UNIVERSITY OF AMSTERDAM

System & Network Engineering MSc
Research Project II

Machine Detectable Network Behavioural Commonalities for Exploits and Malware

Alexandros Stavroulakis
alexandros.stavroulakis@os3.nl

February 7, 2016

Abstract

In this project we study the detectability of patterns in the network behaviour of automatically generated malware.
The computer security tool, Metasploit Framework, offers the feature of automatically developing malicious software for penetration testing purposes. In this paper we theorize that a quick and automated procedure, without much user input, should lead to a certain level of predictability in the behaviour of malware. Our first aim is to determine the validity of this theory. Secondly, we try to discover evidence that suggests the existence of patterns in the manner of operation of such software. Finally, we present our findings of predictable behaviour accompanied by graphical traffic representation. The methodology followed throughout the course of this research, the software tools used and the results acquired during the process of completing this project, are described in detail. Any ethical implications are documented and presented accordingly.

Acknowledgements

At this point I would like to thank my supervisor, Mr. Adrianus Warmenhoven, for his assistance throughout the course of this research.

Contents

Chapter 1

Introduction

Malicious software, more commonly known as malware, has been a menace since the decade of 1980[1]. Ever since then – and as technology evolves – new ways of developing and using malware for more and more targets have made their appearance. According to CNN, nearly a million new malware threats are released every day[2].

In order to mitigate attacks by malicious software, Anti-Virus (AV) software was developed. With the first products being released in 1987[3], AV software has come a long way and currently possesses multiple ways of detection. The different detection methods can be broken into the following categories:

- **Signature Based Detection** - AV software uses a database of known malware signatures with which it compares the contents of certain files. A signature can be a unique hash, or binary string, which can be used to identify certain malware.
- **Heuristic Based Detection** - AV software searches a file for evidence of malicious code. Usually works in combination with the Signature Based technique.
- **Sandbox Detection** - During this process, AV software executes a file in a contained and virtualized environment. It records its actions and in the end reaches a consensus on whether it is malicious or not.
- **Behavioural Based Detection** - AV software looks into the behaviour of a file once it has been executed, in order to determine if it is malicious.
- **Data Mining Based Detection** - The most recent addition to malware detection solutions[4, 5]. AV software attempts to find patterns in large amounts of data, which can then be used to identify similar behaviour in other collected data. This technique takes the Behavioural Based Detection on a larger scale.

This report will relate more to the subject of Behavioural Based Detection and Data Mining Based Detection, as we will be looking into network behaviour of malware once it has successfully infected a system. The main problem is that newly created malware can pass undetected by a Signature Based Detection technique, as its signature will most probably not be included in the known signatures database.

The rest of this chapter is dedicated to describing in detail the problem at hand, providing the motivation behind this research and mentioning any ethical implications that may arise from our study. The second chapter will discuss the research question and scope of the project, while the third chapter will present any previous work on this domain. The fourth chapter discusses the approach and methodology that were used and the fifth chapter demonstrates the results that have been produced by our research. The sixth chapter summarizes our conclusions and the final chapter proposes ways to improve the performed work.

1.1 Problem Description

The Metasploit Framework[6] (MSF) is an open source computer security tool used in penetration testing and vulnerability assessment. It was created in 2003 by H. D. Moore and is currently part of the Metasploit Project as provided by Rapid7[7]. Part of MSF is an open source tool called Armitage[8], designed by Raphael Mudge and released in 2010, which is used for cyber attack management and Red Team collaboration, and serves as a Graphical User Interface for the Framework. One of the most popular features of Metasploit is the development and execution of malicious exploitation code for penetration testing purposes. This gives users the ability to, essentially, develop malware in a quick and automated way, without the requirement of programming experience. This procedure will be further discussed in Chapter 4.

This leads to the generation of new pieces of malicious code, which can prove to be hard to detect by current AV software. However, as this procedure is mostly automated, it is hypothesized that it could cause a certain amount of predictability in the behaviour of such malware.

1.2 Motivation

The aim of this project is to research whether the aforementioned malware generation procedure will produce malicious code, which will have predictable network behaviour, when used by inexperienced users ("hackers"). The reasoning behind targeting such users is the premise of them being more

reliant and dependent upon an automated procedure, rather than developing something from scratch. Since a more experienced user would be able to tweak and modify this procedure, pattern recognition would prove to be more difficult.

If our research could lead to pattern discovery based on the network behaviour of malware and result to their identification by characteristics such as packet transfer ratio, packet size, sequence of ports, payload sizes etc., an additional way of detection would be provided to the currently used detection techniques of AV software.

Furthermore, we will focus on the use of Armitage, the GUI of Metasploit, as its point-and-click manner of operation makes it easier to orientate between commands during this procedure. Therefore making it a more ideal tool to be used by lower level hacking enthusiasts.

1.3 Ethical Concerns

During the course of this research no physical machines or actual user data will be targeted. The experiments will take place in a virtual environment and the appropriate care will be taken during the stage of malware developing and infection, so that the generated malware will not leave the safe space of testing.

Chapter 2

Research Question

From the overall discussion of the significance of this research and to fully explore our hypothesis, the following research question has been designed:

Is it possible to detect the presence of malicious software, generated by Armitage, by identifying its network behaviour?

The results of this research will fit in one of two categories. Proof of knowledge, that automatically generated malware can be detected solely based on their behaviour on the network; which may help a future implementation of a broad-spectrum detector for basic malware generated by hobbyists. Alternatively, the result may be that Armitage, and as an extension Metasploit, in their current form, are enough to create sophisticated malware that can evade detection.

2.1 Scope

As previously explained, this research targets users without much experience in the domains of malware developing or penetration testing. Users who either want to try this procedure out of curiosity, or it is their hobby, or they are taking their first steps in learning about this subject. Therefore, the scope of this project regarding the malware generation procedure needs to be strict.

We will be working under the premise that the targeted users will be more inclined to follow methods of attack that are not too complicated and have a high success rate. In other words, commonly used examples of such attacks, which are easy to find on the Internet and thoroughly explained. The most popularly used methods will be selected to be part of the research methodology.

Moreover, methods such as Deep Packet Inspection will not be used. The network behaviour of malware can be characterized as 'metadata', which has no concerns whether the content is encrypted or not. Therefore, deeming such methods not necessary.

