Master Degree Project

UNIVERSITY OF SKÖVDE

Impersonating a sand-box against evasive malware

A physical target machine impersonating a sand-box analyzing environment against the evasion techniques used by malware

Axel Lindorin

Supervisor: Martin Lundgren
Examiner: Marcus Nohlberg

Master Degree Project (120 ECTS) in Informatics
with a specialization in Data Science/
Privacy, Information and Cyber Security
30 ECTS
Spring term 2022

Abstract

The steadily increasing amount of malware puts an even larger amount of work required to analyze all the gathered samples. The current methods of analyzing malware come with their downsides such as inefficiency as a manual analysis requires a human or dynamic analysis that could be considered unreliable. The usage of dynamic malware analysis where the malware is executed in a sandbox environment is proven to be an efficient method of analyzing malware. As the techniques used to protect the system evolves, so do the attacking techniques. Some of the malware uses advanced evasion techniques to avoid detection from these sandbox analyzing environments, which causes the malware to be cleared and later executed in a real, target environment. These evasion techniques can find certain artifacts in the system which is inherent to a sandbox environment.

Previous studies mention the lack of transparency between the virtual and physical host to be one of the bigger giveaways for the malware when looking for artifacts. There is also a grey area regarding how the malware acts and behaves, trying to assess and figure out if it is in a sandbox or not. This paper focused on creating a sandboxing analyzing environment within a physical machine, using all the dead giveaways by keeping the system as minimal as possible with only analyzing tools and software, in other words creating a fake sandbox environment.

12 samples of malware were analyzed in the two environments and the results show that the malware interacts more within the physical system and uses different APIs, System calls, and dlls compared to the virtual system. The malware samples, after its running process, resulted in similar activities on both systems which indicated that mimicking a sandbox could be effective to deter evasive malware.

Keywords: Evasive, Malware, malware analysis, comparison, sandbox

Table of Contents

1. Introduction 1
1.2 Scope 2
1.3 Experiment 2
1.4 Results and analysis 2
2. Background 3
2.1 Malware analysis methods 3
2.1.1 Static Malware Analysis 3
2.1.2 Dynamic Malware Analysis 3
2.1.3 Hybrid Malware Analysis 4
2.2 Metamorphic and Polymorphic malware 4
2.3 Malware evasion techniques 4
2.4 Research background 6
2.4.1 Hybrid approaches 6
2.4.2 Understanding the Malware environment 7
2.4.3 Overview 7
3. Problem area 9
3.1 Problem background 9
3.2 Research question 9
3.3 Aim and goals 10
3.4 Contribution 10
3.5 Delimitation 11
4. Method 12
4.1 Malware samples 12
4.1.1 Including criteria 12
4.1.2 Excluding criteria 13
4.2 Implementation 13
4.2.1 Preparation 13
4.3 Comparison 13
5. Environment framework 15
5.1 Experiment prerequisites 15
5.1.1 Experiment 16
5.2 Tools and software 16
5.3 Evasive malware 17
5.4 Samples used 19
5.4.1 UnbanTool 19
5.4.2 Wmiprvse 20
5.4.3 YFGGCVyufgtwfyuTGFWTVFAUYVF 20

5.4.4 Exeinjector 21
5.4.5 UnPackMe 21
5.4.6 Pafish 22
5.4.7 Build 22
5.4.8 Discord_Nitro_Generator 23
5.4.9 VimWorld 23
5.4.10 ”22" 23
5.4.11 Setup 24
5.4.12 ObfuscatedName 24
5.5 Setup 25
5.5.1 Hardware 25
6. Results 26
6.1 Running evasive malware 26
6.1.1 Sample 1 26
6.1.2 Sample 2 27
6.1.3 Sample 3 28
6.1.4 Sample 4 28
6.1.5 Sample 5 29
6.1.6 Sample 6 29
6.1.7 Sample 7 30
6.1.8 Sample 8 31
6.1.9 Sample 9 31
6.1.10 Sample 10 32
6.1.11 Sample 11 32
6.1.12 Sample 12 33
7. Analysis 34
7.1 What do the results show 34
8. Discussion 36
8.1 Software and Utilities 36
8.2 Social and scientific aspects 36
8.3 Ethical aspects 36
8.4 Limitations 37
9. Conclusion 39
9.1 Future work 39
References

1. Introduction

The rapidly growing and increasing amount of malware creates a major issue regarding malware analytics. The manual method of analyzing malware is often considered more reliable compared to dynamic malware analysis. In this case where is it seen as reliable because there is a human analyzing the potential malicious code, but the manual method is time consuming. Time consuming methods with a larger amount of advanced malware create this issue with overwhelming amounts of malware to be analyzed. Common methods of malware analysis are signature-based and behavior based, looking at the malware for any known or common attributes with previously known malware or if the malware behaves similarly to previously known malware (Tahir, 2018). More commonly, organizations use dynamic malware analysis where the suspected malware is executed in a testbed or a sandbox environment while studying it. Sandbox or testbed environments are used to mitigate and limit the damage of the potential malware. By containing the malware with the use of sandbox and testbed environments, the malware will not be able to spread and infect other physical systems (Mehra & Pandey, 2015). This way is more efficient as this is done by automating the process of analyzing malware and does not require an individual to go through the malware code and instead let it execute in the contained environment to determine its actions.

There is an ongoing arms-race as the cyber attacker continues evolving malware to adapt to the current security precautions and avoid detection. The results of this are adaptive malware. The malware is now in such a state where it can detect its environments, be able to detect sandbox and testbed environments and therefore adapt to it by laying dormant, later to be executed in a targeted environment (Colon Osorio, Qiu & Arrot 2015). One of these types is called metamorphic malware and is a major issue for security analysts as the adaptable malware renders the sandbox and testbeds environments somewhat unreliable (Sujyothi & Acharya, 2017). The malware uses evasion techniques to avoid detection. These evasion techniques vary in aggression, so different malware can detect different attributes that the virtualized environments inherit. As the amount of malware needed to be analyzed is increasing, combined with a somewhat unreliable dynamic analysis method, the manual method will be overwhelming. This makes it difficult for a security analyst to pinpoint where the actual issue lies, how the malware understands what environment it is in, and what triggers the adaptable malware's behavior, to lay dormant or active.

1.2 Scope

Considering evasive malware, this study will focus on environments infected with malware that evades dynamic automated malware analysis. The Focus will be on how the same malware behaves in a physical, non-sandbox environment and an identical virtual machine. This study helps to provide a better overview and a more detailed understanding of evasive malware and its different behavior in a sandbox and target environment.

1.3 Experiment

This study will infect a physical stand-alone machine and a virtual system with evasive malware. The targeted machine itself will have several tools similar to a sandbox environment with limited applications and no network access. Both systems will run on Windows 10 Home edition with pre-installed tools suitable for malware analysis. The machines will use an identical setup for each malware run and later extract the output from the tools and be further analyzed and compared.

1.4 Results and analysis

By comparing the results from the physical target machine with the results from a virtual machine (System calls, APIs) the analysis helps future research and security analyst to understand what and how evasive malware acts in a targeted system. This conclusion aims to help and understand the evasion techniques used by malware in a physical environment by using tools and software related to a sandbox. By comparing a virtual and a physical system it is possible to observe the evasive malware take on transparency between the systems, in a system where transparency is not an issue.

2. Background

This chapter will go through the different concepts related to malware and its evasion techniques in virtual sandbox environments. Related works and previous studies will also be discussed, what has already been done, and common techniques used by security analysts today.

2.1 Malware analysis methods

There are different methods for analyzing malware, the more common approaches are static malware analysis and automated dynamic malware analysis. In more recent years, the combination of the human interaction of the static analysis method with the automated analysis from the dynamic results in a hybrid solution. This method is combining the best of the static and dynamic methods into one hybrid solution (Jeon, Jeong, Baek & Jeong, 2022).

2.1.1 Static Malware Analysis

There are different ways to analyze malware but some of the most common methods are static malware analysis and signature-based malware analysis (Sujyothi & Acharya, 2017). These methods require a human to analyze malicious code without actually executing it (Egele, Scholte, Kirda & Kruegel, 2012). If the malicious source code is available, it is possible to analyze the code for memory corruption flaws unlike the correctness of the given system (Chen, Dean, Wagner, 2004; Chen & Wagner, 2002). Static analysis can also be useful to extract information from the malicious code, for example where certain functions in the code itself are located for future analyses (Egele, Szydlowski, Kirda,& Kruegel, 2006). Static analysis could be seen as the more reliable method, where a cyber security analyst examines the code for potential malicious functions. Previously mentioned, the larger amount of malware makes this process slow and less efficient. To speed up the process, dynamic malware analysis is used to speed up the process in an automated way.

2.1.2 Dynamic Malware Analysis

Dynamic malware analysis is an automated process to detect unknown malware by executing the file in a safe sandbox environment to verify its actions (Egele et al., 2012). It is more efficient with a larger amount of malware. Dynamic malware analysis is more prone to miss metamorphic malware while analyzing as it looks at the behavior of the suspected file itself. Dynamic malware analysis uses different approaches to perform its analysis (Egele et al., 2012). These are techniques such as:

- Function call monitoring, such as System calls and APIs
- Information Flow Tracking
- Instructions Trace
- Autostart Extensibility

2.1.3 Hybrid Malware Analysis

Both static and dynamic analysis methods has their drawbacks with low efficiency and potentially get deceived by more advanced malware, a combination of the two results in hybrid malware analysis. The hybrid uses both the best aspects of static malware and the best aspects of the dynamic malware method, without any of its drawbacks. For static malware analysis, the drawback is the need for a manual analysis of the malware. To look at the malicious code and find proof or other indications that the malicious code is malicious could be difficult if the malware author obfuscates the code which makes the code either appear unreadable or not malicious (Jadhav, Vidyarthi & Hemavathy, 2016). The dynamic malware analysis drawback is the automated process. The evasive malware is able to detect when it is being dynamically analyzed and therefore can avoid detection. The idea of a hybrid solution may vary but the idea is to perform dynamic analysis and when anomalies occur, a human requires to perform static analysis to confirm the malware intentions (Jeon et al., 2022).

