\PCI\VEN_80EE&DEV_CAFE&
    SUBSYS_00000000&REV_00]
[-HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E97D-E325-11CE-
    BFC1-08002BE10318}\0020]
[-HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CriticalDeviceDatabase\pci#
    ven_80ee&dev_cafe]
[-HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\PCI\VEN_80EE&DEV_CAFE&
    SUBSYS_00000000&REV_00]
[-HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\VBoxGuest\Enum]
```

A.2.3 Script using Floss and stringsifter

Code listing A.3: Python/Jupyter script using Floss & Stringsifter

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import subprocess
import csv
import os
from multiprocessing import Pool
import time


# In[2]:


# Run flarestrings command for file1_path

    ## TThis function runs Flare-Floss (https://github.com/mandiant/flare-floss)
    ## against the executable file that it receives as a variable.
    ## Flare-Floss works more or less like strings but can also de-obfuscate
        strings.
def runFloss(pathToFile):
    commandFlarestring = ['floss','-q', pathToFile]
    flarestringOutput = subprocess.check_output(commandFlarestring, text=True)
    return flarestringOutput
    ## Flare only works for PE files, so this is ran if the file is a MSI file.
    ## this uses stringsifter (https://github.com/mandiant/stringsifter) instead
    ## This basically works as regular string does.
def runFlarestrings(pathToFile):
    commandFlarestring = ['flarestrings', pathToFile]
    flarestringOutput = subprocess.check_output(commandFlarestring, text=True)
    return flarestringOutput

    ## This uses the rank_strings function from stringsifter (https://github.com/
        mandiant/stringsifter)
    ## Ranked_strings essentially just ranks based on ML model trained on malware
        strings etc.
    ## So it should hopefully rank the most suspicious and relevant strings at the
        top.
    ## other than that it loops 4 times as we have 4 intervals (10, 100, 1000, and
        10 000)
    ## then it returns the 4 ranked lists as a single list.
def rankThemStrings(flarestringOutput):
    initialAmount=10
    rankedStringsLists = {}

    for i in range(4):
        stringsAmount=str(initialAmount)
        initialAmount=initialAmount*10
        commandRankStrings = ['rank_strings', '-l', stringsAmount,'-s']
        rankstringOutput = subprocess.check_output(commandRankStrings, input=
            flarestringOutput, text=True)
        rankedStringsList = rankstringOutput.splitlines()
        rankedStringsLists[f'{i+1}'] = rankedStringsList
    return rankedStringsLists

    ## This just compares the strings in the two lists and removes duplicates in 1
        from 2.
    ## then it returns a filtered list, this is relevant as we want the differences
         between the two highlighted
def filterStrings(list1, list2):
    filterComplete = [string for string in list2 if string not in list1]
    return filterComplete

    ## More or less the super function, runs the other functions.
def runFiles(file1, file2):


    ## Checks if the file is a msi file or not, since flare does not work with
        anything else than pe files :(
    ## We also use multiprocessing so that it does not take years to complete... ok
         more like two or three hours.
    ## This cuts it down by a lot. MediaGet still takes ages though... 39minutes...
    if os.path.splitext(file1)[1] == '.msi':
        with Pool(processes=2) as pool:
            results = pool.map(runFlarestrings, [file1, file2])
    else:
        with Pool(processes=2) as pool:
            results = pool.map(runFloss, [file1, file2])

    ## these names are legit. ok?
    LegitList = results[0]
    MalList = results[1]
    ## Multiprocessing for the ranking function. So we can rank both the baddy and
        the nice list
    with Pool(processes=2) as pool:
        LegitLists = pool.map(rankThemStrings, [LegitList, MalList])


    ## Let there be one list!
    ## This function just runs the filter/comparison function 4 times (one for each
         of our intervals!)
    ## This gives us a list of each interval that contains the unique strings in
        the malicious sample.
    filterLists = []
    for i in range(4):