Chapter 3

Related work

The majority of previous work on the subject of malware pattern matching is based on signature detection and code analysis. However, no particular research on malware generated by Armitage (and Metasploit) has been discovered.

One example is the research by Ferdiansyah et al., a collaboration between the Sam Houston State University and Firat University, which focuses on string matching methods for identification of same family malware[9]. The researchers attempted to find similarities between malware that are considered to be byproducts of particular malware groups. Another example is the work of Christodorescu et al., of the University of Wisconsin and the Carnegie Mellon University[10]. Their paper presents a malware detection algorithm, which incorporates instruction semantics to detect malicious program traits.

The most relevant research to this topic was by Liang et al., of the Wuhan University and Remnin University[11]. The research was inspired by advanced persistent threat (APT) attacks. The authors, using Decision Tree and Naive Bayes machine learning techniques, created a novel model aimed at detecting unknown Trojans equipped by APT attacks, based on a software's network behaviour. While the idea of Liang et al. certainly resembles the one of this project, the actual targets are different. In addition, our research does not intend to provide a tool against such threats, rather than test a hypothesis, which will result to a Proof-of-Knowledge.

3.1 Metasploit Framework Payloads

Section 1.1 briefly described the Metasploit Framework and its GUI wrapper Armitage. The Framework includes a large database of available exploits and payloads for penetration testing purposes. The term exploit refers to a piece of software, which takes advantage of a system's vulnerability. While

the term payload, in Metasploit, refers to an exploit module representing the malicious code, which we want to be executed on the victim's system.

In the context of this research, the automatically generated malicious software refers to the MSF payloads. The payloads in MSF can be split into two categories[12, 13]:

- **Staged** - Staged payloads make use of Stages and Stagers. Upon infection, the Stagers create a connection to the attacker and begin to pull the payload components that make up the Stages. That way, the first part of the attack is the setup of a communications mechanism and the second can include a higher level malware, which can infect the victim and perform its actions.
- **Stageless** - These payloads can be characterized as standalone executables, combining the above procedure in a single file. They require no additional files to run. They can create a connection back to the attacker, spawn a new process, or create a new user etc.

During the experimentation phase of this research, both types of payloads were tested.

Chapter 4

Approach and methodology

In order to examine our hypothesis the following approach was taken. A secure testing environment was designed where the experiments could take place on a virtual machine, which was then reverted back to a clean state after each trial. Several ways of generating malware were attempted with the use of Armitage, and a number of different tests was ran. During these tests, the network traffic was captured and then analyzed in order to determine the existence of subtle or not-so-subtle fingerprints.

4.1 Testing Environment

More specifically the testing environment was designed as explained below:

- **Host Machine** - An Ubuntu 15.10 host machine was used as provided by the MSc programme's facilities.
- **Virtual Environment** - Oracle VirtualBox 5.0.12 r104815 was chosen as the, at the time, latest available virtualization environment[14].
- **Victim** - A Windows 7 Service Pack 1 virtual machine was set up as the victim and testbed of the generated malware. This particular operating system was chosen as a victim, as it is an older version of Windows, insecure when not updated, and still supported by Microsoft and used by many around the world[15].
- **Attacker** - A Kali Linux 2.0 virtual machine was set up as the attacker. It was chosen due to its native support of the Metasploit Framework. All malware was generated on this machine. The instance was updated to its latest version after the installation and remained the same for the entire course of this research[16].
- **Armitage** - As previously mentioned, Armitage was used to launch the attacks. The latest available version was used, 1.4.11 released in August 2015[8].

Once the virtual machines were set up, the networking between them and the host machine was set up as a Host-Only network. No updates or upgrades were made to the machines during the course of the tests.

4.2 Malware Generation

Section 2.1 described the scope of this research. Since the target of this project is inexperienced users, a small part of this research was dedicated to identify the most probable ways of attack, which can be launched by them. The premise was that a lower level attacker would try to use a method of attack, which is easy to follow and sure to work. The developers of Metasploit claim that the most popular choices have been the `reverse_tcp` and `reverse_http(s)` payloads[17], which use the respective protocols in their titles as transport types to communicate between the victim and the attacker. The word 'reverse' in their title implies the fact that the attacker is waiting for the victim to connect back to them in order for a connection to be established.

The above payload types make use of the Meterpreter payload[13, 18]. Meterpreter is a payload, which uses in-line DLL injection in order to create a connection between the victim and the attacker; and provide the attacker with an interface, which can be used to dynamically interact with the victim.

The process of generating a malicious executable file is fairly simple. The first step would be for the attacker to have network access to the victim. With the second step being to scan the machine for vulnerabilities and discover possible ways of exploiting it. In our case, as the victim was chosen to be an older implementation of Windows and we already knew the main types of attack to be used, the second step was initially not necessary.

As shown in image 4.1, it is simply a matter of selecting a payload from a list and launching it. Once the target has been discovered and selected, the following action to be taken would be to set up the payload that is going to be used. The most important fields to take notice of would be the `LHOST` and `LPORT`, which specify the attacker's IP address and the port, which will be used in order to establish a connection with the victim. At this point it should be stressed that during all of the tests the attacker will always use address `192.168.56.102` and port `4444`.

The next thing to notice would be the output field, which specifies what the result of this dialogue will be. Initially, the attacker would choose 'exe' in order to produce the file, which will infect the victim and click 'Launch'. Then repeat the procedure as described above and in the final step, instead

of selecting 'exe', select 'multi/handler'. This results in the creation of a handler, which is listening at the aforementioned IP address and port, waiting for the victim to connect. This procedure basically describes the creation of a 'backdoor' type of malware and is one of the most commonly used examples of Metasploit attacks.

In regard to the infection part of this process, the attacker would need to think of a way to infect the victim. During the tests, an Apache2 instance was used. The victim downloaded and ran the executable via the browser. Once the file has been executed, the connection between the attacker and victim is established.