2.2 Metamorphic and Polymorphic malware

Technology advances for securing and protecting information, but so does the attacks. Less advanced malware is often detected by current systems by static or dynamic analyzing methods. But as mentioned, static malware analysis is a challenge to perform due to the larger amount of suspected files which reduces its efficiency. Dynamic is the current solution for higher efficiency. Dynamic malware analysis is where the suspected file is executed in a safe sandbox environment to quickly determine its actions. Malware is now able to understand where it is being executed and therefore adapt depending if it is in a targeted or sandbox environment. This adaptive malware is called metamorphic and/or polymorphic malware. Christodorescu and Jha (2004) mentioned that common signature-based lacks insight into the malware code itself which is then easily defeated by metamorphic malware. Metamorphic uses code obfuscating techniques to transform how the software or file is represented or even obfuscate the whole code and therefore avoid detection (Zhang & Reeves, 2007). Zmist is a perfect example of both of these. Zmist is an advanced malware that shows a set of Metamorphic and polymorphic coding that includes obfuscating the entry point, randomly using polymorphic decryption, code integration and code permutation (Szor, 2005).

2.3 Malware evasion techniques

For malware to avoid detection by sandbox environments it uses different evasion techniques. A survey conducted by Afianian, Niksefat and Sadeghiyan (2020) mentioned that evasion techniques mainly relied on five different evasive tactics used by malware to avoid execution in sandbox environments.

- Fingerprinting
  Tactic is used by malware to detect sandboxes by looking at familiar environmental artifacts inherent to a sandbox. These artifacts are unique traits of the system such as usernames, files, system settings, active processes and

more (Bulazel & Yener, 2017). Artifacts that might indicate and reveal the purpose of the environment the malware is present in and therefore avoid execution. There is a constant push of new techniques trying to hide artifacts in sandboxes in other words reducing the transparency between the sandbox and the physical machine. As much as the transparency is reduced, there are still some instructions in a system that cannot be virtualized (Xu Chen, Andersen, Mao, Bailey & Nazario, 2008).

- Reverse Turing test
  Reverse Turing test is when a computer or software tries to determine if the movement and activity on the system is a machine or actual human interaction (Borders, Xin Zhao & Prakash, 2006). This tactic is looking for any human interaction in the environment. Automated dynamic malware analysis uses no human or operator to conduct these analyzes. If the malware does not detect any human interaction, such as with BaneChant which awaits for the left-click on the mouse the malware will execute after a set amount of time. Reverse Touring Tests can be grouped into three quite broad categories. Passive observation is when the software is passively looking for common human activities, such as process creations, typing patterns and mouse movements for example. The second category is active interactions with the malware where the user is required (or forced) to interact with a dialog box created by the malware. The last category of reverse touring tests is passive traits, these traits could be meta-data in the applications or files, such as Windows ”recently opened” or by looking at meta-data in the copy-paste clipboard for relevance (Bulazel and Yener, 2017).

- Targeted
  In more advanced and more targeted malware, instead of detecting a sandbox environment, this tactic searches for an intended environment for example Stuxnet which scanned the environment for a specific industrial control system (Afianian et al., 2020) which indicates that the malware has reached its targeted environment. Spearphishing is also used to avoid malware analysis and with certainty execute the code to the targeted recipient. Another method is to encrypt the payload and the required key could be a targeted PC's serial number or other specific environmental settings.

- Stalling
  This tactic utilizes the effectiveness of automated sandbox analyses where each sample has a limited amount of time in the sandbox before being forwarded. The aim of the attacker by using stalling evasion techniques is to delay the execution and therefore the ”activity” of the malware for a long amount of time, enough for the dynamic malware analysis to pass the malware as there is no activity for the analysis to be concerned about (Kolbitsch, Kirda & Kruegel, 2011).

- Trigger-based
  Malware that triggers based on, for example, keystrokes that together could be considered as a user working on an application on a workstation PC. Commonly, malware will require certain triggers to execute. Trigger-based malware can be set off by various trigger types such as time and events in the system and network inputs (Brumley, Hartwig, Liang, Newsome, Song & Yin,

2008). These triggers can be on specific dates or key loggers that and when the user inputs a certain keystroke the malware will execute. This helps to avoid automated dynamic malware analysis as there is no human interaction with the sandbox, which is the idea with trigger-based malware, to only trigger on the common user and its behavior.

- Fileless Malware
  This type of malware is never subjected to a malware analysis environment in the first place. This tactic uses a ”drive-by” tactic that uses vulnerabilities in the system or applications and with these vulnerabilities can inject malicious code directly to the system memory. Fileless malware attacks a system without the need to download a malicious executable, instead, it uses different exploits in web browsers, macros, trusted admin tools, or scripts (Tan, Gao, Shi, Wang, Fang, & Tian, 2019; Mansfield-Devine, 2018). Fileless malware has no limitations on what it can plot to the systems undetected such as reconnaissance, persistence, data theft or execution of the malware (Sudhakar & Kumar, 2020).

2.4 Research background

To get a better understanding of the area of malware and its evasion techniques, it is important to look at previous studies in the same area. This previous research has created themes based on the findings.

2.4.1 Hybrid approaches

Both static and dynamic malware analysis is useable with a wide variety of tools. These more traditional methods do however come with their limitations and issues. Mainly, the static method is easily fooled by more sophisticated malware whereas dynamic malware analysis has a significant performance overhead. Propositions have been made to ease and reduce the transparency between the virtualized environment and the running operating system and its processes. Even if studies have been made to reduce the transparency between the host and the virtualized environment, there is still a negligible overhead and the issue is improved but still exists (Leon, Kiperberg, Leon Zabag, Zaidenberg, 2021). To get a better understanding a study by Colon Osorio et al., (2015) looked further into the area of polymorphic and metamorphic malware techniques as it continues to defeat the effectiveness of static malware analyses. By combining what the authors call the ”best parts” of the two methods, static and dynamic, and limiting the respective disadvantages and utilizing the advantages of the two approaches. This resulted in a hybrid approach called ”segmented sandboxing”, a two-step approach where selected code segments were executed in a secure sandbox environment. It is a somewhat reliable method, combining the two but lacks in efficiency as the code has to be manually segmented before execution. Zhang et al., (2020) created their solution SCARECROW. SCARECROW is a solution that camouflages an analyst environment as an end-user environment trying to work around the issue regarding malware trying to detect its environment which is shown to be efficient. The hybrid solution is proven to be efficient but it is the gap or the grey area in

between the platforms, the virtual and physical that aware of the malware and therefore base their actions.

2.4.2 Understanding the Malware environment

To create better protection a malware analysis must understand how the malware operates, how it enters the system, indications of compromise, and also its actions. Yong Wong et al., (2021) performed a study, interviewing cybersecurity personnel aiming at participants with previous knowledge of malware analysis to better understand the challenges and issues with malware analysis. Besides the issues and challenges, the interview gave an insight into understanding the workflow and expanding new research areas. One of the more interesting areas of issues found was regarding adaptive malware, how it was unclear how to create an appealing environment for the malware to be executed. Based on the results from the participants, both tools from static and dynamic analyzing methods is the more reliable method to detect adaptive malware. Another interesting solution that was found is to analyze the victim's environment, trying to copy the targeted environment into or as a sandbox analyzing environment to create this appealing environment for the malware to be executed in. These sandbox environments, as mentioned before are a common tool to be used in malware analysis as it is considered a secure and controlled environment for the analyst. As previously mentioned, copy a targeted environment as a sandbox environment trying to emulate a scenario where the malware would consider a target and execute. The challenge is to reduce the gap in-between the two physical and virtualized environments and to keep a virtualized environment transparent and concealed from the malware. Khalimov et al., (2019) compared virtualized systems with bare metal ones where the malware can still detect these artifacts or attributes corresponding to each of the two systems. Using a container-based solution with transparency that helps to imitate a ”real” host. The results from their container-based environment did show some positive results by for example hiding network artifacts that could be used by the malware to identify. Switching from a common virtualized environment to a docker-sandbox hide still produces these artifacts or signs of a closed-off sandbox environment for the malware.

2.4.3 Overview

For future work, previous studies mention malware detection in a sandbox and virtualized environments. They continue to discuss that they think that it is possible to create a hypervisor with the only purpose to analyze malware without being detected. The pattern found whilst reading these articles is that malware can adapt to environments and has been for a long time. These adaptations enable the malware to lay dormant in the analyst environment by probing or fingerprinting techniques to later be executed in an end-host environment. There are different solutions out there and most of them try, in one way or another, to replicate the environments by reducing the transparency creating this sort of best of both scenarios. From what this short information gathering has found, there are not a great number of papers figuring out what the malware is looking for in a sandbox but rather trying to ”workaround” instead of trying to deal with the issue itself. The work that has been done related to this

study is older work done back in 2010 (Alsagoff, 2010), and the area in question has evolved a bit since then. If a hypervisor will be created for the only reason to analyze malware, it seems like one of the better solutions is to look at what the malware looks for. Based on a study by Afianian et al., (2020) mention an interesting concept where the main goal of many studies looks into the area of trying to mimic a sandbox environment as a targeted environment. To create this better understanding, Afianian et al., (2020) continue to mention an interesting area of research by comparing a sandbox that mimics a targeted environment with a target environment that is trying to mimic a sandbox environment.

3. Problem area

The overwhelming amount of malware samples to be analyzed forces security analysts to use dynamic malware detection. Dynamic malware detection is the most efficient method of analyzing malware but it comes with a drawback. Malware can detect the autonomous dynamic analysis and avoid detection in the sandbox environment. Studies have been made and one of the bigger issues lifted was the transparency problem between a sandbox and artifacts that cannot be virtualized. The idea is to explore evasive malware in a sandbox environments with relevant tools and comparing a physical system that is not affected by the issue of transparency.