        filterList = filterStrings(LegitLists[0][f'{i+1}'], LegitLists[1][f'{i+1}'
            ])
        filterLists.append(filterList)

    return LegitLists, filterLists


    ## Here be writetocsv function. We all know what this does, we've all seen it
        before.
def write_to_csv(folderPath, fileName, data1, data2):
    filePath = os.path.join(folderPath, fileName)
    with open(filePath, 'w', newline='') as csvFile:
        writer = csv.writer(csvFile)
        ## Just writes the rows to the csv file. Makes it a lot easier to read.
        writer.writerow([folderPath, 'malicious_'+folderPath])
        for item1, item2 in zip(data1, data2):
            writer.writerow([item1, item2])


    ## The unique function! What a unique name.
    ## So this function just takes the beginning folder path (so Solarwinds/ from
        nr 1)
    ## This allows us to place the csv file back into the correct folder, which
        makes life easier.
def writeUnique(folderPath, fileName, filteredOutput):
    filePath = os.path.join(folderPath, fileName)
    with open(filePath, 'w', newline='') as csvFile:
        writer = csv.writer(csvFile)
        for string in filteredOutput:
            writer.writerow([string])





# In[3]:


## could I have done this smarter? Yes. Should I? Perhaps
## But at the end of the day there is only 24 binaries,
## so instead of using 5 hours where i can spend 1 minute and all that stuff.
## from this is also becomes obivous that this file is SUPPOSED to be in the same
    folder as the folders
## to the binaries.
## Everything is coded with the expection that first comes the legitimate binary,
    then second comes the malicious.
file_paths = {
```

```
0: 'SolarWinds/SolarWinds.Orion.Core.BusinessLayer.dll',
1: 'SolarWinds/Malicious␣-␣SolarWinds.Orion.Core.BusinessLayer.dll',
2: 'M.E.doc␣-␣NotPetya/ZvitPublishedObjects.dll',
3: 'M.E.doc␣-␣NotPetya/Malicious␣-␣ZvitPublishedObjects.dll',
4: 'ccleaner/CCleaner_v5.32.6129.exe',
5: 'ccleaner/CCleaner_v5.33.6162.exe',
6: 'Mesa/SwissrangerSetup1.0.14.706.exe',
7: 'Mesa/Malicious-SwissrangerSetup1.0.14.706.exe',
8: 'eGrab/egrabitsetup.exe',
9: 'eGrab/Malicious-egrabitsetup.exe',
10: 'DoFoil/mediaget.exe',
11: 'DoFoil/Malicious␣-␣mediaget.exe',
12: 'eCatch/eCatcherSetup.exe',
13: 'eCatch/Malicious-eCatcherSetup.exe',
14: 'Darkside/DH_SMARTPSS-Win32_ChnEng_IS_V2.002.0000009.0.R.190426.exe',
15: 'Darkside/Malicious-SMARTPSS-Win32_ChnEng_IS_V2.002.0000007.0.R.181023-General-
     v1.exe',
16: '3CXd3dcompiler/d3dcompiler_47.dll',
17: '3CXd3dcompiler/Mal_d3dcompiler_47.dll',
18: '3CXffmpeg/ffmpeg.dll',
19: '3CXffmpeg/Mal_ffmpeg.dll',
20: '3CX32/3CXDesktopApp32.exe',
21: '3CX32/Mal_3CXDesktopApp32.exe'
#22: '3CX64/3CXDesktopApp64.exe', # Commented these out as technically Floss not
     supposed to run against files
#23: '3CX64/Mal_3CXDesktopApp64.exe'# Larger than 16MB, and these are almost 150.
}## Dont mind that there is other file /almost/ as large, as these just would not
     finish.


# In[ ]:


## The main function! CODE PRACTICES!
## So this is a messy function, but its mine and i love it.
## It goes from the first file_paths dictrionary to the last
## as each sample is a pair, we increase it by 2.
## Included a timer to have some perspective of how long everything takes.

if __name__ == '__main__':
    startTime = time.time()
    i = 0
    while i < 22:
        print(file_paths[i], file_paths[i+1])
        LegitLists, listOfFilter = runFiles(file_paths[i], file_paths[i+1])

        file_path = file_paths[i]
        folder_name = file_path.split('/')[0]

        ## Writes in total 8 csv files, 2 for each interval.
        ## 1 is the ranked strings from the samples, while the other is the unique
             strings in the malicious sample.
        for c in range(4):
            a = str(c+1)
            write_to_csv(folder_name, a+folder_name+'.csv', LegitLists[0][a],
                 LegitLists[1][a])
            writeUnique(folder_name, a+'Differences'+folder_name+'.csv',
                 listOfFilter[c])
        i=i+2
        elapsedTime = time.time() - startTime
        print(f"{folder_name}␣finished␣at:␣{elapsedTime}␣seconds")


# In[ ]:




# In[ ]:
```

