Master of Science in Engineering: Computer Security
June 2020


# The Development and Effectiveness of Malware Vaccination : An Experiment

**Oskar Eliasson**
**Lukas Ädel**

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Engineering: Computer Security. The thesis is equivalent to 20 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

**Contact Information:**
Author(s):
Oskar Eliasson
E-mail: osel15@student.bth.se

Lukas Ädel
E-mail: luad15@student.bth.se

University advisor:
Professor of Computer Engineering, Håkan Grahn
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

# Abstract

**Background.** The main problem that our master thesis is trying to reduce is malware infection. One method that can be used to accomplish this goal is based on the fact that most malware does not want to get caught by security programs and are actively trying to avoid them. To not get caught malware can check for the existence of security-related programs and artifacts before executing malicious code and depending on what they find, they will evaluate if the computer is worth infecting. The idea is that by identifying these checks we could "vaccinate" a system with data-points that trigger these checks and trick the malware into believing that a system is protected and skip it.

**Objectives.** This thesis will research common malware evasion techniques to find what data-points malware avoids and develop a vaccine with the found data-points. To test the effectiveness of the vaccine an experiment will be conducted where malware will be executed on different systems to observe their behavior.

**Methods.** The vaccine concept will be tested by gathering data-points with a background review of related works and performing an experiment. In the experiment a virtual machine without protective measures is used as a baseline which can be compared to a virtual machine with the vaccine. It is also interesting to see how a vaccine compares to an antivirus solution and how / if it would cooperate with an antivirus solution, so two more virtual machines are added to the experiment, one with just an antivirus software installed, and a second one with antivirus installed plus the vaccine. On these four systems, a set of malware will be executed and their behavior and activity (Windows API calls) will also be measured and compared.

**Results.** This experiment showed that our vaccine was effective in reducing malware behavior, 70% of the malware did reduce their activity when exposed to the vaccine compared to the baseline. The results also indicate that the vaccine was effective in cooperation with an antivirus program, 85% of the malware did reduce their activity on this virtual machine compared to the baseline.

**Conclusions.** From the results, we can conclude that of our created systems the system that reduced the most malware activity was the system with antivirus plus vaccine. This shows that vaccination can be a viable option for researchers to further study.

**Keywords:** vaccine, data-points, malware.

# Sammanfattning

**Bakgrund.** Huvudproblemet som undersöks i denna masteruppsats är skadan som skadlig programvara kan utföra på ett system och hur den kan minskas. En metod som kan användas för att utföra den här typen av undersökningar baseras på det faktum att skadlig programvara inte vill bli upptäckt av säkerhetsprogram och försöker aktivt att undvika dem. Skadlig programvara kan undvika att bli upptäckt genom att leta efter säkerhetsrelaterade program samt andra datapunkter, baserat på vad den skadliga programvaran hittar kan den utvärdera om det är värt att infektera systemet eller inte. Vår metod går ut på att identifiera dessa säkerhetsrelaterade program samt datapunkter och "vaccinera" ett system med dem i syfte att få den skadliga programvaran att ta beslutet att systemet inte är värt att infektera.

**Syfte.** Denna masteruppsats kommer att titta på vanliga tekniker som skadlig programvara använder för att undvika datapunker och kommer med hjälp av denna information skapa ett vaccin. För att testa hur effektivt vaccinet är, kommer ett experiment utföras och skadlig programvara kommer att köras på olika system för att undersöka hur deras beteende förändras.

**Metod.** Vaccinkonceptet kommer att testas genom att samla datapunkter från publicerade forskningsartiklar inom relaterade områden. Därefter kommer ett experiment att utföras. I experimentet kommer skadlig programvara att först exekveras på en virtuell maskin som inte har några skyddsmekanismer, detta kommer skapa en utgångspunkt för hur mycket aktivitet som den skadliga programvaran genererar. Denna aktivitet jämförs sedan efter att samma skadlig programvara har exekverats på flera andra system. Ett system med vaccin, ett system med virusskydd och ett system med både virusskydd och vaccin.

**Resultat.** Experimentet visar att vaccin minskar aktiviteten som genereras vid exekvering av skadlig programvara jämfört mot ett system utan några säkerhetsmekanismer. Resultaten visar att 70% av den skadliga programvaran minskade sin aktivitet i denna jämförelse. Vidare pekar experimentet mot att vaccin och virusskydd tillsammans är det system som får den skadliga programvaran att minska sin aktivtet mest jämfört mot systemet utan säkerhetsmekanismer. Resultaten visar att 85% av den skadliga programvaran minskade sin aktivitet i denna jämförelse.

**Slutsatser.** Våra resultat säger att av våra skapade system är det systemet med virusskydd och vaccin som är det mest effektiva systemet för att reducera aktiviteten som skadlig programvara gör. Detta visar att vaccination kan vara en bra lösning att göra vidare undersökningar inom.

**Nyckelord:** vaccin, datapunkter, skadlig programvara.

# Acknowledgments

We would like to thank our academical supervisor Håkan Grahn from Blekinge Institute of Technology and our supervisor David Olander from Orange Cyberdefense for their advice and guidance during the research, experimentation, and writing of our master thesis.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In today's society, there are almost daily headlines about a company or a program being hacked. These breaches often result in the loss of individuals' records which damages both the image and trust of the company. An additional loss is a financial loss, Hammouchi et al. [16] analyzed 9000 public breaches from 2005 which resulted in the loss of 11.5 billion individuals record and an average cost of 148 USD per stolen record.

To prevent these hacks and breaches a lot of money is invested in different computer security solutions. When you hear about companies being hacked, it is usually the big companies. Small companies are also being hacked frequently and are relatively more affected. In some of these cases, the companies have also declared bankruptcy because of it [13]. Because of this, software security solutions need to be available on all scales, both to large companies as well as the small family-owned company.

To prevent malware an antivirus program typically needs to detect and then stop the malware from executing. This can be done using two techniques: looking at known malware identifiers (signature-based detection) and typical behavior of malware (heuristic-based detection) [8].

Malware uses various techniques to avoid being detected by antivirus programs and caught in honey-pots / sandboxes, which is an environment where the malware is isolated and can not do any real harm. These environments are often used to study the malicious intents of malware. These avoiding-techniques involve looking for certain data-points (files/registries/applications etc) that malware tries to identify to abort its malicious behavior (we will refer to this as "the general approach") [8]. Besides avoiding detection, malware also tries to identify data-points to avoid infecting the same system again (we will refer to this as "the specific approach"). By creating and inserting both these kinds of data-points in a system you could trick the malware to abort its malicious behavior. This would be called "vaccinating" your system [7].

In our master thesis the focus will be on the general approach of vaccination which is based on data-points that malware looks for to avoid detection. To vaccinate against re-infecting (specific approach) you would need to have knowledge from specific malware and what data-points that malware created and then looks for [7]. This would also mean that in the best case scenario the vaccination will only stop the specific malware in which data-points have been added to the vaccine. To create an effective specific vaccine you would have to analyze a large amount of malware to identify re-infection checks and what they look for. The specific approach would

theoretically be better in the case where new malware is detected but the antivirus' signature database is not yet updated. In that case, you could update the computer with the malware's unique identifiers.

It could be bad if we disclose a useful method to prevent malware if the research is then read by malware creators which then could adapt and avoid this prevention technique. This however, is the never-ending race of malware prevention and detection vs malware infection, once one side has invented something new the other side is not far behind in adapting to the new reality [4].

If this experiment indicates that a small defensive effort with general vaccination can prevent malware activity then this technique could be interesting to look further into.

The aim of this project is to gather information about what data-points malware are looking for in general, to avoid detection. With this information, we will develop a vaccine that we can do an experiment with and evaluate its effectiveness.

This will be done with the following objectives:

- **O**$_1$: Research common evasion techniques to find what data-points malware looks for to avoid being detected.
- **O**$_2$: Develop a program that vaccinates based on previously found data-points.
- **O**$_3$: Evaluate effectiveness of vaccine and look for data-points that can be used in future versions of the vaccine.

The research questions we seek answers to in our master thesis are:

- **RQ**$_1$: Which are the data-points that malware tries to identify to abort its malicious behavior?
- **RQ**$_2$: How effective can vaccination against existing malware be, both compared to a system with antivirus and as a complement on a system with antivirus?

In order to answer the research questions the following null hypothesis will be used:

- **H**$_1$: A comparison in malware behavior on the systems will show no difference in malware behavior between systems.

By focusing on the general approach of vaccination the hope is that more malware and previously unknown malware could be prevented by a small defensive effort. By using published research on what malware analysts are hiding from malware to analyze them and previous research about vaccine, a set of data-points can be gathered and used in an experiment. In this experiment the effectiveness of vaccination against 32-bit binaries will be compared with an antivirus program in Windows 10.

In order for our master thesis to have a realistic workload, some parameters/areas are excluded from the experiment, for example no consideration is taken with regards to how the vaccine affects performance and legitimate software.

The rest of the report contains: Chapter 2 Background which introduces some terminology and concepts that are important to understand. Chapter 3 Related

Work contains a number of summaries of related works and how our master thesis contributes to the area. Chapter 4 Method describes the steps used to create our vaccine and to evaluate it. Chapter 5 Results and Analysis will bring up the results and try to visualize it. Chapter 6 Discussion will focus on our thoughts about the experiment and its results. Chapter 7 Conclusions and Future Work will conclude the paper and the results and also bring up items that can be the focus of future research.

# Chapter 2

# Background

### Data-points and Vaccine

In an attempt to visualize the concept of how vaccines work and what we in our master thesis mean when mentioning data-points figure 2.1 is used to showcase an made up example of a malware. This made up malware fetches a list of all running programs on the computer and checks whether the data-point Avast, a popular antivirus program, is running. If the malware found that the data-point existed (meaning that Avast is running on the system), the made up malware would abort its execution in fear that it would be discovered. If the data-point was not found on the system, the made up malware decides it is worth to infect the system and executes all of its malicious code.

```
programs = getRunningPrograms()
If "Avast" in programs:
    exit()
else:
    exploit("everything")
```

Figure 2.1: Trivial made up example of how a malware can check for data-points and how it can be used as a vaccine.

Other data-points that malware could check for to decide if it is worth infecting a system could really be anything on a computer for example a file, a process, a register, or hooking a API call to return a value that can be considered a data-point. The purpose of a vaccine is to place these data-points on the system to fool the malware into aborting its execution [10], [11].

### Malware infection

Virus, trojans, and worms are all different types of malicious software, also known as malware [6] which is the term that will be used throughout our master thesis. The different malware types differ between each other and it is therefore hard to find a general way to analyze them. Even in the same type classification samples differ from each other. This fact makes it difficult to analyze malware infections with regards to how they behave when exposed to vaccine.

The best-case scenario would be to get a binary result so we can conclude that a malware successfully infected the system or not. However one would need detailed knowledge about every tested malware and its behavior to do that. For example, what malicious operations are the malware capable of and what is its end goal. But even if the malware does not reach its goal the system will still be infected to some degree if any harmful operations were carried out so the result can not be considered binary in its nature. One method to handle this problem is by observing how many operations the malware executes on one system versus another. Xu et al. [35] looks at the numbers of operations that malware executes and can then compare and assume that less executed operations are better.

### Virtual Machines

According to Smith et al. [27] a Virtual machine (VM) is a machine that runs on a layer of software on another machine to support the desired architecture. The main platform that runs the virtualization is called Host and the platform that is running within the Host is called Guest.

One functionality within Virtual machines that allowed for automated and efficient malware execution is "snapshots". According to the VirtualBox documentation [24] snapshots essentially let you save the current state of the computer, including VM settings, power state, and disk state. By the use of virtual machines and snapshots we can run the malware in an isolated and protected environment which easily can be reset to its original state.

### DLL & Windows API call hooking

From Microsoft [21] we can read that a Dynamic Linked Library (DLL) is a file that contains a set of functions that can be used by different programs so that a specific function does not have to be reinvented every time someone wants to use it. This leads to there being a lot of Windows Application Programming Interface (API) DLLs which contains functions that a program can use to interact with Windows and other running programs. A program that wants to use a DLL function needs to import the DLL first. When the DLL is imported, the program knows where in memory it can call the DLL functions.

Willems et al. [33] describe Windows API call hooking, of which multiple methods can be used. One method is inline hooking which is a user-mode hook. With inline hooking, a custom DLL (with the replacements of the Windows API calls that you want to hook) needs to be injected into a process. What inline hooking then does is to find the target function that it wants to hook and modify it so that a call to that function is redirected to a custom function. The change of the original DLL-function does not affect other processes' use of that DLL-function since DLLs are protected by something called copy-on-write-protection [22]. If a process modifies the DLL code the original code is left untouched and a new changed version is created somewhere else in memory and the process that created the change will continue to use the new version while other processes continue to use the original.

It is not as straight forward as it can seem, to use inline hooking. One way to achieve inline hooking is to create a helper-program which is built so that it can

execute a malware and load the DLL that contains the inline hooking. However, this means that the malware will not be executed from the original process "explorer.exe" which is the process that a program starts from if executed by a user. Instead its parent process will be the helper-program.

Figure 2.2 shows an example of how an API call before and after inline hooking is executed. Before inline hooking, a call to a DLL-function is executed and the called DLL-function executes its body and then returns to the caller. With inline hooking, an unconditional jump is placed at the start of the DLL-function, so that when the function is called the body is not executed and the execution is instead transferred to the hook-function. The hook-function then can call the body of the DLL-function and get the return or skip the DLL-function all together and in the end return to the caller.