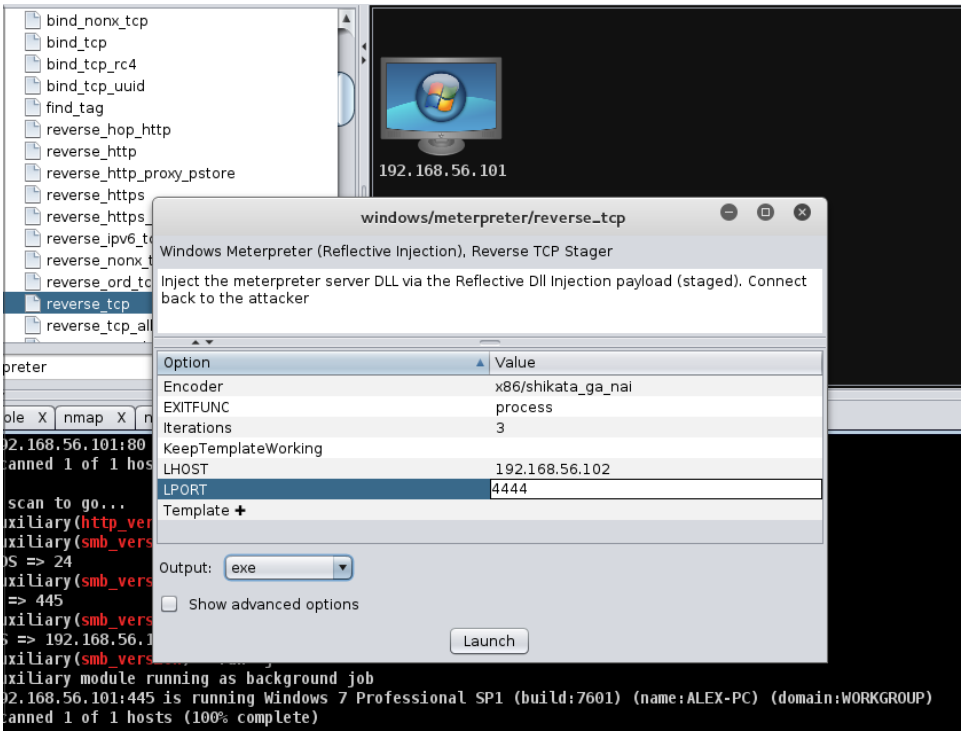


Figure 4.1: How to generate a malicious executable file with the `reverse_tcp` payload in Armitage

This procedure was used in order to generate malware for each test. The creation method remains the same, even for the different payloads that were used during this project, such as, among others, `reverse_http` and `reverse_https`.

4.3 Capturing Traffic

In each test, while the attacks were taking place the traffic was captured. This process was split into two parts; first, during the infection and while commands were issued to the victim from the attacker's side. And second, while the connection between the two virtual machines was idle, in order to determine how the malware behaves in the network while it is still running on the victim. To capture and analyze the traffic, WireShark 2.0 was used.

Once the traffic was captured, we moved on to the analysis part of the experiments. The traffic was filtered according to the transport type of each payload and visualized. The goal of this part was to detect any possible repetition in the transmission of packets, packet lengths, sequence of ports etc. In general, an attempt to find any evidence of predictability in the network behaviour.

To make sure that a well rounded experiment was performed, three types of tests were ran for each type of malware that was generated. The inaugural test was a simple capture of traffic from the virtual machines, while the victim was in a clean state and no attack was taking place. This was done for comparison reasons, in order to see the difference between clean and 'infected' traffic. Next, was the above described procedure, of capturing the traffic during the attack and capturing the traffic during an idle connection between the attacker and the victim. Lastly, Internet connection was allowed on the victim side, in order to simulate normal browsing and an attack was launched at it through the local network. The traffic was captured on both interfaces so that a better comprehension could be achieved of how such an attack would look like under normal circumstances of operation.

Chapter 5

Results

The results of the antecedently described experiments will be presented in this chapter. The chapter is divided in the sections representing the most popular payloads, as explained previously, `reverse_tcp` and `reverse_http(s)`. The `reverse_http` and `reverse_https` payloads are grouped together due to their similar behaviour, as it was detected by the experiments and will be discussed further below. Section 5.1 will provide a small analysis of the clean traffic captures. At the end of the chapter, the possible implications of evasion techniques will be pointed out.

However first, it should be noted that both Staged and Stageless types of payloads were tested. As their post-infection network behaviour was identical, no separate sections have been dedicated to each one, rather they are described as one.

5.1 Clean State

In order to know what the normal traffic representation on a network between two machines looks like, and use this as base of comparison with the later tests, the normal network behaviour of the virtual machines was observed. Two examples follow with the traffic representation between the idle virtual machines, and simple Internet browsing.

Image 5.1 shows the first test of traffic capture between the two idle virtual machines. The capture lasted one hour. The traffic consists of network announcements, DHCP and ARP requests and acknowledgments, etc. Traffic that has mostly to do with the operation of the Windows 7 machine.

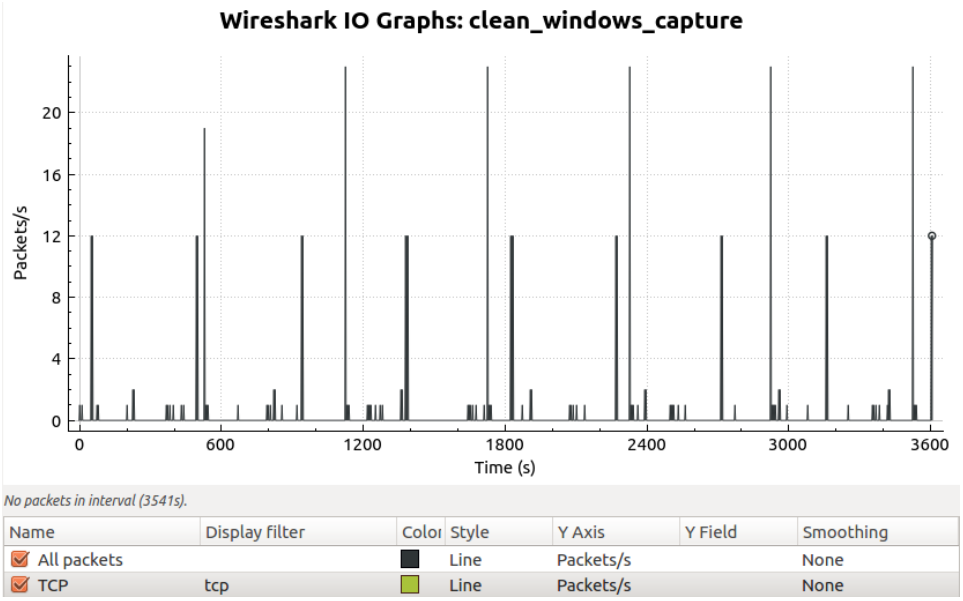


Figure 5.1: Idle network traffic capture. Representation of solicitations on the LAN. No TCP traffic.