3.1 Problem background

Several analyses have been performed in the last year where researchers concluded that fingerprinting tactic used by malware looks for artifacts in the system. Previous work has been trying to get one step ahead in this arms race between malicious actors and security analysts by reducing the transparency, in other words, reducing the imprint of a sandbox or virtual machines. But there has been a focus on the common analysis environment, a sandbox system mimicking a system that the evasive malware would consider as a target. A study by Afianian et al., (2020) highlights this issue in their survey. To create this better understanding, Afianian et al., (2020) continue to mention an interesting area of research by comparing a sandbox that mimics a targeted environment with a target environment that is trying to mimic a sandbox environment.

3.2 Research question

What is the behavioral difference of evasive malware between a virtualized sandbox mimicking a physical system and a physical machine mimicking a virtualized sandbox system?

As seen in previous studies, there has been a focus on the sandbox environment mimicking a target machine and on how evasive malware acts in such an environment. The question for this study is to look at how sandbox evasive malware acts in a targeted environment where there is full transparency but has the tools and functionality of a sandbox, will it deter the malware to be executed also would similar approaches raise efficiency for future evasion attempt analytics? Will a target environment with artifacts of a sandbox deter the execution of malware, and limit its actions, or does the malware simply look for pre-determined artifacts on which to base its behavior?

3.3 Aim and goals

The goal of this study is to dig deeper into the behavior and thinking process of evasion techniques and look at what steps the evasion techniques follow to determine their actions. This will be done by running different malware samples in both a targeted environment and a virtual one, both with the same hardware, operating system and software later to be compared with each other. By comparing the two systems, the data provided will show if the evasive malware takes the same routes throughout the system, queries the same APIs, and so on. This will also show if a targeted system that is identical to a sandbox can deter the malware compared to the virtual sandbox system. Information that will be evaluated based on the behavior of the evasive malware, to either execute or lay dormant in the environment but also the steps the malware takes from start to finish. The information is the result of installed software, such as virtual hypervisors, debuggers, or other tools used to dynamically analyze malware. In other words, tools and software are expected to be found in a sandbox testbed environment. The question is, based on a previous study, how the malware will behave in an environment, where the system acts like its counterpart which will be an interesting focus for this paper. There have been previous studies trying to work around this issue by fooling the malware with increasing transparency, but not so many articles locate the issue itself which is the main goal of this study. The malware can detect where it is being executed and therefore avoid detection from security analysts. To better understand malware adaptability in environments, two infected systems to monitor the malware in a targeted environment and a sandbox environment. The goal is to observe how the malware behaves regarding, System Calls, APIs, and dlls in an environment that mimics its intended counterpart. Comparing these two The idea is to focus on the physical, targeted environment mimicking a sandbox and running malware samples. Simply looking at the malware in the physical machine will not help to understand the malware as this gives a one-sided view of the malware process. This requires something to compare the physical machine with, and in this case, set a baseline of the evasive malware actions in a sandbox.

3.4 Contribution

The contribution of this paper is to provide information about evasive malware and how it behaves in an environment counter to its intended use. A deeper look into the malware actions with and without the transparency created by virtualized environments will give a good comparative view of the environments and the adaptive malware behavior. This information will help in the future analysis of adaptive malware, on how to tackle the issue but also in the creation of more advanced sandbox testbeds platforms. Besides the platforms, software used to mitigate the execution and running of malware in targeted environments can also use this information for future development protection against adaptive malware. The results from this paper show how fingerprinting evasive techniques are used in a physical environment that is using tools and software similar to a testbed environment compared to a virtual environment.

3.5 Delimitation

This paper will solely focus on the evasive malware differences between a physical machine and a virtual machine with close to identical hardware and software configurations. Malware using evasion techniques to avoid detection in sandbox environments will be used to compare the two systems. This paper will focus on the differences between the two systems and not on how the malware and its evasion techniques necessary works. There will be no focus on the malware samples but instead on how they perform, behaves, or acts in the systems, and what is happening when the process is executed and running in one machine compared to the other. Therefore there are requirements on both systems, the tools, software, and malware to avoid any unnecessary analyzes and unusable information.

4. Method

The study will experiment to be able to compare a physical target system with a virtual system. The experiment method will be the most viable method to conduct this study as there is a somewhat grey area regarding previous studies as many of the previous studies have focused on sandbox and virtualized environments. Previous work has performed experiments, testing malware if different environments or other proprietary solutions. This study will not reinvent the wheel but instead, follow previous experiments and studies and test malware in a contained environment. The study will instead test evasive malware in a physical machine that mimics a sandbox analyzing the environment and comparing it to a virtualized sandbox system. An experiment is also a viable method to get a more detailed and unbiased view of the procedures done and at the same time build upon the current research in the related area. By creating a lab environment, it is possible to use new and modern sets of malware samples but also receive relevant information for this study.

4.1 Malware samples

The malware that is used has some criteria to it avoid any unnecessary analyzes. These criteria will be divided into including and excluding criteria on the malware samples. To ensure that the malware follows the criteria, the malware will be gathered from previous studies focusing on evasive malware. Some of the malware used in previous work will not be included due to their inaccessibility of them or lack of description, which makes it difficult to guarantee the malware uses evasive techniques. The study will focus on detection-dependent evasive malware as these evasive tactics differ in actions based on their environment by performing fingerprinting, reverse Turing tests, and looking for specifically targeted artifacts in systems. The information regarding the malware samples was retrieved from VirusTotal (VirusTotal, 2022) as it is a viable source of information about malware samples (Galloro, Polino, Carminati, Continella & Zanero, 2022). The sample files were retrieved from FileScan which is a large database with reported malware samples. FileScan allows sorting samples with evasive attributes which are necessary for this experiment (FileScan, 2022).

4.1.1 Including criteria

The focus of this study is evasive malware explicit. Evasive malware in this study requires the malware to be able to avoid detection by using one or more evasion techniques mentioned in chapter 2.3 from either virtual machines or hypervisor testbeds. The samples are required to be tagged as evasive and a description of their evasive technique to avoid any irrelevant malware types or evasive techniques. The test environment will mimic a targeted system and as mentioned previously, the most common operating system for a home user is the Microsoft Windows operating system so it is necessary that the malware needs to be executable on a Microsoft Windows platform.

4.1.2 Excluding criteria

The malware targeted for a specific system will not be used either such as Win32Industroyer as its specifically created for Siemens electrical control systems (Cherepanov, 2017) and not the common user personal computers which this study focuses on. Evasive malware of ransomware sort will not be used as this study utilized the live physical machine for information gathering. If the machine would be locked up by ransomware, no tools can be used and no information can be retrieved. Ransomware is a type of malware that will not be used in this study because of that. Malware that is not flagged as malicious will not be used to ensure that the samples used are evasive malware and are not simply suspected as malicious files.

4.2 Implementation

Based on the information from the study by Afianian et al., (2020) regarding fingerprinting (reverse Turing test), an understanding of how the evasive malware fingerprints the system for artifacts. By performing similar tests on this physical test environment, it is possible to answer the research question of how evasive malware performs and acts in a targeted environment mimicking a sandbox. This will be done by using evasive malware samples from the previous study and its results where the malware did not reveal its malicious intentions and run the same samples in a physical environment. By running the same samples in the opposite environment, see if the evasive malware relies on the same APIs and System calls to in this case reveal the malicious payload or if the tools and software will deter it.

4.2.1 Preparation

The physical system was installed with the software and prerequisites mentioned in chapter 4.1. BIOS settings have virtual machine support and XMP activated. The tools and software were installed using a separate external hard drive as the machine itself is never connected to live internet. The virtual machine was installed with identical hardware to the physical machine and neither connected to the internet nor updated. The operating system is not updated after being installed and windows defender is deactivated as the evasive malware samples are of age. These steps are done for each of the malware samples to ensure that the system is the same for each information gathering.

4.3 Comparison

To understand the evasive malware behavior in a physical machine mimicking a sandbox environment it is important to understand the evasive malware in a virtual sandbox environment. The study done by Afianian et al., (2020) highlights the usage of API, system calls, and DLLs made by evasive malware to locate artifacts in the system. The study will use system calls, APIs, DLLs, and events created for comparison in this paper as it focuses on identical malware samples in two different environments. Focusing on detection-dependent eva-

sion techniques, meaning evasion techniques that look for signs or artifacts inherent to a VM-based, Hypervisor-based, or emulation-based environment in a physical environment. Looking at the evasive malware detection-dependent levels by Afianian et al., (2020). Hardware, drivers, and devices are created by virtual machines, in this case, the software that is installed on the physical machine. Execution environment, compared to a sandbox a physical kernel memory space differs for example. The one interesting for this study is behavior. Behavior, in this case, is for example performance differences between a physical and a sandbox system measured by the evasive malware or the imperfect emulation of a CPU in a virtualized environment (Galloro et al., 2022). Network and application will be somewhat of an interest in this study as the machine itself will not be connected to live internet at the same time the samples of malware found do not use a network for indication. The application will be used as one of the systems will run in a virtual machine, this means applications such as virtual tools or other will be present on the system itself. From previous work, there is an understanding of how evasive malware performs in a sandbox environment.

The study will look at the malware behavior in the same three areas, to look for differences in quires made, APIs, and system calls as it is used to leak information about the system environment (Galloro et al., 2022) but also see if the mimicked sandbox environment will help to deter the malware to execute even if it is in a physical machine, in other words, will it expand its activities, create a new process or simply continue to run after its execution. Besides specific files or folders, the paper will also mention obvious differences and behaviors.

5. Environment framework

The lab environment will be based on previous similar studies conducted with the main goal to observe evasive malware in different environments. The paper will perform a comparative study to explore and observe evasive malware in a targeted physical environment and a virtualized sandbox environment. As the main purpose is to study evasive malware and its behavior, the most suitable method is to create a lab environment and analyze the systems infected with evasive malware.

5.1 Experiment prerequisites

The lab environment setup will use two different systems. The first one will be used as an analyzing platform that will perform these tasks:

- Deploy two systems, a physical machine, and a virtual sandbox. The system will install the same operating system and the same tools and software identical to a sandbox analyzing the environment. The systems will later be infected with 12 samples of malware using evasion techniques.