A.3 Analysis

A.3.1 Swiss Ranger Tables

A.3.2 eCatcher Tables

A.3.3 eGrabit Tables

A.3.4 MediaGet Tables

A.3.5 SmartPSS Tables

A.3.6 3CX Desktop App tables

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	7,99	7,981	**-0,009**	PeStudio
File-size	1181500	1311927	**+130427**	PeStudio
Version Info	Yes.	None.	**Changed.**	PeStudio
PDB Path	None.	`d:\Projects\WinRAR\SFX\build\sfxzip32\Release\sfxzip.pdb`	Very Slight change.	Cuckoo Static
Packer	Nullsoft	Zip Archive	**Changed.**	Exeinfo PE
Double Packed	When unzipped contains a directory	When unzipped contains a packaged setup executable & an additional executable dll file	**Change.**	7zip
Secion Changes	.ndata	.CRT	**Name & Content**	PeStudio
.text Entropy	6,44	6,511	**+0,071**	PeStudio
.text File-Ratio	2,04	4,02	**+1,98**	PeStudio
.text Raw-Size	24064	52736	**+28672**	PeStudio
.text Virtual-Size	23628	52299	**+28671**	PeStudio
.rdata Entropy	5,047	4,967	**-0,08**	PeStudio
.rdata File-Ratio	0,43	0,55	**+0,12**	PeStudio
.rdata Raw-Size	5120	7168	**+2048**	PeStudio
.rdata Virtual-Size	4764	7029	**+2265**	PeStudio
.data Entropy	4,801	1,320	**-3,481**	PeStudio
.data File-Ratio	0,09	0,04	**-0,05**	PeStudio
.data Raw-Size	1024	512	**-512**	PeStudio
.data Virtual-Size	154712	121304	**-33408**	PeStudio
.rsrc Entropy	4,773	5,129	**0,356**	PeStudio
.rsrc File-Ratio	0,82	1,21	**0,39**	PeStudio
.rsrc Raw-Size	9728	15872	**6144**	PeStudio
.rsrc Virtual-Size	9464	15400	**5936**	PeStudio
Overlay Size	1140540	1234103	93563	PeStudio
Overlay Signature	Nullsoft	PKZIP	**Changed.**	PeStudio
Overlay File-Ratio	96,53	94,07	**-2,46**	PeStudio

Click 4.1.3 to go back to Section 4.1.3

Table A.1: Havex-SwissRanger: Static Differential Analysis General File & Sections

Signature	**Instruction**	**Comment**
A process attempted to delay the analyst task	rundll32.exe tried to sleep 168 seconds, actually delayed analysis time by 168 seconds	The malicious sample attempts to sleep, while the benign does not.
Allocates Read-Write-Execute Memory (Events)	NtProtectVirtualMemory	51 in malicious compared to 21 in benign.
Creates a suspicious Process	`C:\Windows\System32\cmd.exe"/cC:\Users\rick\AppData\Local\Temp\setup.exe` `c:\windows\system32\rundll32.exeC:\Users\rick\AppData\Local\Temp\tmp687.dll,RunDllEntry`	Two files ran by command line. Same we saw in static analysis.
Creates a thread using CreateRemoteThread in a non-child process indicative of process injection	2296: `c:\windows\system32\rundll32.exeC:\Users\rick\AppData\Local\Temp\tmp687.dll,RunDllEntry` 2540: `C:\Windows\system32\rundll32.exe""c:\users\rick\appdata\roaming\sydmain.dll"",AGTwRec"`	n/a
Creates a windows hook that monitors keyboard input	SetWindowsHookExW, hook_identifier: 13 (WH_KEYBOARD_LL)	Uses a hook to monitor user input
Creates executable files on the filesystem	setup.exe & tmp687.dll	The two files that the other signatures detect being ran.
Drops a binary and executes it	`C:\Users\rick\AppData\Local\Temp\setup.exe`	The setup file that is ran.
Drops an executable to the user AppData Folder	`C:\Users\rick\AppData\Local\Temp\setup.exe` `c:\Users\rick\AppData\Roaming\NSDS.dll"`	One file was dropped in AppData by the legitimate, however, these two were not.
Installs itself for autorun at Windows Startup	`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\load` `C:\Windows\system32\rundll32.exe"c:\users\rick\appdata\roaming\sydmain.dll",AGTwLoad`	Sets rundll32 to run the sydmain.dll's AGTwLoad function at startup
Manipulates memory of non-child process indicative of process injection	Process 2296 (setup.exe) manipulating memory of non-child process 2020 (explorer.exe) Process 2540 (sydmain.dll) manipulating memory of non-child process 2020 (explorer.exe)	setup.exe & sydmain.dll manipulates the explorer.exe process.
Potential code injection by writing to the memory of another process	Process 2296 injected into non-child 2020 Process 2540 injected into non-child 2020	setup.exe & sydmain.dll injects into explorer.exe process
Repeatedly searches for a not-found process	Process32NextW: rundll32.exe	183 events detected by Cuckoo
Searches running processes potentiall to identify processes for sandbox evasion, code injection or memory dumping	Process32FirstW	40 events detected by Cuckoo. Malware getting a handle on running processes and checking name to inject into
This executable has a PDB path	`d:\Projects\WinRAR\SFX\build\sfxzip32\Release\sfxzip.pdb`	PDB path that is not present in the legitimate sample

Click 4.1.3 to go back to Section 4.1.3

Table A.2: Havex-SwissRanger: Cuckoo Dynamic Analysis

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File-size	44232128	43971440	**-260688**	Pefile Script
Signature	Yes.	No.	**Change.**	Cuckoo Static
Version Info	Yes.	No.	**Change.**	Cuckoo Static
Packer	Inno/Borland Delphi	Nullsoft PiMP Stub	**Changed.**	Exeinfo PE
Double Packed	When unzipped contains a directory	When unzipped contains a packaged setup executable & an additional executable dll file	**Change.**	7zip
Secion Changes	CODE, DATA, BSS, .idata, .tls, .rdata, .reloc, .rsrc	.text, .data, .ndata, .rdata, .rsrc	**Sections missing and added.**	Pefile Script
CODE/.text Entropy	6,561	6,380	**-0,181**	Pefile Script
CODE/.text File-Ratio	0,09%	0,05%	**-0,04%**	Pefile Script
CODE/.text Raw-Size	37888	23040	**-14848**	Pefile Script
CODE/.text Virtual-Size	37732	22590	**-15142**	Pefile Script
DATA/.data Entropy	2,739	4,986	**+2,247**	Pefile Script
DATA/.data Virtual-Size	588	111572	**+110984**	Pefile Script
.rdata Entropy	0,204	5,036	**+4,831**	Pefile Script
.rdata File-Ratio	0%	0,01%	**+0,01%**	Pefile Script
.rdata Raw-Size	512	4608	**+4096**	Pefile Script
.rdata Virtual-Size	24	4324	**+4300**	Pefile Script
.rsrc Entropy	5,617	5,402	**-0,215**	Pefile Script
.rsrc File-Ratio	0,05%	0,03%	**-0,02%**	Pefile Script
.rsrc Raw-Size	24064	11776	**-12288**	Pefile Script
.rsrc Virtual-Size	23732	11456	**-12276**	Pefile Script

Click 4.1.3 to go back to Section 4.1.3

Table A.3: Havex-Talk2M eCatcher: Static Differential Analysis General File & Sections

Type	**Benign**	**Malicious**
Resources Amount	6	14
Manifest	XML 1.0 document, ASCII text, with very long lines, with no line terminators	XML 1.0 document, ASCII text, with CRLF line terminators
Libraries	5	8
Total Imports	96	151
USER32.dll	12	60
AD-VAPI32.dll	6	9
OLEAUT32.dll	5	0
COMCTL32.dll	1	4
GDI32.dll	0	8
VER-SION.dll	0	3
ole32.dll	0	4
SHELL32.dll	0	6

Click 4.1.3 to go back to Section 4.1.3

Table A.4: Havex-Talk2M eCatcher: Static Differential Analysis Resources & Library