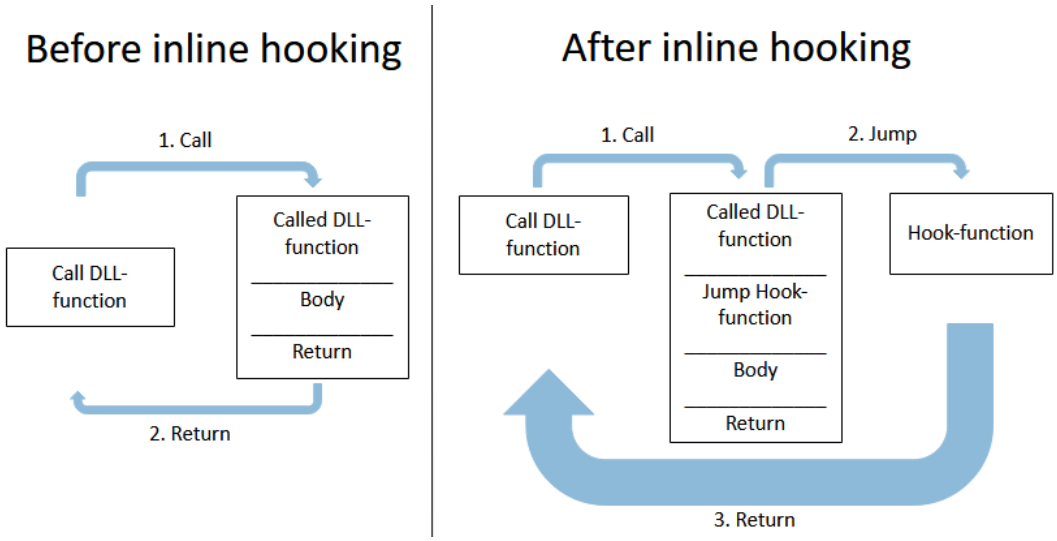


Figure 2.2: Inline hooking example

# Chapter 3

# Related Work

In a study from 2008 by Chen et al. [11] they tested how common it was for malware to use anti-VM and anti-debugging techniques and then also tested if malware could be deterred if a system on purpose looks like it is ready to analyze malware. With a dataset of 6222 malware, they concluded that 4% of malware exhibit less activity when run in a VM compared to a normal system and 39.9% of malware had less activity when run in a debugger compared to normal execution. To test deterring malware they set up a new system armed with a few data-points that imitate different monitoring setups. Here they tested with 6205 malware and 25% of the malware reduced their activity when running in this system compared to a normal system. This test proved that by making it look like you have malware analysis capabilities the malware behaved less maliciously. The idea and setup behind their experiment are more or less the same as in our master thesis but we also include a comparison with an antivirus program and since more than a decade has passed since their test it is also interesting to see if vaccination can still be as effective.

Wichmann et al. [32] conducted an experiment where malware was tested to see if they had logic to not re-infect an already infected system. If the malware had that capability the data-points that it created to mark the computer as infected where extracted. With these extracted data-points they constructed a vaccine for each malware which they tested by vaccinating a computer and then running the malware on it. The results showed that 59% of their dataset used data-points and their experiment managed to verify 95% of those data-points and successfully vaccinate a computer. This experiment explores vaccination which we also do but they focus on a different approach where they only focus on vaccination against re-infection.

Xu et al. [35] developed a program called AUTOVAC which aims to analyze and extract data-points and generate vaccines from malware. The basis for extracting a data-point was that if the data-point in someway could be related to the malware exiting its execution. The used dataset contained 1716 malware. AUTOVAC managed to create 536 vaccines to 210 of the malware in the dataset. The vaccines were then evaluated by running and monitoring the malware activity in a clean environment compared to an environment that had the vaccine. The results showed that in the worst case the malware activity was still reduced by 24% when running on a vaccinated system. This experiment, unlike our experiment, aims to automate the process of extracting vaccine data-points from malware. The results of their experiment, that in the worst-case malware activity was reduced by 24% can be compared with the result of our experiment - which creates data-points from other sources than the tested dataset.

Techniques available to detect and mitigate when malware identifies emulated monitoring systems and avoids them were something that Papazis et al. [25] researched. They found that by design, monitoring systems have indicators that can be detected. They also present a list of those indicators, tools that malware uses to find the indicators and evasive methods to mitigate the indicators of a monitoring system. In this paper the malware problem is viewed from the opposite direction, they want to identify data-points to hide them better while we do not want to hide them at all - we can extract some data-points that they tested. We can also verify that emulated monitoring systems can minimize the malicious activity of the malware. This is something we can have in mind when evaluating the experiment since our experiment is going to have malicious files running on a VM.

Shi et al. [26] wrote a paper about identifying malware evasion techniques and ways to defeat those evasion techniques when using WinDbg to analyze malware. 881 malware from 2006-2015 were analyzed and 79 evasion techniques were found. Of these techniques, they identified seven categories and the most used evasion technique from each category varies from being used by 2% of the malware to 58%. This paper focused on identifying evasion techniques while malware were analyzed with a debugger. The identified categories that malware used to evade being analyzed can also be used in our experiment to fake the presence of analyzing tools. Their result on how frequently the malware utilized evasion techniques can also be references for minimum and maximum effectiveness of a vaccine that mimics debugger behavior.

A paper by Lim et al. [18] tries to use static analysis to identify and classify if malware uses packing, anti-VM, or anti-debugging techniques. In their experiment, they used 1603 malware and found that 86% of the malware used evasive techniques related to anti-debugging while only 1% had anti-VM capabilities. Since 53% of the malware was packed and analyzed statically these number might increase since some malware's true functionality can not be detected statically. The fact that the results indicate that a low percentage of malware uses anti-VM evasion is good news for our experiment since we can not avoid VM data-points in our "clean" system which will be the baseline for the experiment. They also found that 86% of malware had anti-debugging capabilities which is a number that could be a maximum number for how effective a vaccine can be.

Sun et al. [28] present a solution to detect malware that uses anti-VM techniques. In an experiment, they calculate the difference between executing the malware in a real and virtual environment to distinguish if the malware has anti-VM capabilities. The results showed that when executing the samples in a real machine compared to in a VM the calculated malware behavior changed by a factor of 16 to 240 depending on the malware sample. During the experiment they collected the following: File system activity, windows registry access activity, process activity, networking activity, and thread activity. This paper gives us some ideas for what to monitor when conducting the experiment and it also shows that our VM will introduce some accidental vaccination that we can not measure.

Lindorfer et al. [19] worked with automated screening for malware samples with evasive behavior. A program is created where malware is being run in multiple sandbox environments multiple times and the behavior is then compared. Two different monitoring methods are used, one method monitors from within the sandbox and the other from outside the sandbox. The result showed that 77% of the behavior stayed

the same. 9% evaded the monitoring from the outside and 3% evaded the inside monitor. Another interesting result is that 2% of malware traffic was not executed because of the language in the environment (German). Lindorfer et al. looks at malware and tries to find those who indicate evasive behavior, and in contrast to others, they run their malware multiple times in the same and different environments. From this, they gather that 23% of malware behave differently even when running under the same circumstances. This fact might affect others and our results if malware only are executed once in each environment.

Yokoyama et al. [36] creates *SANDPRINT* which they deploy on several different sandboxes. *SANDPRINT* intends to detect if it is being executed in a sandbox or not by observing several features. With supervised machine learning algorithms, they can tell sandboxes apart from a real system with an accuracy of 97.8%, although this accuracy is reached on the training data. The researchers then asked themselves with these results if they could use it to create a program that detects sandboxes without being marked for evasive behavior. They tested this on three new sandboxes, and by removing one of their used checks they were able to detect all three of them without being detected themselves. The features they are looking for in their attempts to fingerprint sandboxes are not directly VM related artifacts as others have looked for. They instead focus on other indirect data-points in these three categories: hardware, history, and execution. For example the execution path of the file, RAM size, screen resolution, and system uptime.

Bulazel et al. [10] did a survey on published research from the last decade that focuses on malware which utilizes evasion techniques. First, they explain the advantages and disadvantages of four types of analysis systems and follow up with eight categories of methods that malware uses to detect that it is being analyzed. As an answer to this, they explain how to detect evasive behavior from malware and how to mitigate it. The paper discusses the relevance of evasion techniques used by malware and says that it is still a relevant parameter, that the field has not moved forward in any fundamental ways in the last decade.

By reading the work done in this field one can observe that most of the work is done to try to hide monitoring software from malware by identifying malware evasion techniques. We found a total of three papers where the concept is reversed and malware evasion techniques are exploited to trick the malware into aborting their malicious behavior [11][32][35]. Of those three only Chen et al. [11] focus on a general type of vaccine, the others explore a more targeted approach. These facts give relevance to the aim of our master thesis since it is to compare the effectiveness of a general vaccination both in terms of versus an antivirus program and as a complement to an antivirus program.

# Chapter 4

# Method

To answer our research questions we will work according to the following steps:

1. **Conduct a background review**. This section will describe the process of finding sources and what kind of information we want to extract from those sources.

2. **Obtain a dataset**. The dataset that will be selected should be well motivated and well fitted to use for the experiment with our limitations.

3. **Organize the environmental setup**. This includes multiple parameters such as how we are going to execute malware in a secure and automated manner and how the malware will be observed to get a result that can decide the effectiveness of a vaccine. We also need to find out which different states of our environment that we need to have in order to evaluate the effectiveness of the vaccine.

4. **Prepare the experiment**. This section will describe the experiment design and motivate why we chose to do an experiment.

## 4.1 Background review

To find relevant data-points that malware tries to identify to abort its malicious behavior a background review is to be conducted. We are going to use the databases in BTH-summons [9], for example, ACM Digital Library, IEEE Xplore, and Scopus. We are then going to narrow it down by mainly looking at journal articles and conference proceedings. To find relevant papers we need relevant keywords to filter with - to find these keywords we will search for expressions and words describing how malware avoids detection and how researchers counter that. We plan on using the found keywords together with the snowballing technique (backward and forward) to find papers about data-points that we can use in the vaccine.

After identifying potentially relevant papers we will start looking for information about how malware does identify security software that it wants to avoid. What methods are malware using to detect security software so that we can create the vaccine to counter those methods.

For each found search method that malware uses we look at data-points that we can include in the vaccine. To find the most relevant data-points we will look for general data-points which if applied correctly should deter many malware infections

and not just specific malware. Another criterion that is interesting for us is how frequently used the data-points are. By using the most frequently used data-points in our vaccine, the general vaccine should be as effective as possible.

To get more data regarding what methods malware utilizes in its search and what kind of data-points they are interested in, we will also look for some malware analysis reports which are not in conference proceedings or journal papers. These are typically produced and presented by security companies or professionals. The idea behind researching these kinds of reports is to get an idea of what kind of data-points malware might be looking for to help with the construction of the vaccine.

Why are we looking for the same data-points as malware are doing? If malware is looking to identify the existence of a data-point as a deciding factor on whether to continue its execution of malicious code, we can introduce those data-points with the vaccine. This will exploit the malware's behavior and make them abort their own malicious activity.

From the identified papers we will also be looking for which methods the researchers used to execute malware securely, how they monitored the malware, and how they evaluated their results. This knowledge is crucial for us to understand how to properly execute our experiment and gather data to evaluate the effectiveness of the vaccine.

## 4.2 Dataset

To gather a dataset for the experiment we were interested in a publicly available dataset so that our results could be reproduced. A dataset containing a reasonably large amount of malware - large enough so that conclusions could be drawn but small enough so that our time frame would allow us to execute every malware multiple times. The size of the dataset is also a relevant factor because of hardware limitations. The dataset should also contain recently found malware, not multiple years old malware so that the vaccine will be tested on up to date malware and thus provide us with relevant results. If an old dataset of malware would be used the results might not at all reflect the current climate of malware and it would be difficult to conclude how relevant the result would be today.

With all these criteria in mind, we found Virusshare [3] which collects malware and publishes datasets of their findings. From their available datasets we chose VirusShare_00358.zip uploaded 2019-05-15 as can be seen in figure 4.1, this contained 65536 malware initially. First of all, we are only interested in 32-bit Windows executables (.exe), *Windows* executable files because Windows 10 is the system we are interested in deploying the vaccine on. *Executable files* because we do not want to test malicious DLLs or any other type of files because they depend on special conditions to work, for example, they are not executable on their own, this leaves 2315 malware. *32-bit* executables because they are compatible to run on both 32-bit systems and 64-bit systems. After removing 32-bit executable files, 2291 malware remained in the dataset. The third step for these binaries is to test them against Virustotal.com which was done with their API [31]. The goal is to delete all binaries which do not get detected as malicious by at least one security product. The Virustotal score criterion is used to reduce the possibility that we accidentally include a

non-malicious file in our dataset, this left us with 2238 malware. The last step was to execute the malware and removing the ones that crashed. This left us with 1923 malware.

To visualize the steps and how many malware was left after each step one can look at figure 4.1

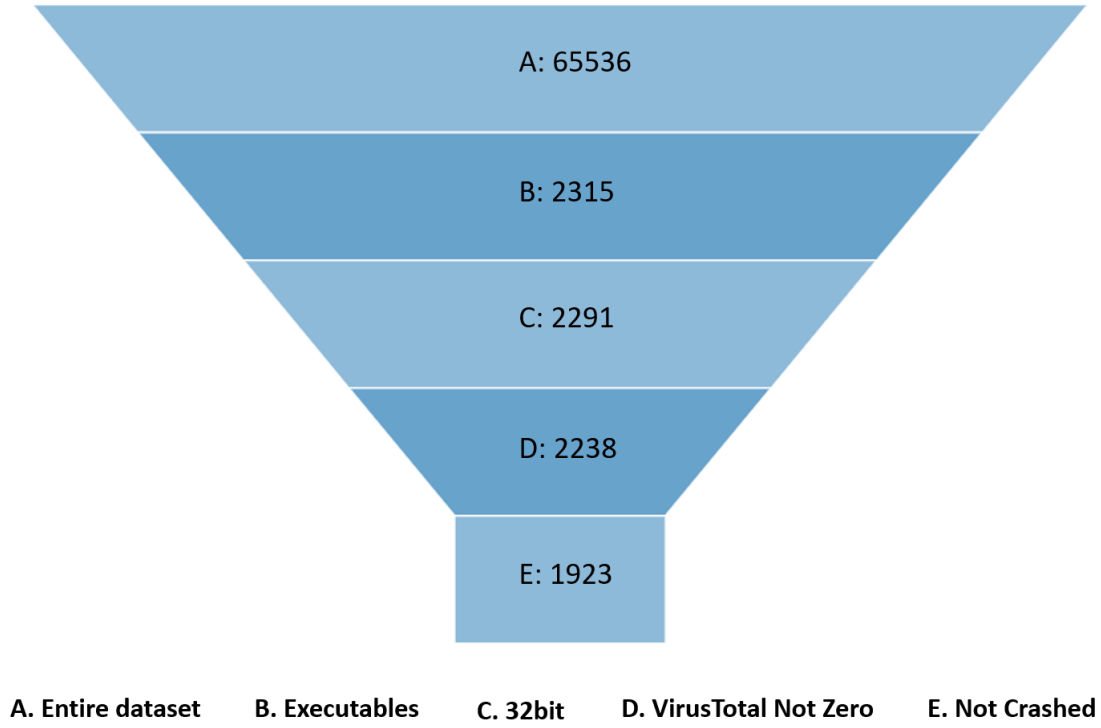


Figure 4.1: Dataset Selection

## 4.3 Environment setup

We wanted to execute the malware in a contained environment that was as similar as possible to a normal desktop computer so that the results could be compared with what would happen in reality. We also needed functionality to analyze the behavior of the executed malware and quickly reset the system to its default state to execute the next malware in the dataset. We also wanted to do this automated to make it more time-effective. The following sections will present what kind of analysis system we used and what kind of monitoring system we used and how they were setup.