- Use analytic tools to monitor the system
  To gather data and information on the two systems infected, various sets of tools described below will be used to monitor the system. This will gather information about the malware, what System Calls it invokes and what APIs and dlls are being used by the malware. System calls are used by a program to send requests for a service from the kernel which can be used to create new or execute processes or access hardware in the machine itself. Malware uses system calls to operate inside an infected machine (Shobana & Poonkuzhali, 2020). APIs work similarly to as a bridge or an interface between software.

- Receive the results of the monitoring
  The results will be the system calls, API, and DLL which will be compared with a physical and virtual environment running these malware samples. These analyzed differences will be the results of the study. The results will provide a greater overview and a deeper understanding of how adaptable malware can detect and analyze its environment. These differences will tell where the malware looks to determine if it is targeted or in a sandbox environment and how it will behave when in a system and vice versa to the system that the malware tries to avoid. The main goal of this study is to take a deeper look into malware adaptation depending on the environment. Comparing a targeted environment without the issues regarding transparency, with previous analyses and work of test beds analysis will give the results to answer the research question on how complete transparency is affecting the behavior of malware in a testbed environment.

5.1.1 Experiment

This 8 steps is taken for each malware sample on both systems. For a more detailed description on how the two system were setup or how the systems where reset after each iteration, look at chapter about how the systems where setup for the experiment, 5.5 Setup.

1. After installation and snapshots for the virtual, the system was restarted to ensure that all the software was installed successfully. The software and tools used were loaded up.

2. Start up process monitor and setup for the current malware sample.

3. Take a snapshot of the registry before malware is extracted and executed.

4. One of the malware samples is extracted from an external device and later executed with APIMiner. The samples were also executed without APIMiner as some of the samples did not run with APIMiner.

5. While the sample is running in the system, API, system calls, drivers, and DLLs are monitored by the APIMiner and Process monitor.

6. After the sample has been running and the tools do not show anything new regarding API, system calls, drivers, and DLLs, the system is idle for 10 minutes. If something new occurs, another process monitor log is taken.

7. The results from the tools were inserted onto an external drive and extracted. On the virtual system the file where moved between the host machine and the virtual.

8. The machine is then reset to the state before the sample is executed for the next iteration of testing.

5.2 Tools and software

The tools that will be used might variate during the comparison, but to give a transparent overview of the tools regarded as relevant and useful a list will be made. The system will use Microsoft Windows as the operating system as it is the most commonly used, up to 75% by July 2022 (StatCounter, 2022), and will be a suitable operating system to observe. The expected results are to find attributes, such as APIs, system calls, and processes that differentiate from a sandbox environment in other words the malware behavior. The first steps were taken by the malware, in other words, the procedure the malware takes to determine the environment. To clarify the measurements, it will be an analysis, for example, the logs created by each tool. These tools will be used a certain amount of time to get an idea of what system calls or API the malware invokes. This difference will be useful information to analyze the malware procedure to locate its surrounding environment. The tools used will be based on Mohanta, and Saldanha (2020) and Talukder and Talukder's (2020) papers on malware analysis lab setups and analyzing tools.

The criteria of the software or tools used are that they should be free to download and use but also well documented and guides to avoid errors while conducting the study and therefore get inconsistent or false information as result. This list will be updated as other tools might be added or replace current ones. The tools will be tested to see if it performs as mentioned and satisfies the expected results.

- APIminer, a tool used to log APIs in the command line which can run on an analysis machine and is not necessary on the infected device itself. The tool can log every API that is used by any windows executable which will be of great use while doing this research. The results from using this tool will be different logs and traces of the malware in the system. These traces will be used to analyze how the malware moves in the system and what it does with these APIs. The difference in the logs between the system will be used as a result. The logs themselves include information such as processes, and handles.

- Regshot is a tool used to take snapshots of the system, registry which then makes it possible to compare the two registries. This will be a useful tool to snapshot the systems before the execution and after in both the system to later be analyzed. Not only to see what the malware has done but also how the systems compare with each other after the malware execution.

- Process monitor. This tool will follow a given process throughout its run. Looking at queries done, file modifications, API calls, dlls queries, and folder interactions. It shows real-time process and thread activity regarding the given process. In this case, to compare two different actions and behaviors between two systems, this tool is useful as it gives a great overview of what is done in the system by the malware.

5.3 Evasive malware

This chapter is intended to show what some of the queries made by the evasive malware could look like, and what the malware queries for. This information is based on JoeSecurity which analyses evasive malware samples in their sandbox environment. These are lines from the related analysis report done. The different evasion techniques use various sets of APIs and System calls to gather information about the current environment. It is possible to hide some artifacts related to sandbox environments, such as screen size or storage space, but some of the computational artifacts are not as simple to hide without losing efficiency (Yokoyama et al., 2016). According to Afianian et al., (2020) fingerprinting is the most common evasive tactic in both manual and automated analysis by looking for debuggers in the environment the malware is present. Other methods for malware to detect virtual and sandbox environments can be done by querying for physical network adapters and logical disks and lastly queries sensitive information about video cards as shown in figure 1, which is quite uncommon in sandbox environments. (1, JoeSecurity, 2022).

Network adapter:
```
IWbemServices::ExecQuery - root\cimv2: select * from win32_-
NetworkAdapterConfiguration where ipenabled=true
```

Disk:
```
IWbemServices::ExecQuery - root\cimv2: select * from
win32_logicaldisk
```

Video controller / Graphical Processing Unit:
```
IWbemServices::ExecQuery - root\cimv2 : select * from
win32_VideoController
```

Figure 1: Hardware queries example

This would be considered fingerprinting, as the malware tries to poke the system with queries, looking for artifacts that indicate that the environment is virtual. Other indications of a sandbox environment are the malware looking for artifacts related to more common sandbox or virtual environments such as Cuckoo sandbox, VMWare, or VirtualBox as shown in figure 2. This can be done by looking for running or dormant executables (4, JoeSecurity, 2022).

Virtual machine and sandbox artifacts:
`vboxtray.exe vboxservice.exe vboxservice.exe vmwareuser.exe vmwaretray.exe vmtoolsd.exe vmtools.exe` or `Code function: vm-ware qemu qemu vbox`

Figure 2: Virtual and sandbox artifacts queries

This is possible as many sandboxes have these small artifacts located in the system as standard or as a requirement such as the network adapters or video card information.

Some of these artifacts can be modified by the user, such as username or system name which could be an indication of the malware. Targeted malware uses specific lookups on the system to more specifically target a so-called targeted system. For example, this malware looks for strings in the processor's name related to more server-based hardware, in this case, as shown in figure 3 Intel Xeon processors which are not as common in a private user environment. This information about the system can be difficult to manage and modify. Other targeted malware can be specified by the keyboard layout on the system to target a specific computer in a specific location or person (5 JoeSecurity, 2022). Malware specifically looks for systems with the French keyboard layout, one could argue that most analytic environments do not use French keyboard layouts in their analytic environments. This malware example does have some other evasive tactics for sandbox environments but its target is French and therefore will be considered as targeted malware, see figure 3.

Processor unit:
```
Function Chain: Key Value Queried:HKEY_LOCAL_MACHINE\HARDWA-
RE\DESCRIPTION\System\CentralProcessor\0 ProcessorNameString
String check: Xeon.
```

Keyboard layout:
```
GetKeyboardLayout
```

Figure 3: CPU and keyboard layout query

Stalling is also, as previously mentioned, a common evasive method for malware to avoid sandboxes by exploiting the limited analyzing time of the sandbox. An example of this is that the malware creates a function chain (2, JoeSecurity, 2022) Other cases, later when the malware has created its threads and starts an evasive loop that delays the execution of the malware with various amounts of ”sleep” time, see figure 4. (3, JoeSecurity, 2022).

Stalling techniques:
```
GetTickCount - Sleep - GetTickCount

Thread sleep time: -60000s >= -30000s
```

Figure 4: Stalling techniques

5.4 Samples used

Here is a description of the twelve malware samples that were used in the experiment. The description also includes the motivation as to why the evasive malware sample was chosen, how it operates, what artifacts it looks for and what evasion techniques it uses to avoid detection. All the samples fulfill the requirements of the malware samples set by this study. The malware samples are detected in a majority of the anti-malware software vendors.

5.4.1 UnbanTool

UnbanTool is a malicious executable using evasion techniques to avoid detection in virtual environments. The executable is imitating and utilizes a common communication platform called Discord. The malware is using several discovery techniques while in the system. The relevant techniques, virtualization, and sandbox evasion techniques used by the malware:

- Continuously queries and opens the same files over and over
- Checks sensitive information about the network adapter used by the system which is often not virtualized
- Checks sensitive information about the operating system
- Contains longer sleeps and utilizes evasive sleep loops to hinder dynamic analysis

The reason to why this was chosen is that it is used on the common communication platform Discord which is an interesting approach to utilizing its web hooks, APIs and DNS to perform its attack. Moffitt, Karabiyik, Hutchinson, and Yoon (2021) wrote in their article about the state of malware and discord in their article which raised the issue of the attackers' influence on the communication platform which further motivated the use of this sample.

5.4.2 Wmiprvse

Wmiprvse is an executable, remote access tool (RAT) type hidden in the system32 folder. The sample utilizes uncommon combinations of evasion techniques of the found samples:

- Queries for sensitive information about the BIOS
- Queries for sensitive information about the disk used by the system
- Queries for sensitive information about the processor used by the system
- Contains longer sleeps and utilizes evasive sleep loops to hinder dynamic analysis

The malware sample fulfilled all the requirements this study presented. The sample is also flagged as malicious by many of the anti-malware vendors which confirms its maliciousness. From the malware, that fulfilled the requirements, this sample inherent an interesting package of evasive techniques, both that it queries the disk and the processor, hardware which is often hard to emulate in a sandbox environment. This is seen as a perfect example to compare the two systems.

5.4.3 YFGGCVyufgtwfyuTGFWTVFAUYVF

YFGGCVyufgtwfyuTGFWTVFAUYVF is malware with the goal to steal credentials, browser history and FTP credentials that use an interesting evasion technique based on the time zone on the system but also looks for system information related to the operating system, current versions, NETFramework and more. The following evasive techniques used by this sample:

- Collect sensitive information about the network adapter used by the system.
- Collect system hardware print information
- Collect information about the systems time zone
- Contains longer sleeps and utilizes evasive sleep loops based on the time zone information to hinder dynamic analysis

The malware sample fulfilled all the requirements this study presented. The sample is also flagged as malicious by many of the anti-malware vendors which confirms its maliciousness. This sample had the uncommon evasion technique of looking at the system information which would be interesting to compare two identical systems with the same version of the operating system. The comparison would give an interesting result as the sample as the action of the sample as both of the systems, from the malware perspective should be, in theory identical.

5.4.4 Exeinjector

Exeinjector is a trojan with the goal to steal credentials, browser history, browser hijacking, stored passwords and mail stored on the system. The evasion techniques used by this sample check systems information on a more hardware level. Information such as computer name, serial number, MAC-address and more. The evasion techniques used by the sample:

- Query the system for its name (computer and system name)
- Query the system for its serial number
- Query the system for its MAC-address
- Continuously queries and opens the same files over and over
- Utilizes evasive sleep loops to hinder dynamic analysis

The malware sample fulfilled all the requirements this study presented. The sample is also flagged as malicious by many of the anti-malware vendors which confirms its maliciousness. The sample uses system information, such as name, serial number and MAC-address. Sandbox and virtualized environments often have a limited set of MAC-addresses and serial numbers that are used in the environment. The comparison between the two systems, one using a common vendor serial and MAC-address and the other system using emulated serial and MAC-addresses will be an interesting addition to this study.

5.4.5 UnPackMe

UnPackMe uses various sets of system checks to avoid detection while in a system. This evasive malware checks queries for the serial number of the system but also checks the video controller. Video controllers, such as graphical processing units are difficult to emulate in a virtualized environment. These are the evasion tactics used by the malware:

- Queries the system for its serial number
- Queries the system for its video controller
- Queries the operating system for its serial number

This malware is of interest as its scans the system's hardware for its serial number and also the operating system serial number to ensure that the environment is not emulated and that the operating system is activated and within the scope of the malware. As mentioned, hardware is difficult to emulate (Jadhav et al., (2016), and video controllers or more common graphics cards will be emulated by the sandbox. This creates an opportunity for the malware to check if there is a video controller available and if so, what kind of video controller.

5.4.6 Pafish

Pafish is a software that could be considered more as a tool than malware, still, it is still classified as very intruding malicious software. Pafish uses almost all of the current evasive tactics that malware would use to avoid detection. These are the tactics used:

- Detects human interaction with the system
- Queries the system for plug-and-play device information
- Queries the system for disk information
- Queries the system for BIOS information
- Performs registry checks to detect sandbox / dynamic analysis environments
- Queries the system for installed software inherent to virtualized environments. For example VMtools or Virtual Box Guest additions.
- Queries the system for its network adapter information
- Queries the system for processor information
- Queries the system for video controller information
- Queries the system for the system serial number

This sample is perfect for this kind of experiment as its thorough with its environment scanning and detection. This also helps to set a good baseline when comparing the two systems. Even if the sample itself could be considered a tool for this kind of analysis, it is still a helpful and informative tool to use.

5.4.7 Build

Build is an evasive malware that focuses more on the hardware of the system it is in. It combines common evasive tactics to detect different hardware parts from the system. These are the parts of the hardware the malware looks for:

- Queries sensitive video device information
- Queries sensitive processor information
- Queries processor information
- Queries sensitive operating system information
- Queries the current volume of sensitive information

The two systems differ the most when looking at the hardware installed. The physical machine has nothing in between whereas the virtualized environment has the hypervisor separating, in this case, the malware from the physical hardware. This makes this sample a good candidate to examine and compare between these two environments as in theory, the results would be very different and not be affected by installed software.

5.4.8 Discord_Nitro_Generator

Discord_Nitro_Generator is similar to the UnBanTool sample where it utilizes the communication platform Discord to spread. Compared to UnBanTool, this sample uses evasion techniques to detect hardware used by the system. These are the evasion techniques:

- Queries sensitive information about the physical memory
- Queries information about the memory
- Queries sensitive information about video controllers
- Queries sensitive information about the processor
- Checks the disk volume and free space.

As previously mentioned, the reoccurring platform Discord is used to spread its malicious software. The use of Discord and the use of the physical memory in the machine and the system's disk space are two oddities looking at the samples which will be a good addition to the experiment.

5.4.9 VimWorld

VimWorld, is a malware that focuses on fingerprinting the system for software and tools inherent to hypervisors. It also checks for video controllers and the running operating system. These are the artifacts the malware looks for to evade detection:

- Queries for sensitive information about the operating system
- Queries for sensitive information about the video controller
- Queries the system for software inherent to hypervisors
- Queries the system for tools inherent to hypervisors

The sample checks the video controller, similar to previous samples but also checks the operating system for installed files and names. These files and names are related to three different hypervisors and sandbox software, QEMU, VMware, and VirtualBox. Besides names and files, the malware continues to search for tools and software that are installed by these sandboxes, looking for tools and guest additions from VMware and VirtualBox which are used to create a better experience while in a sandbox environment.

5.4.10 ”22"

22 is quite similar to previous samples with some exceptions. The sample looks for network adapters, video controllers, and installed tools related to sandboxes but more interestingly it has a focus on the disks installed and used in the system. These are the artifacts the sample looks for:

- Queries for sensitive information about the video controller
- Queries for sensitive information about the operating system
- Continuously queries and opens the same files over and over
- Checks interactions with the malware process

- Queries for sensitive information about the disk
- Queries for sensitive information about the volume

These samples focus on the disk and volume results in the malware scanning for the disk serial number, name, and its allocated letter which is required to be C. Instead, as shown in previous samples, this malware does a detailed look at the disk installed and used. As for many parts of hardware, the disk is difficult to emulate in a virtual environment. The size of the volume, if it is dynamic by running and creating new files over and over, the name, serial number, and more are the focus of this sample. This creates an interesting addition to compare a physical and emulated disk, with the same volume size, name, letter, and free space.

5.4.11 Setup

Setup uses evasive techniques to look at the hardware and operating system, similar to the previous samples. The samples also look for specific strings, files, DLLs, and processes related Here are the different artifacts the samples look for:

- Specific string dll, sbiedll.dll
- Specific string related to VMware and VirtualBox
- Queries the system for information about the computer system
- Queries the network adapter for configuration information

To have a sample not simply look for the network adapter but instead looks at how the network adapter is configured, for example, DNS and subnet. The sample also scans for specific strings and dlls in the system to decide its actions. The sample is of interest as both the system have identical network adapter configurations besides that one is emulated and the other is a physical one.

5.4.12 ObfuscatedName

ObfuscatedName utilizes NET framework to execute its code. The sample uses similar evasion techniques to fingerprint the operating system, and network adapter and creates and opens files continuously. These are the artifacts and defensive techniques used:

- Queries the system for sensitive operating system information
- Queries the network adapter for sensitive information
- Continuously creates and opens new files

This is another sample, one of two that uses somewhat of an overflow with new files and continuously opens them which is an indication that it tries to detect sandbox environments. This combined with the hardware fingerprinting, operating system and network adapter creates an interesting sample to compare between the systems.

5.5 Setup

The goal is to create and analyze an environment that would for the malware to be considered as a target machine. To create an environment that is a targeted machine trying to mimic a sandbox it is necessary to explicitly explain and in detail describe the environment used. The system itself will be running Microsoft Windows 10 Home edition as an operating system, not updated and without any anti-malware software. This is the basis of the system, as the idea is to create the ”common user environment” where Windows 10 Home is more focused on the home user or consumer (Windows Experience Blog, 2015). Besides the basics of the target system, the idea is for the machine to mimic a sandbox environment, it is therefore important to understand what indicates a sandbox.

All software will be installed with standard settings, no other configurations were made while installing or running the software and tools. Microsoft was installed with standard settings and no alterations to keep it as simple as possible and true to a common user. The tools and software were located on an external hard drive that is installed on the machine. The results from these tools are then stored on the same external hard drive for further analysis. After each run of the evasive malware, two different approaches were taken for the two systems. The physical machine was reinstalled with an external drive as installing media. The disk was formatted before each installation to ensure it is as clean and identical volume sizes as it first where, after each testing iteration. The virtual sandbox system used snapshots of the system before the malware was present on the system. This method of formatting and reinstalling the operating system does not ensure that

5.5.1 Hardware

The system is running an older 2-core Intel process, 8GB of memory, and a hard drive with 900gGB of usable space after the installed operating system. The virtual machine is indented to be as identical as possible with 2 core virtual processors, 8GB of memory, and 900GB of useable volume space.

6. Results

These are the results from the experiment by running the malware samples in the physical machine and the virtual sandbox. The information gathered is from two close to identical systems running the same software and malware samples and later compared. For each of the malware samples, an indication of compromise, if it was executed, the amount of created files, created threads, and total events are identified. Execution is considered a step that the malware application after it ran, in other words, did the malware code do anything with the system? This helps to give an overview of how the two systems compared to each other. The sample evasion techniques are also mentioned and also some of the key differences between the systems while the malware process was running.

6.1 Running evasive malware

The 12 malware samples were identified based on the requirements and gathered from the information in chapter 5.4. These samples fulfill the criteria and requirements. To avoid any identification of the malware, the samples will be identified as Samples 1-12 while running instead of their respective name.