Signature	**Instruction**	**Comment**
The executable contains unknown PE section names indicative of a packer (could be a false positive	.ndata added, CODE, DATA, BSS missing	the .ndata is the packer from the nullsoft installer.
A process attempted to delay the analyst task	rundll32.exe tried to sleep 591 seconds, actually delayed analysis time by 591 seconds	Malicious sample attempts to sleep. Common evasion tactic[30]
Allocates Read-Write-Execute Memory (Events)	NtProtectVirtualMemory	20 compared to 8
Checks adapter address which can be used to detect virtual network interfaces	GetAdaptersAddresses	Is called once by the malicious sample.
Creates a windows hook that monitors keyboard input	SetWindowsHookExW, hook_identifier: 13 (WH_KEYBOARD_LL)	Monitoring keyboard input of user
Creates executable files on the filesystem	`C:\Users\rick\AppData\Local\Temp\TmProvider.dll` `C:\Users\rick\AppData\Local\Temp\eCatcherSetup.exe`	Creates two additional files compared to usual, these are also the files being ran for nefarious purposes.
Creates hidden or system file	`C:\Users\rick\AppData\Local\Temp\TmProvider.dll` `C:\Users\rick\AppData\Local\Temp\eCatcherSetup.exe`	Hides the two files dropped in the AppData folder with SetFileAttributesW.
Expresses interest in specific running processes	process: potential process injection target explorer.exe	The signature looks for process API calls and tracks them. It triggers based on a list (where explorer is part of it) and alerts if one or more triggers
HTTP traffic contains suspicious features which may be indicative of malware related traffic	POST `http://zhayvoronok[.]com/wp-includes/pomo/idx.php?id=6163674211069016526008BFC60-25&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	POST method with no referer header
Installs itself for autorun at Windows Startup	rundll32 `C:\Windows\system32\TMPprovider038.dll",RunDllEntry` rundll32 `C:\Windows\system32\TMPprovider038.dll",RunDllEntry`	Malicious dll file being sat to run at startup
Manipulates memory of non-child process indicative of process injection	NtAllocateVirtualMemory, Process injection: Process 2892 manipulating memory of non-child process 2020. 21 Events	process 2892 is rundll32 running 'TmProvider.dll, RunDllEntry'. 2020 is explorer
Performs some HTTP Request	POST `http://zhayvoronok[.]com/wp-includes/pomo/idx.php?id=6163674211069016526008BFC60-25&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	Triggers once more due to the request
Potential code injection by writing to the memory of another process	NtAllocateVirtualMemory, Process injection: Process 2892 injected into non-child 2020, 21 events"	process 2892 is rundll32 running 'TmProvider.dll, RunDllEntry'. 2020 is explorer
Searches running processes potential to identify processes for sandbox evasion, code injection or memory dumping	Process32FirstW, 26 events	Another alert for process injection
Sends data using the HTTP POST Method	POST `http://zhayvoronok[.]com/wp-includes/pomo/idx.php?id=6163674211069016526008BFC60-25&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	Triggers once more due to the same request
Sets of modifies WPAD proxy autoconfiguration file for traffic interception	RegSetValueExA, WpadDecisionReason	WPAD being configured.

Click 4.1.3 to go back to Section 4.1.3

Table A.5: Havex-Talk2M eCatcher: Cuckoo Dynamic Analysis

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	7,995	7,998	**+0,003**	PeStudio
File-size	2376808	2525510	**+148702**	PeStudio
Signature	Yes.	No.	**Changed.**	PeStudio
Version Info	Yes.	No.	**Changed.**	PeStudio
Packer	Wise Installer	Nullsoft PiMP Stub	**Packer/installer changed**	Exeinfo PE
Double Packed	When unzipped contains a directory	When unzipped contains a packaged setup executable & an additional executable dll file	**Change.**	7zip
Secion Changes		.ndata	**.ndata added**	Pefile Script
Overlay	No.	Yes, nullsoft.	**Changed.**	PeStudio
.text Entropy	5,554	6,38	**+0,826**	PeStudio
.text File-Ratio	0,02	0,91	**+0,89**	PeStudio
.text Raw-Size	512	23040	**+22528**	PeStudio
.text Virtual-Size	510	22590	**+22080**	PeStudio
.rdata Entropy	2,839	5,036	**+2,197**	PeStudio
.rdata File-Ratio	0,04	0,18	**+0,14**	PeStudio
.rdata Raw-Size	1024	4608	**+3584**	PeStudio
.rdata Virtual-Size	533	4324	**+3791**	PeStudio
.data Entropy	0,269	4,986	**+4,717**	PeStudio
.data File-Ratio	0,02	0,004	**-0,016**	PeStudio
.data Raw-Size	512	1024	**+512**	PeStudio
.data Virtual-Size	20	111572	**+111552**	PeStudio
.rsrc Entropy	7,996	3,919	**-4,077**	PeStudio
.rsrc File-Ratio	99,63	0,1	**-99,53**	PeStudio
.rsrc Raw-Size	2368000	2560	**-2365440**	PeStudio
.rsrc Virtual-Size	2371584	2304	**-2369280**	PeStudio
Flagged Strings	18	36	**+18**	PeStudio
MITRE Strings	7	14	**+7**	PeStudio

Click 4.1.3 to go back to Section 4.1.3

Table A.6: Havex-eGrabit: Static Differential Analysis General File & Sections

Type	**Benign**	**Malicious**
Resources Amount	4	6
Manifest	XML 1.0 document, ASCII text, with very long lines, with no line terminators	None.
Libraries	2	8
Library Imports	15	151
KERNEL32.dll	14	58
USER32.dll	1	60
GDI32.dll	0	8
COMCTL32.dll	0	4
SHELL32.dll	0	6
ADVAPI32.dll	0	9
ole32.dll	0	4
VERSION.dll	0	3

Click 4.1.3 to go back to Section 4.1.3

Table A.7: Havex-eGrabit: Static Differential Analysis Resources & Library

Type	**New Strings found**	**Method**
MITRE Execution through API	ShellExecute, LoadLibraryEx	PeStudio
MITRE Modify Registry	RegSetValueEx, RegCreateKeyEx	PeStudio
MITRE Data Destruction	RegDeleteValue, RegDeleteKey	PeStudio
MITRE Process Injection	SetWindowLong, FindWindowEx, SendMessageTimeout, GetWindowLong	PeStudio
MITRE Query Registry	RegEnumKey, RegEnumValue, RegQueryValueEx	PeStudio
MITRE File and Directory Discovery	FindFirstFile, FindNextFile, GetSystemDirectory.	PeStudio
MITRE Remote File Copy	MoveFile, MoveFileEx, CopyFile	PeStudio
MITRE Clipboard Data	CloseClipboard, SetClipboardData, EmptyClipboardData, OpenClipboard.	PeStudio
MITRE Sandbox Evasion	Sleep	PeStudio
MITRE System Information Discovery	ExpandEnvironmentStrings	PeStudio
MITRE System Time Discovery	GetTickCount	PeStudio
Top 10k Ranked Strings difference	The malicious sample had 9931 different strings than the benign out of the top 10 000 strings.	FLOSS & Stringsifter