The capture represented in image 5.2 lasted 15 minutes, during which a few webpages were visited and a YouTube video was played. In a simulation of regular network usage of a machine, we can observe that vast majority of the traffic consists of TCP traffic (WireShark includes HTTP traffic in its TCP filtering).

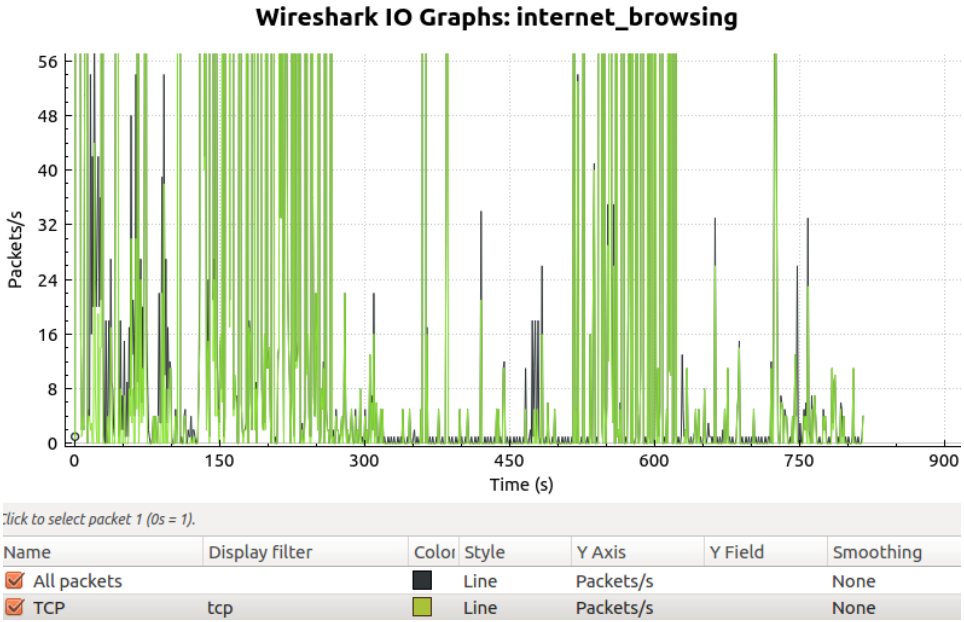


Figure 5.2: Internet browsing. TCP traffic most prominent.

5.2 `reverse_tcp`

As it was briefly mentioned before, this payload is used to generate a 'backdoor', which will then initiate a TCP connection between the victim and the attacker. A listener is set up at the side of the attacker and awaits for the victim to connect to it, using the IP address and port number that were provided to it upon creation time.

After several trials, it was made clear that there exists a form of repetition in the behaviour of malware generated with the automated use of this payload. Further explanation will be given in the following subsections.

5.2.1 Contained Environment

As shown in image 5.3, there occurs a transmission of packets in a fixed period of time. More specifically, five TCP packets are exchanged between attacker (port `4444`) and victim (port `49163`) every sixty seconds. In all tests that were ran, it was observed that each transmission round starts and ends with a packet from the attacker. In addition the same packet types ('PSH, ACK' and 'ACK') and packet lengths are repeated in the same order, in every round. This results to a transmission of 652 Bytes every sixty seconds.

No.	Time	Source	Destination	Protoco	Length	Info
1	0.000…	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=1 Ack=1 Win=636 Len=154
4	0.053…	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=1 Ack=155 Win=256 Len=74
5	0.053…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=155 Ack=75 Win=636 Len=0
6	0.053…	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=75 Ack=155 Win=256 Len=154
7	0.053…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=155 Ack=229 Win=639 Len=0
46	60.06…	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=155 Ack=229 Win=639 Len=154
47	60.11…	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=229 Ack=309 Win=255 Len=74
48	60.11…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=309 Ack=303 Win=639 Len=0
49	60.11…	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=303 Ack=309 Win=255 Len=154
50	60.11…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=309 Ack=457 Win=642 Len=0
82	120.3…	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=309 Ack=457 Win=642 Len=154
83	120.4…	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=457 Ack=463 Win=254 Len=74
84	120.4…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=463 Ack=531 Win=642 Len=0
85	120.4…	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=531 Ack=463 Win=254 Len=154
86	120.4…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=463 Ack=685 Win=644 Len=0

Figure 5.3: Transmission of 5 packets every 60 seconds.

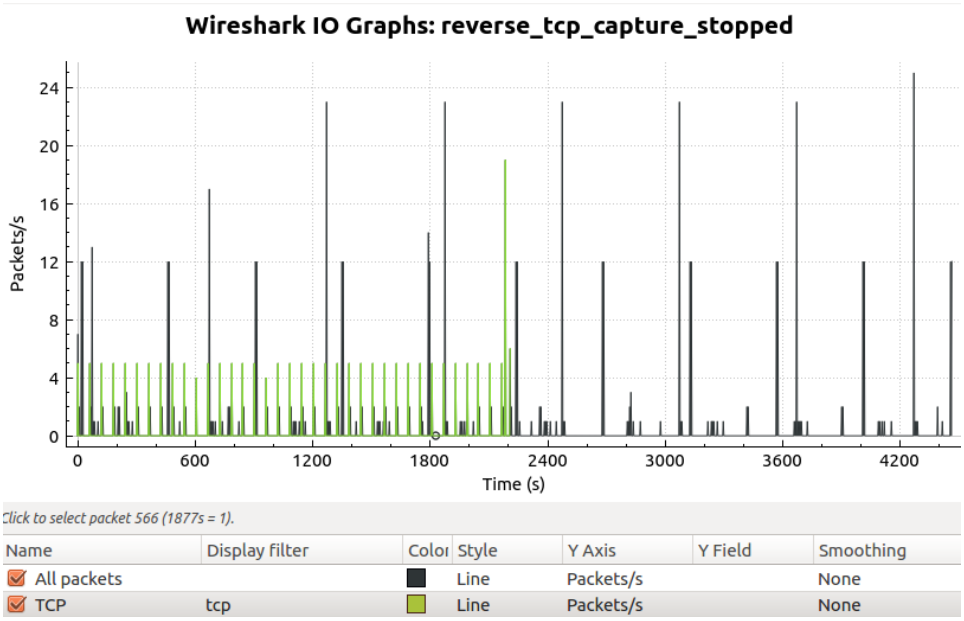


Figure 5.4: Graphical representation of the traffic. Malware behaviour shown with green colour.