### 4.3.1 Analysis system

Bulazel et al. [10] talk about four analysis systems used for analyzing malware. The method that resonates the most with our specifications is virtual machines with the use of snapshots. This section will explain how the virtual machines were set up and what software was installed. VirtualBox version 6.1.4 on an Ubuntu 18.04 desktop is used for all the virtual machines which are based on Windows 10 home x64, together

with Windows Defender which is the antivirus software that will be tested, with the following parameters:

- System version 1909 (released November 2019, no updates done).
- OS build 18363.418.
- Antimalware client version: 4.18.1902.5.
- Engine version 1.1.15600.4.
- Antivirus version 1.285.74.0.
- Antispyware version 1.285.74.0.

To restore the machine state after executing a malware we used snapshots. For us to be able to compare and test the effectiveness of our vaccine we have to set up different VMs with different settings. We created four snapshots with the following states:

1. **Clean**: Not vaccinated and antivirus is inactive.
2. **Antivirus**: Not vaccinated and antivirus is active.
3. **Vaccine**: Vaccinated but antivirus still inactive.
4. **Antivirus + Vaccine**: Vaccinated and antivirus is active.

After creating the vaccine and inserting it on a computer we would need other computers without the vaccine to be able to compare them. Since we want to test the effectiveness of our vaccine we first need to get a baseline of what is considered normal behavior. This is to be done by analyzing how the malware behaves on a computer without any protection, hence the *Clean state*. We also want to test how our vaccine works compared to a computer that has an antivirus program active, hence the *Antivirus state*. Both the *Clean state* and *Antivirus state* are going to be used twice, once with the helper program and once without to test if the helper-program affects the results. When using the helper-program in these states it will inject an empty DLL which will not interfere with the execution of the malware, but the effects of executing the malware from a helper-process and injecting a DLL will still be measured. Finally, we want to be able to find out whether our vaccine can work as a complement to an already existing antivirus, this is done by using the *Antivirus + Vaccine state*. In table 4.1 and 4.2 we explain what settings differ from state to state or what settings and installed programs are similar.

*Spoofed processes* are "empty" processes, meaning they do not do anything, they are placed in their original installation paths and executed from that path, making the malware think that we have the program installed and running. This is a part of our vaccine and is therefore "off" on systems that do not contain the vaccine. *Antivirus* - This is the anti-malware software used in the *Antivirus* and *Antivirus + Vaccine* states. *Keyboard layout* - US keyboard layout was installed by default. however, we also installed the Swedish keyboard layout to make it more realistic, this goes for every system. *Region* - Default was US, we never changed it on any

system. *Windows API call interception* - This is also a part of our vaccine, to interrupt the Windows API calls made by the malware. Because of this, the systems containing the vaccine is the only systems that have this on. *Spoofed registries* - This is the third part of the vaccine, we imported registries that could work as a vaccine. Because of this, the systems containing the vaccine are the only systems that have this functionality on. *Google Chrome* - A typical web browser that we could use. To make it even we installed it on all the systems. *Python 2.7.9 (32-bit)* - CAPE has a python script named "agent.py" which has to be running in order for CAPE to analyze the activity caused by the malware, to run this we had to install Python 2.7.9 (32-bit), therefore it is installed on all the systems. *Executed with Helper-program* explains if the helper-program was used to execute the malware or not within the four snapshots.

Table 4.1: Difference between the VM states

| **Settings** | **Clean without helper** | **Clean with helper** | **Antivirus without helper** | **Antivirus with helper** |
|---|---|---|---|---|
| Spoofed processes | Off | Off | Off | Off |
| Antivirus | Off | Off | On | On |
| Keyboard layout | Swe (Active) + US | Swe (Active) + US | Swe (Active) + US | Swe (Active) + US |
| Region | US | US | US | US |
| Windows API call interception | Off | Off | Off | Off |
| Spoofed registries | Off | Off | Off | Off |
| Google Chrome | installed | installed | installed | installed |
| Python 2.7.9 (32-bit) | installed | installed | installed | installed |
| Executed with Helper-program | No | Yes | No | Yes |

Table 4.2: Difference between the VM states

| **Settings** | **Vaccinated** | **Antivirus + Vaccine** |
|---|---|---|
| Spoofed processes | On | On |
| Antivirus | Off | On |
| Keyboard layout | Swe (Active) + US | Swe (Active) + US |
| Region | US | US |
| Windows API call interception | On | On |
| Spoofed registries | On | On |
| Google Chrome | installed | installed |
| Python 2.7.9 (32-bit) | installed | installed |
| Executed with Helper-program | Yes | Yes |

To see all the spoofed programs used in the vaccine, see table A.1. For all imported registries used in the vaccine, see table A.2. For all the hooked Windows API calls used in the vaccine, see table 5.1.

### 4.3.2 Monitoring System

To monitor the behavior of the executed malware inside the virtual machines we needed something to monitor what was happening inside the virtual machine or better - something that monitors only what the malware did. One system that provides functionality for these needs is the CAPE sandbox [12]. CAPE also provides functionality that allows it to be scripted and automated. CAPE can be instructed to execute a file on a VM and save the executed file's behavior for later analysis. The default settings of CAPE were not exactly what we needed, so we had to tweak it to suit us with the following changes:

1. Timeout 120 seconds.
2. Critical timeout 40 seconds.
3. Not hiding process IDs (PIDs).
4. Save Windows API calls to CSV files.
5. Turned off autosubmit CAPE-samples.
6. Disabled network access to the VM.

Timeouts were set to match the two minute run-time for the malware as Chen et al. [11] did, this was done to minimize the risk of the malware being cut off during its execution. The critical timeout is related to how much time our VM:s are allowed to take on startup and shutdown. CAPE also has functionality that hides certain PIDs from being searched for - PIDs that we have included in our vaccine and want to be detectable. The Windows API calls were saved to CSV files to enable better result analysis. Autosubmit of CAPE-samples was turned off - this is something CAPE

otherwise uses to submit new malware for testing if a malware downloads or extracts a new executable. This was turned off to save time - if the malware did execute its created file that file's behavior would be monitored anyway. We do not want CAPE to submit and analyze that file again. Network access was disabled to ensure that no malicious activity would be spread beyond the controlled virtual machines.

As we mentioned in tables 4.1 and 4.2 we have used Windows API call interception in the vaccinated state. This is done to target specific Windows API calls that malware uses to detect security software. Our implementation of this is done by injecting a DLL file into the malware which can hook the Windows API calls (more detailed explanation in chapter 2 Background). The next problem is that we somehow need to inject the DLL into the malware. One solution to this problem is the fact that CAPE offers the functionality of submitting malware samples of multiple types and with arguments. One way we could utilize this functionality is by compressing our DLL, malware, and DLL injecting program into a ZIP archive and then use CAPEs python submit-script with the following command:

*folder.zip –package zip –options file=Helper-program.exe,arguments='Hook.dll malware.exe'*

This instructs CAPE to use their zip-package and to start Helper-program.exe with the argument to inject Hook.dll into malware.exe and start malware.exe. To isolate and test if this method had any effect on the malware we also chose to use this method but with a DLL that did not do anything on the *Clean* and *Antivirus* systems. The *Clean* and *Antivirus* systems also were tested without the helper-program and empty DLL.

Instead of trying to navigate and make accurate decisions about the infection rate, we will observe the malware behavior, how many operations does the malware execute on one system versus another. As Xu et al. [35] we look at the numbers of operations that the malware executed and can then compare between systems and assume that less is better.

The fact that one extra process is created in the cases where we use the DLL-injecting process, there will be extra measured activity. This is something that is going to be taken into consideration. This extra activity is excluded from the final result since it is activity caused by our vaccine and not the malware. We can from the result see what activity was caused by each process and simply not count any activity from the DLL-injecting process.

One Windows API call that will differ between our virtual machines because of our vaccine is Process32NextW. This will cause more API calls the more processes are running on the system, which affects the result of the comparison since our vaccinated computer have a lot of spoofed/fake processes running. This is something that we take into account by not counting the Process32NextW call towards the malware activity number.

Another thing besides activity that we will measure is the amount of started sub-processes (PIDs) each malware produces. Because it is an indicator of prevented malicious behavior if a malware that spawned sub-processes in the *Clean state* does not start them in other states.

*Activity* will be based on the number of Windows API calls the malware is ex-

ecuting. Only Windows API calls made by the malware will be captured, nothing from other processes. This is done by CAPE which injects a monitoring DLL into the malware process which hooks Windows API call functions and monitors them (Explained in Background, figure 2.2). CAPE stores the execution results in a Mongo database (MongoDB) [23] by default, we also activated CSV file storage, which meant we got access to all Windows API calls that the malware executed. The information in the Mongo database together with the Windows API calls allows us to analyze both the Windows API calls but also how many PIDs were created.

Except monitoring the total activity exhibited by the malware we will also specifically look for how many of the tested malware did look for the implemented data-points, and if the malware searched for any data-points that were not implemented in our vaccine. This will be done by observing the Windows API call logs for API calls related to searching for data-points, for example findWindow() and openFile().

## 4.4 Experiment execution

From Wohlin et al's [34] definition of an experiment, we see that they are in line with what we are aiming for. We wanted to set up an environment with fixed independent variables, this is our *Clean system* which all other systems then add something onto. In the other systems, the only changes made are either installing the vaccine or turning on the antivirus or both. From this, we want to measure the dependent variables which are malware activity and created PIDs, and see how they depended on the vaccine and antivirus. These goals do not align with the goals of a case study which is more observational in its nature and does not seek to test a hypothesis according to Wohlin et al.

We will use the previously presented virtual machines, see table 4.1 and 4.2, and their different states during the experiment. Each malware will be executed twice on each VM so that an average activity can be calculated and used when calculating the results. This is done to get a more accurate result if the behavior of the malware changes from one run to the other. According to Lindorfer et al. [19] 77% percent of their dataset contained the same amount of activity when run multiple times. Meaning that 23% changed their activity when run in the same environment multiple times.

Since the VM will be reset after each malware execution the starting order of the malware does not have any impact on the result. However, the malware will be picked sequentially sorted by name from the dataset (The naming convention of the malware are VirusShare_MD5Hash of the malware. The fact that the names are MD5 hashes means that the order of the malware already are random and not depending on the original name, size or date) and then executed twice on each state:

1. **Clean without helper**: Not vaccinated and antivirus is inactive.
2. **Clean with helper**: Not vaccinated and antivirus is inactive.
3. **Antivirus without helper**: Not vaccinated and antivirus is active.
4. **Antivirus with helper**: Not vaccinated and antivirus is active.

5. **Vaccine**: Vaccinated but antivirus still inactive.

6. **Antivirus + Vaccine**: Vaccinated and antivirus is active.

The activity of the malware (Windows API calls) during run-time will be monitored with the help of CAPE sandbox [12]. We can not say that malware infected the system without manually inspecting every malware - which time restrictions do not allow. Instead the level of activity each malware produce will be measured. The level of activity will be less if the malware decides not to infect the system. We will also look at the created number of PIDs which also will be less if the malware decides not to infect the system.

The effectiveness of the vaccination will be evaluated by comparing generated malware activity and PIDs between the different systems to get an understanding of how much malicious behavior different systems prevented. After each system has executed every malware twice and we have averaged each malware's behavior we will calculate average rank, standard deviation, median, and average values for each system. Average rank stems from individual ranks which are calculated by gathering the activity / PIDs generated by each malware and then rank the systems from 1-6, lowest to highest activity / PIDs. Standard deviation shows how much the activity / PIDs for each system differs from the mean.

With the generated activity and number of PIDs from each malware on the different systems a Friedman test [15] will be used to test our null hypothesis mentioned in chapter 1 Introduction. This is done to see if the outcome in activity and PIDs on the different systems did statistically differ from each other. If the null hypothesis can be rejected, the systems differed from each other, a post hoc test can be applied to do pair-wise comparisons between systems. In this case we will use the Nemenyi test [15] to evaluate whether two systems statistically differed from each other.

# Chapter 5

# Results and Analysis

This chapter will present the results from the background review, experiment, and analysis of the results. The results are gathered by following the method presented in chapter 4 Method.

## 5.1 Background review

The following keywords were found that we could use for screening papers: malware evasion, anti-VM, anti-debugging techniques and sandboxing. By using the keywords one paper stands out as the first published paper which we found that talks about malware evasion and also proposes vaccination as a counter to malware. This paper from Chen et al. [11] was then used as the focus point in the snowballing technique which we utilized to search for other papers that included the keywords that we found.

From the survey made by Bulazel et al. [10] which covers this area of research, we have gathered information regarding general themes and selected specific areas to focus on. They describe eight different categories that malware uses for fingerprinting analysis systems and we have chosen to implement vaccine data-points that target two of these categories: environmental artifacts and process introspection. Examples of environmental artifacts are files on disk and running processes, one example of process introspection is Windows API call hooking.

The Windows API call hooks can catch a wide area of malware that uses a specific technique for checking certain parameters, such as if debugging is present, or if a window is open/running. The debugging part is impossible to fake with processes, however fake processes are a good base to use for searches that are not covered by the Windows API call hooks. One example of this would be if a malware uses a non-hooked Windows API call function to search for a process. In this case a spoofed process will work, if we spoofed the specific process that the malware searched for.