6.1.1 Sample 1

This sample did not create any new running processes after its execution on either of the two infected systems. From a user's point of view, there was no immediate sign of infection in either of the systems. Looking at the output from the process monitor and APIMiner there are some differences. Between the two systems, the physical machine had more events done by the malware. Whilst the virtual machine had 917 events, the physical machine had a total of 1587 events. It is possible to find early in the process where the virtual machine fails to find two WMI (Windows Management Instrumentation) security keys compared to the physical machine. The bigger difference between the systems for the monitoring software is that the virtual machine already has the DLLs the malware tries to access whereas the physical machine needs to create the required DLLs. In the last step of the malware process, it looks for a wow64 DLL responsible for emulating 32-bit software in the virtual machine and shortly thereafter sends out an error message. In the physical machine, the malware quires NETFramework, PowerShell, PowerShell utility, Microsoft open authentication, ”friendly URLs” and various SignalR. The malware queries for different NETFramework policies, if it does not exist, the malware creates these files with reading access all over the main drive and later on does the same procedure in assembly instead of NETFramework. The malware did execute both by running APIminer and without it, the results were not affected by the tools of the system based on the events created by the malware. As mentioned earlier, no new processes were created in the process. Based on the error message displayed after running the malware, the reason why the malware did not execute as intended was because of the NETFramework version installed on the systems.

Table 1: Results from sample 1

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	336	5	3683
Virtual	No	No	170	5	1716

6.1.2 Sample 2

This sample, sample 2 did execute in both of the machines but still failed to create new processes. This sample did not show any indication of compromise from a users point of view except for the last few seconds on the virtual machine where it explicitly said that the software cannot run in an emulated or debugging environment, whereas in the physical machine the application ran and ended itself without any required interaction from the user. The malware is using two specific strings called VirtualProtect and WritePrivateProfileSection which would be regarded as fingerprinting evasive techniques which is the reason for the error. The process compared to the two systems is similar to each other, both systems load the dlls used by the malware. There are some anomalies where the malware queries for wow64 dlls but at the same step in the process, the malware looks at windows locale settings, and hard drive volume. Like the previous sample, queries for dlls do not exist in the physical machine and therefore are required to create it whereas in the virtual the malware only queries for the dll and then closes the file. The malware utilized the paging I/O substantially more in the virtual machine compared to the physical machine. Paging I/O and I/O flags are often used in the same line as the wow64 folder that emulates 32-bit software. For example, when the malware reads in the system32 folder on the virtual machine, the physical machine malware is in the WindowsNT folder disabling buffer size checks, and disabling meta files. In the end, the file on the virtual machine is closed at the desktop, which would explain the error message ending up on the screen but also once again the last step it takes is closing a file in the wow64 folder. On the physical machine on the other hand the desktop file is never closed but instead closes the temporary folder.

Table 2: Results from sample 2

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	304	4	2536
Virtual	No	No	132	2	963

6.1.3 Sample 3

Sample 3 utilizes the hardware differences to locate the environment it is in. This is done by looking and video controllers, system information, and logical. This is the first sample where it was obvious that the system was compromised. For the virtual machine the malware tried to open and access the connected Microsoft account, mail, and OneDrive, at the same time it disabled one of the modules in Windows Security. Besides these steps, a process was started and suspended. For the virtual machine, no process and Windows security were unmodified but a prompt for installing FileZilla windows 32 setup was without any follow-up. The process starts in a similar pattern for the two systems. The malware sent queries about the system regarding hardware, such as the number of processors available in the system but also the family and model of the processor. This is the first malware sample that acknowledges the tools and software present in the systems. APIMiner did not successfully execute the malware application and therefore was not usable for this sample. The malware also noticed the installed virtual machine software on the physical machine. The larger difference is done with reg query values and keys where the malware queries for TCP/IP parameters, hostnames, DNS caches, windows sockets (WinSock2) adapter names, interfaces, and more. This list contains about 100 different queries done by the malware in DNS and TCP/IP folders. None of these queries was performed on the virtual machine which has already closed the file, this resulted in about 2000 more events for the physical machine. Both machines had actions done by the malware, even though the systems are identical besides the fact that one is virtual and the other is not, the same sample of malware resulted in two different outcomes.

Table 3: Results from sample 3

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	Yes	Yes	460	9	5213
Virtual	No	Yes	405	7	4585

6.1.4 Sample 4

This sample did not execute in any of the environments and looking at the events created by the malware process, the reason is the older version of NET-Framework installed on the systems. This is shown by the queries for the framework files and folders that do not exist. Comparing the two machines, the difference is minimal except in two places where the physical machine receives more events from the malware. These events, about 100 of them started with a query about standard information about an image in assembly. They thereafter created files in assembly with reading write and delete attributes where the virtual machine gets queried with its security file in assembly twice and then closes the file. Between the two systems, there is not that big of a difference between the systems regarding the number of events. Neither of the systems received a new process by running the application.

Table 4: Results from sample 4

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	414	5	4207
Virtual	No	No	425	11	4065

6.1.5 Sample 5

The sample did run in both environments, without any indication of compromise. Neither of the systems received any new processes by running the malware. 70 events into the running malware the virtual machine creates a new thread and loaded some dlls from sysWOW64 with success. While that is occurring on the virtual machine, the malware created new dlls, queries information about the file, closed in, and created new ones. These dlls were kernel32 dll, ntdll, kernel base, and more whereas the virtual machine loaded dlls such as user32, win32u dlls. From what is shown is that the physical machine creates these dlls, creates file mappings, and accesses them when the virtual machine simply loads in dlls, dlls that are not present in the physical system. The two systems create new files back and forth and in the end even out. By looking at the events, the malware process tries to identify the version of the running operating system, querying the current version of the installed language packs.

Table 5: Results from sample 5

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	229	5	1555
Virtual	Yes	No	174	3	1192

6.1.6 Sample 6

Sample 6 fingerprints for several artifacts in a system. These artifacts are disks, drivers, processors amount, network adapters, and general system product information such as the manufacturer. Besides the artifacts, it also fingerprints for Windows defender presence. This malware sample did run in both environments but only without the use of APIMiner which was not used in this run of the sample. Both the systems ran the malware process in fewer amounts of events compared to other samples with both processes ending up under 500 events. Both systems regarding the events are similar to each other. There are some oddities where the physical machine reads for specific mscoreei dll in the NETFramework folder with success. This dll is the main reason for these oddities in the comparison. Besides the dll, another interesting difference between the systems is that the virtual machine catches modifications, such as creating file mapping whereas the physical machine does not. Both systems received a pop-up mentioning updating the NET framework. The virtual machine did also

receive a new process where the application ran called WMI provider hosts that were suspended by windows.

Table 6: Results from sample 6

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	Yes	Yes	76	3	483
Virtual	Yes	Yes	76	2	449

6.1.7 Sample 7

This sample of malware did run on both environments but only without API-miner which therefore was not used in this sample. Immediately by running the processes on the machine a pop-up of the command line shows as it's failing to connect to a remote server, which is explained by the larger amount of IP-related queries made before the window is closed. This happens on both systems for a brief moment. On the virtual machine, a new process was created which never was suspended or marked as malicious. The events show that the physical system reads and queries two DLLs mscoree and mscoreei dll several times the virtual system simply loads the dll without any reading. This, and a large number of events reading a CLR DLL in the physical system. The physical machine created and read a larger amount of files in the windows assembly folder compared to the virtual machine. The malware also reads the TCP/IP services on the physical machine and goes through each interface on the system. This is done on both IPv4 and IPv6 interfaces. This sample also shows that the physical machine does not cache changes done while the virtual machine does. Besides the dlls, in the physical machine, the malware did locate the installation path and folders related to virtualization software but not in the virtual machine.

Table 7: Results from sample 7

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	Yes	553	23	8545
Virtual	Yes	No	487	22	5311

6.1.8 Sample 8

The sample did run in both environments and both systems received a new process that was suspended by windows. From a user's point of view, there is an interesting pop-up in the virtual machine, a command line window mentioning that the process has failed and resulted in ”Error in Anti VM” and shortly closed. Compared to the physical machine, this is the only noticeable difference between the systems without looking at the events. The events show the opposite as the two systems show a bigger difference. The main dlls causing the larger amount of events are clr, clr0400, mscoreei, and clrjit dlls. These dlls are simply loaded or do not even exist in the virtual machine whereas in the physical these dlls are read over and over several times. This is also the first malware sample that queries the system and hardware for specific tools and software. At event 1359 for the physical and line 699 on the virtual, the malware process starts querying Scsi ports and busses but then more interestingly it queries for virtual device drivers and disks and storage units using VMware-related names. The malware process continues to search for VMware Tools and VirtualBox Guest additions, where the malware first identifies the indication of a sandbox. The fingerprinting process continues and the malware identifies the string VBOX in the disk name and while a query for more hardware descriptions about VideoBiosVersion which the physical machine does not respond with anything compared to the virtual machine that responds with Oracle VM VirtualBox and its current version. This is the last step the malware takes to identify the environment it is present. The next event is the physical machine modifying the assembly and native images and TCP/IP interfaces, and once again it does all this without caching. The software did not run with APIMiner.

Table 8: Results from sample 8

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	Yes	428	18	7477
Virtual	Yes	Yes	387	15	4660

6.1.9 Sample 9

Sample 9 looks for installed guest additions and Vmware Tools in the system. The sample provides the least different outcome. Neither of the systems received any new processes nor showed any indication of compromise by the malware. Where the few differences reside familiar dlls, such as mscroee and clrjit compared to clr between the systems. An interesting note is that only on the physical machine is hardware and system name information queried. On the physical machine a lot of information regarding the cryptography is performed, for example, cryptographic provider and configuration, whereas on the virtual machine it is not existing. A similar difference in queries could be made whilst the malware queries the physical system for TCP/IP parameters and interfaces where the virtual machine only gets one interface queried.

Table 9: Results from sample 9

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	375	11	3766
Virtual	No	No	290	9	3028

6.1.10 Sample 10

Sample 10 uses fingerprinting of hardware on the system. This sample did not work with APIMiner and therefore could not be used with this sample. Once again, the differences are again related to clra and clrjit dlls and native images in assembly. Another interesting note about the physical machine is the number of XML files used, especially the specific file called system.xml. In length, both processes in the system had a similar amount of threads compared to the other samples tested.

Table 10: Results from sample 10

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	Yes	284	5	1933
Virtual	No	Yes	268	4	1895

6.1.11 Sample 11

Sample 11 used fingerprinting technique to identify system information, such as hostnames and FTP connections. This is the first sample where the process from the malware application was close to identical to each of the systems. The five lines that make the difference in total events are queries regarding the current version of Windows in the virtual machine. The sample ran in both environments and neither of the systems received a new process.

Table 11: Results from sample 11

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	No	No	144	3	1754
Virtual	No	No	144	3	1759

6.1.12 Sample 12

The last sample, sample 12, did both execute and created a new process for the machine. A pop-up was present for both systems. This sample found the presence of virtualization software on both the systems but also performed a reverse touring test locating user input to the system. The differences vary, whereas there is no obvious place of indifference. The sample starts with a deeper look into the previously mentioned Wow64 folder and queries security files related to kernel32 dlls and continuous queries the virtual system in the explorer folder and current control sets. At the same time, the physical machine gets its syswow64 folder queried eight times on dlls not relevant in the virtual system. The software continues to query installed languages which are done later on in the physical system. Another oddity related to the physical machine is the two-step query for identifying the hostname whereas it is performed in one and more compact query on the virtual. There are some points where it is possible to observe what steps are related to the process discovery, for example when both systems are queried about their current version of OOBE, while on the physical system the same results are found in the local/ temp folder. Another way that the virtual machine was exposed is the number of processors used, which was the same amount as the physical machine. The sample did also query SystemBiosVersion where the virtual machine VBOX whereas the physical responded with LENOVO. This is the last step the virtual machine does before the process ends at the same time the physical machine continues getting buried in its Core UI components, user settings, and the current version of compatibility32.

Table 12: Results from sample 12

System	Indication of compromise	Executed	Created files	Created Threads	Total events
Physical	Yes	Yes	587	8	6537
Virtual	Yes	No	343	6	4585

7. Analysis

Twelve samples of malware were tested in both environments. All the malware samples used evasion techniques to avoid being executed in a virtual environment. The testing showed that there is a difference between the two systems by running all malware samples. These differences vary in the amount where some sample runs are more evident than others regarding the differences. This is somewhat expected as it is close if not impossible to create a complete identical psychical environment in a sandbox environment, but as this study focuses on what the malware process does when it is running in a system, it gives an understanding of what it does and behaves in the system in question. While running the samples in the machine, there were some reoccurring events between the malware. On the virtual machine, a dll called wow64 was queried more compared to the physical system in the majority of the malware samples used. Wow64 is used by 64-bit operating systems to run 32-bit applications in windows. It is common for malware to run in 32-bit for this same reason, it helps the malware to expand its attack surface as it can be run in both environments.

Other dlls, clr, clrjit, mscoreei, and mscoree were also where there were differences when the virtual machine simply loaded these dlls as the physical machine often reiterated the loads and queries for these specific dlls. Both of the machines were running a 64-bit operating system during the observation. On the physical machine, there were more events regarding windows assembly in many cases. Assembly is used by NET applications which were commonly seen in some samples where the malware prompted about outdated or missing NET Framework files. As both machines in some samples failed to run NET framework applications it should be that the malware at least queries for the same files in assembly but these events related to assembly were in the majority in the physical system. The last common finding whilst comparing the systems is that many of the queries done to different folders or dlls are never cached by the physical machine. Events that are somewhat identical between the systems are more than often non-cached by physical machines, these are events that were specifically mentioned to be non-cached by the system. In the virtual machine, however, non-caching was existent but in a much lower volume and never non-cached without the physical machine doing the same.

7.1 What do the results show

By looking at the results, it is possible to determine that the physical machine in 10 out of 11 cases has more actions performed. This means that the physical machine had more events in total, created more files and created more threads than the compared environment Even if the physical machine had more events done by the malware, in a majority of the samples run the two machines performed in the same manner regarding the indication of compromise and created processes after the execution. By looking at the events created, it is also possible to observe, in most cases, the differences in the number of events. The events show where the malware process loads a dll and closes the file. In the physical machine, the same dll loads but often reiterates the loading and requires more dlls to load or be created before it closes the file. Almost as if the

physical version is missing the required components which are created when it is queried while the virtual system simply loads the queried dll. This should not be an operating system issue as both systems have identical installation files and neither of them has been connected to live internet. The difference is the hardware, that one is a physical machine while the other is a virtual machine without any physical components. The results help answer the research question as it shows that there is a difference between the two systems and that malware takes different approaches to perform the same actions in the system. In many of these cases, different approaches either fail to create a new process or expand their actions throughout the system. This could also be because of the setup environment running the samples and why the samples did behave, such as an older version of the Windows operating system.

8. Discussion

This part of the paper discusses some parts of the study about some of the decisions made while performing these malware runs. Limitations are also discussed in the lab setup, tools, software used in malware samples and also the different aspects of the study, ethical, social and scientific aspects.

8.1 Software and Utilities

Mentioned in the paper are tools and software that were used in the physical environment to capture information about evasive malware. In the end, three software were used in the experiment phase. The limitation of tool usage was due to different factors such as the malware samples used but also the usage of the tools themselves. Some of the tools were in a state where they could not be considered finished and where necessary setups were required, in other words where misinterpretation and misconfiguration of the software could reflect on the results. Therefore, tools that required any installation without any clear guide were not used to avoid faulty usage of the tool. Other tools, such as Wireshark and the fake net were not used due to the samples of malware used. These malware samples did not use network reconnaissance as their evasion technique and therefore would render the usage of these tools somewhat useless for this experiment.

8.2 Social and scientific aspects

From a social perspective, this could provide insight, raise awareness of this evasive malware, and provide some helpful protection against it. As the results show, installing tools related to virtual or sandbox systems will likely help deter some malware that can be used both for home users and enterprise solutions. One could argue that this could contribute to the ninth goal of the UN regarding building resilient infrastructure. For scientific purposes, this study will provide information regarding evasive malware and how it behaves and acts in different environments. This information can be used for further research on evasive malware, and research on the creation of new sandbox analyzing environments. This paper is intended to add to the current studies and articles trying to better understand evasive malware with a focus on the environments with the hopes of getting one step closer to lacking defenses.

8.3 Ethical aspects

As the study revolves around analyzing malware in a physical environment, it is important to ensure that the information in this paper can and will not be used for any malicious actions. The information provided is used for future research on the topic of evasive malware in sandbox environments. The information is about how the malware behaves in the physical and virtual environment with tools similar used in a sandbox. To ensure the validity of the paper, as much information is necessary to answer the research question but to avoid informa-

tion that could help malicious actors improve their malware. The paper is focused on information about how the malware interacts with the systems and how the malware process is different between the physical and virtual, there will be no focus on how the malware works in detail. As the malicious actors most likely understand their software, but not necessarily the target environment or the fine details about a sandbox environment. It was important to avoid any form of information that could help to exploit or work around the current sandbox environments During the whole process of the study it was important to limit the amount of irrelevant information and limit it to the research question. The malware samples were downloaded and extracted to an external storage device where the machine running these samples never connected to the live internet. This is a precaution done to avoid any form of spreading of the malware samples used and causing infection of devices outside of the testing environment.

8.4 Limitations

The main focus of this study was to observe how evasive malware tackled a targeted and a sandbox environment to observe the malware behavior in these two systems. This paper uses somewhat a previous study by Afianian et al., (2020), trying to figure out if a targeted environment mimicking a sandbox would deter the malware, which in this case it did. Still, using previous studies creates a limitation on what to focus on. Another limitation was the time to complete this study as 12 samples of malware were used, whereas more time on the paper would provide more samples and therefore more detailed work. The study also focuses on a home user environment using Microsoft Windows 10 Home with a set of tools as software. The software and tools could also be argued to be limited as mentioned. The samples of malware and the usability of the tools used for gathering information are free software and vary in their state of completion. Some of the tools are in a non-finished state whereas some tools are in a complete state and some tools did not work as described and are therefore never used. Due to the time limit, a more in-depth look into the system by using software that takes an image of the system and looks throughout the whole system for indications of malware or evasive fingerprinting.

This section is for malware using evasive tactics that fingerprint the environment for sandbox artifacts related to a specific system or performs reverse Turing tests to detect human interaction. The main purpose is to see how evasive malware interacts and behaves in a targeted environment mimicking a sandbox. Therefore only detection-dependent evasive techniques used as stalling or trigger based are not affected by their environment. To use these types of evasive techniques in this study will not be utilized because the malware would work in the same way independently where it is executed and therefore not help to answer the research question. There are mainly two challenges with the usage of evasive malware. The first reason is the challenge of finding working evasive malware samples that are useable for the study and the second reason is the lack of information surrounding it and therefore hard to guarantee that fulfills the criteria. The stages for retrieving malware were to ensure that it is evasive, find a sample for download, ensure that the file is the malware in question, and also make sure that the file is intact and working. If the malware sample fails in

one of these steps, it cannot be used for this study. The study does not consider malware operating in BIOS. The study did a thorough investigation of the malware samples and based on the samples used, no sample utilizes BIOS when the system is formatted and reinstalled. Because of the lack of information regarding malware samples staying in the malware, further actions were done to prevent or mitigate this.

9. Conclusion

The goal of the paper is to locate and observe the differences that occur between when evasive malware is acting in a physical and a virtual environment. This has been done by looking at the malware process while it is running, gathering APIs, files created, threads created, and more to get an understanding of how the evasive malware samples deploy their content and get running in the two systems, a physical and virtual system. The results had some key differences, the systems loaded and queried mainly clrjit, clr, mscoreei, mscoree, and wow64 are some of the largely utilized dlls, created more files, created more threads, and cached queries which led to a larger amount of events. As more events were created, in 11 out of 12 samples in the physical machine, the running of the malware in the systems remained alike as there are very few samples where only one of the systems is compromised. Samples that occurred differently from each other were often command-line pop-ups that presented information and error messages or in some few cases, only one of the systems received a new running process related to the malware. By analyzing the results it is possible to answer the research question: how does complete transparency is affecting the behavior of malware in a testbed environment and will a target environment with artifacts of a sandbox deter the execution of malware, limit its actions, or does the malware simply look for pre-determined artifacts to base its behavior? These malware samples have shown that by taking 12 different samples of malware, malware using evasion techniques and running them in two separate environments close to identical to each other does result in different behavior and approaches by the malware. These different approaches and behaviors do not necessarily imply that the evasive malware will perform as intended because of the different behavior but acts similarly. This could indicate that a target system using software and tools used by analyzing the environment deterred the malware from actually expanding its actions and ” doing some damage”. There are only a few cases where the malware process looks for pre-determined artifacts but is solely related to third-party software, such as VirtualBox. From the results, it is not possible to determine any artifacts from the operating system that helps the malware identify the environment.