In images 5.4 and 5.5 we can see the graphical representation of the traffic. The capture lasted circa 70 minutes. The Y axis shows the amount of packets and the X axis shows the time in seconds. In image 5.4 the malicious traffic is shown with a green colour, while the following image focuses solely on the malicious traffic. It is important to notice that in every test, when the session between the attacker and the victim ends, an increase in the amount of packets was witnessed; ranging in all tests between ten to twenty packets. This was due to multiple 'reset' (`RST`) packets being sent, sometimes leading to a spurious retransmission, ending, of course, with a packet containing 'finish'(`FIN`) flag. Once the session ended, the malware moved to its exit

function and stopped its operation, leading to, as expected, no network presence whatsoever.

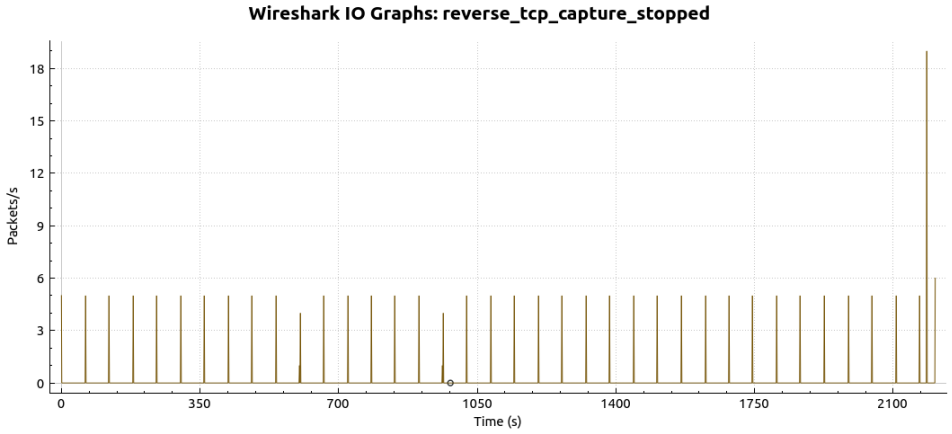


Figure 5.5: Focus on the part of the traffic generated by the malware.

Image 5.6 depicts the network traffic representation of a stageless payload. As can be seen, it is identical to the one above. The same applied between the staged and stageless types of payloads in Section 5.3.

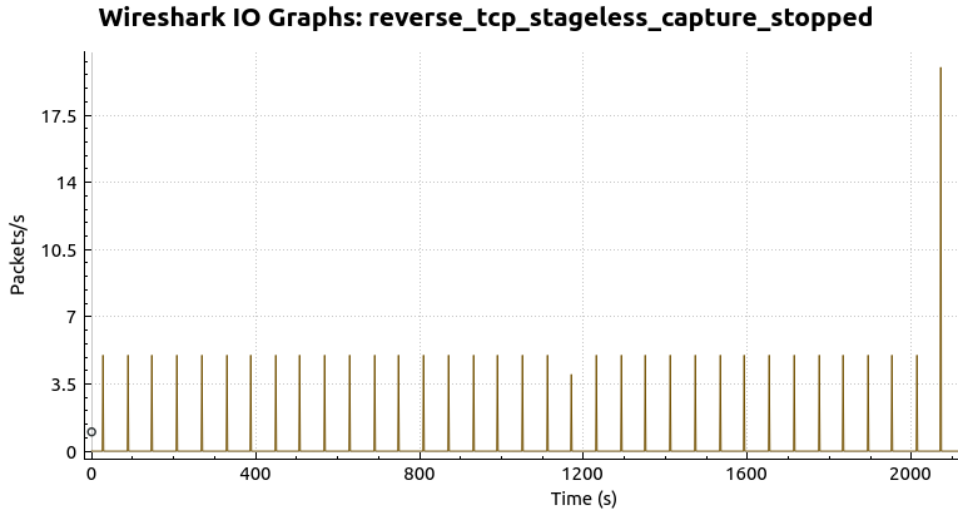


Figure 5.6: Focus on the part of the traffic generated by the malware. Stageless payload

5.2.2 During Web Browsing

While the behaviour of the generated malware remained the same during this test, the distinction has been made in order to emphasize on its detectability. Image 5.7 depicts the malicious traffic, shown in red colour, laid over the entire TCP traffic that was captured, shown in green colour. It is evident that, without knowing the source of the traffic, detection would be rendered very difficult, under regular network usage circumstances.

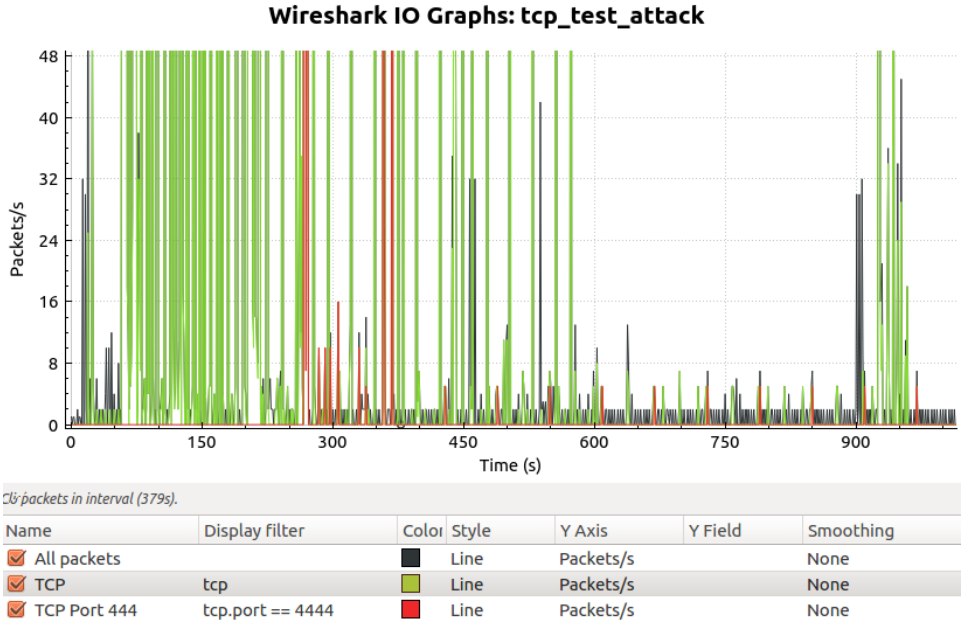


Figure 5.7: Graphical representation of the traffic. Malware behaviour shown with red colour.