One thing we chose not to look further into is mutexes. According to Xu et al. [35] it is possible to vaccinate a system by creating mutexes with specific names to fool the malware that it has already installed itself. For example, malware might create a mutex called "InfectionCheck" and if it finds that mutex when executing, it will stop its execution to not infect the same system again and reduce the risk of being detected. By installing a specific mutex they found in Conficker, which is a big malware family, they managed to prevent the initial infection phase. However, this does not prove that mutexes can work as a good general vaccine. Which is why we chose not to include it.

By going over the papers found during this background review a list of potentially usable data-points is created. Because of time limitations and difficulty of implementing and testing int3 [18], UnhandledExceptionFilter [18] and KiUserExceptionDispatcher [11] was disregarded after some unsuccessful implementations even though they were recognized as some of the most used checks by malware.

The next step was to create a list of data-points (environmental artifacts) that can be effective against malware. To generate such a list that is relevant to our experiment we had to look for data-points that were general (not malware specific) enough that they could be applied and work against multiple malware. In search of specific data-points or a theme of data-points, we looked at some reports that analyzed specific malware. One report about TrickBot [5] indicates searches for commonly used analyzing tools. Another report, about PlugX [29] does indicate searches for antivirus software. Lindorfer et al. [19] searches for artifacts that relate to the malware analysis system Anubis. With this information, we can see that malware seems to be interested in searching for malware analysis tools and antivirus software. To obtain a list of malware analysis software to include in the vaccine we look at FlareVM [17]. FlareVM is a malware analysis virtual machine that comes preinstalled with malware analysis tools. By examining FlareVM we managed to extract process names, file paths, and registries that we could import into our vaccine.

Since we saw searches for antivirus programs we installed and gathered file paths and process names for several popular antivirus programs and antivirus programs that we identified malware searches for. Because of the time limitations we could not install and extract data-points from all antivirus programs. This means that there exist many popular antivirus programs excluded from the ones we used so there exists room for improvement in this part of the vaccine. Another common theme for malware is the detection of being executed inside a virtual machine, something that for example Papazis et al. [25] have brought up, so we also looked at VM-specific processes and registries that appeared in a VMWare VM and a VirtualBox VM. The complete list of environmental artifacts (both files and registries) can be seen in the following tables A.1 and A.2 which are located in the appendix.

## 5.2 Vaccine creation

With the results from the background review and by targeting the vaccine on the two categories environmental artifacts and process introspection, this section will describe the contents of the vaccine.

We found process names of security products that could be used in the vaccine. Processes where created with the same names and executed on the system. The found processes' file paths were also duplicated so that if a security program was found in C:/Programs/Program1/Program1.exe that is where the spoofed program was inserted and started from. The result of this is a spawned process called Program1.exe, with the window name C:/Programs/Program1/Program1.exe. The Windows registries related to security products and virtual machines were also imported.

A lot of the papers from the background review focused on anti-debugging techniques and presented popular Windows API calls, the ones we selected are presented

in table 5.1. Windows API calls related to anti-debugging were hooked with the help of Microsoft Detours [30] which is the same tool that Chen et al. [11] used to make the creation of the hooks easier. With Detours, functions can be created with the same name as the original system functions but with own defined behavior. The functions created are compiled into a DLL which is injected into the malware and thus overwriting Windows API calls to our created functions instead of the corresponding original Windows functions. Some of our created functions would not apply in a live system and must be reconstructed to work in such environments. For example, malware might use FindWindow to look for open security programs and our hook of this function just returns the first found window which would mean that anyone who uses this function will always believe the window they looked for exists.

One limitation here is how we use the Windows API call hooks. Since we know every malware we intend to execute we can start the malware with another process that injects the hooking-DLL into the malware at startup. In a real-world scenario, you do not know if an executable file is malicious or not so some other strategy would be needed for the injection of the hooking-DLL. One strategy would be to include the hooking-DLL in Windows AppInit which would enable the hooks in any program that uses the Win32 API [20]. Another strategy would be to hook every executable on the system - or only non-recognized applications.

From the found papers and their findings we chose to implement the following Windows API call hooks mentioned in table 5.1. They were chosen based on the criteria that they were frequently used by malware and reasonably general.

Table 5.1: Hooked Windows API calls

| **Windows API calls** | **Monitorable** |
|---|---|
| IsDebuggerPresent [18], [11] | yes |
| FindWindowA [18] | yes |
| FindWindow [18] | yes |
| OutputDebugStringW [18] | yes |
| OutputDebugStringA [18] | yes |
| OutputDebugString [18] | yes |
| CheckRemoteDebuggerPresent [18], [11] | yes |
| NtQueryInformationProcess [37],[14] | yes |
| PEB BeingDebugged [37],[14] | no |
| NTGlobalFlag [37],[14] | no |

Explanation of how the hooked Windows API calls are modified:

- **IsDebuggerPresent** - Always return true.
- **FindWindow*** - Returns a filehandle to the first found open window.
- **OutputDebugString*** - Sets last error to 1 before returning - indicating that a debugger caught the text but also somewhere generating an error and thus revealing its presence.

- **CheckRemoteDebuggerPresent** - Always return true.
- **NtQueryInformationProcess** - If queried for ProcessDebugPort or ProcessDebugObjectHandle the returned values are altered to indicate the precense of a debugger.
- **PEB BeingDebugged** - BeingDebugged bit was set to 1 by executing some assembly instructions. So if a malware checks its PEB BeingDebugged bit it will believe a it is being debugged. This data-point is not monitorable in our setup since malware checks it by executing assembly instructions.
- **PEB NTGlobalFlag** - NTGlobalFlag bit was set to a non-zero value by executing some assembly instructions. So if a malware checks its NTGlobalFlag value it will believe it was started by a debugger. This data-point is not monitorable in our setup since malware checks it by executing assembly instructions.

## 5.3 Effectiveness Evaluation

The *Clean system* is used as the base to which other systems' activity can be compared to. Since this system has no security measures the malware should be able to execute its entire execution path from start to finish without any interruptions and therefore generate the most amount of activity. The *Vaccinated system* will have our vaccine solution enabled but no antivirus software enabled. The *Antivirus system* will only have antivirus enabled. The *Vaccine + Antivirus system* will have both our vaccine solution and antivirus enabled.

When results are presented in the sections below the measured *activity* from the 1923 malware will be based on the number of Windows API calls that the malware has executed. Only Windows API calls made by the malware will be captured, nothing from other processes. This is done by CAPE which injects a monitoring DLL into the malware process which hooks Windows API call functions and monitors them (explained in chapter 2 Background, figure 2.2). All calculations below is based on the average activity for each malware which is calculated by using equation 5.1.

$$Activity = \frac{run1 + run2}{2} \tag{5.1}$$

To give a view of how the malware behaved on the four different systems we use average rank, median activity / PIDs, average activity / PIDs, and the standard deviation of malware activity / number of PIDs in table 5.3 and table 5.5. Average rank stems from individual ranks which are calculated by gathering the activity generated by each malware and then rank the systems from 1-6, lowest to highest activity. Standard deviation shows how much the activity for each system differs from the mean. *Clean* and *Antivirus* have two rows each, one with and one without helper (explanation of helper can be found in chapter 2 Background) - this is because we executed these systems both with and without executing the malware from a helper-program which injects an empty DLL into the malware. This was done to be able to find out the impact of the helper-program vs the vaccine data-points. The amount of activity each malware produces - since they are malware, indicates how much

malicious actions are taken. The lower the amount of activity the better in this case since we want to prevent malicious activity. As mentioned in chapter 4 Method, we consider the helper-program to be a part of the vaccine and from now on we will only focus on analyzing the systems without helper.

To evaluate whether there is any statistical difference between the different systems with regards to generated activity we test our null hypothesis, as described in chapter 1 Introduction, that the systems will perform identically. To test this we first use the Friedman test [15] to find out if there were any difference between the systems by ranking how they performed on each malware. With the Friedman implementation [2] it can be concluded that the Friedman test shows significance ($p < 0.01$) so we can reject the null hypothesis, showing that the systems does statistically differ from each other. Because of this it is viable to do a pairwise comparisons between systems.

A test that can be applied after the Friedman test shows significance is the Nemenyi test [15] which does pairwise comparisons of the systems. With the Nemenyi implementation [1] we get the results presented in table 5.2. The table shows with the color blue which comparisons between systems did show statistical significant difference with a 95% confidence interval. The displayed numbers show to what degree the comparison showed statistical difference, if the number is less than 0.05 it showed a statistical significant difference with at least 95% confidence interval. The comparisons marked with red are the ones that had a number higher than 0.05 and thus did not show a statistical significant difference between the systems with a confidence interval of 95%. This means that the systems that did differ from each other can be compared by looking at their average activity numbers or average rank, the lower the average activity number or average rank is, the better the system performed in reducing malicious behavior.

Table 5.2: Statistical differences between activity in systems with the Nemenyi test

| | Clean without helper | Clean with helper | Vaccine | Defender without helper | Defender with helper | Vaccine + defender |
|---|---|---|---|---|---|---|
| Clean without helper | -1 | | | | | |
| Clean with helper | 0.001 | -1 | | | | |
| Vaccine | 0.001 | 0.001549 | -1 | | | |
| Defender without helper | 0.001 | 0.001 | 0.0192 | -1 | | |
| Defender with helper | 0.001 | 0.001 | 0.001 | 0.001 | -1 | |
| Vaccine + defender | 0.001 | 0.001 | 0.001 | 0.001 | 0.777194 | -1 |

From the numbers in table 5.3 regarding the *Clean system without helper* we get an idea of where the activity levels are when no security products are in place to prevent the malware. Moving on to the numbers from *Vaccine* we can see that both the median and the average number of generated activity have been reduced. Median reduced by 67% and average reduced by 84% compared with *Clean without helper*. This proves that it is better to have the vaccine than having no defense at all installed or activated on the system, according to our experiment. We can also see that the numbers generated during the Antivirus-execution are reduced compared to

the *Clean without helper* numbers. Median reduced by 73% and average was reduced by 50%. This proves that *Antivirus without helper* is also better than just running your computer without any type of defense. Finally, we see that when the vaccine is deployed as a complement to *Antivirus without helper* the numbers are reduced even more. Median reduced by 92% and average reduced by 89%.

From the displayed standard deviation for each system, we can see that in the *Clean system without helper* the malware activity differs the most from the average. In the other systems, the activity differs less from the average and *Antivirus + Vaccine* is the system with the lowest difference in malware activity from the average. So there is an effect from the defensive efforts which reduces the maximum activity and thus the standard deviation.

Table 5.3: Virtual Machines malware activity levels

| **system** | **Average rank** | **Median activity** | **Average activity** | **Standard deviation** |
|---|---|---|---|---|
| Clean without helper | 4.82 | 4861 | 37309 | 111613 |
| Clean with helper | 3.84 | 2171 | 9985 | 38792 |
| Vaccine | 3.60 | 1636 | 5996 | 29330 |
| Antivirus without helper | 3.42 | 1315 | 18817 | 93605 |
| Antivirus with helper | 2.69 | 465 | 4783 | 27089 |
| Antivirus + Vaccine | 2.61 | 389 | 4332 | 26881 |

Another factor than malware activity to monitor is the number of processes that the malware produces. If the malware does not spawn any extra processes the number will be one. If the number of processes is reduced it is a clear indicator of reduced activity since the malware did choose **not** to create another process. Table 5.4 is based on the same principles as table 5.2 but instead of measuring activity, PIDs are measured.

Table 5.4: Statistical differences between PIDs in systems with the Nemenyi test

| | Clean without helper | Clean with helper | Vaccine | Defender without helper | Defender with helper | Vaccine + defender |
|---|---|---|---|---|---|---|
| Clean without helper | -1 | | | | | |
| Clean with helper | 0.001 | -1 | | | | |
| Vaccine | 0.001 | 0.270054 | -1 | | | |
| Defender without helper | 0.001 | 0.001 | 0.001 | -1 | | |
| Defender with helper | 0.001 | 0.001 | 0.001 | 0.124421 | -1 | |
| Vaccine + defender | 0.001 | 0.001 | 0.001 | 0.001234 | 0.687588 | -1 |

Table 5.5 is based on the same principles as table 5.3 but instead of measuring activity, PIDs are measured. Again the *Clean without helper* numbers are used as a base level for comparison, and we can also see that the median values are the same for every system. The average values however differ and we can see that the average number of PIDs for the *Vaccinated system* was reduced by 23%. This shows that according to our experiment, it is better to use our vaccine than nothing in regards to

PIDs. The average number of PIDs for the *Antivirus system without helper* was also reduced in comparison to the *Clean system without helper*. It was reduced by 29%. This means that using an antivirus is better than using nothing and also better than using only vaccine when considering PIDs. When looking at the average number of PIDs in the *Antivirus + Vaccine system* the number is reduced by 36% which is the highest decrease.

Table 5.5: Virtual Machines malware PID levels

| **System** | **Average rank** | **Median PIDs** | **Average PIDs** | **Standard deviation** |
|---|---|---|---|---|
| Clean without helper | 4.13 | 1 | 2.32 | 2.26 |
| Clean with helper | 3.81 | 1 | 2.02 | 1.83 |
| Vaccine | 3.68 | 1 | 1.79 | 1.45 |
| Antivirus without helper | 3.25 | 1 | 1.64 | 2.03 |
| Antivirus with helper | 3.10 | 1 | 1.53 | 1.80 |
| Antivirus + Vaccine | 3.01 | 1 | 1.49 | 1.77 |

With the raw data of how much activity the malware generated on each of the tested systems, the following sections will compare the different results to give a more detailed view of how the malware activity changed between the different systems. The calculation for each malware to get the activity change in percentage between the two systems is calculated with equation 5.2.

$$Percentage\_change = \frac{sys2\_malwareX\_activity \quad sys1\_malwareX\_activity}{sys1\_malwareX\_activity} \tag{5.2}$$

### 5.3.1 Clean without helper compared with Vaccine

To give some perspective on how much activity the malware generated on the *Vaccinated system* it can be compared with the activity generated on the *Clean system without helper*. Using equation (5.2), the *Clean system without helper* will be *sys1* and the *Vaccinated system* will be *sys2*.

