Computers, Materials & Continua
DOI:10.32604/cmc.2021.014596
Article

Tech Science Press

MMALE—A Methodology for Malware Analysis in Linux Environments

José Javier de Vicente Mohino[1], Javier Bermejo Higuera[1], Juan Ramón Bermejo Higuera[1], Juan Antonio Sicilia Montalvo[1,*], Manuel Sánchez Rubio[1] and José Javier Martínez Herraiz[2]

[1]Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Logroño, 26006, La Rioja, Spain
[2]Departamento de Ciencias de la Computación de la Escuela Politécnica Superior, Universidad de Alcalá de Henares, Alcalá de Henares, Madrid, Spain
[*]Corresponding Author: Juan Antonio Sicilia Montalvo. Email: juanantonio.sicilia@unir.net
Received: 01 October 2020; Accepted: 28 November 2020

Abstract: In a computer environment, an operating system is prone to malware, and even the Linux operating system is not an exception. In recent years, malware has evolved, and attackers have become more qualified compared to a few years ago. Furthermore, Linux-based systems have become more attractive to cybercriminals because of the increasing use of the Linux operating system in web servers and Internet of Things (IoT) devices. Windows is the most employed OS, so most of the research efforts have been focused on its malware protection rather than on other operating systems. As a result, hundreds of research articles, documents, and methodologies dedicated to malware analysis have been reported. However, there has not been much literature concerning Linux security and protection from malware. To address all these new challenges, it is necessary to develop a methodology that can standardize the required steps to perform the malware analysis in depth. A systematic analysis process makes the difference between good and ordinary malware analyses. Additionally, a deep malware comprehension can yield a faster and much more efficient malware eradication. In order to address all mentioned challenges, this article proposed a methodology for malware analysis in the Linux operating system, which is a traditionally overlooked field compared to the other operating systems. The proposed methodology is tested by a specific Linux malware, and the obtained test results have high effectiveness in malware detection.

Keywords: Malware analysis; methodology analysis; Linux malware; IoT malware

1 Introduction

Linux is considered one of the safest operating systems (OSs) [1] due to several advantages, including the frequent release of security patches and design and built principles. As is well-known, there has been no bug-free software, and any computer, regardless of its OS, can be attacked by a different type of malware. Many people believe there is no malware for Linux

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

OS, but this is only one of the many misconceptions about Linux. Although Linux's architecture represents a huge challenge for attackers, it can be compromised.

Despite being less secure than Linux, Windows has traditionally maintained dominance in the desktop OS quota. According to recent statistics, Windows accounts for about 77% of the entire desktop OS market. However, this is a double-edged sword because it attracts greater attention and interest of cybercriminals and, thus, suffer more attacks. However, the UNIX-based systems, such as Linux or Mac OS, have been involved in a growth usage tendency, which cannot be ignored. In addition, the popularity of Linux in Web servers (34.8%) and IoT devices [2] have made it more attractive for cybercriminals. According to the report in [3]: "Most known IoT malware indeed can be labeled more generically as Linux malware."

Recent reports have suggested that malware cost has increased in recent years, and it is projected to increase further. In [4], it is stated that: "The digitalization of industrial operational technology and IoT systems without enough protection offers quick-growing potential for targeted attacks and a wide, open flank." In 2018, the number of new malware programs for Linux-based IoT Systems increased dramatically, reaching the number of 15,730 new malware. In the first quarter of 2019, the figure was much worse, and in only three months, this number increased to 40,000 new malware per month.

Malware can adopt different forms, such as executable files, scripts, and codes, and their main objective is to harm or infiltrate into a host computer, so they can usually be described as hostile. Sometimes, malware cannot be classified in only one category as the same malware can develop behaviors of various categories [5].

As mentioned above, Windows is the most employed OS, so most of the research efforts have been focused on its malware protection rather than on other operating systems. As a result, hundreds of research articles, documents, and methodologies dedicated to malware analysis have been reported [2,6]. However, there has not been much literature concerning Linux security and protection from malware. Namely, there have been only a few blog entries and several technical reports. This was our main motivation for developing the proposed malware analysis methodology for Linux OS.

This paper proposed a methodology for specific Linux malware analysis. The proposed methodology is evaluated by a case study, including a full analysis of the Linux malware. Also, the proposed methodology can be regarded as a great starting point for a better understanding of Linux malware.

The rest of the paper is organized as follows. In Section 2, the background and related work are presented. In Section 3, the proposed methodology for malware analysis in Linux is introduced. Detailed results of applying the proposed methodology to the analysis of the malware Linux.Encoder.1 are provided in Section 4. Finally, conclusions are given in Section 5.

2 Background and Related Work

According to [2], malware for Linux systems can be developed to target diverse objectives, such as cameras, routers, printers, home appliances, and other IoT devices. Most of them are used to form botnets that enable distributed denial of service attacks (DDoS), since the mentioned equipment usually has little or no security mechanism. The objective of malware analysis is to determine the malware main features [7], find what changes are made in a victim device, and determine the malware family type. In addition, this analysis provides useful information needed to respond to a network intrusion and determine what happened or which files have been infected.

There are many ways of analyzing malicious software [8,9]. They have different analysis speeds, and some of them require a user to count on abilities and/or previous experience. Two of the most used are static analysis, where there is no execution of malware, and dynamic analysis, where it is necessary to run malware [10]. An attacker can evade static analysis by using obfuscation [11] or even anti-debugging techniques [12,13]. However, dynamic analysis has its own limitations, including the impossibility to deal with kernel malware, inability to deal with unusual behaviors that are produced only under specific circumstances, and limited behavior, as it cannot think like an analyst [14,15].

The next sections present different methodologies of malware analysis: observation, static, and dynamic analyses.

2.1 Observation

Observation is the simplest way to analyze malicious software, and it is performed by comparing the states before and after executing malware in a victim system. This approach is based on finding differences between the two states, for instance, which files have been modified, created, or erased. The main disadvantage of this approach is poor performance [16,17].

2.2 Static Analysis

The static analysis includes no execution of the malware. The aim is to obtain as much data as possible from the binaries. Although many people believe that the execution should be the first step to do with a sample that is suspected to be malware, some authors argue that it is best to gain a deeper comprehension first [18].

According to [19], the static analysis has a double perspective: Identify and detect any malicious behavior in software and detect any security failure that could lead to a system compromise. Static analysis includes the following types of analyses:

- Basic static analysis that is to examine an executable file without observing actual instructions.
- Advanced static analysis that consists of applying reverse engineering to the internal parts of the malware and observing the instructions of the code, thus discovering what the program does [7].
- String analysis that consists of scanning the malware binary file for ASCII text strings, UNICODE, or both [20].

Static analysis helps to reveal what malware can do, how to stop it, and which of its features do not depend on file execution [21]. In this type of analysis, it is not required for an analyst to understand every single line of a malware code. However, it would be helpful to know whether it connects to the Internet or not (and which sites) or some specific details, such as if it has encoding functions or auto replication features. Namely, all these small pieces of information together lead to a better comprehension of a specimen.