Click 4.1.3 to go back to Section 4.1.3

Table A.8: Havex-eGrabit: Static Differential Analysis Strings

Signature	**Instruction**	**Comment**
A process attempted to delay the analyst task	rundll32.exe tried to sleep 547 seconds, actually delayed analysis time by 547 seconds	Attempts to sleep most likely to delay analysis
Allocates Read-Write-Execute Memory (Events)	NtProtectVirtualMemory	39 Compared to 28 in benign
Checks adapter address which can be used to detect virtual network interfaces	GetAdaptersAddresses	Is called once by the malicious sample
Creates a windows hook that monitors keyboard input	SetWindowsHookExW, hook_identifier: 13 (WH_KEYBOARD_LL)	Monitoring keyboard input of user
Creates executable files on the filesystem	`C:\Users\rick\AppData\Local\Temp\egrabitsetup.exe` `C:\Users\rick\AppData\Local\Temp\TmProvider.dll`	Creates two additional files compared to the benign.
Creates hidden or system file	`C:\Users\rick\AppData\Local\Temp\egrabitsetup.exe` `C:\Users\rick\AppData\Local\Temp\TmProvider.dll`	Uses SetFileAttributesW to hide the dropped files.
Drops an executable to the userAppData Folder	`C:\Users\rick\AppData\Local\Temp\egrabitsetup.exe` `C:\Users\rick\AppData\Local\Temp\TmProvider.dll`	Another signature triggers on these files being created
Expresses interest in specific running processes	potential process injection target:explorer.exe	The signature looks for process API calls and tracks them. It triggers based on a list (where explorer is part of it) and alerts if one or more triggers
HTTP traffic contains suspicious features which may be indicative of malware related traffic	POST `http://www.pc-service-fm.de/modules/mod_search/src.php?id=25697426326281793500C8F590-891062d5c51294011447f8168bc4437c&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	The pattern behind v2 matches the one from Talk2m, which makes sense, as they are part of the same campaign. POST method with no referer header.
Installs itself for autorun at Windows Startup	`HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\TmProvider` `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\TmProvider`	Sets itself to start at startup.
Manipulates memory of non-child process indicative of process injection	NtAllocateVirtualMemory, Process 2800 manipulating memory of non-child process 2020	2800 is rundll32 running 'TmProvider.dll, RunDllEntry'. 2020 is explorer
Performs some HTTP Request	POST `http://www.pc-service-fm.de/modules/mod_search/src.php?id=25697426326281793500C8F590-891062d5c51294011447f8168bc4437c&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	Another generated event due to outbound post.
Potential code injection by writing to the memory of another process	Process TmProvider.dll 2800 into non-child 2020	2800 is rundll32 running 'TmProvider.dll, RunDllEntry'. 2020 is explorer
Searches running processes potentiall to identify processes for sandbox evasion, code injection or memory dumping	Process32NextW, process_name: audiodg.exe	Alert for process injection
Sends data using the HTTP POST Method	POST `http://www.pc-service-fm.de/modules/mod_search/src.php?id=25697426326281793500C8F590-891062d5c51294011447f8168bc4437c&v1=038&v2=498139398&q=5265882854508EFCF958F979E4`	Another generated event due to outbound post.
Sets of modifies WPAD proxy autoconfiguration file for traffic interception	RegSetValueExA, RegSetValueExW	Runs 15 times, changing 15 wpad keys.
The executable contains unknown PE section names indicative of a packer	section: .ndata	.ndata is there due to nullsoft installer

Click 4.1.3 to go back to Section 4.1.3

Table A.9: Havex-eGrabit: Cuckoo Dynamic Analysis

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	7,225	7,206	**-0,019**	PeStudio
File-size	14040816	14117376	**+76560**	PeStudio
Signature	Yes.	No.	**Change.**	Cuckoo Static
PDB Path	`E:\mediaget-adframes-release\release\mediaget.pdb`	`X:\MediaGet\src\Desktop.3745\build-ide\release\mediaget.pdb`	Very Slight change.	PeStudio
.text Entropy	6,296	6,285	**-0,011**	PeStudio
.text File-Ratio	42,95%	42,77%	**-0,18%**	PeStudio
.text Raw-Size	6030336	6038528	**+8192**	PeStudio
.text Virtual-Size	6038357	6030251	**+8106**	PeStudio
.rdata Entropy	7,484	7,454	**-0,03**	PeStudio
.rdata File-Ratio	52,04%	52,32%	**+0,28%**	PeStudio
.rdata Raw-Size	7306240	7386112	**+79872**	PeStudio
.rdata Virtual-Size	7305847	7385751	**+79904**	PeStudio
.data Entropy	6,194	6,174	**-0,02**	PeStudio
.data File-Ratio	0,83%	0,81%	**-0,02%**	PeStudio
.data Raw-Size	115224	114688	**-536**	PeStudio
.data Virtual-Size	137376	135808	**-1568**	PeStudio
.reloc Raw-Size	443392	445952	**-2560**	PeStudio
.reloc Virtual-Size	443036	445630	**-2594**	PeStudio

Click 4.1.4 to go back to Section 4.1.4

Table A.10: DoFoil-MediaGet: Static Differential Analysis General File & Sections