In table 5.3 we saw that the median and average malware did decrease in activity from the *Clean system without helper* to the *Vaccinated system*. But how many of the malware did decrease their activity and how much did they decrease it with? Figure 5.1 tries to answer that question by showing how many percentages of the malware did decrease their activity by a certain threshold percentage. Because we do not know the goal of each executed malware and thus we have not defined if they reached their goal this diagram was created to allow the reader to decide what threshold level is important.

From the figure, we can see that a majority (70%) of the malware was affected by the vaccine and reduced their activity by at least 5% compared with the *Clean system without helper*. We can also see that 16% of the malware did reduce their activity by at least 95%.

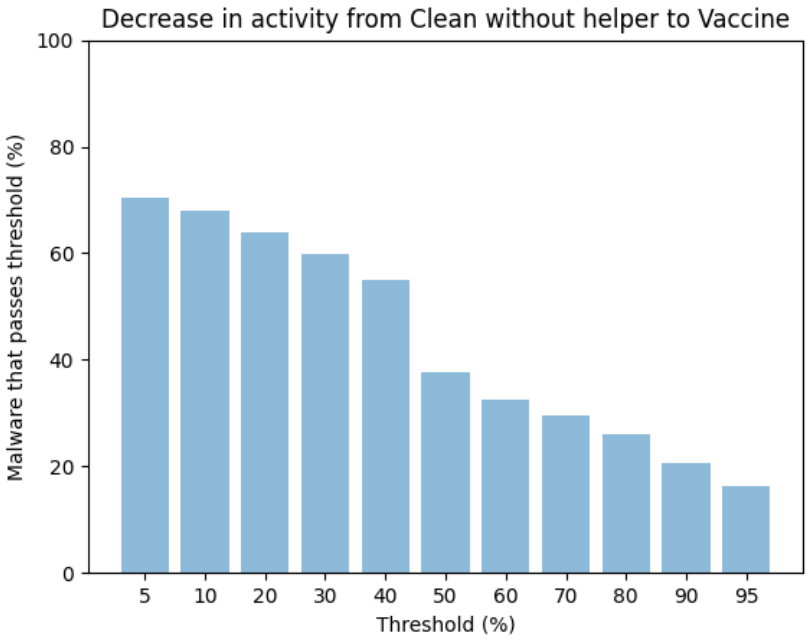


Figure 5.1: Vaccine: How many percentage Y of malware decreased their activity considering a threshold X percentage decrease

Another metric is the number of processes created by the running malware, if the number of processes decreased it would also indicate less malicious activity. When running on the *Vaccinated system* 26.9% of the malware had at least reduced their number of processes with one compared to the *Clean system without helper*.

### 5.3.2 Clean without helper compared with Antivirus without helper

To give some perspective on how much activity the malware generated on the *Antivirus system without helper* it can be compared with the activity generated on the *Clean system without helper*. Using equation (5.2), the *Clean system without helper* will be *sys1* and the *Antivirus system without helper* will be *sys2*.

Regarding the activity levels we can see in table 5.3 that the activity levels decrease from the *Clean system without helper* to the *Antivirus system without helper*. Figure 5.2 will be used to get a better view of how the activity decreased from being executed on the two systems.

Figure 5.2 tries to show how many of the malware decreased their activity and by how much by showing how many percentages of the malware did decrease their activity by a certain threshold percentage. Because we do not know the goal of each executed malware and thus we have not defined if they reached their goal this diagram was created to give the reader the opportunity to decide what threshold level is important.

From the figure we can see that 54% of the malware was affected by *Antivirus without helper* and reduced their activity by at least 5% compared with the *Clean*

*system without helper*. We can also see that 34% of the malware did reduce their activity by at least 95%.

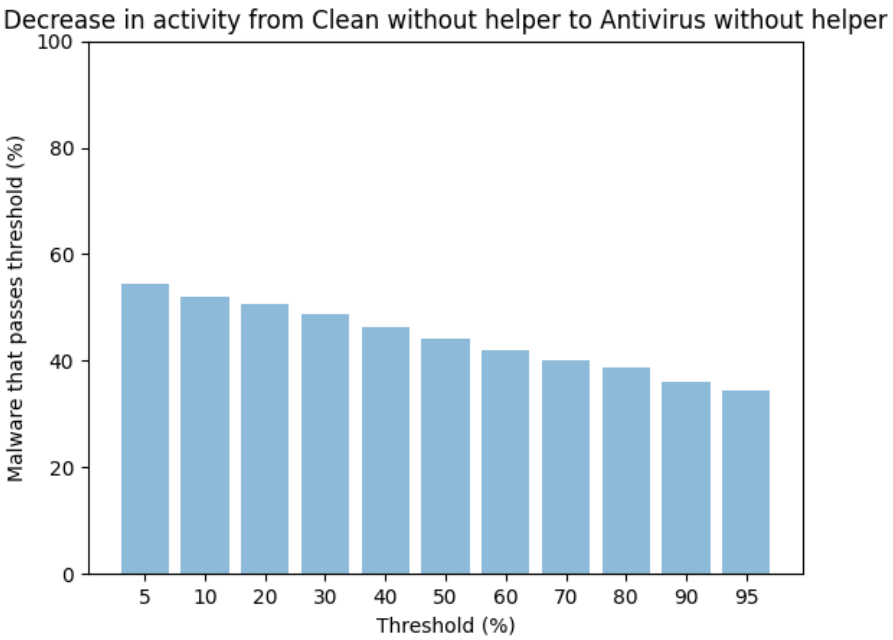


Figure 5.2: Antivirus: How many percentage Y of malware decreased their activity considering a threshold X percentage decrease

Another metric is the number of processes created by the running malware. If the number of processes decreased it would also indicate less malicious activity. When running on the *Antivirus system* 35% of the malware had at least reduced their number of processes with one compared to the *Clean system without helper*.

### 5.3.3 Clean without helper compared with Antivirus and Vaccine

To test if the vaccine would be of any help to an existing antivirus this section will compare how much activity the malware generated on the *Antivirus + Vaccine system* with activity generated on the *Clean system without helper*. Using equation (5.2), the *Clean system without helper* will be *sys1* and the *Antivirus + Vaccine system* will be *sys2*.

Regarding the activity levels we can see in table 5.3 that the activity levels decreased from the *Clean system without helper* to the *Antivirus + Vaccine system*. Figure 5.3 will be used to get a better view of how the activity decreased from being executed on the two systems.

Figure 5.3 tries to show how many of the malware decreased their activity and by how much by showing how many percentages of the malware did decrease their activity by a certain threshold percentage. Because we do not know the goal of each executed malware and thus we have not defined if they reached their goal this

diagram was created to give the reader an opportunity to decide what threshold level is important.

From the figure we can see that 85% of the malware was affected by *Antivirus without helper* and reduced their activity by at least 5% compared with the *Clean system without helper*. We can also see that 40% of the malware did reduce their activity by at least 95%.

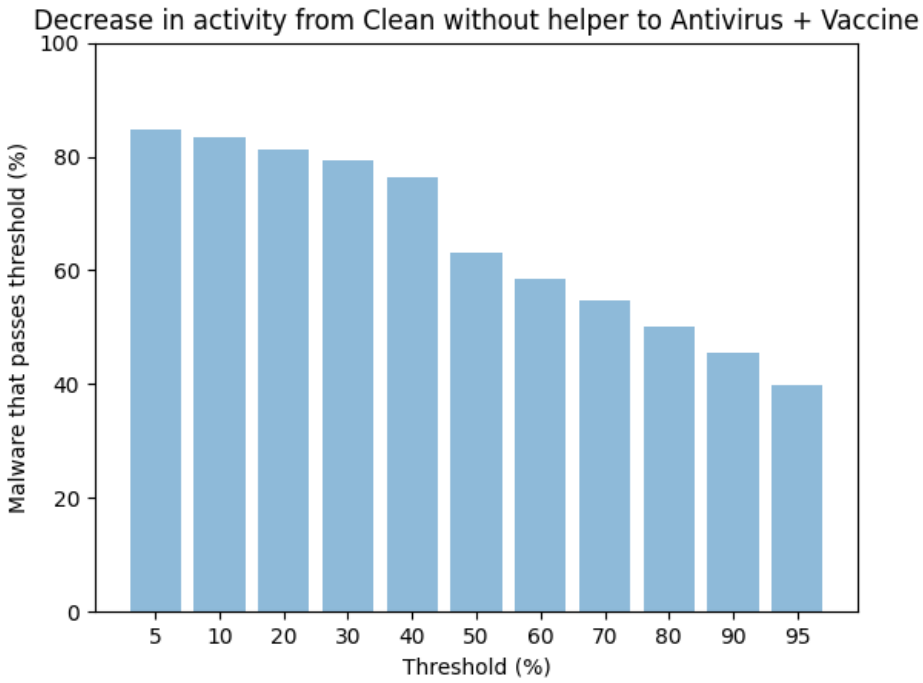


Figure 5.3: Antivirus + Vaccine: How many percentage Y of malware decreased their activity considering a threshold X percentage decrease

Another metric is the number of processes created by the running malware. If the number of processes decreased it would also indicate less malicious activity. When running on the *Antivirus + Vaccine system* 44% of the malware had at least reduced their number of processes with one compared to the *Clean system without helper*.

### 5.3.4 Vaccine data-points

From the data gathered during the execution of malware we can look deeper at what the malware actually did and by doing so finding out which of the injected vaccine data-points the malware searched for whether it was in our vaccine or not.

Based on the malware executed on the vaccinated system we can see how frequently each API call got hooked in our DLL. This is presented in table 5.6.

Table 5.6: Hooked Windows API calls

| **Windows API calls** | **percentage** |
|---|---|
| IsDebuggerPresent | 35.3% |
| FindWindow | 5.7% |
| FindWindowA | 0.5% |
| OutputDebugString | 0.3% |
| OutputDebugStringA | 0.1% |
| OutputDebugStringW | 0% |
| CheckRemoteDebuggerPresent | 0% |
| NtQueryInformationProcess | 0% |
| PEB BeingDebugged | ?% |
| NTGlobalFlag | ?% |

Except from the Windows API calls that where hooked by the vaccine we could also find other interesting system functions that were used by the malware by reading the logs. Table 5.7 shows the made calls on the *Clean system without helper* and how many percentage of the malware used that call.

Table 5.7: Used windows API's but not vaccinated

| **Windows API calls** | **percentage** |
|---|---|
| NtOpenFile | 95% |
| SetUnhandledExceptionFilter | 74% |
| NtQuerySystemInformation | 62% |
| FindWindowW | 13% |
| FindWindowExA | 2.0% |
| FindWindowExW | 2.7% |

NtOpenFile and findWindow-variants are two Windows API calls worth looking more into. They are used to check if a file exists and if a window is open. The Windows API calls reveals what file or window was searched for. To get as many hits as possible on interesting searches the following data are gathered from the 1923 malware executed on the *Clean system without helper*.

### NtOpenFile

There are no indications that malware looks for vaccination-points via this method. The data shows that this API call is used to get a filehandle to system DLL's and other files created by the malware. But there are three searches that stands out and where made by 33% of the malware:

- FileName->\\??\\STORAGE#Volume#{e7c5d359-57ff-11ea-a5f4-806e6f6e6963} #0000000000100000#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

- FileName->\\??\\SCSI#CdRom&Ven_VBOX&Prod_CD-ROM#4&2617aeae&0&010000 #{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

- FileName->\\??\\STORAGE#Volume#{e7c5d359-57ff-11ea-a5f4-806e6f6e6963} #0000000024400000#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

These are specific hardware identifiers that exists in the used virtual machine, two concerning hard drives and one concerning a cd rom from Virtual Box.

### findWindow-variants

This Windows API call exists in multiple variants (FindWindow, FindWindowW, FindWindowA, FindWindowExW, FindWindowExA) here they are looked at aggregated. With these calls you can either look for the window name that is observable in an open window or the classname of that window, most malware searched by classname rather than windowname, the Ex-variants looks for child-windows while the other looks for top level windows.

In our activity logs we found these searches. Some look for our own vaccine data-points and others can be used in the future of vaccines.

Table 5.8: FindWindow searches

| **Search** | **percentage of malware** |
|---|---|
| ClassName->ApplicationManager_DesktopShellWindow | 11.6% |
| ClassName->MS_AutodialMonitor | 5% |
| ClassName->MS_WebCheckMonitor | 5% |
| ClassName->Shell_TrayWnd | 3.2% |
| ClassName->EDIT | 2.0% |
| ClassName->SDM_Singleton | 2.0% |
| WindowName->SDM_Singleton | 1.8% |

The following findings are also relevant but all found in under one (1) % of the malware.

Table 5.9: FindWindow-variants searches found in under 1 % of malware

| **Search** | **percentage of malware** |
|---|---|
| ClassName->OLLYDBG | < 1% |
| ClassName->SysListView32 | < 1% |
| ClassName->PROCMON_WINDOW_CLASS | < 1% |
| ClassName->Progman | < 1% |
| ClassName->GBDYLLO | < 1% |
| ClassName->pediy06 | < 1% |
| ClassName->FilemonClass | < 1% |
| ClassName->RegmonClass | < 1% |
| WindowName->File Monitor - Sysinternals: www.sysinternals.com | < 1% |
| WindowName->Process Monitor - Sysinternals: www.sysinternals.com | < 1% |
| WindowName->Registry Monitor - Sysinternals: www.sysinternals.com | < 1% |
| WindowName->Piriform Defraggler | < 1% |
| WindowName->Defraggler | < 1% |

### Register data-points

None of our register vaccination data-points was searched for by NtOpenKey. One common search by malware was to check HKEY_LOCAL_MACHINE\\SOFTWARE \\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options for themselves.

# Chapter 6

# Discussion

This chapter will discuss the limitations, results and validity threats. The first section is the limitations and will contain what restricts the experiment. The second section is the results section where our result in regards to data-points, activity, and our expectations will be compared to the actual result. The last section will go through validity threats.

## 6.1 Limitations

### 6.1.1 Background review

One limitation of the background review is the fact that we might not have found all relevant related works. If we found all relevant related works it would have provided us with more information regarding data-points for the vaccine and other researchers' results to compare our results against. Since we chose to utilize the snowballing technique starting from Chen et al. [11] we might have ended up in one part of the research and missed other parts.