Static analysis has certain constraints and limitations. Therefore, malicious agents could thwart it by means of:

- Packers: Packers denote utilities or tools that are designed to modify file format. The purpose is to evade the detection of antivirus engines, thus making reverse engineering harder. Additionally, packers usually include some advanced subroutines for virtual environment detection, so unless a virtual machine is carefully prepared, the analysis will not be performed successfully.

- Crypters: Crypters are very similar to packers. Once malware is running, the payload will be unpackaged and loaded into the victim's machine memory.
- Metamorphism: Metamorphism is a technique that allows the malware to rewrite itself in each interaction, so a new malware version is not equal to the previous one. Several tactics are involved in metamorphism, including code permutation, expansion and shrinking, garbage code insertion, and register renaming.
- Polymorphism: Polymorphism is a technique that allows the malware to change both shape and signature. However, since a part of the malware shape remains the same, this technique is not as effective as metamorphism; in other words, this technique can be recognized easier than metamorphism.

The most effective way to deal with the anti-static analysis tactics and techniques is a multi-layer approach that includes joint action of people, technology, and processes as well as behavior-based detection tools, which are more advanced than the traditional antivirus tools. Heuristic analysis mechanisms can also help by means of searching several shared known components of different threats.

2.3 Dynamic Analysis

Dynamic analysis, as opposed to static analysis, includes running malware. This type of analysis is performed in a controlled and safe environment to analyze results and adopt a step-by-step approach. A wide variety of methodologies can be used in dynamic or behavioral analysis [22,23]. It should be noted that dynamic analysis is required to be performed in a safe environment, such as a secured lab, to prevent malware spreading. Some of the dangers of dynamic analysis include damage or deletion of information, propagation of malware to other hosts, or even network traffic saturation.

Dynamic analysis can be performed either automatically or manually. On the one hand, when this analysis is performed manually, an investigator is required to learn how it works by interacting with the malware with is supported by monitoring tools [24]. However, before malware can be run safely, it is necessary to set up a testing environment that does not pose a risk to a system or a network [25]. One advantage of this type of analysis is that it can contribute significantly to a faster understanding of malware [26].

On the other hand, the automatic methodology is done by means of sandboxes, i.e., testing, isolated environments where software can be monitored, assessed, and executed securely. Sandboxes are safe because they are isolated [27]. There are certain tools that can provide this functionality, such as CWSandbox [28], which is a Windows-oriented software that meets the three requirements for automation, efficiency, and correctness, Zero Wine [29], and Cuckoo. However, according to [30], a virtual environment provided by automatic dynamic analysis systems differs from the real one. Also, in [31,32], it was noted that the main drawback of this type of analysis is that results may be inaccurate as malware may behave differently, so some malware functions may not be detected.

There are also several online malware scanning systems where analysts can upload suspicious files for analysis, such as VirusTotal, Comodo, Joe Sandbox, Malwr, and Eureka [33].

Debuggers are the most reliable tools for malware detection due to their capacity to trace and analyze variable values. According to [34], malware analysis is performed by employing different techniques, such as function call monitoring, parameter analysis, and tracking of information flow and instructions.

In [35], a review of different dynamic analysis techniques proposed for or implemented in systems with the Linux OS was presented. Specifically, the review included five analysis techniques: System-call base approach, Process Control Block based approach, Executable and Linkable Format (ELF) based approach, Linux kernel-based approach, and Hybrid approach. The authors highlighted difficulties in analyzing malware, including evasion techniques, and pointed out the current problems of the use of these techniques.

Accordingly, a good approach to dynamic analysis is to use artificial intelligence and machine learning techniques. In [36], a framework for automatic analysis of malware behavior by means of machine learning was proposed. The proposed incremental approach for dynamic analysis consists of processing behaviors of thousands of binaries. This helps to identify new malware classes with similar behavior. This incremental approach combines clustering and classification techniques, offering the advantages of reduced run-time and memory overhead and increased accuracy in detecting new malware behavior.

The main drawback of learning techniques is an inability to endure super-linear increase in the amount of input data, which makes them potentially inapplicable for malware analysis. This could be solved by submitting groups of similar behavior prototypes.

In [37], the dynamic analysis was conducted with the assistance of various machine learning techniques. The algorithms included the *k*-Nearest Neighbors, Naïve Bayes, Support Vector Machine, J48 decision employed tree, and even neural networks by means of a Multilayer Perceptron (MLP). According to the results, the final accuracy of the machine learning process raised up to 96.8%, while employing machine learning could help to detect malware behavior accurately.

In [38], an approach of dynamic analysis based on artificial intelligence (AI) and neural networks was used in a Windows environment. The authors proposed a way to train neural networks to detect malware behaving in known patterns. This means that when malicious behavior is learned, it is possible to detect similar conduct faster.

As exposed, dynamic analysis can provide a deeper comprehension of malware compared to static analysis, but some precautions should be adopted to avoid possible security risks.

2.4 Related Work

Besides the investigation of the current malware analysis methodologies, it is important to note that there is not a great variety of these methodologies. A few recent publications have pointed out a lack of details in the use of tools for the execution of an analysis and an unstructured approach.

Regarding the existing methodologies, it should be noted that without the application of a logical order of steps, the analysis will not be effective, resulting in unreliable information.

Research results on the existing malware analysis methodologies suggest that **static analysis should be performed before dynamic analysis** since the former allows gathering useful information for achieving better performance in the latter phases.

Another conclusion drawn based on the research results suggests that the **memory analysis phase should be placed after dynamic analysis** because the RAM memory dump should be obtained once malware has performed changes on a victim. If needed, a packet analysis stage should be added before the static analysis; otherwise, it will be performed on a compressed sample (packet) and not on actual malware. Finally, other activities worth including in malware analysis are

analysis of obfuscated code and encryption techniques, as well as polymorphism. Therefore, it is important to describe briefly the most relevant methodologies, which are listed in the following.

One of the first steps in malware analysis is to review the structure of a malicious specimen to understand how it interacts with a victim machine. An operating system forensic analysis methodology that can help to perform this review on Windows machines was presented in [39].

MARE (Malware Analysis and Reverse) is a state-of-the-art malware analysis methodology for Windows machines, and it is based on the following four processes [40]: malware detection, isolation, behavioral analysis, and code analysis. MARQUES (Malware Analysis & Reverse Engineering system Quick Evaluation System) automates behavior and analysis of a suspicious malware code [25]. MARQUES is a system designed to create a preliminary analysis report that can provide malware analysts with immediate insight into malware functionality and Indicators of Compromises (IOC). As an innovative feature, it can automate malware analysis not only at the behavioral level but also at the code level.

SECFENCE is another malware analysis methodology that was introduced in [41] in a generic way but not documented. Namely, in [41], phases of the analysis were defined, but neither steps nor tools were detailed, and tools to be used were not properly indicated. As for static analysis, SECFENCE includes only integrated Cuckoo sandbox static malware analysis features, including file name, size, type, hash MD5, and compile time.

Pipeline is a Linux-specific malware analysis methodology that was presented in [2]. In [2], the author focused mainly on describing the analysis results of 10.548 malware samples, indicating their main characteristics. Their research also included a pipeline analysis of involved processes.

This methodology conducts code analysis before behavior analysis. However, the latter is performed automatically with a sandbox. Also, this approach is not the most optimal one because it does not allow the analyst to use the information gathered in the static analysis since it cannot interact with malware dynamically with the help of sandbox as it could do that manually. Therefore, it can be impossible to run different paths of malware execution, causing a loss of information.

SAMA [42] is another malware analysis methodology whose objective is to provide complete information about how malware works, and it is applicable to any type of malware. This methodology is based on a developed structure of process analysis and the use of tools.

The comparison of the malware analysis methodologies is given in Tab. 1. The MARE and SAMA methodologies can be considered as complete methodologies among the compared methodologies as they provide the necessary basis for proper and systematic malware analysis.

Table 1: Comparison of malware analysis methodologies

Comparison type	MARE [41]	SECFENCE [39]	Pipeline [2]	SAMA [42]	Observation
Initial actions	No	No	No	Yes	Actions to obtain evidence of a secure machine not infected by previous analysis

(Continued)

Table 1: (Continued)

Comparison type	MARE [41]	SECFENCE [39]	Pipeline [2]	SAMA [42]	Observation
Classification	No	Partial	Yes	Yes	Analysis of malware executable file without executing its code
Static code analysis	Yes	No	Yes	Yes	SECFENCE include the Cuckoo sandbox features integrated static malware analysis functions
Dynamic code analysis	Yes	No	No	Yes	Only is performed by SAMA and MARE
Behavioral analysis	Yes	Yes	Yes	Yes	Pipeline performs it with a developed sandbox; it does not allow manual interaction with malware. SECFENCE uses the Cuckoo sandbox
Memory analysis	No	Yes	No	Yes	Performed by only SAMA and SECFENCE
Use of sandbox	Yes	Yes	Yes	Yes	Online sandbox in MARE and SAMA
Obfuscation analysis	Yes	Partial	No	Yes	SECFENCE performs only packet analysis
Documented	Yes	No	Partial	Yes	
Operating system	Windows	Linux	Linux	Windows	

3 Proposed Methodology

The main objective of this work is to develop and test a methodology that can standardize the necessary stages and steps of analyzing malicious software in the Linux OS. The proposed methodology includes several phases that have to be conducted in the malware analysis process, as shown in Fig. 1.

The proposed methodology includes the following steps:

- **Initial actions.** This stage includes several substeps: Gathering evidence that all changes performed by malware are possible to track, the possibility to return easily to the clean state of a victim machine, and guarantee that there are no traffic outbounds.
- **Binaries analysis.** The purpose is to gather initial information about malware, such as if there is obfuscated code or dependencies. This should help to conduct static analysis better.
- **Static analysis.** This step includes a malware code analysis, which is performed by browsing the malware code but not running it. The purpose is to obtain a better understanding

of how malware works. It is a complex process that requires analysts to employ reverse engineering techniques. Static analysis is conducted before dynamic analysis because the information collected in static analysis can provide better performance in the latter analysis.

- **Dynamic analysis.** The main objective of this analysis is to study how malware behaves in an execution environment. Therefore, malware analysts could know what actions malware performs on the victim machine. This phase should include all necessary tools for capturing malicious activities and changes in order to collect all actions done on an infected system. The dynamic analysis also includes a review of changes made in the victim machine.
- **Memory analysis.** This step is completed after dynamic analysis to detect and gather all modifications made by malware in RAM memory, which typically denote new processes running. In order to not define too many steps, a victim machine is restored to a clean state.
- **Information gathering.** Once malware has been conveniently studied by an analyst, the proposed methodology performs an extensive search on the Internet. The objectives of this search are as follows: To confirm whether the drawn conclusions are right, to clear out those aspects of the investigation that may be doubtful, and to face code analysis with a fresh mind approach, i.e., there is no partiality for a malware analyst.

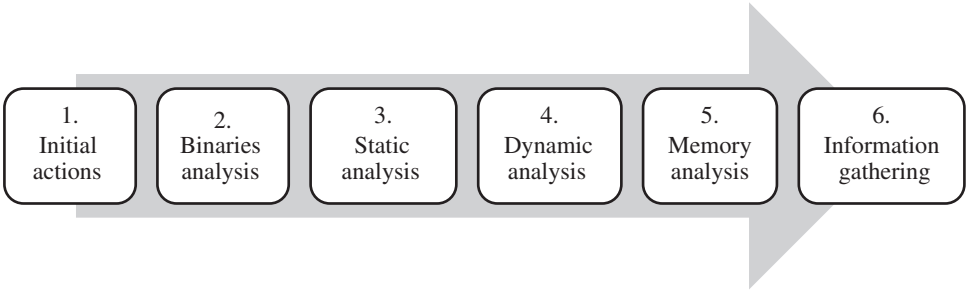

Figure 1: Block diagram of the proposed methodology

After describing the steps of the proposed methodology, it is important to outline several important aspects of the proposed methodology, which are as follows:

- The methodology for malware analysis in Windows should involve different procedures, considering its nature, specific features, and idiosyncrasies. Unlike Windows, Linux keeps a structure as simple as possible based on permissions. Therefore, all components, including equipment and processes, are built up into a file and core system settings and can be configured at any time. This simplifies the preparation and infection of malware compared to Windows, but it requires more specific knowledge to be able to conduct a satisfactory attack of malicious code infection.
- Compared to Windows, tools to be used in each stage differ significantly, so it is necessary to carry out extensive research to decide which tools suit best.
- File systems, system knowledge, and low-level libraries and components are other critical points that have to be considered. From the point of view of an attacker, malware preparation and its propagation are completely different [43].
  There are certain differences between Windows and Linux OSs:
  —Regarding files: In Windows, a user can typically move, create, and delete files and folders regardless of he or she is or is not the owner of that information. However, Linux requires privileges to perform these actions.

—Windows relies on DLLs while Linux provides four library functions (dlopen, dlerror, dlsym, dlclose), two shared libraries (static library libdl.a and dynamic library libdl.so), and one include file (dlfcn.h) to support dynamic linking loader.
—Malware propagation is totally different in these two OSs, considering their architecture and features. For instance, Linux is designed with protection rings with different levels of privileges.

- Finally, in Windows, an attacker simply needs the execution of an .exe file to cause the infection, while in Linux, execution privileges are necessary to run files.

Consequently, the preparation of an attack vector and exploitation of a vulnerability to malware are different in these two environments, so the same malware analysis methodology cannot be used in both OSs due to specific procedures in each of them.

3.1 Initial Actions

The purpose of this first stage is to obtain detailed information on a victim system before malware makes any change. It is necessary to ensure that the working environment is clean and uninfected by the analysis of previous malware samples. This stage could be described as taking a photograph of the initial state of a system to have something to compare with the following state. The processes conducted in this stage are presented in Fig. 2.

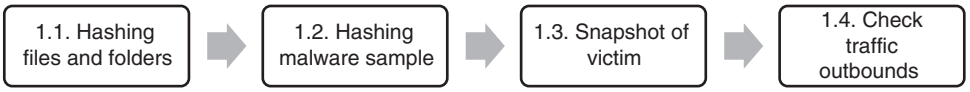

Figure 2: Initial actions

3.1.1 Hashing Files and Folders

Hashing is a process whose objective is integrity checking. It consists of creating a short, fixed-length code called a hash using mathematical functions. Any change done to the original element means that the calculated hash will be different from the original one. Hashing works in only one way, so it is not possible to obtain the original input from the hash code. Although there have been studies that suggested to hash the entire system, the proposed methodology hashes only several important directories.

3.1.2 Hashing Malware Sample

MD5 hash of a malware sample has to be obtained with the aim of both checking integrity of the download and, most importantly, quickly identify a malware sample; namely, a hash identifies malware like a fingerprint. MD5 has been chosen because its speed will not slow down the process of hashing. Additionally, the probability that two hashes accidentally colliding is relatively low, so it is very helpful to use hashes in malware detection.

3.1.3 Taking Snapshot of Victim

A snapshot represents a restoration point or a clear state to which a system should be returned in the case of need. It allows a researcher to reset the machine without installing the OS from scratch. It is a good practice to restore the system to a clean, secure point after studying malware.

3.1.4 Check Traffic Outbounds

This measure controls and assures that there is no traffic that outbounds a victim system. The purpose is to avoid malware spreading and infecting the host system (when a victim system is virtualized) or any other external machine (when a victim system is in a physical environment).

3.2 Binaries Analysis

Its objective is to gather all possible information from a binary file. This should provide an analyst with a convenient starting point. The steps of this stage are illustrated in Fig. 3.

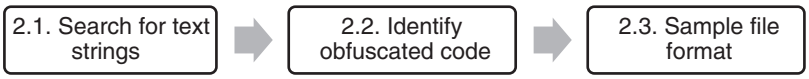

Figure 3: Binaries analysis substages

3.2.1 Text Strings Searching

This action can provide an analyst with useful information, including UNICODE or ANSI strings found on binaries. For instance, if encoding-related strings, such as AES, cipher, or similar, are found in a sample, possibly ransomware is present.

3.2.2 Obfuscated Code Identification

Obfuscated code is understood as any code that has been modified with the aim of making reverse engineering harder or evading the inspection of anti-malware software. When dealing with an obfuscated code, a higher level of sophistication and intelligence is to be credited to the attacker.

3.2.3 Sample File Format

This action is performed with the aim of gathering information on file format and/or dependencies of samples. For instance, file headers are an undervalued and overlooked source of data, and some headers contain information about how malware code is organized that could be of the utmost.

3.3 Static Analysis

Static analysis is conducted to understand malware structure and operation better without executing it better. A researcher has to be careful with obfuscated code and employ mainly a disassembler to dissect the files of a sample totally.

Static analysis, which is also called code analysis, includes the machine code analysis that is conducted to understand how malware has been designed by an attacker. Machine code is essentially a bunch of basic instructions, which are the lowest-level instructions, such as add, sum, and jump, so it can be very useful for a researcher to have previous experience or knowledge in dealing with machine code to make the most of the analysis. Reverse engineering is the most common method to obtain information about the functionality of the malware, including data that can execute the malware, data installation, execution commands, passwords, control commands, execution routes, IRC channel and connection password, data to avoid restrictions on operation in virtual environments, and hidden functionalities. The starting point is a sample(s) of malware, from which data can be gathered to determine malware's main functions or components, how they are connected to each other Reverse engineering is the most pragmatic approach because it is the best way to split a code in its essential instructions and then analyze them.

3.4 Dynamic Analysis

Dynamic analysis, which is also known as behavioral analysis, has high importance due to the great amount of information expected to be obtained. Dynamic analysis includes three main steps, as shown in Fig. 4.

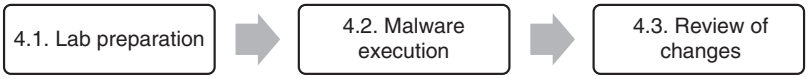

Figure 4: Dynamic analysis substages

As its name suggests, malware execution involves malware execution in a secure laboratory environment. This is a complex process in which debuggers are used. Namely, while malware is running, debugging is performed to gather as much information as possible, including the files downloaded from the Internet, websites the malware tries to navigate to, accessed URLs, commands, paths of execution, any change made in the file system (added, modified, or removed files), communications with other machines (if any), and processes running in RAM. Since a malicious code lacks the comments of a developer, it can be a big challenge for an analyst to understand it, and it is useful to have previous experience in programming.

3.4.1 Lab Preparation

The objective is to boot all required software: Gather all actions performed by malware and delude the malware to believe it has free access to the Internet, so it can behave normally; in other words, software should provide fake network capabilities.

3.4.2 Malware Execution

The objective of this substage is to run malware in a lab environment and see which changes are made in a victim system, including created or deleted files and started RAM processes.

3.4.3 Changes Review

The main objective of this phase is to detect and study any change done to the victim machine by malware, no matter how insignificant it is. Review of changes seeks to obtain as much information as possible about any action performed by malware, including:

- Determining how filesystem has changed: Searching for created, modified, or erased files. This also includes to examine whether any file has been downloaded from the Internet;
- URLs accessed by the malware;
- Communications with other machines. Typically, malware will communicate with a Command & Control server.

Although a review of changes includes an analysis of new processes running in RAM, this will be explained in detail in the following section.

3.5 Memory Analysis

The objective of this stage is to gather all possible information on RAM memory once malware has been executed. It is also suggested to review changes in a victim machine as well as restore the victim machine to the clean state. The two main processes conducted in this stage are presented in Fig. 5.

Figure 5: Memory analysis substages

3.5.1 RAM Forensics

RAM forensics can be roughly defined as an exhaustive RAM memory analysis. Namely, it is necessary to examine the victim machine's memory dump. This is useful in several situations; for instance, when there is a suspicion that malware is not leaving tracks on the hard disk; also, it can provide an extra source of information, given that it is possible to detect suspicious network activity, malicious processes running in the background, or recently executed commands. The steps of this stage are presented in Fig. 6.

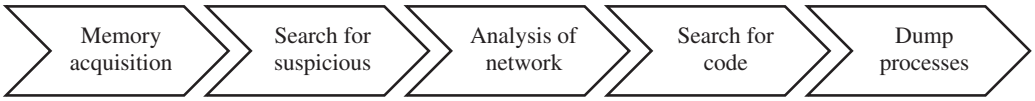

Figure 6: RAM forensics substages

According to researchers, once Linux kernel symbols have been determined, it is possible to obtain live system information. Gathering information about structure layouts can play an important role as far as the extraction of memory modules information is concerned [44]. This gives an insight into how important forensics can be in security investigations.

3.5.2 Restoration

This step is performed to restore a victim machine to the clean state to make it ready for the latter malware analysis. In this step, the previously taken snapshots are used. Once a machine has been restored, a researcher has the clean machine ready to conduct another malware test.

3.6 Information Gathering

This stage aims to obtain information about malware functionality, typically through extensive internet research. The malware information gathering at the end of the methodology can help the analyst to conduct sample analysis without bias. Furthermore, this stage can help to either re-affirm on drawn conclusions or sharpen obtained results that may lay unclosed. It is a great opportunity to know almost everything about a certain malware. In addition, the Internet is a wide-open source of information where an analyst can gather a large amount of information on malware, such as which level of danger implies certain malware or if the analyst is dealing with a new or an evolved malware.

4 Case Study

One of the biggest risks when developing a methodology is to remain it in the theoretical stage. In order to prevent this, it is important to test the developed methodology with a case study, which requires:

- Malware sample. The Internet offers a variety of sites where malicious software is available to download for investigation purposes. GitHub is one of the most famous ones.
- Testing environment. This environment is also called a laboratory environment, and it represents a secure, isolated group of machines, either physical or virtualized, where malware can be executed and studied.

4.1 Malware Sample

In this work, malware Linux.Encoder.1 was used to test the proposed methodology. This malware belongs to the ransomware family, and it has the following features:

- Ability to encode user's personal files and/or folders,
- Request for payment to unlock these files. This payment is a ransom.

One of the biggest mistakes that the ransomware usually takes advantage of is either the absence of a backup of important files or the outdated state of the existing one. In these cases, a victim can either pay an attacker to unlock files or accept losing them. So, it is always a good practice to do a regular full backup of our most important files and folders. Linux.Encoder.1 also has a distinctive feature that the ransom is demanded in virtual coins, also known as bitcoins.

Linux servers are the main target for Linux.Encoder.1. This is both due to the number of data servers usually contain and the possibility to hijack information from many users. Linux.Encoder.1 takes advantage of a vulnerability in an open-source e-commerce web application called Magento.

Once a server has been infected, Linux.Encoder.1 starts encoding files and folders. The starting point is critical directories such as /home, /root or /var/lib/mysql. Linux.Encoder.1 employs the AES algorithm, considering that it provides enough strength in encryption at a reasonable speed.

4.2 Testing Environment

The testing environment or laboratory is the place where malware is executed and analyzed. The lab environment used in this case study was built as follows:

- There was a **dedicated virtual network to isolate and run the malware**. Both victim and analysis machines were hosted into this network. The victim machine had free internet access. InetSim software provided required fake network capabilities: Full simulation of described services and the possibility to create a log to register any network action.
- Host machine **was isolated from the virtual network**. To prevent malware from spreading to the host, there was **only one point of communication** between the host and analysis machine, a shared folder. By default, it had read-only privileges and required a root password to be mounted on the analysis machine.
- Host machine had free internet access without any restrictions.

The OS on the victim machine was vulnerable to the malware, and both patches and software updates were not activated.

4.3 Performed Analysis

4.3.1 Initial Actions

The proposed methodology began with the initial actions, including hashing files and folders, obtaining a snapshot, and monitoring traffic outbound.

There were two hashing processes to perform. The first one was related to the files and folders of the operating system, and the second one was related to the malware sample. It should be noted that it is important to have the sample downloaded on the victim machine.

There are many different algorithms to obtain a hash, but MD5 was used in this work because it is one of the fastest. In Linux OS, some of the most important files and folders to be hashed are the following ones:

- /boot, /home, /etc., /bin, /lib, /usr, /var, /proc/meminfo, /proc/ and directories hashed: */boot, /home, /etc., /usr* and */var*.

Since it is possible to obtain malware sample's MD5 hash on GitHub, in the performed analysis, it was recalculated with the help of md5sum command. The snapshots were used to reset the machine to the clean state without the need to install the OS from scratch. In the case study, VirtualBox was employed to obtain a snapshot of the victim machine.

It is important to monitor traffic from a victim machine, both inbounds and outbounds. The outbounds traffic is especially important because malware is different from other types of attacks, such Denial of Service (DoS) or jamming, where all the traffic is directed to the target system. Thus, checking outbound network traffic may be helpful in identifying external communications the sample may establish. As the victim virtual machine was bridged, in the analysis, **Wireshark** was employed to detect any possible connection to the Internet.

4.3.2 Binaries Analysis

In this step, several actions were performed with the purpose of gathering information on malware binaries.

Search for Text Strings In Linux, search for text strings can be conducted in several different ways, but *strings* command is one of the best options due to its simplicity and customization. It can find any printable string (at least four characters long, but customizable) in a binary file.

The output of the *strings* command produced a combination of words such as *encrypted private key, AES, padding*, or even name of files, such as *lockfile.c*. This could lead the analyst to think about Linux.Encoder.1 as **ransomware**.

Obfuscated Code, Sample File Format, and Dependencies Searching malware binaries to detect whether they were obfuscated or not was conducted by Detect-It-Easy, which is a freeware that can deobfuscate and identify file types, providing information on the analyzed file. The information extracted from binaries included:

- Type of archive: malware sample was an Executable and Linkable (ELF) file,
- 64-bit architecture,
- How binaries were compiled,
- It could be concluded that the malware code was not obfuscated.

The information on libraries that are employed by a file is always instructive and useful. Binary file dependencies were investigated with the assistance of the ELF library viewer. Unfortunately, the analysis did not provide any outstanding dependencies to be considered.

4.3.3 Static Analysis

Static analysis has great importance since several current malware analysis methodologies do not provide a proper analysis, which makes malware identification harder.

As previously described, static analysis investigates a malware sample code to understand how it has been developed and, consequently, how the spreading process could be stopped.

Given that reverse engineering techniques are the best way to conduct static analysis, a good approach to static analysis should include the use of IDA Pro. This allowed gathering advanced information, including:

- Linux.Encoder.1 could potentially have file-ciphering capabilities. This was already a possibility indicated by its name (Encoder).

- If really ciphering files, malware could employ padding. This is a way to make things more difficult for a potential cryptanalyst, i.e., avoid predictability.
- There were some clues about AES being the type of algorithm employed with CBC (Cipher Block Chaining). CBC is a way of ciphering where plaintext is mixed with the previous block with an XOR function before encrypting it.
- There also existed some hints about SALT that had been considered when designing Linux.Encoder.1. SALT denotes random data mainly employed to make passwords in storage more secure and difficult to be guessed.
- Reverse engineering techniques allowed gathering some information that led to thinking that malware could need an internet connection, such as Address not available or Network is down.

Therefore, the following conclusions can be drawn:

- It can potentially use ciphering encryption capabilities;
- As a result, malware family hints to be ransomware or similar;
- There are some clues about AES CBC and the introduction of SALT.

Objdump is very convenient to analyze the sample file format. This is a Linux-native command whose purpose is decomposing a file, i.e., getting a better understanding of file format (including headers). It is instructive to go in-depth into command output:

- Size is about how long the corresponding section is;
- VMA stands for Virtual Memory Address, i.e., a location in the main memory;
- LMA or Local Memory Address;
- File off: represents a shift of entry (reference is file header);
- Algn or alignment. This must be an integer that is a power of two. This field is normally used to make memory-mapping of file more efficient; in that case, the larger the value is, the better the memory-mapping will be.
- Other values such as CONTENT, ALLOC, LOAD, READONLY, and DATA are flags indicating the behavior of its section.

This helps in understanding how a file is going to be loaded in memory.

4.3.4 Dynamic Analysis

Dynamic analysis is performed to understand malware behavior. It is performed by **executing malware in a secure environment** and collecting as much information as possible.

Lab Preparation There are several actions to be considered when preparing a lab: configure analysis machine as a default gateway of a victim machine, unzip sample malware files since they are password-protected to avoid risks, and boot network simulator software.

It is also advisable to check whether a victim machine can infect the host machine by the malware under the analysis.

Malware execution Unlike Windows, where files are usually executed by double-clicking on them, Linux requires execution privileges to run a file. GitHub provides some instructions about performance. That is also the case with Linux.Encoder.1 it was required to use chmod command first:

- *Chmod +x Linux. Encoder.1 ./Linux.Encoder.1*

Wireshark helped to gather information on possible malware network activities. Based on the static analysis results, the malware was believed to perform network activities. However, after an exhaustive study with the assistance of Wireshark, it was surprisingly noted that:

- Linux.Encoder.1 did not try to connect to any Command & Control server;
- Malware did neither communicate with other machines nor tried to download additional malware.

Review of Made Changes InetSim logs were checked, but **there was no trace of access to Command & Control** server or similar. This result leads to the same conclusion as Wireshark pointed. However, by navigation through files and folders, it was found that there **were several newly created files** that included the text *README_FOR_DECRYPT*, as shown in Fig. 7.

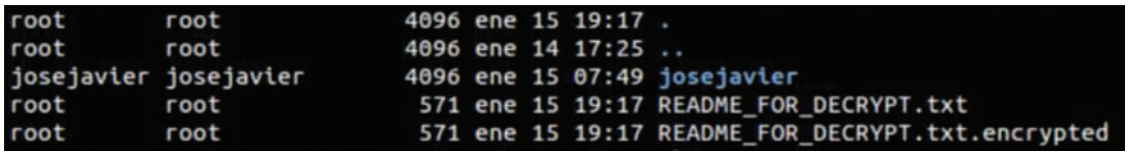

Figure 7: Newly created files

Furthermore, by employing *diff* and *find* commands, it is possible to know comprehensively what files and folders in the entire system have been modified.

The following conclusions can be drawn:

- Analyzed malware definitely belongs to the ransomware family, and this is proof of some of its capabilities.
- There seems to exist no connections to a Command & Control server.
- Malware does not show any attempt to interact with network or downloading additional malware.

Hybrid Analysis Results Static analysis, although being lightweight compared to dynamic analysis, can be defeated by means of tactics and techniques, such as packing, other obfuscation techniques, or even by metamorphism. Dynamic analysis is typically performed in an emulator, such as a virtual or physical environment, but it is more expensive in terms of both time and resources.

Since both analyses have certain weaknesses, it was decided to use both of them in the case study. Also, this hybrid analysis can increase the effectiveness and accuracy of obtained results and reduce the impact of analyses' weaknesses. The hybrid analysis that combines static and dynamic analyses involves mixing static features gathered by analyzing the code with the information extracted when executing the malware. The results obtained by static, dynamic, and hybrid analyses are presented in Tab. 2.

4.3.5 Memory Analysis

This process consists of extracting and analyzing a RAM memory dump of a victim machine. Linux offers several useful tools to perform memory analysis, including *Memdump*, which is the command employed in Linux to make a RAM dump, Linux Memory Extractor (LiME), FTKImager, and Volatility that is an open-source framework whose main purpose is analyzing RAM dumps to gather information. LiME is an open-source tool that works at the kernel level

and allows the acquisition of RAM in an easy way. The syntax required for memory extraction when using the LiME is shown in Fig. 8.

Table 2: Static, dynamic, and hybrid analyses results

Factor	Static analysis	Dynamic analysis	Hybrid analysis
Based on	Analysis and study of code	Execution of malware	All information gathered by static analysis helps an analyst to perform better dynamic analysis
Time required	Low	High	High
Resources	Low	High	High
Affected by packing, obfuscation, etc.	Yes	No	No
Disadvantages	Some malware types can be unidentified. High dependency on malware analyst's skills	Requires appropriate lab preparation. High demands for time and resources	High cost, mainly in terms of time
Advantages	Low time and cost	High detection rate, provides the information on malware behavior	More accurate results as the information of both analyses is combined
Effectiveness	Lower than dynamic analysis	Medium	High: Combines both types of analyses, so conclusions are more accurate; minimizes every approach weaknesses

Figure 8: LiME extraction of memory

LiME dump is a *.mem* file. With the assistance of FTKImager some useful information can be gathered. A hint of malware employing some type of library for ciphering purposes, which is related to the *AES* algorithm, is presented in Fig. 9.

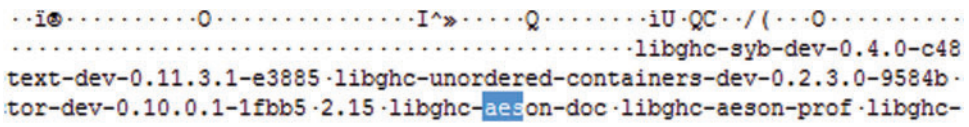

Figure 9: Information extracted from RAM

Based on the memory analysis results, the following conclusions can be drawn:

- Malware seems to have encryption capabilities: memory dump offers suggestions on using the AES encryption libraries, and there are several references or hints to cryptographic terms, such as public key DH.
- Memory dump references to some directories, such as System settings or Home folder. This leads to the conclusion that these folders could possibly have been encrypted.

4.3.6 Information Gathering

Some internet research can help sharpen the information gathered by the code analysis (static and dynamic analyses). This can help the analyst to either confirm what has been suggested or to detect which things can been done differently. Although, at this point, an **analyst has to know what family malware belongs to**, it is important to confirm it. One of the best ways to achieve this is by using means of a utility, which can be either an antivirus (ClamAV, Sophos, Kaspersky, Avast, ESET…) or a web application such as VirusTotal.

Once a sample was analyzed and identified as malware, or more precisely, like ransomware, it was possible to know:

- The analyzed malware was a specific Linux malware (ELF file) named Linux.Encoder.1;
- Linux.Encoder.1 belonged to the ransomware **family**.