5.3 `reverse_http(s)`

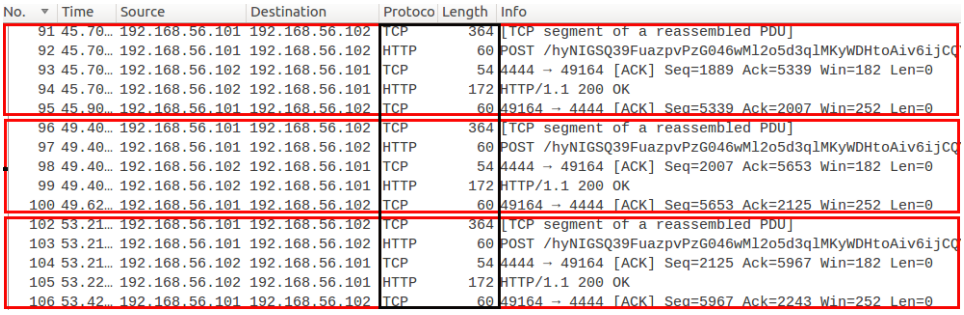
Similar to the section above, these two payloads are used for the same purpose, with the only difference being that they tunnel their communication over `HTTP` and `HTTPS` respectively. And just like with the previous payload, after several tests, it was observed that here too exists a form of repetition in the network behaviour of malware generated with the automated use of this payload. Further explanation will be given in the following subsections.

5.3.1 Contained Environment

Image 5.8 illustrates the exchange of packets between the attacker (port `4444`) and the victim (port `49164`). Once again five packets are exchanged,

however for these payloads, the transmission starts and ends with victim. In all tests it was show that the victim sends a `TCP Segment of a reassembled PDU` packet and an `HTTP POST` message. In response, the attacker would send an `ACK` packet and an `HTTP OK` message. The round would end with the an acknowledgement sent by the victim.

Moreover, image 5.8 shows a transmission of 710 Bytes per round. Throughout the experiments, this value varied due to the fluctuation of the `TCP Segment of a reassembled PDU` packet's length. Three separate lengths were observed during the course of the experiments: 293, 323 and 364. This lead to a total transmission o 639, 669 and 710 Bytes per round respectively.



No.	Time	Source	Destination	Protoco	Length	Info
91	45.70…	192.168.56.101	192.168.56.102	TCP	364	[TCP segment of a reassembled PDU]
92	45.70…	192.168.56.101	192.168.56.102	HTTP	60	POST /hyNIGSQ39FuazpvPzG046wMl2o5d3qlMKyWDHtoAiv6ijCQ
93	45.70…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=1889 Ack=5339 Win=182 Len=0
94	45.70…	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
95	45.90…	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=5339 Ack=2007 Win=252 Len=0
96	49.40…	192.168.56.101	192.168.56.102	TCP	364	[TCP segment of a reassembled PDU]
97	49.40…	192.168.56.101	192.168.56.102	HTTP	60	POST /hyNIGSQ39FuazpvPzG046wMl2o5d3qlMKyWDHtoAiv6ijCQ
98	49.40…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=2007 Ack=5653 Win=182 Len=0
99	49.40…	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
100	49.62…	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=5653 Ack=2125 Win=252 Len=0
102	53.21…	192.168.56.101	192.168.56.102	TCP	364	[TCP segment of a reassembled PDU]
103	53.21…	192.168.56.101	192.168.56.102	HTTP	60	POST /hyNIGSQ39FuazpvPzG046wMl2o5d3qlMKyWDHtoAiv6ijCQ
104	53.21…	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=2125 Ack=5967 Win=182 Len=0
105	53.22…	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
106	53.42…	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=5967 Ack=2243 Win=252 Len=0

Figure 5.8: Transmission of 5 packets per 4 seconds.

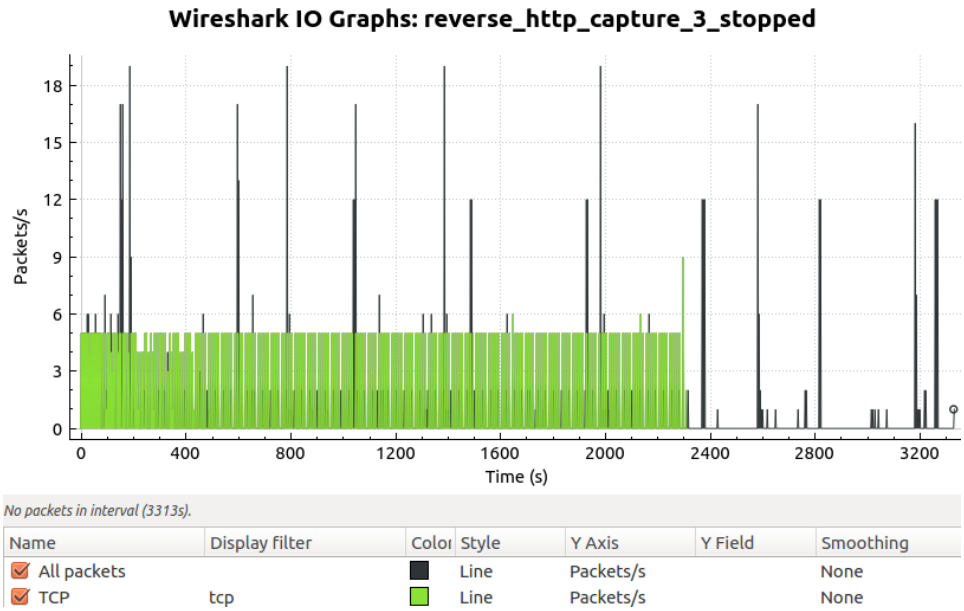


Figure 5.9: Graphical representation of the traffic. Malware behaviour shown with green colour.