No data-points were extracted from our dataset to create the vaccine, because of this our vaccine is not optimized against our dataset and will not generate perfect results. However, we consider that this is a strength for the experiment since the same results should be achievable with other datasets as well. We have not used every general data-point available in our vaccine, for example, the ones we did not manage to implement: int3, KiUserExceptionDispatcher, and UnhandledExceptionFilter. Since we chose to create a general vaccine that does not have all data-points, it would mean that if our vaccine yields positive results it would be possible for future research to be conducted and make the vaccine perform even better. The reason why we did not create a vaccine with specific data-points is that it is not as proactive as a general vaccine. For the specific vaccine to work, new data-points from new malware has to be found and inserted continuously into the vaccine. We also noticed some papers in our related works, for example, Xu et al. [35] that has already tried the specific approach and proved that it works.

### 6.1.2 Dataset

The chosen dataset might not represent the general characteristics of the malware population. If the chosen dataset does not fairly represent the current malware population the conclusions drawn from the results can not be generally applicable.

The dataset contains Windows 32-bit executable files from a public dataset published in mid 2019 so the results and conclusions drawn from this experiment should not be extrapolated too far from this.

### 6.1.3 Environment

We wanted each virtual machine state to have the same base conditions to make the experiment as fair as possible. This is because we did not want anything except our vaccine to prevent the activity in the *Vaccine VM* and we did not want anything in the antivirus environment to be prevented by anything else but antivirus. Also, if we were to change some base parameters after executing the malware on the *Clean system without helper* this would make the results invalid. This can be visualized in table 4.1 and 4.2 in the environment setup. The comparison between the vaccine and antivirus program is only done with one antivirus program and does not reflect the effectiveness of other antivirus programs.

To make it possible to run and analyze the dataset within the time frame we had to automate the malware execution to make it more time-efficient. In our case, this means that we used virtual machines, which in comparison to running the malware on a physical computer means that there are some accidental vaccination data-points already implemented [28]. However, the biggest risk with this is the fact that our results would show vaccination to be less effective than it is. This also relates to the chosen sandbox environment which is being run inside of the VM and will introduce some data-points in the system that are impossible to avoid as well as the Virtual Machine itself. The malware might identify these which would create accidental vaccination which will change the results compared to if the malware were run without virtual machines or with another sandbox.

### 6.1.4 Experiment

According to the results, one would think that it is always positive to install the vaccine, however, we have not considered any potential performance-related issues with running the vaccine. Chen et al. [11] did consider this and they also used Detours and found the overhead to be 15s for intercepting a Windows API call. Neither have we considered the effect vaccine would have on legitimate software. One example of this would be gaming related programs which might contain anti-reverse and debug features to detect cheaters - these games might not run properly on a vaccinated computer.

## 6.2 Results

### 6.2.1 Data-points

This section aims to explore and discuss which data-points could be identified that the executed malware searched for to abort its malicious behavior. We will use the results presented in Vaccine data-points 5.3.4 together with the vaccine created and described by following our method presented in chapter 4 Method.

The focus of this experiment is to create a general vaccination and test the effectiveness of that. One thing to consider when reading these results is the fact that we are not able to correctly trace the impact of each vaccine data-point. For example, there may be and most certainly are methods for malware to check whether one of our data-points is in the system or not in a way that we did not monitor and therefore missed. There definitively exists more general approach vaccine data-points that we did not use in this experiment and also did not find searches for by the malware. However if anything this would mean that the numbers presented for how many of the malware did certain data-point searches could be more in reality.

When comparing our results to the related works, we found out that they check for the frequency of each Windows API call used by the malware instead of just counting once per malware, as we do. This makes it impossible to tell how many percentages of the malware checked that data-point since one malware can make multiple checks and all of them will be counted.

### Implemented data-points for the vaccine

We can see that the most used data point that we implemented in our vaccine was the Windows API call *IsDebuggerPresent* which was present in 35.9% of the malware.

This Windows API call is present in a high percentage of the malware although it is relatively easy for a real debugger to evade this detection. Since it does not take a lot of time to implement this Windows API call it might be worth it just to avoid the simplest debuggers. However, this Windows API call is very observable so anyone who uses this Windows API call can be detected and marked as a program that is using evasive behavior. The malware which uses this Windows API call probably does not belong in the category of the stealthiest malware, by implementing this we catch the lowest hanging fruit.

We found that 6% of the malware in our dataset used the FindWindow Windows API call to search for specific open windows. Any program exposed to the vaccine which uses this Windows API call would be led to believe that the searched-for window exists. In a real-world case, this vaccine would have to be created so that it would only step in and interfere when it detects certain searches that malware would search for.

Following this, we also have implemented two vaccines (PEB BeingDebugged and NTGlobalFlag) that we in this experiment can not validate whether they made a difference or not. These were chosen because they are stealthy and hard to detect checks for which might be used by malware that tries a bit extra to hide its evasive behavior. Our experiment does not evaluate whether they actively contribute to the vaccine, the focus is more on the fact that we want to create a general vaccine that is as good as possible. PEB BeingDebugged and PEB NTGlobalFlag were set up by executing several assembly code operations that changed values in the running malware process which indicates that a debugger is present. We can not measure the impact of these vaccine data-points since the malware can check for them by executing assembly operations which are much more transparent and difficult to monitor compared to Windows API calls. Future research on these parameters needs to be done where they are isolated to be able to answer how big of an impact they had on the vaccine. Although modifying PEB bits are mentioned by [11] and [37] no

data on how effective they are is presented.

The rest of our Windows API call vaccines did only get activated by less than one % of the malware and some of them by zero malware. Meaning that for this dataset the general vaccine could do without them but further research with larger datasets would be needed to fully decide their impact on malware as vaccine data-points.

One data-point that was not intentionally used but rather used as a result of implementing Windows API call hooks was the use of a helper-program which injects a DLL into the malware and executes the malware. When reviewing our method one item was different between the different systems, the helper-program did not execute the malware in every system. We therefore executed the malware with the helper-program in every system. This proved to generate a big difference in malware activity. However, in normal conditions, the malware most likely will be executed normally by a user i.e "explorer.exe" which means that executing the malware with the helper-program should be viewed as a vaccine. This is the reason we have included both "with helper" and "without helper" in our result sections. From the results, we can see that malware executed on the *Clean system without helper* did reduce their average activity by 73% with the helper compared to without. We do not know why this data-point had such a huge impact but a program started by a user usually has "explorer.exe" as its parent-process, whereas now "helper-program" is the parent-process which indicates that the program was not started normally by a user.

### Further improvements of the vaccine

We saw that 95% of the malware did use NtOpenFile which is an API call that is not malicious in itself self, however, 33% of the malware used NtOpenFile to search for specific hardware identifiers. These hardware identifiers are associated with our virtual machine. Specifically, the CDROM identifier gives away that this is a virtual box machine, and it might be that the malware can read out something from the hard drive identifiers. Further research is needed to include these data-points in a future version of the vaccine to find out what the malware are interested in finding out when using these hardware identifiers and trying to open them. It could be that the malware is trying to identify if the string "VBOX" is included in the CDROM identifier, which would be in line with what Chen et al. [11] says when mentioning that VMWare leaves well-known strings in the names of hardware devices. If it turns out that malware are looking for the "VBOX" string, that could possibly be inserted in hardware identifiers as a future vaccine data-point.

When analyzing the calls made by the malware we noticed that apart from NtOpenFile multiple other interesting calls were frequently used that we did not include in our vaccine. They are SetUnhandledExceptionFilter (74%), NtQuerySystemInformation (62%), and different types of FindWindow (16.7%). Implementing these in a future version of this vaccine would probably increase the effectiveness of the vaccine as they are checked by such large percentages of the malware. It should be said that these checks might be present in malware that also does other checks so if for example, they trigger on IsDebuggerPresent they might not execute these other checks.

### 6.2.2 Effectiveness Evaluation

This section will discuss how effective a general vaccine can be on its own and also how effective it is while deployed on the same system which has an antivirus solution running. As mentioned earlier the activity is measured as Windows API calls made by the malware and counted Windows API calls are only those made by the malware and its sub-processes which CAPE monitors by injecting a monitoring DLL.

Our results show that about 70% of the malware reduced their activity when using our vaccine compared to running a computer with Windows 10, completely without protection which can be seen in figure 5.1. However, in most cases, people tend to have an antivirus or some other protection against malware. We tested with antivirus and our results show that the activity decreases in 54% of the malware compared to when those malware were executed on the *Clean system without helper*, which can be seen in figure 5.2.

To answer the question of whether the vaccine would assist in a system where antivirus software is installed we executed the malware on a system that has antivirus activated and is vaccinated with our vaccine. The results from this execution shows that 85% of the malware decreased their activity compared to the *Clean system without helper*, which can be seen in figure 5.3. This is the highest number of malware that did decrease their activity - it beats the *Antivirus system without helper* and also the *Vaccinated system.*

Another parameter that also indicates that *Antivirus + Vaccine* is the best combination is the average number of activity each malware produces on the different systems. In table 5.3 we can see that compared with *Clean without helper*, *Vaccine* reduced the average activity with 84%, *Antivirus without helper* reduced average activity with 50%, and *Antivirus + Vaccine* reduced average by 89%. By performing the Friedman test and Nemenyi test we also concluded that there were significant statistical difference with a 95% confidence interval between the different systems.

Except using activity to indicate if our vaccine was relevant we used the number of process IDs spawned for each malware (PIDs). With vaccine compared to *Clean without helper* the number of PIDs was reduced by 26.9%, with *Antivirus without helper* compared to *Clean without helper* the number of PIDs was reduced by 35%, and with *Antivirus + Vaccine* the number of PIDs was reduced by 44%. We also have an average of PIDs for each system and they all did reduce their average compared to the *Clean system without helper*, *Vaccine* by 23%, *Antivirus without helper* by 29%, and *Antivirus + Vaccine* by 36%. The differences in number of PIDs also showed a significant statistical difference with a 95% confidence interval between the systems.

By looking at percentage of malware that reduced their activity, average malware activity, percentage of malware that reduced their number of PIDs, and the average number of malware PIDs, the best way to protect a computer from malware infection is to have both antivirus software installed and vaccination.

**Results compared to existing work** When attempting to determine how effective our vaccine is, it is important to understand how much activity could have been prevented with an optimal vaccine. An optimal vaccine would decrease activity in 100% of the malware, but this will probably never happen since all malware does not have evasive behavior. According to Lim et al. [18] 86% of their dataset contained evasive behavior, but it might be more since 53% were packed and harder

to analyze. Shi et al. [26] did also examine malware evasion techniques and they identified that each technique they found varies to being used between 2% and 58% of the malware. These kind of numbers are individual per every dataset but indicates the fact that not all malware uses evasive behavior.

Some researchers that have implemented and tested other variants of vaccination are Chen et al. [11], Wichmann et al. [32], and Xu et al. [35]. The findings of Chen et al. [11] shows that 25% of malware reduces its activity with their vaccine - our vaccine does reduce the activity by up to 70% but also includes more vaccine data-points than what was used by Chen et al. According to Wichmann et al.'s findings [32], 59% of their dataset contained data-points that prevents reinfection (specific approach). Xu et al. [35] also examines the specific approach where they create data-points targeted against reinfection. They found that they could impact the behavior of 12% of the malware with their vaccine. Both Xu et al. and Wichmann et al. have examined their datasets and created a specific vaccine targeted against their datasets and specifically against the reinfection of malware. They attempt to stop the malware from performing any malicious activity at all while our vaccine is not as strict in regards of stopping the activity completely. Since their methods differ so much compared to our method the results are difficult to compare but all results show that vaccination could be used to reduce malicious behavior.

**Expectations compared to reality** When reading up on related works we started to assume that our vaccine would result in a lower amount of activity compared to running the malware on a clean system. We did not expect the reduction to be as high as it is. In regards to how the vaccine would perform when combined with an antivirus program, we did not think it would help, we thought that the antivirus program would stop most of the malware before the vaccination could have any effect. But it turned out we were wrong about this assumption since our results show that the vaccine did in fact help when installed beside the antivirus program.

## 6.3 Validity threats

**Conclusion validity.** We do not have the needed infection details about every malware to evaluate whether the malware has infected the system or not to evaluate the effectiveness of the vaccine. Instead we measure Windows API calls and number of PIDs as the independent variables. This might not be the most optimal way but it is a reliable method that works with our limitations.

**Internal validity.** Some parameters that we could have missed during our experiment is during the background review, where we might not have found all relevant related works and thus not all relevant data-points. However this only means that there is room for improvement on the vaccine. Another parameter is the malware execution which might behave "randomly" and produce different outcomes on different systems. To deal with this problem we executed each malware twice on each system to get an average.

**Construct validity.** To deal with the fact that some malware randomly crashed, either in one of the two executions on one system or between different systems, every malware that was not capable of executing on all executions were removed from the dataset before analyzing the results.

**External validity.** Since virtual machines were used instead of bare metal computers in the experiment there inevitably exist accidental vaccination to some degree from the virtual machines that can affect the results. However this would only make the vaccination better than it seems to be, since the vaccination can mimic these accidental vaccinations.

# Chapter 7

# Conclusions and Future Work

This chapter will present the conclusions of our experiment as well as the most important parameters that were found during the experiment that would be interesting to change and test to see the differences in the results. We will also answer our research questions and lastly provide some ideas for future works.

## 7.1 Conclusion

There exist many data-points that are related to malware evasive behavior but from our results, the most important one is to execute malware from another process to reduce malware activity the most. We also saw that when comparing average malware activity between a *Clean system without helper*, *Vaccinated system*, *Antivirus system without helper*, and *Antivirus + Vaccine system* the *Antivirus + Vaccinated system* did have the lowest average and showed statistically significant different activity from the other systems when performing a Friedman test and then a Nemenyi test. When comparing the average number of malware PIDs between a *Clean system without helper*, *Vaccinated system*, *Antivirus system without helper*, and *Antivirus + Vaccinated system* the *Antivirus + Vaccinated system* did have the lowest average and showed statistically significant different number of PIDs from the other systems. With this we can answer the research questions.