The majority of engines (up to 44 out of 59) can identify a sample as malware. In addition, even more useful is information about Linux.Encoder.1 provided on the community tab. After extensive research on the internet, some other useful pieces of information were obtained, including:

- Ransomware **main target are Linux servers**, which is probably because in this way, more users would be affected, so the pressure to pay the ransom would be higher. This is the first ransomware targeting computers running Linux OS [45].
- Linux.Encoder.1 **takes advantage of a vulnerability in Magento**, an open-source application for e-commerce.
- Linux.Encoder.1 can also identify other target files to be encrypted, such as *.js, .php, .html.*, if they are in other less known directories.

In addition, one of the most important findings from the security point of view is as follows: there is a **misuse of cryptography** in the design of Linx.Encoder.1. According to some researchers, it is **easy to recover files due to the way IVs are generated**. These date and time dependencies are a huge flaw in the implementation of cryptography.

4.4 Analysis Results Summary

A summary of the analysis results is presented in Tab. 3.

Table 3: Summary of analysis results

Malware analysis results		
Analysis summary	Key observations	Analyzed malware belongs to the ransomware family. Analyzed malware is focused on Linux servers. It takes advantage of vulnerabilities in Magento.
	Limitations	Analyzed malware targets Linux servers. It has not been tested against the Windows server.
Classification	File type	Malware samples are ELF files with the 64-bit architecture.
	Antivirus identifications	Online utilities and/or antivirus, such as *VirusTotal*, classify the analyzed malware as ransomware.
	Diffusion mechanisms	Self-replication is not needed because this malware is designed to infect servers.
	Ability to data leakage	Linux.Encoder.1 is engineered to encrypt files. Therefore, it is not needed to leak sensitive information since the damage is done by the hijacking information.
	Interaction with a remote attacker	When analyzing network activity, there is no proof of connection with the Command & Control server of the malicious agent.
Dependencies	Supported OSs	Malware is designed for Linux OS, but there is no guarantee that it is not affecting other OSs.
Code analysis and behavior	Static code analysis	Attackers designed malware to employ AES CBC 128 to encrypt files. Padding has been considered when cyphering.
	Dynamic code analysis	Linux.Encoder.1 does not seem to connect to any Command & Control server. Misuse of cryptography: it is easy to recover files because of the way seeds are generated (dependency on date and time).
Support elements	Logs	InetSim logs indicate that malware artificial intelligence is not sophisticated enough to detect fake network services.
Incident recommendations	Steps to eradicate	Due to the bad implementation of cryptography, it is easy to recover files knowing the date and time of the first file encrypted. It is always a good idea to backup files frequently to fight ransomware.

4.5 Lessons Learned

Based on all the obtained results, some helpful conclusions can be drawn, and they are given in Tab. 4. These conclusions can provide a good retrospective of the work presented in this paper.

Table 4: Useful lessons

Problem/success	Lesson learned
Information gathering is conducted after code analysis.	This approach is successful because it allows performing static and dynamic code analyses with no bias.
Not all malware capabilities are discovered.	Dynamic analysis performed in a lab environment cannot retrieve all malware features, mainly because Linux.Encoder.1 targets Linux servers.
Not all directories are hashed.	There can be a discussion about what system directories should be hashed to study what changes are done by malware. In this work, only some of the most important folders are hashed, but in certain cases, it can be desirable to hash all system folders.

5 Conclusions

The most important conclusions of this study are as follows:

- It is important to analyze malware in a structured way, which means that it is necessary to **rely on a methodology** to have the steps defined. This increases the possibility of **getting better analysis results** as well as **reducing the complexity** of the overall analysis process.
- According to the obtained analysis results, it is useful to perform the code analysis before gathering the information on the Internet. This allows conducting the investigation without any bias or prejudice of what other researchers have already discovered.
- The presented research is quite useful to sharpen results and conclusions. Sometimes neither static nor dynamic analysis can provide all required details, and malware can be too complex to be fully understood. In these cases, internet-based research can help analysts complete their findings.
- Malware is complex and constantly changing. Although Linux.Encoder.1 is not sophisticated enough to detect fake network services, this does not mean it is not a threat. According to the conducted analysis, an analyst should update the knowledge about new cybersecurity threats and capabilities. In addition, having a programming background as a developer can definitely help.

6 Future work

Our current work includes developing a malware analysis methodology for Android OS since there has been a spectacular increase in malware infections in this OS in recent years. Our future work will also examine whether different analysis tools for Linux and Android OSs have equivalent versions.

Acknowledgement: The authors extend their appreciation to the Software Engineering and Security research group (SES) of Universidad Internacional de La Rioja.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. Taylor, *Why Linux is Better than Windows or macOS for Security*. USA: Computer World from IDG Communications, 2018. [Online]. Available: https://www.computerworld.com/article/3252823/why-linux-is-better-than-windows-or-macos-for-security.html.

[2] E. Cozzi, M. Graziano, Y. Fratantonio and D. Balzarotti, "Understanding linux malware," in *39th IEEE Sym. on Security and Privacy*, San Francisco, pp. 161–175, 2018.

[3] Costin and J. Zaddach, 'IoT malware: Comprehensive survey analysis framework and case studies," in *BlackHat Conf.*, USA, 2018.

[4] AV-TEST, *Security Report 2018/19*. Germany: AV-TEST. Institute, 2019. [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf.

[5] K. A. Monnappa, *Learning Malware Analysis: Explore the Concepts, Tools and Techniques to Analyze and Investigate Windows Malware*, 1st ed. Birmingham, England: Packt Publishing Co., Ltd., pp. 20–45, 2018.

[6] K. Mathur and S. Hiranwal, "A survey on techniques in detection and analyzing malware executables," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, pp. 422–428, 2013.

[7] M. Honig and M. Sikorski, "Part 1: Basic analysis," in *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*, 1st ed. San Francisco, USA: No Starch Press, pp. 9–39, 2012.

[8] L. Cavallaro, P. Saxena and R. Sekar, "On the limits of information flow techniques for malware analysis and containment," in D. Zamboni, *Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA*, *Lecture Notes in Computer Science*, Berlin: Springer, vol. 5137, pp.143–163, 2008.

[9] C. K. Moser and E. Kirda, "Exploring multiple execution paths for malware analysis," in *IEEE Sym. on Security and Privacy*, Berkeley, CA, USA, pp. 231–245, 2007.

[10] Y. Heng and S. Dawn, *Automatic Malware Analysis: An Emulator-Based Approach*, 1st ed. NY, USA: Springer Science & Business Media, 2012.

[11] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Int. Conf. on Broadband, Wireless Computing, Communication and Applications*, Fukuoka, Japan, pp. 297–300, 2010.

[12] X. Chen, J. Andersen, Z. M. Mao, M. Bailey and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *IEEE Int. Conf. on Dependable Systems and Networks with FTCS and DCC*, Anchorage, AK, USA, pp. 117–186, 2008.

[13] T. Shields, *Anti-Debugging: A Developers' View*. USA: Veracode Inc., 2010.

[14] D. Sgandurra, L. Muñoz-González, R. Mohsen and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," Ithaca, NY, USA: Cornell University, 2016. [Online]. Available: https://arxiv.org/abs/1609.03020.

[15] M. Sihwail, K. Omar and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *International Journal on Advanced Science, Engineering and IT*, vol. 8, no. 4-2, pp. 1662–1671, 2018.

[16] R. Grégio, D. S. Fernandes Filho, V. M. Afonso, R. Santos, M. Jino *et al.*, "Behavioral analysis of malicious code through network traffic and system call monitoring," in *Proc. SPIE Defense, Security, and Sensing, Evolutionary and Bio-Inspired Computation: Theory and Applications V*, Orlando, Florida, USA, 2011.

[17] M. Brand, "Analysis avoidance techniques of malicious software," Ph.D. dissertation, Edith Cowan University, Australian, 2010. [Online]. Available: https://ro.ecu.edu.au/theses/138.

[18] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa and K. Nakao, "Automated malware analysis system and its sandbox for revealing malware's internal and external activities," *IEICE Transactions on Information and Systems*, vol. 92, no. 5, pp. 945–954, 2009.
[19] M. Christodorescu, S. Jha, D. Maughan, D. Song and C. Wang, *Malware Detection*. Wisconsin, USA: Springer, 2007.
[20] D. Distler and C. Hornat, *Malware Analysis: An Introduction*. USA: SANS Institute InfoSec Reading Room, 2007.
[21] G. Díaz and J. R. Bermejo, "Static analysis of source code security: Assessment of tools against SAMATE tests," *Information and Software Technology*, vol. 55, no. 8, pp. 1462–1476, 2013.
[22] A. Shabtai, K. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, "Weiss Andromaly: A behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.
[23] J. Hegedus, Y. Miche, A. Illin and A. Lendasse, "Methodology for behavioral-based malware analysis and detection using random projections and K-nearest neighbors classifiers," in *7th Int. Conf. on Computational Intelligence and Security*. Hainan, China, pp. 1016–1023, 2012.
[24] I. Firdausi, C. Lim, A. Erwin and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *2nd Int. Conf. on Advances in Computing, Control, and Telecommunication Technologies*, Jakarta, Indonesia, 2010.
[25] C. Q. Nguyen, J. E. Goldman and A. E. Smith, "Malware analysis & reverse engineering quick evaluation system," in *Proc. of the 12th Annual Information Security Sym.*, West Lafayette, IN, EEUU: Purdue University, 2011.
[26] L. Zeltser, *Mastering 4 Stages of Malware Analysis*. USA: SANS Institute, 2019. [Online]. Available: https://www.sans.org/blog/dominando-las-4-etapas-del-anailisis-de-malware/.
[27] C. Gutiérrez, *Adelantándonos a los atacantes con el análisis dinámico de malware*. Slovak Republic: Welivesecurity by ESET, 2014. [Online]. Available: https://www.welivesecurity.com/laes/2014/08/13/adelantandonos-atacantes-analisis-dinamico-de-malware/.
[28] C. Willems, T. Holz and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.
[29] J. Koret, *Zero Wine Malware Analysis Tool*. USA: Synopsys Inc., 2020. [Online]. Available: https://www.openhub.net/p/zerowine.
[30] E. Gandotra, D. Bansal and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 2, pp. 56–64, 2014.
[31] M. Sharif, A. Lanzi, J. Giffin and W. Lee, "Impeding malware analysis using conditional code obfuscation," in *Proc. of the Network and Distributed System Security Sym.*, San Diego, California, USA, 2008.
[32] C. Liangboonprakong and O. Sornil, "Classification of malware families based on N-grams sequential pattern features," in *IEEE 8th Conf. on Industrial Electronics and Applications*, Melbourne, VIC, Australia, 2013.
[33] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras and W. Lee, "Eureka: A framework for enabling static malware analysis," in S. Jajodia, J. Lopez (eds.), *Computer Security—ESORICS, Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer, pp. 481–500, 2008.
[34] M. Egele, T. Scholte, E. Kirda and C. Kruegel, "A survey on automated dynamic malware analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2008.
[35] G. Damri and D. Vidyarthi, "Automatic dynamic malware analysis techniques for Linux environment," in *3rd Int. Conf. on Computing for Sustainable Global Development*, New Delhi, India, 2016.
[36] P. T. Rieck, C. Willems and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.
[37] C. L. Firdalusi, A. Erwin and A. Satriyo, "Analysis of machine learning techniques used in behavior-based malware detection," in *2nd Int. Conf. on Advances in Computing, Control and Telecommunication Technologies*, IEEE, Jakarta Indonesia, pp. 201–203, 2010.

[38] C. Pascariu and I. Barbu, "Dynamic analysis of malware using artificial neural networks," in *9th Int. Conf. on Electronics, Computers and Artificial Intelligence*, IEEE, Targoviste, Romania, pp. 1–5, 2017.

[39] R. O. Ramírez and O. A. Reyes, "Implementación de un laboratorio de análisis de malware," Ph.D. dissertation, Autonomous University of Mexico, 2013.

[40] C. Q. Nguyen and J. E. Goldman, "Malware Analysis Reverse Engineering (MARE) methodology & Malware Defense (M.D.) timeline," in *InfoSecCD 10: 2010 Information Security Curriculum Development Conf.*, New York, USA: ACM, 2010.

[41] 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*, vol. 15, no. 1, pp. 39–60, 2019.

[42] C. Abad Bermejo, J. R. Bermejo, M. A. Sicilia and J. A. Sicilia, "Systematic Approach to Malware Analysis (SAMA)," *Applied Sciences*, vol. 10, no. 4, pp. 1360, 2020.

[43] A. Sicilia, J. Bermejo-Higuera, E. García-Barriocanal, S. Sánchez-Alonso, D. Domínguez-Álvarez *et al.,* "Querying streams of alerts for knowledge-based detection of long-lived network intrusions," in H. Christiansen, H. Jaudoin, P. Chountas, T. Andreasen, H. Legind Larsen (eds.), *Flexible Query Answering Systems, FQAS, Lecture Notes in Computer Science*, vol. 10333. Cham: Springer, pp. 186–197, 2017.

[44] S. Zhang, X. Meng and L. Wang, "An adaptative approach for Linux memory analysis based on kernel code reconstruction," *EURASIP Journal on Information Security*, vol. 14, no. 1, pp. 1–13, 2016.

[45] D. Bisson, *Website Files Encrypted by Linux.Encoder.1 ransomware? There is Now a Free Fix*, UK: Cluley Associates Limited, 2015. [Online]. Available: https://www.grahamcluley.com/website-files-encrypted-linux-encoder-1-ransomware-free-fix/.