Type	**Benign**	**Malicious**
Libraries	33	28
Library Imports	3559	3529
LIBEAY32.dll	74	69
MSVCP100.dll	81	61
MSVCR100.dll	123	112
opencv_core320.dll	7	0
Opencv_img_hash320.dll	8	0
opencv_imgproc320.dll	1	0
opencv_videoio320.dll	5	0
opencv_videoio320.dll	2	0
Qt5Core.dll	910	922
Qt5Network.dll	153	175
Qt5Widgets.dll	1418	1416
SSLEAY32.dll	64	61

Click 4.1.4 to go back to Section 4.1.4

Table A.11: DoFoil-MediaGet: Static Differential Analysis Resources & Library

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File-size	143921828	132620403	**-11301425**	Pefile Script
Double Packed	When unzipped contains a install directory	When unzipped contains a packaged install executable & an additional executable file	**Change.**	7zip
Renamed Executable	Not Applicable for this.	The malicious sample contains a renamed executable file. The file with the name Smartpss.exe, has the version info for a microsoft file named MSHTA.exe	**New**	7zip
.text Entropy	6,433	6,435	**+0,002**	Pefile Script
.text Raw-Size	23040	26112	**+3072**	Pefile Script
.text Virtual-Size	22738	25687	**+2949**	Pefile Script
.rdata Entropy	5,180	5,261	**+0,081**	Pefile Script
.rdata Raw-Size	4608	5120	**+512**	Pefile Script
.rdata Virtual-Size	4496	4992	**+496**	Pefile Script
.data Entropy	4,618	4,134	**-0,484**	Pefile Script
.data Raw-Size	1024	1536	**+512**	Pefile Script
.data Virtual-Size	110456	152888	**+42432**	Pefile Script
.ndata Virtual-Size	278528	32768	**-245760**	Pefile Script
.rsrc Entropy	6,101	6,321	**+0,220**	Pefile Script
.rsrc File-Ratio	0,14%	0,13%	**-0,01%**	Pefile Script
.rsrc Raw-Size	205312	177152	**-28160**	Pefile Script
.rsrc Virtual-Size	205048	176824	**-28224**	Pefile Script

Click 4.5 to go back to Figure 4.5

Table A.12: Darkside-SmartPss: Static Differential Analysis General File & Sections

Changed Variable	**Benign**	**Malicious**	**Change**	**Method**
File Entropy	6,683	6,704	**+0,021**	Pefile Script
File-size	2789376	2814976	**+25600**	Pefile Script
Secion Changes	.voltbl	Missing	**.voltbl missing**	Pefile Script
.text Entropy	6,690	6,718	**+0,028**	Pefile Script
.text File-Ratio	80,18%	80,12%	**-0,06%**	Pefile Script
.text Raw-Size	2236416	2255360	**+18944**	Pefile Script
.text Virtual-Size	2236128	2254956	**+8828**	Pefile Script
.rdata Entropy	5,810	5,802	**-0,008**	Pefile Script
.rdata File-Ratio	16,94%	17,01%	**+0,07%**	Pefile Script
.rdata Raw-Size	472576	478720	**+6144**	Pefile Script
.rdata Virtual-Size	472548	478396	**+5848**	Pefile Script
.data Entropy	3,544	3,541	**-0,003**	Pefile Script
.data Virtual-Size	1433832	1433880	**+48**	Pefile Script
.pdata Entropy	6,112	6,065	**-0,047**	Pefile Script
.pdata Raw-Size	44544	45056	**+512**	Pefile Script
.pdata Virtual-Size	44424	44784	**+360**	Pefile Script
.00cfg Entropy	0,429	0,511	**+0,082**	Pefile Script
.00cfg Virtual-Size	40	56	**+16**	Pefile Script
.gxfg Entropy	5,047	5,214	**+0,168**	Pefile Script
.gxfg Virtual-Size	10800	11248	**+448**	Pefile Script
_RDATA Entrop	2,472	3,261	**+0,789**	Pefile Script
_RDATA Virtual-Size	244	348	**104**	Pefile Script
.reloc Entropy	5,442	5,412	**-0,030**	Pefile Script
.reloc File-Ratio	0,44%	0,45%	**+0,01%**	Pefile Script
.reloc Raw-Size	12288	12800	**+512**	Pefile Script

Table A.13: VEILEDSIGNAL-3CX-ffmpeg.dll: Static Differential Analysis General File & Sections