9.1 Future work

To continue this work, more samples of malware could be useful to get an even better overview and more information to either strengthen this work or improve upon it. More samples would provide more useful information but also help discover more trends related to each machine or more information related to the malware. As cybercrime and malware keep evolving, there will always be newer, more advanced evasive malware samples that could be used in the same or similar environment setups. As for the setups, the current environment could be improved upon, with newer or more advanced software and tools used to analyze the malware or a more recent version of the running operating system. There could also be another approach done, instead of taking a larger amount of samples as in this study, take a lesser amount and look in-depth into the infected system, for example by taking memory dumps and examining it in detail.

References

Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2020). Malware dynamic analysis evasion techniques. *ACM Computing Surveys*, *52*(6), 1–28. https://doi.org/10.1145/3365001

Alsagoff, S. N. (2010). Malware self protection mechanism issues in conducting malware behaviour analysis in a virtual environment as compared to a real environment. *2010 International Symposium on Information Technology*. https://doi.org/10.1109/itsim.2010.5561600

Borders, K., Xin Zhao, & Prakash, A. (2006). Siren: Catching evasive malware. *2006 IEEE Symposium on Security and Privacy (S&P'06)*. https://doi.org/10.1109/sp.2006.37

Bulazel, A., & Yener, B. (2017). A survey on automated dynamic malware analysis evasion and counter-evasion. *Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium on - ROOTS*. https://doi.org/10.1145/3150376.3150378

Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., & Yin, H. (2008). Automatically identifying trigger-based behavior in malware. *Botnet Detection*, 65–88. https://doi.org/10.1007/978-0-387-68768-1_4

CHEN, H., DEAN, D., AND WAGNER, D. 2004. Model Checking One Million Lines of C Code. In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04).

CHEN, H. AND WAGNER, D. 2002. MOPS: an infrastructure for examining security properties of software. In Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS). 235–244.

Cherepanov, A. (2017). *Win32/industroyer - welivesecurity*. WIN32/INDUSTROYER A new threat for industrial control systems. Retrieved March 30, 2022, from https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf

Christodorescu, M., & Jha, S. (2004). Testing malware detectors. Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA '04. https://doi.org/10.1145/1007512.1007518

Colon Osorio, F. C., Qiu, H., & Arrott, A. (2015). Segmented sandboxing - A novel approach to malware polymorphism detection. *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. https://doi.org/10.1109/malware.2015.7413685

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on Automated Dynamic malware-analysis techniques and Tools. *ACM Computing Surveys*, *44*(2), 1–42. https://doi.org/10.1145/2089125.2089126

Egele, M., Szydlowski, M., Kirda, E., & Kruegel, C. (2006). Using static program analysis to aid intrusion detection. *Detection of Intrusions and Malware & Vulnerability Assessment*, 17–36. https://doi.org/10.1007/11790754_2

FileScan. FileScan.IO - next-gen malware analysis platform. (2022). Retrieved October 4, 2022, from https://www.filescan.io/scan

Galloro, N., Polino, M., Carminati, M., Continella, A., & Zanero, S. (2022). A systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security*, *113*, 102550. https://doi.org/10.1016/j.cose.2021.102550

Jadhav, A., Vidyarthi, D., & Hemavathy M. (2016). Evolution of evasive malw ares: A survey. *2016 International Conference On Computational Te chniques In Information And Communication Technologies (IC CTICT)*. doi: 10.1109/icctict.2016.7514657

Jeon, J., Jeong, B., Baek, S., & Jeong, Y.-S. (2022). Hybrid malware detection based on Bi-LSTM and SPP-net for smart IOT. *IEEE Transactions on Industrial Informatics*, *18*(7), 4830–4837. https://doi.org/10.1109/tii.2021.3119778

1 JoeSecurity. (2022). *Analysis report pQlSDfwyYkf.js*. Automated Malware Analysis Report for pQlSDfwyYkf.js - Generated by Joe Sandbox. Retrieved March 22, 2022, from https://www.joesecurity.org/reports/report-6cdad3b5ac021d3dbf0fb6159831cdce.html

2 JoeSecurity. (2022). *Automated Malware Analysis Report for booking_request.exe ...* Retrieved March 22, 2022, from https://www.joesecurity.org/reports/report-ff17014cbb249e173309a9e1251e4574.html

3 JoeSecurity. (2022). *Analysis reports of evasive malware*. Deep Malware Analysis - Joe Sandbox Reports Evasive. Retrieved March 22, 2022, from https://www.joesecurity.org/index.php/joe-sandbox-reports-evasive

4 JoeSecurity. (2022). *Analysis report: DoNotOpen2.doc*. Automated Malware Analysis Report for DoNotOpen2.doc - Generated by Joe Sandbox. Retrieved March 23, 2022, from https://www.joesecurity.org/reports/report-f12fc711529b48bcef52c5ca0a52335a.html

5 JoeSecurity. (2022). *Automated Malware Analysis Report for ... - joesecurity.org*. Retrieved March 23, 2022, from https://www.joesecurity.org/reports/report-fe1214a06ffc40b1ebb524f185894487.html

Khalimov, A., Benahmed, S., Hussain, R., Kazmi, S. M. A., Oracevic, A., Hussain, F., Ahmad, F., & Kerrache, C. A. (2019). Container-based sandboxes for malware analysis. *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. https://doi.org/10.1145/3344341.3368810

Kolbitsch, C., Kirda, E., & Kruegel, C. (2011). The power of procrastination. Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS '11. https://doi.org/10.1145/2046707.2046740

Leon, R. S., Kiperberg, M., Leon Zabag, A. A., & Zaidenberg, N. J. (2021). Hypervisor-assisted dynamic malware analysis. *Cybersecurity*, *4*(1). https://doi.org/10.1186/s42400-021-00083-9

Mansfield-Devine, S. (2018). The Malware Arms Race. *Computer Fraud & Security*, *2018*(2), 15–20. https://doi.org/10.1016/s1361-3723(18)30016-2

Mehra, M., & Pandey, D. (2015). Event triggered malware: A new challenge to Sandboxing. *2015 Annual IEEE India Conference (INDICON)*. https://doi.org/10.1109/indicon.2015.7443327

Moffitt, K., Karabiyik, U., Hutchinson, S., & Yoon, Y. (2021). Discord Forensics: The Logs Keep Growing. *2021 IEEE 11Th Annual Computing And Communication Workshop And Conference (CCWC)*. doi: 10.1109/ccwc51732.2021.9376133

Mohanta, A., & Saldanha, A. (2020). Malware analysis lab setup. *Malware Analysis and Detection Engineering*, 25–50. https://doi.org/10.1007/978-1-4842-6193-4_2

Shobana, M., & Poonkuzhali, S. (2020). A novel approach to detect IOT malware by system calls using Deep Learning Techniques. *2020 International Conference on Innovative Trends in Information Technology (ICITIIT)*. https://doi.org/10.1109/icitiit49094.2020.9071531

Sujyothi, A., Acharya, S., & Acharya, S. (2017). Dynamic malware analysis and detection in virtual environment. *International Journal of Modern Education and Computer Science*, *9*(3), 48–55. https://doi.org/10.5815/ijmecs.2017.03.06

StatCounter. (2022). *Desktop Operating System Market Share Worldwide*. StatCounter Global Stats. Retrieved September 11, 2022, from https://gs.statcounter.com/os-market-share/desktop/worldwide

Szor, P. (2005). The Art of Computer Virus Research and Defense. Addison-Wesley.

Tahir, R. (2018). A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, *8*(2), 20–30. https://doi.org/10.5815/ijeme.2018.02.03

Talukder, S., & Talukder, Z. (2020). A survey on malware detection and Analysis Tools. *International Journal of Network Security & Its Applications*, *12*(2), 37–57. https://doi.org/10.5121/ijnsa.2020.12203

Tan, Q., Gao, Y., Shi, J., Wang, X., Fang, B., & Tian, Z. (2019). Toward a comprehensive insight into the eclipse attacks of Tor Hidden Services. *IEEE Internet of Things Journal*, *6*(2), 1584–1593. https://doi.org/10.1109/jiot.2018.2846624

Yokoyama, A., Ishii, K., Tanabe, R., Papa, Y., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Brengel, M., Backes, M., & Rossow, C. (2016). SandPrint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. *Research in Attacks, Intrusions, and Defenses*, 165–187. https://doi.org/10.1007/978-3-319-45719-2_8

Yong Wong, M., Landen, M., Antonakakis, M., Blough, D. M., Redmiles, E. M., & Ahamad, M. (2021). An inside look into the practice of malware analysis. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3460120.3484759

Virustotal. VirusTotal. (2022). Retrieved October 4, 2022, from https://www.virustotal.com/

Windows Experience Blog. (2015, May 13). *Introducing windows 10 editions*. Windows Experience Blog. Retrieved March 28, 2022, from https://blogs.windows.com/windowsexperience/2015/05/13/introducing-windows-10-editions/

Xu Chen, Andersen, J., Mao, Z. M., Bailey, M., & Nazario, J. (2008). Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. https://doi.org/10.1109/dsn.2008.4630086

Zhang, J., Gu, Z., Jang, J., Kirat, D., Stoecklin, M., Shu, X., & Huang, H. (2020). Scarecrow: Deactivating evasive malware via its own evasive logic. *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. https://doi.org/10.1109/dsn48063.2020.00027

Zhang, Q., & Reeves, D. S. (2007). MetaAware: Identifying metamorphic malware. *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. https://doi.org/10.1109/acsac.2007.9