**$RQ_1$: Which are the data-points that malware tries to identify to abort its malicious behavior?** Defining every data-point that could be used in a vaccine to prevent malicious behavior from malware is not possible within the given time frame. With the limitations of this experiment in mind, this result is not in any way absolute but answers the question in this specific setting. Each of our data-points is not tested on their own except for the helper-program. Therefore we can not say that a single data-point had any impact besides the helper-program. We can only tell the impact of the rest of the data-points together. When isolating the helper-program we noticed that it is an effective data-point to execute the malware from another process than the standard explorer.exe. By isolating and testing this we could find out if the chosen method for injecting the hooking-DLL and intercepting Windows API calls impacted the vaccine or not. Because of the positive results from using this method it might not be beneficial to change the method in future vaccines.

Even though we did not isolate every data-point we can still observe how many percentages of the malware did check for some of the data-points and argue the possibility that those specific data-points affected the malware behavior. From the results, we can see that the following data-points which all are Windows API calls

have been used and most likely did affect the malware behavior: IsDebuggerPresent, FindWindow, FindWindowA, OutputDebugString, and OutputDebugStringA.

**$RQ_2$: How effective can vaccination against existing malware be, both compared to a system with antivirus and as a complement on a system with antivirus?** According to our experiment vaccination has showed to be effective against malware. When comparing our *Vaccinated system* with a non-vaccinated system (the *Clean system without helper*), a difference in malware activity can be calculated. Our results show that 70% of the malware reduced their activity. When executing the same malware on a system with antivirus 54% of the malware decreased their activity. We also saw that when comparing the malware activity for the *Clean without helper*-, *Vaccine*-, and *Antivirus without helper*-systems they were statistically different with a 95% confidence interval which means that the *Vaccinated system* is the best of those three systems at reducing malware activity.

When testing the vaccine installed together with the antivirus program 85% of the malware reduced their activity compared to the *Clean system without helper*. This was also the system that had the lowest malware average activity, this system's malware activity is statistically different in comparison to the previously best system (the *Vaccinated system*) with a confidence interval of 95%. *Antivirus + Vaccine* is the best system in reducing malware activity of our created and tested systems.

In summary vaccination shows promising results in its capabilities of reducing malware activity, both on its own and together with an antivirus program. In this study, only one antivirus software is tested so future research should include more antivirus software. It would also be interesting to redo this experiment but without virtual machines to get rid of some accidental vaccination data-points.

## 7.2 Future Work

First of all, there is a possibility that our dataset favors our experiment, therefore it would be a good idea to execute the experiment on a different and larger dataset. By doing this the results would be strengthened if the same conclusions could be drawn. Another way that the experiment could be improved would be by testing with more antivirus software than just Windows Defender. Other antivirus software might catch different types of malware which might prove vaccine to be better or worse as a complement. One thing that would come from this kind of expansion is a list of antivirus programs that would be best to complement with a vaccine.

In our experiment, we saw that some malware looked for a keyboard string that corresponded to the "USA-keyboard". This could be worth looking into, what happens if you are from for example Sweden and only have the Swedish keyboard installed? Will this affect the malicious activity of the malware? As referred to earlier in our master thesis there is already a scenario where a situation like this is tested, but it is for German OS language installation [19]. Since we decided to shut off internet access for the VM, which is not a reasonable action to do on a computer that is in production, it would be interesting to see if there are any differences in the efficiency of the vaccine when the internet is available.

Another interesting aspect of our experiment is that ours is executed in a virtual environment which means that some vaccine data-points might be introduced auto-

matically. This is something that does not affect our outcome that much since all of our testing environments are in virtual environments meaning they all have the same data-points. But, it would be a great experiment to test the vaccine on physical hardware compared to virtual, one could even include some virtual environment data-points in the vaccine.

To get an understanding of what vaccine data-points has what effect in the vaccine, the experiment could be rerun where each data-point is isolated. Some examples that could be interesting are PEB BeingDebugged and NTGolbalFlag which this experiment has no method of measuring how often they were checked or how much impact they had. It would also be interesting to test how our vaccine affects legitimate programs in their execution. For example, if some application thinks that our DLL is doing something suspicious, they might not execute.

The vaccine approach as mentioned throughout the entire report has been branched into two different approaches, specific and general. We have chosen to create a general vaccine while Wichmann et al. [32] and Xu et al. [35] chose the specific approach. What would be interesting is to combine the two and use the general vaccine as a base and then update the data-points in real-time when new malware are found and examined for data-points. This kind of general + specific vaccine should be the most effective one and cover a reasonably large percentage of the malware population.

# References


[1] scikit_posthocs.posthoc_nemenyi_friedman — scikit-posthocs 0.6.1 documentation. https://scikit-posthocs.readthedocs.io/en/latest/generated/scikit_posthocs.posthoc_nemenyi_friedman/. Accessed: 2020-04-30.

[2] scipy.stats.friedmanchisquare — SciPy v1.4.1 Reference Guide. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.friedmanchisquare.html#scipy.stats.friedmanchisquare. Accessed: 2020-04-30.

[3] Virusshare. https://virusshare.com/. Accessed: 2020-01-29.

[4] Vaccinating systems against VM-aware malware. https://blog.rapid7.com/2013/05/13/vaccinating-systems-against-vm-aware-malware/, May 2013. Accessed: 2020-01-23.

[5] Latest Trickbot Campaign Delivered via Highly Obfuscated JS File - TrendLabs Security Intelligence Blog, August 2019. Accessed: 2020-03-31.

[6] Eduardo de O. Andrade, José Viterbo, Cristina N. Vasconcelos, Joris Guérin, and Flavia Cristina Bernardini. A Model Based on LSTM Neural Networks to Identify Five Different Types of Malware. *Procedia Computer Science*, 159(Journal Article):182–191, 2019. Publisher: Elsevier B.V.

[7] Pieter Arntz. Malware vaccination tricks: blue pills or red pills. https://blog.malwarebytes.com/cybercrime/malware/2017/08/malware-vaccination-tricks-blue-pills-or-red-pills/, August 2017. Accessed: 2020-01-22.

[8] Fabrizio Biondi, Thomas Given-Wilson, Axel Legay, Cassius Puodzius, and Jean Quilbeuf. Tutorial: An overview of malware detection and evasion techniques. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11244, pages 565–586, 2018. Conference Proceedings.

[9] BTH. Bth library. https://bibliotek.bth.se/databases, 2020. Accessed: 2020-04-06.

[10] Alexei Bulazel and Bülent Yener. A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web. In *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, pages 1–21. ACM, 2017. Conference Proceedings.

[11] Xu Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 177–186. IEEE, 2008. Conference Proceedings.

[12] ctxis. Cape: Malware configuration and payload extraction. `https://github.com/ctxis/CAPE`, 2020. Accessed: 2020-04-07.

[13] Kurt Fanning. Minimizing the cost of malware. *Journal of Corporate Accounting & Finance*, 26(3):7–14, 2015.

[14] Peter Ferrie. The "ultimate" anti-debugging reference. `https://anti-reversing.com/Downloads/Anti-Reversing/The_Ultimate_Anti-Reversing_Reference.pdf`, 2011. Accessed: 2020-03-20.

[15] Peter A. Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, Cambridge ; New York, 2012. p. 355-357.

[16] Hicham Hammouchi, Othmane Cherqi, Ghita Mezzour, Mounir Ghogho, and Mohammed E. Koutbi. Digging deeper into data breaches: An exploratory data analysis of hacking breaches over time. *Procedia Computer Science*, 151:1004–1009, 2019.

[17] Peter Kacherginsky. Flare vm: The windows malware analysis distribution you've always needed! `https://www.fireeye.com/blog/threat-research/2017/07/flare-vm-the-windows-malware.html`, 2017. Accessed: 2020-04-09.

[18] Charles Lim and Nicsen. Mal-EVE: Static detection model for evasive malware. In *2015 10th International Conference on Communications and Networking in China (ChinaCom)*, pages 283–288. IEEE, 2015. Conference Proceedings.

[19] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting Environment-Sensitive Malware. In *Recent Advances in Intrusion Detection*, volume 6961, pages 338–357. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[20] mcleanbyron. AppInit DLLs and Secure Boot - Win32 apps. `https://docs.microsoft.com/en-us/windows/win32/dlls/secure-boot-and-appinit-dlls`. Accessed: 2020-04-01.

[21] Microsoft. What is a dll? `https://support.microsoft.com/en-us/help/815065/what-is-a-dll`, 2019. Accessed: 2020-04-08.

[22] Microsoft. Memory protection. `https://docs.microsoft.com/en-us/windows/win32/memory/memory-protection`, 2020. Accessed: 2020-04-15.

[23] Inc MongoDB. The database for modern applications. `https://www.mongodb.com/`, 2020. Accessed: 2020-04-09.

[24] Oracle. Snapshots. `https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/snapshots.html`, 2020. Accessed: 2020-04-08.

[25] Kon Papazis and Naveen Chilamkurti. Detecting indicators of deception in emulated monitoring systems. *Service Oriented Computing and Applications*, 13(1):17–29, 2019.

[26] Hao Shi and Jelena Mirkovic. Hiding debuggers from malware with apate. In *Proceedings of the Symposium on applied computing*, volume 128005, pages 1703–1710. ACM, 2017. Conference Proceedings.

[27] J. E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005. Publisher: IEEE.

[28] Ming-Kung Sun, Mao-Jie Lin, M. Chang, Chi-Sung Laih, and Hui-Tang Lin. Malware Virtualization-Resistant Behavior Detection. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 912–917. IEEE, 2011. Conference Proceedings.

[29] Gabor Szappanos. The PlugX malware revisited : introducing "smoaler". `https://sophosnews.files.wordpress.com/2013/07/sophosszappanosplugxrevisitedintroducingsmoaler-rev1.pdf`, 2013. Accessed: 2020-03-31.

[30] Virustotal. Microsoft detours. `https://github.com/Microsoft/Detours`, 2019. Accessed: 2020-04-26.

[31] Virustotal. Virustotal. `https://developers.virustotal.com/v3.0/reference`, 2020. Accessed: 2020-04-21.

[32] A. Wichmann and E. Gerhards-Padilla. Using Infection Markers as a Vaccine against Malware Attacks. In *2012 IEEE International Conference on Green Computing and Communications*, pages 737–742. IEEE, 2012. Conference Proceedings.

[33] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy*, 5(2):32–39, 2007. Publisher: IEEE.

[34] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén, Sektionen för datavetenskap och kommunikation, and Blekinge Tekniska Högskola. *Experimentation in software engineering*, volume 9783642290442. Springer, Berlin, 2012 edition, 2012.

[35] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. AUTOVAC: Automatically Extracting System Resource Constraints and Generating Vaccines for Malware Immunization. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 112–123. IEEE, 2013. Conference Proceedings.

[36] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9854, pages 165–187, 2016. Conference Proceedings.

[37] Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. Using Hardware Features for Increased Debugging Transparency. In *2015 IEEE Symposium on Security and Privacy*, volume 2015-, pages 55–69. IEEE, 2015. Conference Proceedings.

# Appendix A

# Supplemental Information

Table A.1: Spoofed programs

| **Spoofed program** | **Process names** | **Filepaths** |
|---|---|---|
| Wireshark | wireshark.exe | C:\Program Files\Wireshark |
| Olly debugger | OLLYDBG.EXE | C:\Program Files (x86)\OllyDbg |
| Olly debugger 2 | ollydbg.exe | C:\Program Files (x86)\OllyDbg2 |
| Windows Debugger | windbg.exe | C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\, C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\ |
| x32dbg | x32dbg.exe | C:\Program Files\x64dbg\release\x32\ |
| x64dbg | x64dbg.exe | C:\Program Files\x64dbg\release\x64\ |
| cutter | cutter.exe | C:\ProgramData\chocolatey\lib\ cutter.flare\tools\cutter\ |
| IDAg | idag.exe | C:\Program Files (x86)\IDA Free\ |
| Ghidra | ghidraRun.bat | C:\ProgramData\chocolatey\lib\ ghidra.fireeye\tools\ghidra_9.0.2\ |
| IDA64 | ida64.exe | C:\Program Files\IDA Freeware 7.0\ |
| fakenet | fakenet.exe | C:\ |
| apimonitor-x64 | apimonitor-x64.exe | C:\ProgramData\chocolatey\bin\ |
| apimonitor-x86 | apimonitor-x86.exe | C:\ProgramData\chocolatey\bin\ |
| Process-Hacker | ProcessHacker.exe | C:\Program Files\Process Hacker 2\ |
| Sysinternal programs | procexp64.exe, Procmon.exe, Autoruns.exe | C:\ProgramData\chocolatey\lib\ sysinternals\tools |
| Resource-Hacker | ResourceHacker.exe | C:\ProgramData\chocolatey\lib\ reshack.portable\tools |
| Brupsuite | burpsuite.jar | C:\ProgramData\chocolatey\lib\ burp-suite-free-edition\tools\app |

| | | |
|---|---|---|
| Ghidra | ghidraproject.exe | C:\ |
| VBox | VBoxTray.exe, VBoxService.exe | C:\Windows\System32 |
| VMWare | VMacthlp.exe, VMtoolsd.exe | C:\Program Files\VMware\VMware Tools |
| VGAuth-Service | VGAuthService.exe | C:\Program Files\VMware\VMware Tools\VMware VGAuth |
| LordPE | LordPE.EXE | C:\ProgramData\chocolatey\lib\lordpe.flare\tools\ |
| PE Detective | PE Detective.exe | C:\Program Files\NTCore\Explorer Suite\ |
| PE-bear | PE-bear.exe | C:\ProgramData\chocolatey\lib\pebear.flare\tools\ |
| PEiD | PEiD.exe | C:\ProgramData\chocolatey\lib\peid.flare\tools\ |
| pestudio | pestudio.exe | C:\ProgramData\chocolatey\lib\pestudio.flare\tools\pestudio\ |
| PEview | PEview.exe | C:\ProgramData\chocolatey\lib\peview.flare\tools\ |
| PPEE | PPEE.exe | C:\ProgramData\chocolatey\lib\ppee.fireeye\tools\PPEE\PPEE(puppy) 1.12\ |
| Task Explorer-x64 | Task Explorer-x64.exe | C:\Program Files\NTCore\Explorer Suite\ |
| Task Explorer | Task Explorer.exe | C:\Program Files\NTCore\Explorer Suite\ |
| UniExtract | UniExtract.exe | C:\ProgramData\chocolatey\lib\uniextract2.fireeye\tools\UniExtract\UniExtract\ |
| SilkETW | SilkETW.exe | C:\Program Files (x86)\SilkETW\SilkETW\ |
| Scylla | Scylla_x86.exe, Scylla_x64.exe | C:\ |
| Immunity-Debugger | ImmunityDebugger.exe | C:\ |
| tcpdump | tcpdump.exe | C:\ |
| ga | ga.exe | C:\ |
| Agent-Simulator | AgentSimulator.exe | C:\ |
| Fiddler | Fiddler.exe | C:\Users\User\AppData\Local\Programs\Fiddler |
| Anubis | popupkiller.exe, exec.exe | C:\\exec\ |