The captured traffic is illustrated in images 5.9 and 5.10. The capture lasted circa 50 minutes. Again, in the Y axis the number of packets is shown, and in the X axis the time in seconds. The above graph depicts the malicious traffic in green colour over the entirety of captured packets shown in black. Image 5.10 concentrates on the malicious traffic. What is obvious from the graph below is that the transmission of packets gradually becomes sparser. Each round of packet exchange takes place roughly every four seconds in the beginning and slowly increases to ten seconds. The limit of ten seconds is never exceeded.

As with the previous payload, when the connection between the attacker and the victim was terminated, an increase of packets was observed; ranging between nine and ten packets, with one outlier of ninety. The cause of the increase in packets was the same as with the `reverse_tcp` payload. It is theorized that this behaviour has to do with the way Armitage handles and ends its connections between the payload control and the malicious code at the victim's side. However, due to time constraints we were not able to examine this theory.

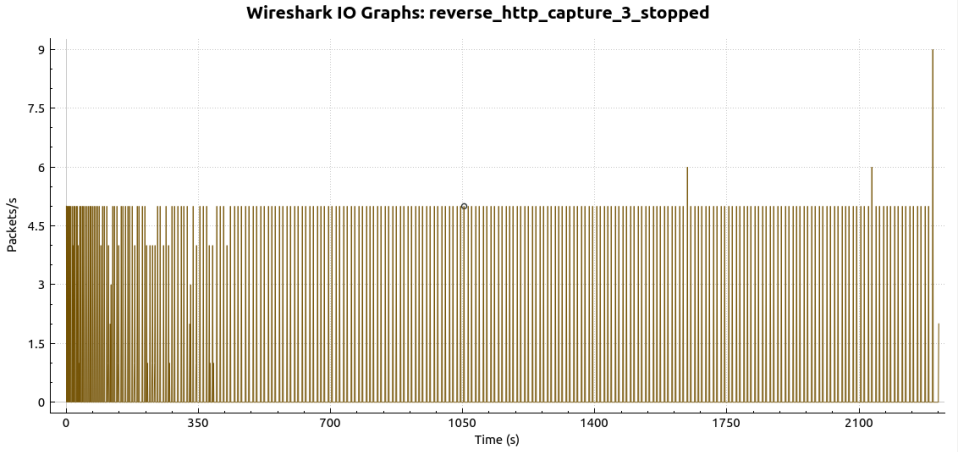


Figure 5.10: Focus on the part of the traffic generated by the malware.

5.3.2 During Web Browsing

As subsection 5.2.2 stated, without knowing the source of the malicious traffic, detection seems to be difficult. While the same applies here, the fact that the transmission of packets takes place in shorter periods of time could prove to be helpful enough to make a difference in detection. Image 5.11 presents the malicious traffic, shown in red colour, laid over the TCP traffic that was captured, shown in green colour.

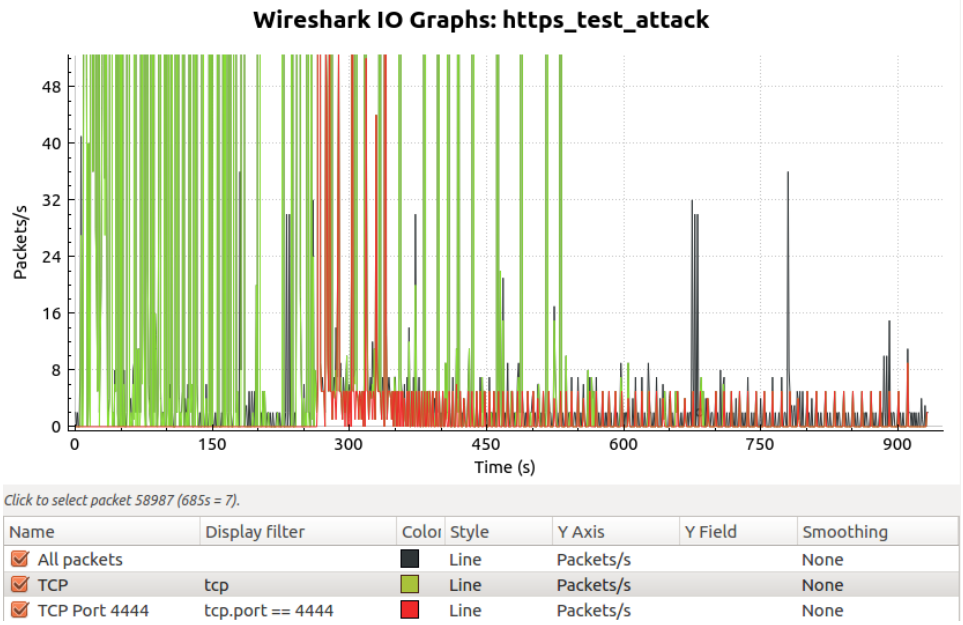


Figure 5.11: Graphical representation of the traffic. Malware behaviour shown with red colour.

At this point, we should underline the connection between the victim and the attacker. We witnessed the selection of the same port (`49163` for the `reverse_tcp` payload, port `49164` for the `reverse_http` payloads, and port `49165` for the `reverse_https` payload) from the malicious code running on the victim, in all tests in the contained environment. However, once the victim was allowed Internet access, even though the attack still took place over the LAN, the port selection was randomized. At the point in time of writing this paper, the reason for selecting the same ports in every test for each payload has not been identified. The source code of the Metasploit Framework (MSF) that is used for the payload generation procedure was examined; and the process is randomized. This was observed in the Web Browsing tests where, as mentioned above, different ports were randomly selected in each test.

5.4 Evasion Techniques

It could be argued that the malware generation procedure can include more steps, in order to make the malicious output less detectable. That, of course, would be in regard to what the attacker is trying to avoid, AV software or Intrusion Detection/Prevention Systems.

5.4.1 AV Software

In order to avoid AV software, an attacker could attempt to encode the payload with one of the provided encoders of Metasploit. This means that the code of the executable is altered in such a way, so it looks different to AV software, while still operating in the same manner. Taking this a step further and using a polymorphic encoder multiple times, would result to not only the executable, but the payload changing at each run of the encoder.

However, while this technique alters the code of the malicious output, it does not change its modus operandi. Consequently, the generated malware is functionally the same as it was before the encoding took place. Which means that, while it can trick AV software into believing it is benign, its network behaviour would not change and therefore still be as predictable as before. This premise was proven by the results acquired from the experimentation phase of this project. Each used payload was tested with multiple encoding rounds and the observed behaviour remained unchanged.

5.4.2 IDS/IPS

Intrusion Detection/Prevention Systems can be configured to pick up on Metasploit traffic[19]. Of course, MSF can also be configured to avoid such tools. One example would be to use a transport type of traffic over a protocol (e.g. `HTTPS`), which would not seem suspicious, or out of place in a user's network data. As it was shown in the above sections, common protocol types were used as part of the experiments. Another example would be to fragment the packets in smaller sizes, while also increasing the time delay between them. That way, the manner of behaviour of the malware will change and not fit the patterns discussed above. However, due to time constraints and to the fact that the last technique was deemed too specialized and above the level of inexperienced users, it was left out of scope. Though, we believe that it would be a good starting point for a potential continuation of this research.

Chapter 6

Conclusions

Throughout the course of this research, we learned how malicious code is generated via Armitage, with the use of specific payloads. We focused on their manner of behaviour on the network and attempted to identify patterns in their operation. A research question was formed in order to examine the hypothesis of this project and a number of experiments was designed to test it.

A certain amount of repetition in their respected behaviour was observed. In short, we came to the conclusion that there is evidence to suggest the existence of patterns in the network behaviour of certain automatically generated malware. We delved into the specifics in the previous chapter and described the exact findings of identical behaviour in each of the tests. Furthermore, we presented graphical representation of the network traffic that was captured, which illustrated the network behavioural patterns observed in the generated malware.

As the images above showed, similarities between each type of malware do exist. However, not all malware behaves the same way, which means that there is no single detection solution. It appears that the only way to determine the existence of such malware based solely on network behaviour would be for the session to be idle for a certain period of time. While the attacker is interacting with the victim, there is more traffic in the network, which makes it more difficult to identify patterns.

Moreover, identifying such an attack under normal circumstances of browsing was determined to be difficult. If we cannot know where the attack is coming from, it would prove particularly challenging to identify its pattern in the network. At least with the naked eye. If this procedure can be automated and monitored, then we could compare suspicious traffic with pre-sampled traffic and check for patterns that we know have a high chance of being

malicious. One example would be to keep track of the data flows and perform pattern matching.

Chapter 7

Future Work

In conclusion, we believe our research could very well be continued, as our limited amount of time did not allow us to research all the aspects that we initially hoped to examine at the conception of this project. The Metasploit Framework has a large list of exploits and payloads at the users' disposal, targeted towards different platforms and architectures, which have yet to be analyzed. Thus, a future step would be to actually analyze more of the available ways of attack and create a database of the obtained results.

In addition we believe that an important step would be to automate the pattern detection procedure, rather than manually searching for suspicious behaviour. One could apply statistical methods on the traffic data, in order to aid in the discovery of possible behaviour patterns.

Metasploit is a tool that is constantly evolving and updating. For that reason we believe that this is a rather open research topic since more methods of attack can be designed and added. Moreover, the automated generation procedure can change and improve. Consequently, these changes should be monitored, in order to maintain an up to date idea of the generation procedure and the behavioural commonalities it leads to.

It would also be important to understand whether penetration testing software can be detected by artifacts of its own infrastructure. If this could prove to be the case, then AV software and/or IDS vendors could possibly skew the success rate of their products in their favour by targeting the testing frameworks. This is analogous to the 'optimizations for testing' done by various companies in the automotive sector[20]. Therefore, this would warrant the continuation of such research, by following a procedure such as: testing vendors with this particular type of malware, modifying the malware, and lastly testing the vendors once again. Then determine which vendors show a sudden drop in their malware detection results.

References

[1] "History of Viruses", NIST, http://csrc.nist.gov/publications/nistir/threats/subsubsection3_3_1_1.html

[2] "Nearly 1 million new malware threats released every day", CNN, Article http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/

[3] "Inventors and Inventions, Volume 4", Marshall Cavendish

[4] "Data Mining Methods for Malware Detection", Muazzam Ahmed Siddiqui , University of Florida http://etd.fcla.edu/CF/CFE0002303/Siddiqui_Muazzam_A_200808_PhD.pdf

[5] "Data Mining Methods for Detection of New Malicious Executables", Shultz et al., Columbia University, State University of New York http://ids.cs.columbia.edu/sites/default/files/binaryeval-ieeesp01.pdf

[6] "Metasploit, Rapid 7", Website, http://www.metasploit.com/

[7] "Rapid7", Website, http://www.rapid7.com/

[8] "Armitage, Cyber Attack Management for Metasploit", Website, http://www.fastandeasyhacking.com/

[9] "Comparison of Pattern Matching Techniques on Identification of Same Family Malware", Ferdiansyah Mastjik et al., Sam Houston State University, Firat University www.ijiss.org/ijiss/index.php/ijiss/article/download/141/pdf_31

[10] "Semantics-Aware Malware Detection", Mihai Christodorescu et al., University of Wisconsin, Carnegie Mellon University http://www.eecs.berkeley.edu/~sseshia/pubdir/oakland05.pdf

[11] "An Unknown Trojan Detection Method Based on Software Network Behavior", Liang et al., Wuhan University, Renmin University, 2013 https://www.researchgate.net/publication/271917158_An_unknown_Trojan_detection_method_based_on_software_network_behavior

[12] "Metasploit Unleashed", Offensive Security, Website https://www.offensive-security.com/metasploit-unleashed/

[13] "How Payloads Work", Rapid7 Github Page https://github.com/rapid7/metasploit-framework/wiki/How-payloads-work

[14] "Oracle VirtualBox", Website https://www.virtualbox.org/

[15] "Windows 7 and Windows XP show no signs of dying", Article, PCWorld, http://www.pcworld.com/article/2878774/windows-7-and-windows-xp-show-no-signs-of-dying.html

[16] "Kali Linux", Website https://www.kali.org/

[17] "Payload Types", Rapid7 Github page https://github.com/rapid7/metasploit-framework/wiki/Meterpreter-Transport-Control

[12] "About the Metasploit Meterpreter", Offensive Security, Website https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

[18] "Snort IDS Ability to Detect Nmap and Metasploit Framework Evasion Techniques", M. Papadaki et al., Plymouth University, UK, 2012 https://www.cscan.org/download/?id=918

[19] "Volkswagen: The scandal explained", BBC, Article http://www.bbc.com/news/business-34324772