| | | |
|---|---|---|
| Avira | Avira.Spotlight .UI.Application.exe, avshadow.exe, avguard.exe, sched.exe, avscan.exe, protectedservice.exe, Avira.Spotlight .Service.exe, avgnt.exe | C:\Program Files (x86)\Avira\Security |
| Avira | Avira.Systray.exe, Avira.ServiceHost.exe | C:\Program Files (x86)\Avira\Launcher |
| F-Secure | fshoster32.exe, fsdfwd.exe | Program Files(x86)\F-Secure\SAFE\ |
| F-Secure | fshoster64.exe, fsulprothoster.exe, fsorsp64.exe | Program Files(x86)\F-Secure\ SAFE\Ultralight\Ulcore\1581514205\ |
| Trend micro | uiWinMgr.exe | C:\Program Files\Trend Micro\Titanium\UIFramework |
| Trend micro | uiSeAgnt.exe, uiWatchDog.exe, uiUpdateTray.exe | C:\Program Files\Trend Micro\UniClient\UiFrmwrk |
| Trend micro | PtSessionAgent.exe, PtWatchDog.exe | C:\Program Files\Trend Micro\Titanium\plugin\Pt |
| Trend micro | coreFrameworkHost.exe, coreServiceShell.exe, AMSP_LogServer.exe | C:\Program Files\Trend Micro\AMSP |
| Trend micro | TmsaInstance64.exe | C:\Program Files\Trend Micro\AMSP\module\ 10011\8.1.1670\8.1.1670 |
| McAffee | McUICnt.exe | C:\Program Files\Common Files\McAfee\Platform |
| McAffee | mcapexe.exe | C:\Program Files\Common Files\McAfee\VSCore_19_5 |
| McAffee | McAWFwk.exe | C:\Program Files\Common Files\McAfee\ActWiz |
| McAffee | MfeAVSvc.exe | Files\McAfee\MfeAV |
| McAffee | mfefire.exe | C:\Program Files\Common Files\McAfee\SystemCore |
| McAffee | McCSPServiceHost.exe | C:\Program Files\Common Files\McAfee\CSP\3.1.286.0 |
| McAffee | McCBEntAndInstru.exe | C:\Program Files\Common Files\McAfee\Platform\CommonBuild |
| McAffee | McPvTray.exe | C:\Program Files\McAfee\MAT |

| | | |
|---|---|---|
| McAffee | mchost.exe | C:\Program Files\Common Files\McAfee\Platform\Core |
| McAffee | McInstruTrack.exe | C:\ProgramData\McAfee\McInstruTrack |
| McAffee | mfemms.exe | C:\Program Files\Common Files\McAfee\SystemCore |
| McAffee | MMSSHOST.exe | C:\Program Files\Common Files\McAfee\MMSSHost |
| McAffee | ModuleCoreService.exe | C:\Program Files\Common Files\McAfee\ModuleCore |
| McAffee | PEFService.exe | C:\Program Files\Common Files\McAfee\PEF\CORE |
| McAffee | ProtectedModuleHost.exe | C:\Program Files\Common Files\McAfee\ModuleCore |
| McAffee | mcshield.exe | C:\Program Files\Common Files\McAfee\AMCore |
| McAffee | uihost.exe | C:\Program Files\McAfee\WebAdvisor |
| McAffee | mfevtps.exe | C:\Windows\System32 |
| BITDefender | bdagent.exe | C:\Program Files\Bitdefender Antivirus Free |
| BITDefender | ProductAgentService.exe, ProductAgentUI.exe | C:\Program Files\Bitdefender Agent |
| Windows Defender | MsMpEng.exe, MpCmdRun.exe, starter.exe | C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2001.7-0 |
| Windows Defender | smartscreen.exe, SecurityHealthHost.exe, SecurityHealthService.exe | C:\Windows\System32 |
| BullGuard | BullGuard.exe, BullGuardCore.exe, BullGuardHelper.exe, BullGuardSentry.exe, BullGuardTray.exe | C:\Program Files\BullGuard Ltd\BullGuard\ |
| MalwareBytes | mbamtray.exe, mbam.exe | C:\Program Files\Malwarebytes\Anti-Malware |
| Panda | PSUAConsole.exe, PSUAMain.exe | C:\Program Files (x86)\Panda Security\Panda Security Protection\ |
| Panda | AgentSvc.exe | C:\Program Files (x86)\Panda Security\Panda Devices Agent\ |

| | | |
|---|---|---|
| AVAST | AvastUI.exe, wsc_proxy.exe, AvLaunch.exe, aswidsagent.exe, AvastSvc.exe | C:\Program files\AVAST Software\Avast\ |
| AVG | AVGUI.exe, aswidsagent.exe, wsc_proxy.exe, AvLaunch.exe | C:\Program Files\AVG\Antivirus\ |
| Kaspersky | avp.exe, ksdeui.exe, avpui.exe, ksde.exe, klwtblfs.exe, wmifw.exe | C:\Program Files (x86)\Kaspersky Lab\Kaspersky Anti-Virus 20.0 |

Table A.2: Created registries

| **Registry** | **value** |
|---|---|
| HKEY_CURRENT_USER \Software \McAfee \FileInsight | C: \\Program Files (x86) \\FileInsight |
| HKEY_LOCAL_MACHINE \SOFTWARE \Oracle \JavaDeploy | |
| HKEY_LOCAL_MACHINE \SOFTWARE \Oracle \VirtualBox Guest Additions | "Version"="6.0.10", "VersionExt"="6.0.10", "Revision"="132072", "InstallDir"="C: \Program Files \Oracle \VirtualBox Guest Additions" |
| HKEY_CURRENT_USER \Software \Sysinternals \AccessChk | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Active Directory Explorer | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ADInsight | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Autologon | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \AutoRuns | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \BGInfo | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \CacheSet | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ClockRes | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Coreinfo | "EulaAccepted"=dword:00000001 |

| | |
|---|---|
| HKEY_CURRENT_USER \Software \Sysinternals \Ctrl2cap | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \DbgView | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Desktops | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Disk2Vhd | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Diskmon | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \DiskView | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Du | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \EFSDump | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \FindLinks | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Handle | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Hex2Dec | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Junction | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \LdmDump | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ListDLLs | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \LoadOrder | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Movefile | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PageDefrag | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PendMove | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PipeList | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Portmon | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ProcDump | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Process Explorer | "EulaAccepted"=dword:00000001 |

| | |
|---|---|
| HKEY_CURRENT_USER \Software \Sysinternals \Process Monitor | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsExec | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \psfile | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsGetSid | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsInfo | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsKill | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsList | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsLoggedon | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsLoglist | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsPasswd | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsService | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsShutdown | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \PsSuspend | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \RamMap | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \RegDelNull | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Regjump | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Regsize | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \RootkitRevealer | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Share Enum | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ShellRunas | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \SigCheck | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \SigCheck \VirusTotal | "VirusTotalTermsAccepted"=dword:00000001 |

| | |
|---|---|
| HKEY_CURRENT_USER \Software \Sysinternals \Streams | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Strings | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Sync | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \System Monitor | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \TCPView | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \VMMap | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \VolumeID | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Whois | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \Winobj | "EulaAccepted"=dword:00000001 |
| HKEY_CURRENT_USER \Software \Sysinternals \ZoomIt | "EulaAccepted"=dword:00000001 |
| Computer \HKEY_LOCAL_MACHINE \HARDWARE \DESCRIPTION \System \BIOS | SystemManufacturer = VMware, Inc. , SystemProductName = VMware Virtual Platform |
| HKEY_CLASSES_ROOT \Applications \VMwareHostOpen.exe \shell \open \command<br><br>@="Open file in VMware host" | @=" \"C: \\Program Files \\VMware \\VMware Tools \\VMwareHostOpen.exe \" –file \"HKEY_CLASSES_ROOT \VMwareHostOpen.AssocFile |
| HKEY_CLASSES_ROOT \VMwareHostOpen.AssocFile \shell \open \command<br><br>@="Open URL in VMware Host" | @=" \"C: \\Program Files \\VMware \\VMware Tools \\VMwareHostOpen.exe \" –file \"HKEY_CLASSES_ROOT \VMwareHostOpen.AssocURL |
| HKEY_CLASSES_ROOT \VMwareHostOpen.AssocURL \shell \open \command<br><br><br>@=" \\\\VMware-host \\Shared Folders" | @=" \"C: \\Program Files \\VMware \\VMware Tools \\VMwareHostOpen.exe \" –url \"HKEY_CURRENT_USER \Software \VMware, Inc. \VMware Tools \Hgfs Usability \MRU RootShare |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware CAF | "Path"="C: \\ProgramData \\VMware \\VMware CAF \\" |

| | |
|---|---|
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware Drivers | "VMciHostDevInst"="ROOT \\VMWVMCIHOSTDEV \\0000", "VMci.status"="1\|1.9.8.6.0.1\|oem4.inf", "vsock.status"="1\|4998961.9.8.12.0.1\|C: \\Windows \\system32 \\DRVSTORE \\vsock_7B561B05841C6140673D7ECE1379E 18A4C891591 \\vsock.inf", "vsockSys.status"="1\|4998961.9.8.12.0.1\|C: \\Windows \\system32 \\DRVSTORE \\vsock_7B561B05841C6140673D7ECE1379E 18A4C891591 \\vsock.inf", "vsockDll.status"="1\|4998961.9.8.12.0.1\|C: \\Windows \\system32 \\DRVSTORE \\vsock_7B561B05841C6140673D7ECE1379E 18A4C891591 \\vsock.inf", "efifw.status"="1\|1.1.0.0.0.1\|oem5.inf", "VMxnet3.status"="1\|1.1.8.10.0.1\|oem6.inf", "pvscsi.status"="1\|1.1.3.10.0.1\|oem7.inf", "VMusbmouse.status"="1\|1.12.5.7.0.1\|oem8.inf", "VMmouse.status"="1\|1.12.5.7.0.1\|oem9.inf", "VMmemctl.status"="1\|4998961.7.4.2.0.1\|C: \\Windows \\system32 \\DRVSTORE \\VMmemctl_5505643C60FE0A4FB0ADAA65E0280A66BE F2AADC \\VMmemctl.inf", "VMhgfs.status"="1\|4998961.11.0.34.0.1\|C: \\Windows \\system32 \\DRVSTORE \\VMhgfs_09F56BC3D38F0BF13DBD88A88D 5273235AEC8D7D \\VMhgfs.inf", "VMrawdsk.status"="1\|4998961.1.1.1.0.1\|C: \\Windows \\system32 \\DRVSTORE \\VMrawdsk_7D8302DFF9F9DDFF719D9C74D79C 26DE6C03FD6D \\VMrawdsk.inf", "svga_wddm.status"="1\|1.8. 16.1.24.1\|oem10.inf" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware Tools | "InstallPath"="C: \\Program Files \\VMware \\VMware Tools \\", "LastBoot"="virtual" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware Tools \VMUpgradeHelper | "NetConfigSaved"=dword:00000001 |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware Tools \VMUpgradeHelper \Adapters \00:0C:29:29:78:E6 DEV_100F REV_01 \\4 | "PNPDeviceID"="PCI \\VEN_8086 SUBSYS_075015AD BBF9765 |

| | |
|---|---|
| 0 | 0888", "InterfaceGUID"="4508A989-D6EF-4C89-B16C-24C7D65FCD50", "ServiceName"="e1g60" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware Tools \VMUpgradeHelper \Adapters \00:0C:29:3E:C8:B9<br>DEV_100F<br>REV_01 \\4<br>0 | "PNPDeviceID"="PCI \\VEN_8086<br><br><br><br>SUBSYS_075015AD<br>BBF9765<br>0888", "InterfaceGUID"="4508A989-D6EF-4C89-B16C-24C7D65FCD50", "ServiceName"="e1g60" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMware VGAuth | "PreferencesFile"="C: \\ProgramData \\VMware \\VMware VGAuth \\vgauth.conf" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMwareHostOpen \Capabilities | "ApplicationDescription"="Open URLs or files in the VMware Host." |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMwareHostOpen \Capabilities \FileAssociations | ".htm"="VMwareHostOpen.AssocFile",<br>".shtml"="VMwareHostOpen.AssocFile",<br>".xhtml"="VMwareHostOpen.AssocFile",<br>".xht"="VMwareHostOpen.AssocFile",<br>".html"="VMwareHostOpen.AssocFile" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMwareHostOpen \Capabilities \Startmenu | "StartmenuInternet"="VMwareHostOpen.exe" |
| HKEY_LOCAL_MACHINE \SOFTWARE \VMware, Inc. \VMwareHostOpen \Capabilities \UrlAssociations | "sftp"="VMwareHostOpen.AssocUrl",<br>"ftp"="VMwareHostOpen.AssocUrl",<br>"news"="VMwareHostOpen.AssocUrl",<br>"http"="VMwareHostOpen.AssocUrl",<br>"mailto"="VMwareHostOpen.AssocUrl",<br>"feed"="VMwareHostOpen.AssocUrl",<br>"https"="VMwareHostOpen.AssocUrl",<br>"telnet"="VMwareHostOpen.AssocUrl",<br>"ssh"="VMwareHostOpen.AssocUrl" |
| HKEY_CURRENT_USER \Software \VMware, Inc. \VMware Tools \Hgfs Usability \MRU RootShare | @=" \\\\VMware-host \\Shared Folders" |


Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden