

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/337274721>

Application Level Rootkit Detection Program for Debian 9.8 Linux Distribution

Thesis · May 2019

DOI: 10.13140/RG.2.2.30930.30404

CITATIONS

0

READS

1,532

1 author:



Batsal Nepal

3 PUBLICATIONS 0 CITATIONS

SEE PROFILE

Acknowledgement

The success of this report required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along with the completion of my project.

I respect and thank my supervisors, Ms. Suman Gupta (first supervisor) and Mr. Akchayat Bikram Dhoj Joshi (second supervisor), my FYP tutors, Mr. Aaditya Khati, Mr. Aaditya Khwakhwali and Mr. Yojan Dhakal for providing me an opportunity to do this project. I am extremely thankful to them for providing such nice support and guidance, although they had a busy schedule.

I heartily thank Mr. Tim Thurlings (Chief Technology Officer at bluedog Security), Mr. Amit Nepal (Information Security Engineer at American Express), and Mr. Michael Boelen (Founder of CISOfy; developer of the rkhunter project) for their valuable suggestions which helped me learn the general working mechanism of the user-mode rootkits which helped me do this project.

Also, I would especially like to thank all the members of the Information Security Community, Security Researchers, Penetration Testers, Students, and anyone in the field of Information Security around the globe who helped to complete the surveys.

I would also like to expand my gratitude to all those who have directly and indirectly guided me in this project.

Batsal Nepal

ID - 16033246

Abstract

This report documents the detailed description of the project “Application-level Rootkit Detection Program for Linux”, user-mode rootkits in simple terms, implemented for both personal and educational use.

Unlike the Windows operating system, the architecture of Linux operating system is classified in 4 rings, protecting user applications from the physical components of the computer system by the help of software layers provided by the operating system itself. The outermost layer is the least privileged (application) and increases with each insider ring (shell, kernel, and hardware respectively). The problem arises when stealthy and malicious programs, like user-mode rootkits, take control of the operating system. This report starts with the introduction to the project, followed by the background scenario. With the different technologies present, a thorough analysis of the similar systems and projects, testing and evaluation of the project with a well-defined conclusion and further work. Also, accompanied by extensive analysis of research papers that have been extracted from repositories such as ACM Digital Library, Google Scholar, IEEE Xplore, SANS, ResearchGate, and other similar platforms.

Abbreviations

API	Application Programming Interface
BASH	Bourne Again Shell
BSD-2-Clause	The 2-Clause BSD License
BSD	Berkeley Software Distribution
DE	Desktop Environment
FOSS	Free and Open Source Software
GPLv2	GNU General Public License, version 2
GPLv3	GNU General Public License, version 3
MALWARE	Malicious Software
MD5	Message Digest Algorithm 5, 128-bit
*NIX	UNIX like OS
OS	Operating System
SHA	Secure Hash Algorithm
SHA1	Secure Hash Algorithm 1, 160-bit
SHA256	Secure Hash Algorithm 256, 256-bit
VM	Virtual Machine
WBS	Work Breakdown Structure

Table of Contents

Chapter 1: Introduction.....	1
1.1 Topic Introduction	1
1.2 Current Scenario	1
1.3 Problem Domain.....	2
1.4 Proposed Solution.....	4
1.5 Project as a Solution.....	4
1.6 Aim and Objectives	5
1.6.1 Aim	5
1.6.2 Objectives	5
1.7 Structure of the Report.....	5
1.7.1 Background.....	5
1.7.2 Development	6
1.7.3 Testing and Analysis	6
1.7.4 Conclusion.....	6
Chapter 2: Background	7
2.1 End Users	7
2.2 Understanding the Solution.....	7
2.3 Review of Similar System	10
2.3.1 chkrootkit.....	10
2.3.2 rkhunter	11
2.4 Review of Similar Project	12

2.4.1 DETECTING USER-MODE ROOTKITS – by Douglas Reed Beck and Yi-Min Wang	
.....	12
2.5 Review of Technical Aspects	13
2.5.1 Software Requirements for End Users	13
2.5.2 Software Used	14
Chapter 3: Development	16
3.1 Considered Methodologies	16
3.1.1 Kanban	16
3.1.2 Prototyping	18
3.1.3 Sig Sigma (6σ)	19
3.2 Selected Methodology	20
3.3 Stages in Selected Methodology	21
3.1.1 Backlog	21
3.1.2 Impact Analysis	21
3.1.3 Build	21
3.1.4 User Acceptance Testing	21
3.1.5 Release	22
3.1.6 Documentation	22
3.1.7 Done	22
3.4 Code Development	23
Chapter 4: Testing, Results, and Evaluation	30
4.1 Unit Testing	30

4.1.1 Unit Test Plan	30
4.1.2 Test Results and Logs.....	31
4.2 System Testing	36
4.2.1 System Test Plan.....	36
4.2.2 Test Results and Logs.....	37
4.3 Comparative Testing.....	58
4.4 Critical Evaluation	61
Chapter 5: Conclusion	62
5.1 Advantages.....	62
5.2 Limitations	63
5.3 Future Work	63
Chapter 6: Legal, Ethical, and Social Issues	64
6.1 Legal Issues	64
6.2 Ethical Issues	64
6.3 Social Issues	65
Chapter 7: References.....	66
Chapter 8: Bibliography.....	79
Chapter 9: Appendix.....	Error! Bookmark not defined.
A. Survey Forms.....	Error! Bookmark not defined.
B. Survey Response	Error! Bookmark not defined.
C. WBS.....	Error! Bookmark not defined.
D. Gantt Chart.....	Error! Bookmark not defined.

E. Flowchart.....	Error! Bookmark not defined.
F. Logs.....	Error! Bookmark not defined.
chkrootkit.log.....	Error! Bookmark not defined.
rkhunter.log	Error! Bookmark not defined.
full.log.....	Error! Bookmark not defined.
G. Scripts.....	Error! Bookmark not defined.
getDE	Error! Bookmark not defined.
advanceSignatureScan	Error! Bookmark not defined.
fullScan.....	Error! Bookmark not defined.
integrityScan	Error! Bookmark not defined.
rootkitSignatureScan	Error! Bookmark not defined.
rtktchk.....	Error! Bookmark not defined.
suspicious	Error! Bookmark not defined.

List of Figures

Figure 1 Prevalence and volume of malware families by platform (Fortinet, 2018).....	3
Figure 2 A survey result performed among the sysadmins.....	8
Figure 3 chkrootkit scanning for rootkits.....	10
Figure 4 rkhunter scanning for rootkits.	11
Figure 5 The Kanban Board (Digité, 2018).....	17
Figure 6 The prototyping model (Ashwin, 2017).	18
Figure 7 The Six Sigma model (Ygraph, 2012).....	19
Figure 8 Listing all the development script files.	23
Figure 9 Snipped code from the advanceSignatureScan script file.	24
Figure 10 Snipped code from the fullScan script file.	24
Figure 11 Snipped code from the getDE script file.	25
Figure 12 Snipped code from the integrityScan script file.	25
Figure 13 Snipped code from the rootkitSignatureScan script file.....	26
Figure 14 Snipped code from the rktchk script file.	27
Figure 15 Snipped code from the suspicious script file.....	28
Figure 16 A survey result about the requirement of a graphical user interface.	29
Figure 17 Finding dependent files.	31
Figure 18 Branching with select case.	32
Figure 19 Output according to the precious exit status code.	33
Figure 20 User dependent output from the os module.....	34
Figure 21 Script responding according to the arguments.	35
Figure 22 Error when executing the script by the user.	37
Figure 23 Successful when executed by the root user.	38
Figure 24 Executing the script via a normal user.	39

Figure 25 Executing the script via root user.	40
Figure 26 Executing the full scan script (PART 1).....	41
Figure 27 Executing the full scan script (PART 2).....	42
Figure 28 Executing the full scan script (PART 3).....	43
Figure 29 Executing the integrity scan script (PART 1).....	44
Figure 30 Executing the integrity scan script (PART 2).....	45
Figure 31 Executing the integrity scan script (PART 3).....	46
Figure 32 Executing the basic rootkit scan script (PART 1).	47
Figure 33 Executing the basic rootkit scan script (PART 2).	48
Figure 34 Executing the advanced rootkit scan script (PART 1).	49
Figure 35 Executing the advanced rootkit scan script (PART 2).	50
Figure 36 Executing the suspicious scan script (PART 1).	51
Figure 37 Executing the suspicious scan script (PART 2).	52
Figure 38 Executing the getDE script by a normal user.	53
Figure 39 Executing the getDE script by the root user.....	54
Figure 40 Displaying one of the logs (etc.log) among the integrity scan log files.	55
Figure 41 All the logs inside the log directory.	56
Figure 42 Displaying one of the logs (advance.log) among the log files.	57
Figure 43 Logs to be analyzed for comparison.....	58
Figure 44 Key points from the chkrootkit.log file.	58
Figure 45 Key points from the rkhunter.log file.....	59
Figure 46 Key points from the full.log file from the author's script.....	59
Figure 47 Survey 1.....	Error! Bookmark not defined.
Figure 48 Survey 2.....	Error! Bookmark not defined.
Figure 49 Survey 3.....	Error! Bookmark not defined.

Figure 50 Survey 4.....	Error! Bookmark not defined.
Figure 51 Response 1.....	Error! Bookmark not defined.
Figure 52 Response 2.....	Error! Bookmark not defined.
Figure 53 Response 3.....	Error! Bookmark not defined.
Figure 54 Response 4.....	Error! Bookmark not defined.
Figure 55 WBS.	Error! Bookmark not defined.
Figure 56 Gantt Chart.	Error! Bookmark not defined.
Figure 57 Flowchart for getDE script.	Error! Bookmark not defined.
Figure 58 Flowchart for integrity scan script.....	Error! Bookmark not defined.
Figure 59 Flowchart for basic and advance scan scripts.....	Error! Bookmark not defined.
Figure 60 Flowchart for suspicious scan script.....	Error! Bookmark not defined.
Figure 61 Flowchart for full scan script.....	Error! Bookmark not defined.
Figure 62 Flowchart for rktchk script.	Error! Bookmark not defined.

List of Tables

Table Number 1 - Unit Test Case 1	31
Table Number 2 - Unit Test Case 2	32
Table Number 3 - Unit Test Case 3	33
Table Number 4 - Unit Test Case 4	34
Table Number 5 - Unit Test Case 5	35
Table Number 6 – System Test Case 1	37
Table Number 7 – System Test Case 2	39
Table Number 8 – System Test Case 3	41
Table Number 9 – System Test Case 4	44
Table Number 10 – System Test Case 5	47
Table Number 11 – System Test Case 6	49
Table Number 12 – System Test Case 7	51
Table Number 13 – System Test Case 8	53
Table Number 14 – System Test Case 9	55
Table Number 15 – System Test Case 10	56
Table Number 16 - Critical Evaluation with Similar Systems	60

Chapter 1: Introduction

1.1 Topic Introduction

"Attack him where he is unprepared, appear where you are not expected" (Tzu, 1971). A type of malware that surely didn't hit the spikes in comparison with the other categories according to various threat reports clearly shows how stealthy a rootkit can be. Generally, a malicious user intending to stay undetected in a system deploys rootkit that specifically makes it easier for an attacker to stay under the hood. In laymen term, a rootkit is a malicious program that is derived from *NIX system where the *root* is the most privileged user and *kit* refers to a set of programs or script (Hoglund & Butler, 2005). In short, the malicious program or a script executed with the root privileges are called rootkits. Keeping that in mind, user-mode rootkits are those which execute in the user space within the operating system. Researchers generally classify user-mode rootkits as a type of rootkit that does not hook any of the kernel components but may hook system APIs for manipulating device drivers and modifying the core files. In most of the cases, however, a malicious version of a legitimate script being injected using various techniques bypassing the security architecture of the software application or even the operating system itself.

1.2 Current Scenario

Most of the user-mode rootkits are embedded within other legitimate applications and file formats. It lurks users in downloading malicious contents along with the so-called "free" contents available on the internet. *"Although to some extent rootkits pose a unique set of challenges to the security industry, the technologies are evolving on both sides of the malware*

war, and while rootkits were, until fairly recently, a specialist security preoccupation — mainly in the UNIX/Linux communities — stealth is nothing new to the anti-virus industry” (Harley & Lee, 2009). However, there has not been any research for the user-mode rootkits in depth even after a decade, at least at the time of publishing this report. As there might be closed source projects that an antivirus company might be distributing of and are still unknown to the most of us, there are significantly lesser in number solely focusing for the user-mode rootkits in Linux systems.

1.3 Problem Domain

Various source proved that rootkits are used by intelligence agencies, government agencies, crackers, hackers, script kiddies, etc. through multiple attack actions and mechanisms using application script (Stiawan, et al., 2012). Moreover, rootkits are one of the biggest threats since they are unpredictable because rootkits are flexible with the growing trend; from the Sony-BMG Copy Protection rootkit exploiting what Sony calls to be an injected for a legitimate reason (BerkeleyTech. L.J., 2006) till the present day. It is now used as an agent to hide other malicious programs (Uppal, et al., 2014) so perfectly that even modern detection systems could not detect the existence (used with Trojans, Botnets, and other categories of malware). Rootkits today are used for various purposes unlike before when it used to be simple trojan programs such *netstat*, *ps*, *ls*, *top*, etc. (Butler, 2004) and are crafted so perfectly that it is almost impossible to detect, making the situation look like a perfect crime.

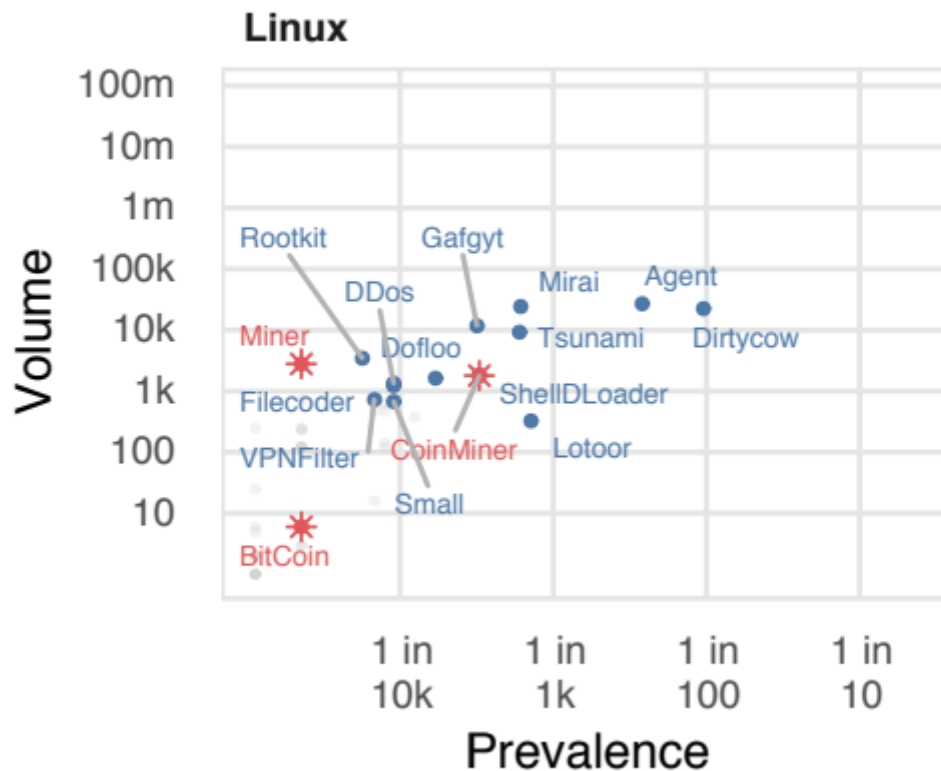


Figure 1 Prevalence and volume of malware families by platform (Fortinet, 2018).

According to the Quarterly Threat Landscape Report (Q3) prepared by Fortinet, the volumes of rootkits for Linux are over than 5,000 in numbers. Similarly, the Jenkins crypto miner is also suspected to be a rootkit to hide the processes created by the miner. Likewise, Alina botnet had an additional rootkit feature that hooked some system API's to download or execute malicious scripts as needed without any user intervention (Fortinet, 2018). The outcomes of jaw-dropping and thrilling facts like this prove that rootkit still exists though it did not evolve in any significant manner.

1.4 Proposed Solution

As there are very less program that specifically performs rootkit scan, i.e. *rkhunter* and *chkrootkit*, and almost none of them solely targeted for the user-mode rootkits, this project is determined totally for detecting the user-mode, aka application-level rootkit, in the Linux system. As mentioned in the proposal and interim report, the project will be focused for the Debian 9, codename “stretch” as for the initial release of the program and will be completely open sourced under the *GPLv2* license in platforms like *GitHub* which enables it to be in the paradigm of FOSS. This project will scan for application-level (user-mode) rootkit performing a signature scan. However, an integrity scan of the core binary and system files are also checked to ensure that the OS is not tampered by modified versions of binary files and libraries.

1.5 Project as a Solution

As there is no such program that solely focuses on user-mode rootkits for the Linux system, this project aims to be helpful for users in detecting user-mode rootkits for the Linux system. The main reason for doing this project is because rootkits are still unpredictable and are adopting the changes as the trend of malware is directing towards. Despite not being able to detect new rootkits, searching by signature scans is one of the efficient ways to detect existing rootkits because it is still being integrated with the modern rootkit detection system. Besides, focusing on the Debian system makes it much more efficient for users using this distribution around the globe. This project can minimize the threats posed by the user-mode rootkits as it will also be looking for the changes in the core system files of the operating system which makes it easier to detect user-mode rootkits as all they do is modify core system files. The program will be handy for students, researchers, and any others who are working in the field of rootkits as one can tweak the program according to their personal desire because of the

dynamism of the project and might be one of the powerful programs to detect rootkits with future releases.

1.6 Aim and Objectives

1.6.1 Aim

This final year project is to develop a user-mode rootkit detection program flexible for Debian Linux distribution which will help to find rootkits in a much simpler and efficient manner.

1.6.2 Objectives

The various objectives followed during this project are listed below in accordance with the implemented software development methodology i.e. Kanban from the Agile Framework. The very first step is to research more on the selected project, conduct relevant surveys in the community and perform a feasibility study of the project. The next step is to prepare the Kanban board that covers and manages the overall project's time and work. Following the WBS and Gantt chart, scripting and testing of the project are executed according to the survey result. Lastly, preparation for presentation and demonstration of the project is rehearsed and executed.

1.7 Structure of the Report

1.7.1 Background

This section of the report consists of an in-depth description of the project. This includes all the information about the end users, understanding the solution, reviews of similar systems and projects, and review of technical aspects of the project.

1.7.2 Development

This section of the report consists of the development stages executed for the project. It discusses the considered methodologies for the project, selected methodology, stages in the selected methodology, and finally the code development phase.

1.7.3 Testing and Analysis

This section consists of testing and review of the project. It includes the demonstration of system testing of the project, unit testing of the project, comparative testing with the similar systems, and critical evaluation of this project.

1.7.4 Conclusion

This section summarizes the project with a distinct conclusion, advantages of the project, limitations of the project, and finally discussion about the future works for the project.

Chapter 2: Background

2.1 End Users

Since this project is focused on the open source community, the users can be anyone using it around the globe involved in almost any domain as each user's perspective can be completely different from others. Being one of the security tools, users can be anyone simply wanting to check for rootkits in their system or an active security researcher wanting to contribute to this project. To achieve enough components for the project, surveys had been performed via *Google Forms* on various platforms like *LinkedIn*, *Twitter*, *Facebook*, and other similar sites. Besides them, various report published at *ACM Digital Library*, *IEEE Xplore*, *ResearchGate*, etc. also helped in gathering ideas to implement in the project. The various user-mode rootkit sample had been downloaded from malware research samples from various sites, including the shared *Google Drive* links. Although it is an unethical behavior downloading malicious contents from malicious *HTTP* (Hypertext Transfer Protocol) and *FTP* (File Transfer Protocol) links, the signature had to be made perhaps the consequences of infecting the author's own personal computers. Everything for the project, including what system to choose for the initial release of the project to what additional features does it require, is gathered from the sources mentioned above that helped me a lot in the development of the project (Refer Appendix for the survey form and result).

2.2 Understanding the Solution

Bounded with a very limited amount of time, this project only checks for user-mode rootkits by performing a signature scan. To make sure that no core files have tampered, an integrity scan of the system is done by comparing the base hash of the installed Debian OS. The program

is scripted according to the survey results and the signature behavior, as this method is still used in the modern detection systems (Mathur & Hiranwal, 2013). According to one of the surveys done among the people who had have worked as a sysadmin, the top 4 critical file system hierarchy in the Linux system is: */bin*, */etc*, */lib*, and */sbin*. Keeping that in consideration, the integrity scan will help to find most of the unknown rootkits that rely on modifying the core binaries.

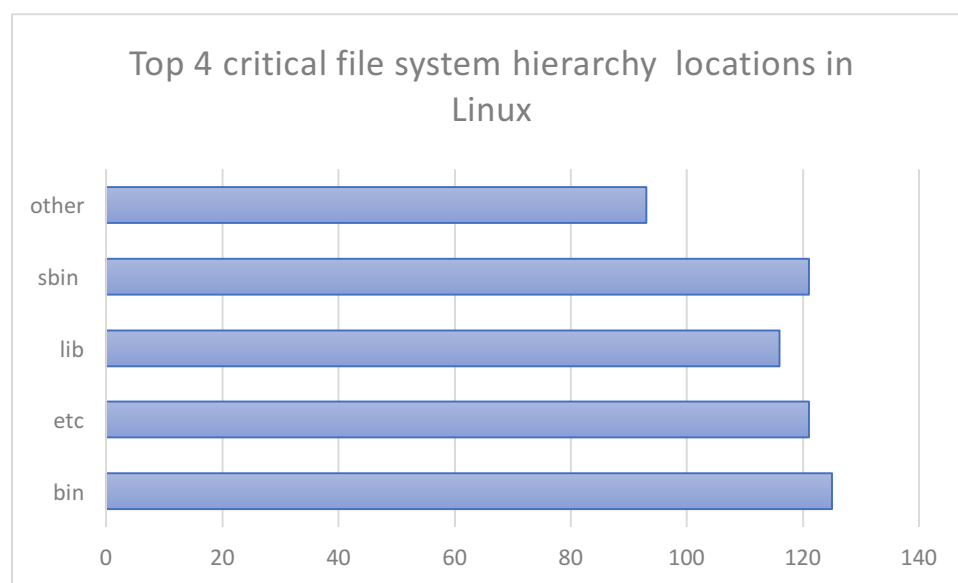


Figure 2 A survey result performed among the sysadmins.

For the purpose of the signature scan, every individual user-mode rootkit had been downloaded from the internet and analyzed in an isolated virtualized environment with the help of two completely open sourced software: *cuckoo sandbox* (licensed under *GPLv3*) and *Oracle VM VirtualBox* (licensed under *GPLv2*) as they are considered as the safest and efficient procedures to analyze signature pattern (Sokol, et al., 2015) of almost any sort of malware. The signature is then scripted in *BASH* according to the results extracted after the analysis.

However, procedures executed in simplifying every rootkit is not documented as it directs the flow of report completely off topic. Because most of the rootkits were downloaded from malicious web servers and FTP (file transfer protocol) servers that most of the modern web

browsers deny accessing them. Moreover, extinct rootkits which only showed up either once or twice had no concrete source of information besides the research reports and reverse engineering blogs written many years ago. Besides, some of the rootkits that are vendor-specific had to be executed on a different machine with the different kernel versions, library releases, and device drivers' versions. Hence, this report does not document any of the contents related to the extraction of the signature from various user-mode rootkits.

2.3 Review of Similar System

2.3.1 chkrootkit

The *chkrootkit* project, developed by Pangeia Informatica; licensed under the *BSD-2-Clause*; is an open-sourced rootkit scanner for *NIX system. The program is written in the shell script that scans for signs of rootkits via its own local database by performing signature scans. Mostly, it is focused on core system binaries of the supported operating systems. Besides, it also checks for worms and loadable kernel modules (LKMs). The best part of using this program is that rootkits can be checked via custom binary files according to user preference from a separate CD-ROM.

Although this system has been running for over a decade now, there are various disadvantages. One of them, sometimes, even ignores some rootkit scans. For instance, the *chkrootkit* program is intended to be executed from the root directory (/). If not, the program skips the scan for certain rootkits and features displaying “*not tested*” right after the binary check. Moreover, the scan outputs some large text while performing suspicious files and directories scan that certainly confuses a user. And finally, it is not solely focused on the user-mode rootkits.

```
tree@house:~$ sudo chkrootkit
ROOTDIR is '/'
Checking amd'... not found
Checking basename'... not infected
Checking biff'... not found
Checking chfn'... not infected
Checking chsh'... not infected
Checking cron'... not infected
Checking crontab'... not infected
Checking date'... not infected
Checking du'... not infected
Checking dirname'... not infected
Checking echo'... not infected
Checking egrep'... not infected
Checking env'... not infected
Checking find'... not infected
Checking fingerd'... not found
Checking gpm'... not found
Checking grep'... not infected
Checking hdparm'... not found
Checking su'... not infected
Checking ifconfig'... not infected
Checking inetd'... not infected
Checking inetdconf'... not found
Checking identd'... not found
Checking init'... not infected
Checking killall'... -
```

Figure 3 *chkrootkit* scanning for rootkits.

2.3.2 rkhunter

The rootkit hunter project developed by Michael Boelen; licensed under the *GPLv2*; is an open-sourced rootkit scanner for *NIX system. The program is scripted in Perl programming language and is currently maintained by SourceForge. It scans for exploits like backdoors, e-mail injection, buffer overflows, format string bugs, and rootkits by the help of signature scan. The best part of using this tool is that it searches for hidden files and directories in the host system. The *sha1* hash is kept in an online remote server hence there is no chance of corruption of the hash. However, it only checks for limited number of system binaries that are usually tampered by the rootkits.

Although this system has been running for over a decade, there are various disadvantages. The program does not clarify the warnings that are shown in the scan. Even in the log file that it generates after the scan, there are no such clue for the warning messages during the scan. Moreover, the program requires user interaction multiple times just to perform a full scan of the system. It does not have any choices to run only specific rootkits scans rather than performing the whole scan. And finally, it is not focused solely on the user-mode rootkits.

```
tree@house:~$ sudo rkhunter --check
[sudo] password for tree:
[ Rootkit Hunter version 1.4.2 ]
Checking system commands...
Performing 'strings' command checks
Checking 'strings' command [ OK ]
Performing 'shared libraries' checks
Checking for preloading variables [ None found ]
Checking for preloaded libraries [ None found ]
Checking LD_LIBRARY_PATH variable [ Not found ]
Performing file properties checks
Checking for prerequisites [ Warning ]
/usr/sbin/adduser [ OK ]
/usr/sbin/chroot [ OK ]
/usr/sbin/cron [ OK ]
/usr/sbin/groupadd [ OK ]
/usr/sbin/groupdel [ OK ]
/usr/sbin/groupmod [ OK ]
/usr/sbin/grpck [ OK ]
/usr/sbin/nologin [ OK ]
/usr/sbin/pwck [ OK ]
/usr/sbin/rsyslogd [ OK ]
/usr/sbin/useradd [ OK ]
/usr/sbin/userdel [ OK ]
```

Figure 4 rkhunter scanning for rootkits.

2.4 Review of Similar Project

2.4.1 DETECTING USER-MODE ROOTKITS – by Douglas Reed Beck and Yi-Min Wang

This patent publication, published in 2011, is mainly focused on finding user-mode rootkits via monitoring the system API calls, both low level and high level. It simplifies the detection mechanism that relies on the state of various resources of a computer system. As a concept for the work-a-round of the project, the authors illustrated some of the techniques that can be used to hunt down rootkits. In short, to identify the hidden resources in the computer system, their approach of the security system invoked a high-level function of the user-mode which is intercepted and modified by the malware to identify the malicious resources and another security system that invokes the low-level function of the kernel mode that is not intercepted and modified by malware to identify the legitimate resources. After both level function is invoked, the security system compares the result of the legitimate and malicious resources list. However, if the legitimate list contains a resource that is not in the malicious list, then the security system may consider the resource to be hidden which can be a sign of a user-mode rootkit being installed by directly invoking low-level functions which are endeavoring to hide resources by intercepting user mode invocations of high-level functions (Beck & Wang, 2011).

However, this publication is slightly more focused towards the user-mode rootkits that use the functionality of the kernel functions. Being the only project focusing the user-mode rootkits in the *NIX systems during the development of this project, it only relies on detecting rootkits via system API calls. Although the signature scan is implemented in the project, doing so creates difficulties in detecting user-mode rootkits in an inactive state as it only detects via system hooks.

2.5 Review of Technical Aspects

To implement this project, readers must have the basic knowledge of the Linux operating system and familiar with the command line interface. No in-depth knowledge about the Linux operating system is required. However, readers wanting to contribute to this project must have an adequate understanding of the *NIX systems, malware analysis, virtualization, and programming; shall be involved in the information security community; either casually or formally. All the changelogs and pieces of code shall be submitted with proper documentation at the author's *GitHub* repo (see README.md attached with the scripts).

2.5.1 Software Requirements for End Users

1. Debian 9 OS

The project is for the moment only available for the Debian 9 OS. Hence, this OS is recommended for the users. However, script can be tweaked, and the base hash can be user defined.

2. BASH Shell

The project needs a command line interpreter in order execute the scripts. However, *BASH* is pre-loaded with the Debian OS and is used as the default environment to execute the script regardless of what terminal being used.

3. Python 3

The project has a *NIX like execution which is made with an ease using Python scripting language, compatible with any of the dot release from the version 3, pre-

loaded with the Debian OS. It uses two of the pre-installed libraries to extract the maximum efficiency that *BASH* couldn't achieve.

a. os

This module provides a portable way of using operating system dependent functionality. The *os* library is used to invoke the OS commands inside the Python script and extract the information about the installed DE of the host OS (Beazley & Jones, 2013).

b. argparse

This module makes it easy to write user-friendly command-line interfaces. The *argparse* library is used to make the program take flags as an argument like in the *NIX systems at the time of execution (Beazley & Jones, 2013).

For the hardware requirements, the end user's personal computer must match the minimal hardware requirements provided by the Debian operating system's official website.

2.5.2 Software Used

1. Debian 9.8.0 OS (with *GNOME* DE)

Since the crucial component of the project is the Debian OS, the GNOME edition of the Debian OS, version 9.8.0 is used.

2. Vim (version 8)

Being one of the most powerful text editors in the Linux systems, it was unarguably selected as the text editor to develop and debug the scripts.

3. GNOME Terminal

The default terminal in the GNOME DE is one of the easiest and efficient command line terminal found in the GNOME environment for executing and displaying the results of the scripts.

4. Cuckoo Sandbox (version 2.0.6)

One of the best isolated sandboxed environments recommended by most of the security professionals in the field of malware analyzing with efficient logging functionality of the occurred events. In this project, all the outcomes of the analyzed user-mode rootkits were scripted from the generated logs by the *cuckoo sandbox*.

5. Oracle VM VirtualBox (version 6)

One of the best hypervisor solutions available for almost all platforms to install relevant operating system mostly used for testing purposes. In this project, it is used to install the OS and execute malware script with some special requirements recommended by the malware researchers and the Oracle's documentation.

6. *dd (data duplicator)*

One of the best tools in Linux systems for burning various disk image formats in various external storage devices. It is used to create a bootable USB drive for installing the Debian OS.

Chapter 3: Development

3.1 Considered Methodologies

There are a handful of software development methodologies that best suits the project.

However, to top-list 3 among them, they would be:

3.1.1 Kanban

“Kanban is not actually a software development methodology, but simply a way of visualizing and tracking work” (Leontranter, 2016). In the early 1940s, Toyota implemented the Kanban technique by getting inspirations from the supermarket that interprets to “visual signal” in English. As the name, development is tracked via visual cards in a Kanban board. Here, the project is done by prioritizing the tasks, developing, and limiting the work in progress. It is one of the chosen Agile based methodology because of its flexibility. The main advantage of Kanban is that it improves the flow of delivery of the project. Additionally, it is one of the easiest and flexible methodologies to be implemented. The main disadvantage is that lack of timeframes like in other methodologies. Besides, the work limit is created by the work in progress according to developers’ preference.

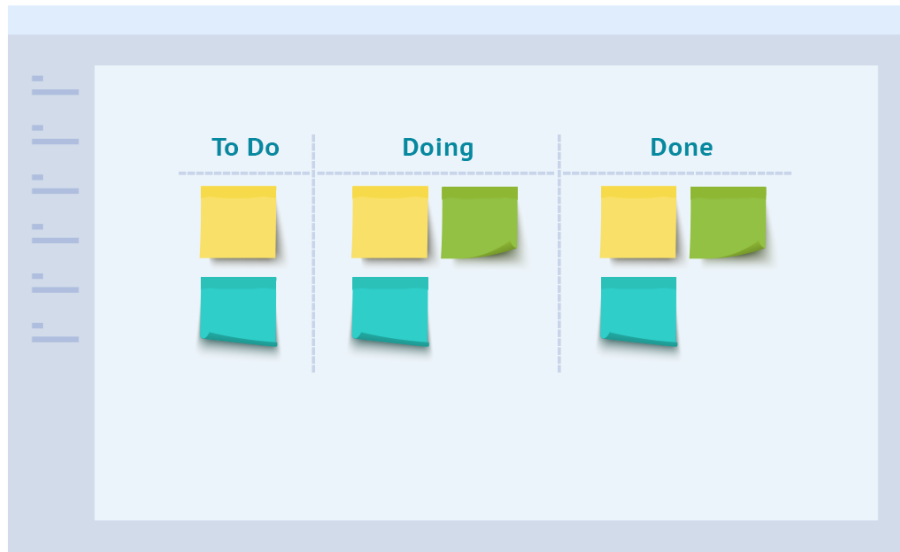


Figure 5 The Kanban Board (Digité, 2018).

3.1.2 Prototyping

“It is the process of implementing the presumed software requirements with an intention to learn more about the actual requirements or alternative design that satisfies the actual set of requirements” (New Horizon College, 2018). Here, a prototype of the system is made beforehand according to the initial requirement provided by the client. Multiple iteration or requirements are done and evaluated by the client. The main advantage of this model is that the prototype of a working system is pre-built and sent for the evaluation to the client which helps developers have a clear view on what must be done and what the user wants. Additionally, there is less chance of errors since the client is involved in the design. The main disadvantage of this model is that the complexity of the project is increased since the project may go way further than expected plans. Besides, this model is time-consuming and is difficult for projects with a tight budget.

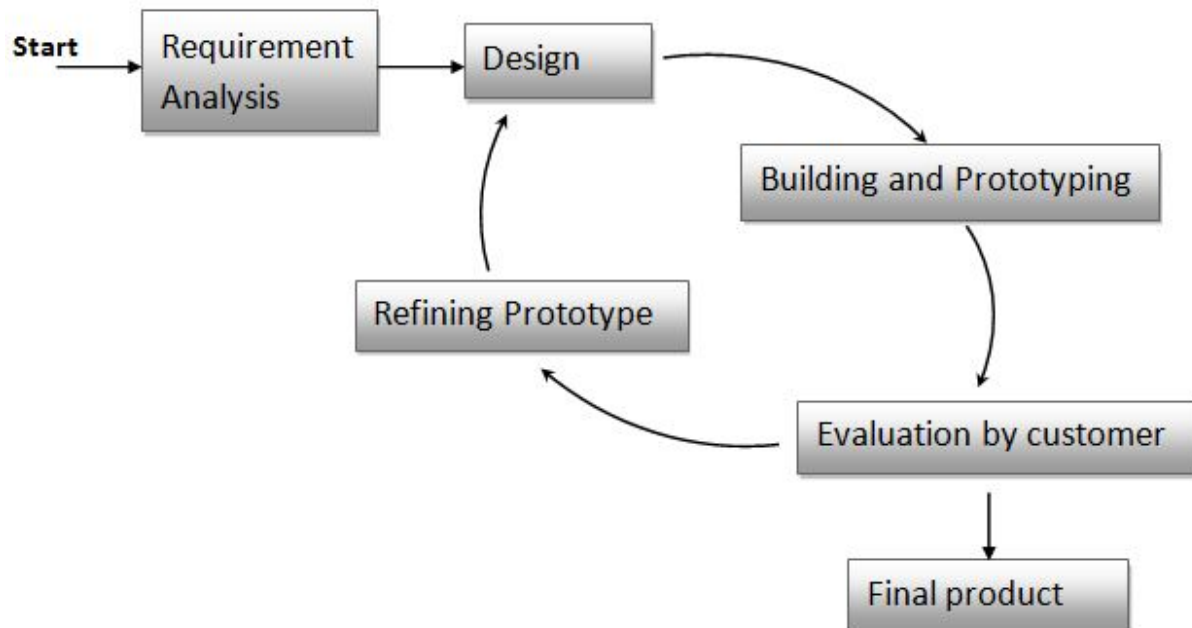


Figure 6 The prototyping model (Ashwin, 2017).

3.1.3 Sig Sigma (6σ)

“Six Sigma is a set of techniques/tools that aim to improve company output, which is typically done by finding and eliminating inconsistencies or defects in service or product processes” (Pearson, 2018). In 1986, Motorola developed this methodology to achieve near perfection quality products. Here, each process is kept at near perfection because quality plays a vital role and defect is almost negligible since its efficiency is 99.99966%. The main advantage is that it focuses on the improvement of every single portion of the process and not just the final product. Additionally, it also helps stand out among competitors because of the quality product. The major disadvantage is that it can be very expensive to small projects because of its training costs. Besides, it is also very difficult to sustain a level of perfection throughout the whole project.

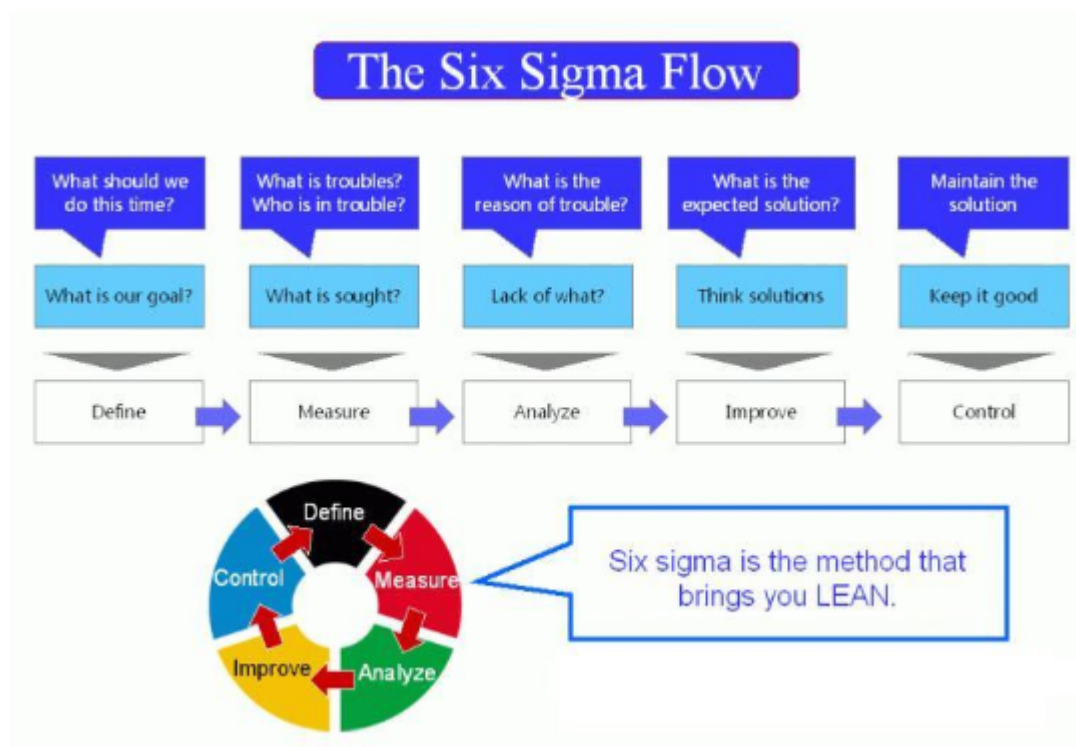


Figure 7 The Six Sigma model (Ygraph, 2012).

3.2 Selected Methodology

The Kanban software development methodology is the most flexible and chosen software development methodology as there is no such control in the flow of tasks at the time of execution. It is an ideal choice whenever there are a lot of changes to be made in the system. Moreover, tasks are presented in a simpler form like a timeline where a task can be further subdivided into various small tasks. It is taken as the right choice for the smaller projects where near persistent development is expected. However, prototyping and sig sigma are lesser flexible than the Agile framework. To justify the reasons for not selecting them is because of the complications they create during the execution of the development phase. Prototyping first leads to implement the project first and then tweaking. As there are many prototypes, the final system might be incomplete without the prototypes designed for the system. Problems like inadequate analysis are one of the major drawbacks for choosing the prototyping methodology. Moreover, this may increase the complexity of the overall system as the main agenda might increase elsewhere than the original plans. On the other hand, sig sigma is used on an enterprise level and more focused on the production side of the product. Though it is a structured way for solving the problem qualitatively, sometimes, too much emphasis on the tool instead of right solutions diverts the flow of the project. Besides, a lot of training is to be required when this methodology is to be implemented and is fully dependent on the statistics.

Hence, the Kanban software development methodology from the Agile framework was chosen and is best suited for this project.

3.3 Stages in Selected Methodology

There are 7 steps that must be followed for the successful implementation of the Kanban methodology for the project. They are:

3.1.1 Backlog

Backlogs are tasks that need to be delivered. They can also be considered as the ideas for the system. In this project, the pieces of stuff required to fully complete the system are kept as backlogs.

3.1.2 Impact Analysis

The impact analysis is just like a research phase where the scope and impact of the tasks are evaluated. Here, performing a feasibility study of the project was executed in this stage.

3.1.3 Build

This stage is where the actual development, testing, and debugging occurs. Here, scripting, testing, and debugging of the system was performed that included algorithms, flowcharts, pseudocode, and integrating scripts.

3.1.4 User Acceptance Testing

As this phase is related to the end consumers' testing, it is skipped since the project is done by the author from scratch with the help of multiple surveys. Hence, this phase is skipped for the project.

3.1.5 Release

This stage is where the system is to be released. The full system along with a readme file is released in platforms like *GitHub* under the *GPLv2* license. Here, the project is released in *GitHub* now for usage and feedback.

3.1.6 Documentation

This stage is only targeted for documenting every procedure in a detailed manner. However, considering the limitation of the word limit in the FYP, only required and limited information is provided following the interim report and the project proposal. Here, the documentation is done under the guidance of the author's supervisor(s).

3.1.7 Done

This stage is the final stage of the project. The system is only to be considered as done if the system fully satisfies the planned outcome. Here, the system is considered as done because it fulfils the requirement proposed in the project proposal including the survey results from the community. However, a modification is done by the help of feedback provided by the user according to their experience being under the license agreement.

3.4 Code Development

During the development phase, aka Built phase in the Kanban software development methodology, the author had to go through countless hectic hours analyzing each user-mode rootkit available on the internet for the *NIX systems. After analyzing all the known user-mode rootkits, there had been a lot of conflicts whether to make a database of the known malicious hash or not. Fortunately, private messages in sites like *LinkedIn*, *Facebook*, *Twitter*, etc. with various brilliant minds in the field of rootkits suggested going according to the behavior resulted by the sandboxed environment rather than just making a database of the known hash.

```
tree@house:~/Documents/FYP_Files/rktchk-master$
tree@house:~/Documents/FYP_Files/rktchk-master$ ls -F
advanceSignatureScan*  fullScan*  home/  LICENSE*  README.md*  rktchk*
base/  getDE*  integrityScan*  logs/  rootkitSignatureScan*  suspicious*
```

Figure 8 Listing all the development script files.

To list all the script files (omitting the directories for the moment), there are just 7 of the scripts; 5 being written in *Bash* and 2 in *Python 3*. Only a short snippet of the scripts is posted below because of the lengthiness of the report (Refer to development folder for full scripts). Python 3 is only used for the scripts that created complications from the *bash* at the time of development.

The first script, “*advanceSignatureScan*” scans for user-mode rootkits in the root file system (/) as it can be found in any of the directories because of its randomized behavior. However, with a simple yet powerful script, it can be easily detected.

```
tree@house:~/Documents/FYP_Files/rktchk-master$  
tree@house:~/Documents/FYP_Files/rktchk-master$ tail -n 18 advanceSignatureScan  
# Ovason Rootkit  
echo "  
!! Checking for Ovason Rootkit in advance mode.... !!"  
find / -type f \( -name 'ovas0n' -o -name 'ovason' \)  
  
if [ $? -eq 0 ]  
then  
    echo "  
    [ Ovason Rootkit Found! ]"  
else  
    echo "  
    [ NONE FOUND! ]"  
fi  
  
# Script end time  
echo "Script ended on $(date +%F) at $(date +%T)"  
## Detection in advance mode ends here. tree@house:~/Documents/FYP_Files/rktchk-master$
```

Figure 9 Snipped code from the *advanceSignatureScan* script file.

The second script, “*fullScan*” executes all the scripts at once. This script is like a call function to other scripts.

```
tree@house:~/Documents/FYP_Files/rktchk-master$  
tree@house:~/Documents/FYP_Files/rktchk-master$ tail -n 18 fullScan  
else  
    echo "  
{ Script for advance rootkit(s) scan not found! } "  
fi  
  
if [ -f suspicious ]  
then  
    echo "  
{ Script for suspicious file/settings scan found! }  
!! Performing suspicious file/setting scan.... !!"  
    chmod +x suspicious  
    ./suspicious  
else  
    echo "  
{ Script for suspicious file/settings scan not found! } "  
fi  
  
# # This program ends here. tree@house:~/Documents/FYP_Files/rktchk-master$
```

Figure 10 Snipped code from the *fullScan* script file.

The third script, “*getDE*” extracts the installed DE from the host system. This script is the only script that *must* be executed by the default logged in user, not with the user id being 0.

```
tree@house:~/Documents/FYP_Files/rktchk-master$
tree@house:~/Documents/FYP_Files/rktchk-master$ tail -n 18 getDE
#!/usr/bin/env python3
import os

if (os.getuid() == 0):
    print ("Please run this program as a normal user.")
    print ("NOTE: The user account that you have created for your personal use.")
    exit()
else:
    print ("Detected Desktop Environment: ")
    os.system("env | grep -i 'xdg_current_desktop' | awk -F\"=\" '{print$2}' ")
# This script ends here.tree@house:~/Documents/FYP_Files/rktchk-master$
```

Figure 11 Snipped code from the *getDE* script file.

The fourth script, “*integrityScan*” performs the integrity check of the Debian host OS. It checks for the *sha256* hash of the core system binaries and libraries which are mostly modified by user-mode rootkits.

```
tree@house:~/Documents/FYP_Files/rktchk-master$
tree@house:~/Documents/FYP_Files/rktchk-master$ tail -n 18 integrityScan
cat host/lib.sha256 | awk '{print $2,$1}' | sort > host/lib.txt
chkLIB

echo "
!!/sbin/ !!
"

cat host/sbin.sha256 | awk '{print $2,$1}' | sort > host/sbin.txt
chkSBIN

echo "
[ DONE! ]

{ See the logs in the log/ directory. }"

# Script end time
echo "Script ended on $(date +%F) at $(date +%T)"

## Integrity scan ends here.tree@house:~/Documents/FYP_Files/rktchk-master$
```

Figure 12 Snipped code from the *integrityScan* script file.

The fifth script, “*rootkitSignatureScan*” performs the rootkit scans in the specific directories that rootkits are found. These user-mode rootkits are found in the same location and hence checks for that directed location only.

```
tree@house:~/Documents/FYP_Files/rtktchk-master$  
tree@house:~/Documents/FYP_Files/rtktchk-master$ tail -n 18 rootkitSignatureScan  
    echo "  
    [ FOUND! ]"  
else  
    echo "  
    [ NONE FOUND! ]"  
fi  
  
echo "  
!! Some more checks.... !!"  
find /usr/include/ -type f -name '*kit*'  
  
echo "  
[ DONE! ]"  
  
# Script end time  
echo "Script ended on $(date +%F) at $(date +%T)"  
  
# This script ends here.tree@house:~/Documents/FYP_Files/rtktchk-master$
```

Figure 13 Snipped code from the rootkitSignatureScan script file.

The sixth script, “*rtktchk*” is the main script that is to be executed by the end user. This script reflects the *NIX execution style with arguments as the flags to be used at the time of the execution. Moreover, this script gives the user an ability to pick a type of scan.

```
tree@house:~/Documents/FYP_Files/rtktchk-master$  
tree@house:~/Documents/FYP_Files/rtktchk-master$ tail -n 18 rktchk  
elif args.full:  
    logstat = os.path.isdir('./logs')  
    if logstat == True:  
        print ("Log directory exists. Using it for storing logs.")  
    else:  
        print ("Log directory does not exists. Making one....")  
        os.system("mkdir logs")  
  
    os.system("bash fullScan | tee logs/full.log")  
elif args.denv:  
    os.system("chmod +x getDE")  
    os.system("./getDE")  
else:  
    parser.print_help()  
# This script ends here.  
tree@house:~/Documents/FYP_Files/rtktchk-master$ _
```

Figure 14 Snipped code from the *rtktchk* script file.

The final script, “*suspicious*” checks for any of the suspicious files and settings in the host OS. Besides the suspicious file checks, the script is totally based on one of the survey results performed by the author.

```
tree@house:~/Documents/FYP_Files/rktchk-master$  
tree@house:~/Documents/FYP_Files/rktchk-master$ tail -n 18 suspicious  
  
cat /etc/passwd | awk -F: '{print $1,$3}'  
  
echo "  
[ DONE! ]"  
  
echo "  
{ Check for suspicious files in the /tmp/ directory.... }"  
  
find /tmp/ -executable -type f  
  
echo "  
[ DONE! ]"  
  
# Script end time  
echo "Script ended on $(date +%F) at $(date +%T)"  
  
# # This program ends here. tree@house:~/Documents/FYP_Files/rktchk-master$ _
```

Figure 15 Snipped code from the suspicious script file.

Only the “*getDE*” script requires non-root privileges to execute. For the rest of the scripts, execution with the user id 0 is mandatory. Moreover, there is **no** graphical user interface version available for the moment because of the results from one of the surveys performed as end users don’t have a problem with the scripts being completely in the command line only.

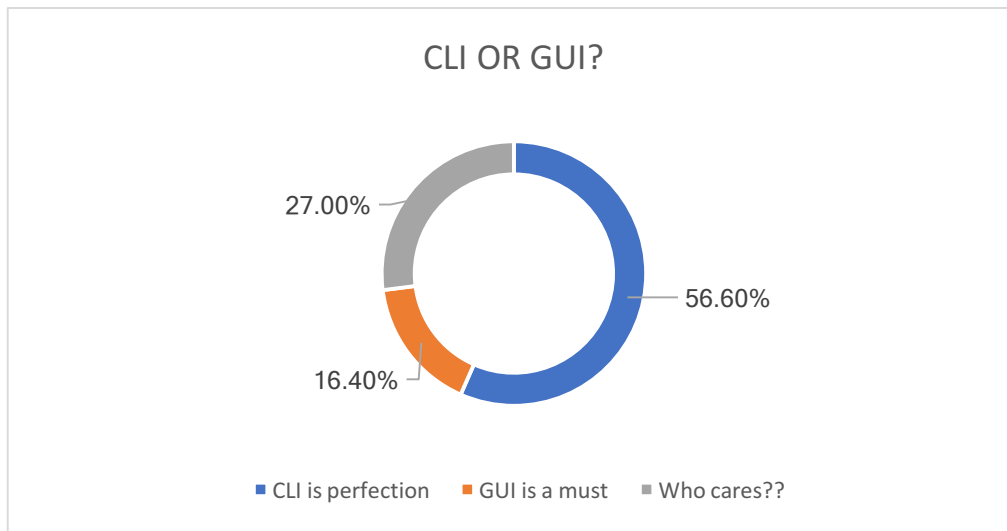


Figure 16 A survey result about the requirement of a graphical user interface.

Chapter 4: Testing, Results, and Evaluation

4.1 Unit Testing

In order to achieve the full system working as intended, there were various smaller tests done before scripting the actual script for the project.

4.1.1 Unit Test Plan

CASES	AGENDA
1	To test if-else statement to search for dependent files in Bash
2	To test select case statement to branch script via Bash
3	To analyze exit code of the previous command and branch output inside a Bash script
4	To invoke system commands via <i>os</i> module in Python 3
5	To test arguments via <i>argparse</i> module in Python 3

4.1.2 Test Results and Logs

Table Number 1 - Unit Test Case 1

Objective	To test if-else statement to search for dependent files in Bash
Expected Result	The script shall check for a specific file, find and output the relevant information.
Actual Result	The script successfully found the file and returned expected relevant output
Analysis	As this small module can find for dependent files at the time of execution, it can be used as a script validation in the real system
Conclusion	Successful Results

```

tree@house:~$ ls | grep selectCase
selectCase
tree@house:~$ ./checkIfElse
!! Checking the existence of the file.... !!
{ File Found! }
tree@house:~$ rm selectCase
tree@house:~$ ./checkIfElse
!! Checking the existence of the file.... !!
{ File not Found! }
tree@house:~$ cat checkIfElse
#!/bin/bash
echo "!! Checking the existence of the file.... !!"

if [ -f selectCase ]
then
    echo "{ File Found! }"
else
    echo "{ File not Found! } "
fi
tree@house:~$

```

Figure 17 Finding dependent files.

Table Number 2 - Unit Test Case 2

Objective	To test select case statement to branch script via Bash
Expected Result	The script must display the correct message according to the input provided
Actual Result	The script displayed the correct message according to the input provided
Analysis	It can be used as a branching point to select the base hash for the final system.
Conclusion	Successful Results

```

tree@house:~$ ./selectCase

  Select Desktop Environment installed in your system:
    1) CINNAMON
    2) GNOME
    3) KDE
    4) LXDE
    5) MATE
    6) XFCE
    7) CLI | WM
7
CLI | WM
tree@house:~$ ./selectCase

  Select Desktop Environment installed in your system:
    1) CINNAMON
    2) GNOME
    3) KDE
    4) LXDE
    5) MATE
    6) XFCE
    7) CLI | WM
0
Invalid Selection!
tree@house:~$

```

Figure 18 Branching with select case.

Table Number 3 - Unit Test Case 3

Objective	To analyze exit code of the previous command and branch output inside a Bash script
Expected Result	The script must find the hidden executable file, storing the recent pipeline exit status as 0, and displaying appropriate output
Actual Result	The script displayed the expected result
Analysis	It can be further used in the script to find the previous exit status and display appropriate output
Conclusion	Successful Results

```

tree@house:~$ ls -al /tmp/
total 0
drwxrwxrwt 1 root root 4096 Apr 23 11:49 .
drwxr-xr-x 1 root root 4096 Feb  4 20:31 ..
-rwxrwxrwx 1 tree tree 177 Apr 23 11:49 .list
tree@house:~$ ./findHiddenFile

!! Checking for hidden executables at /tmp/ !!
/tmp/.list

[ Found! ]
tree@house:~$

```

Figure 19 Output according to the previous exit status code.

Table Number 4 - Unit Test Case 4

Objective	To invoke system commands via <i>os</i> module in Python 3
Expected Result	The output shall be dependent on the user id which must match the /etc/passwd file
Actual Result	The results were dependent on the user id, matching the /etc/passwd file
Analysis	It can be used to execute the getDE script to extract accurate information from the user's environment variables
Conclusion	Successful Results

The figure shows three terminal windows side-by-side. The left window shows a user 'leaf' running a Python script that prints the user ID '1000'. The middle window shows a user 'root' running the same script, which prints '0'. The right window shows a user 'leaf' running a command to list system services from /etc/passwd, showing a list of services and their IDs, with 'leaf' having ID 1000.

```

leaf@tree:~$ python3
Python 3.5.3 (default, Sep 27 2016, 13:06:42) on linux64
Type "help", "copyright", "credits() or license()" for more
>>> import os
>>> os.getuid()
1000
>>>

root@tree:~# python3
Python 3.5.3 (default, Sep 27 2016, 13:06:42) on linux64
Type "help", "copyright", "credits() or license()" for more
>>> import os
>>> os.getuid()
0
>>>

leaf@tree:~$ cat /etc/passwd | awk -F: '{print $1,$3}' | grep root
root 0
leaf@tree:~$ cat /etc/passwd | awk -F: '{print $1,$3}' | grep leaf
leaf 1000
leaf@tree:~$

```

Figure 20 User dependent output from the *os* module.

Table Number 5 - Unit Test Case 5

Objective	To test arguments via <i>argparse</i> module in Python 3
Expected Result	It must branch the script according to the argument passed in either of the two ways
Actual Result	The script successfully handled the given argument
Analysis	It can be used as a choice on what scan to perform
Conclusion	Successful Results

```

leaf@tree: ~
File Edit View Search Terminal Help
    print ("-b | --b triggered!")

elif args.advance:
    print ("-a | --a triggered!")

elif args.suspicious:
    print ("-s | --s triggered!")

elif args.full:
    print ("-f | --f triggered!")

elif args.denv:
    print ("-d | --d triggered!")

else:
    parser.print_help()
    print ("-h | --h triggered!")
leaf@tree:~$ ./arguments -b
Current Working Directory: /home/leaf
-b | --b triggered!
leaf@tree:~$ ./arguments --suspicious
Current Working Directory: /home/leaf
-s | --s triggered!
leaf@tree:~$

```

Figure 21 Script responding according to the arguments.

4.2 System Testing

As the script is built by the author from scratch and is fully intended to be under the paradigm of FOSS, performing a manual test by creating relevant test cases are done.

4.2.1 System Test Plan

CASES	AGENDA
1	To test whether the system is executed with the root user
2	To test the main executable script
3	To perform full scan script
4	To perform integrity scan script
5	To perform rootkit scan script
6	To perform rootkit scan in advanced mode script
7	To check suspicious files and configurations script
8	To get the installed desktop environment script with normal privileges
9	To check the any of the rootkit scan log
10	To check any of the rootkit scan logs

4.2.2 Test Results and Logs

Table Number 6 – System Test Case 1

Objective	To test whether the system is executed with the root user
Expected Result	Executing any of the scans (except the <i>getDE</i> script) shall not be successful
Actual Result	The script did not execute and showed an accurate error(s)
Analysis	The script executes successfully when executed by the root user because of the criteria scripted in the file
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
performs integrity scan.
leaf@tree:~/Documents/rtktchk$ ./rtktchk -f
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.
tee: logs/full.log: Permission denied

!! Checking the existence of the file(s)... !!

{ Script for integrity scan found! }

!! Performing integrity scan... !!

{ Please run the program as a root user. }
{ NOTE: Not even with sudo privileges. PID must be 0. }

{ Script for rootkit(s) scan found! }
!! Performing rootkit(s) scan... !!

{ Please run the program as a root user. }
{ NOTE: Not even with sudo privileges. PID must be 0. }

{ Script for advance rootkit(s) scan found! }
!! Performing advance rootkit(s) scan... !!

{ Please run the program as a root user. }
{ NOTE: Not even with sudo privileges. PID must be 0. }

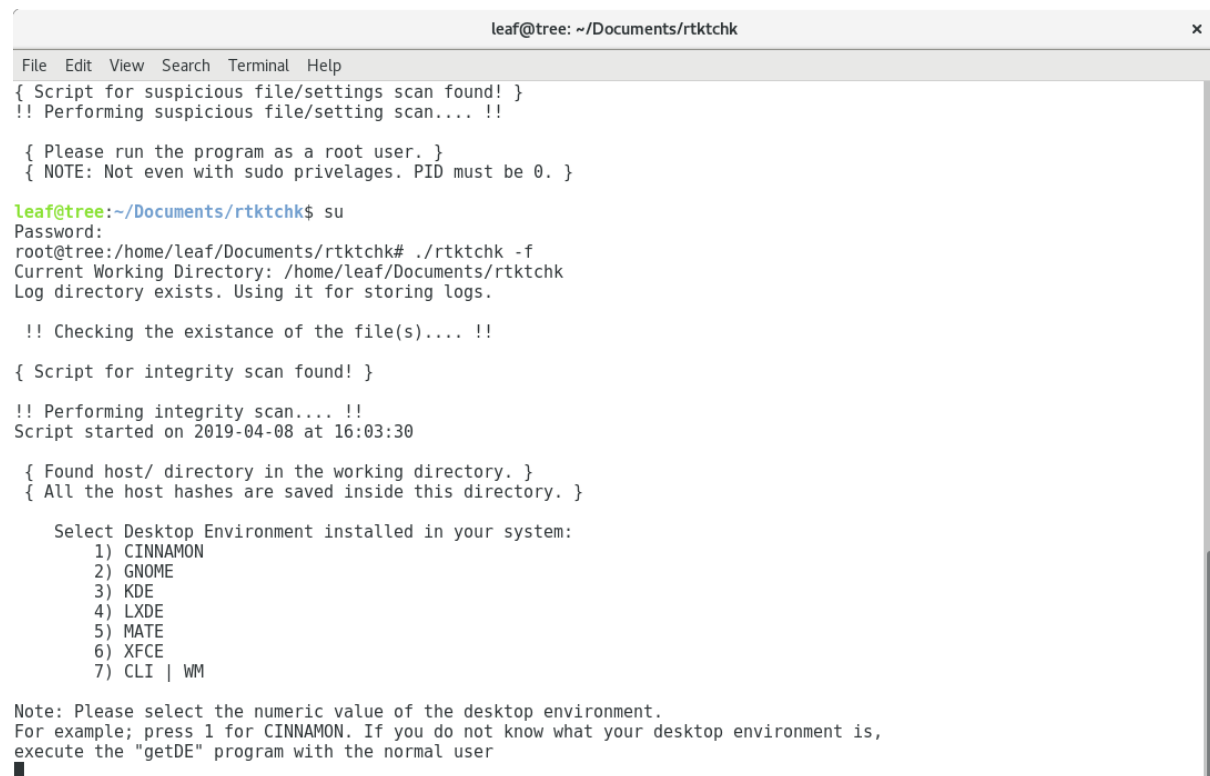
{ Script for suspicious file/settings scan found! }
!! Performing suspicious file/setting scan... !!

{ Please run the program as a root user. }
{ NOTE: Not even with sudo privileges. PID must be 0. }
leaf@tree:~/Documents/rtktchk$

```

Figure 22 Error when executing the script by the user.

Application-level Rootkit Detection Program for Linux



```
leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
{ Script for suspicious file/settings scan found! }
!! Performing suspicious file/setting scan.... !!

{ Please run the program as a root user. }
{ NOTE: Not even with sudo privileges. PID must be 0. }

leaf@tree:~/Documents/rtktchk$ su
Password:
root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -f
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.

!! Checking the existence of the file(s).... !!

{ Script for integrity scan found! }

!! Performing integrity scan.... !!
Script started on 2019-04-08 at 16:03:30

{ Found host/ directory in the working directory. }
{ All the host hashes are saved inside this directory. }

    Select Desktop Environment installed in your system:
    1) CINNAMON
    2) GNOME
    3) KDE
    4) LXDE
    5) MATE
    6) XFCE
    7) CLI | WM

Note: Please select the numeric value of the desktop environment.
For example; press 1 for CINNAMON. If you do not know what your desktop environment is,
execute the "getDE" program with the normal user
█
```

Figure 23 Successful when executed by the root user.

Table Number 7 – System Test Case 2

Objective	To test the main executable script
Expected Result	The script shall execute the default help option when executed without passing any of the arguments from a normal user as well as the root user
Actual Result	The script showed the default help option with the <i>argparse</i> module
Analysis	The script was intended to be executed by both root and normal user because of the <i>getDE</i> script as it is meant to be run with user privileges while others must need root privileges
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
leaf@tree:~/Documents/rtktchk$ ./rtktchk
Current Working Directory: /home/leaf/Documents/rtktchk
usage: rtktchk [-h] [-a] [-b] [-d] [-i] [-f] [-s]

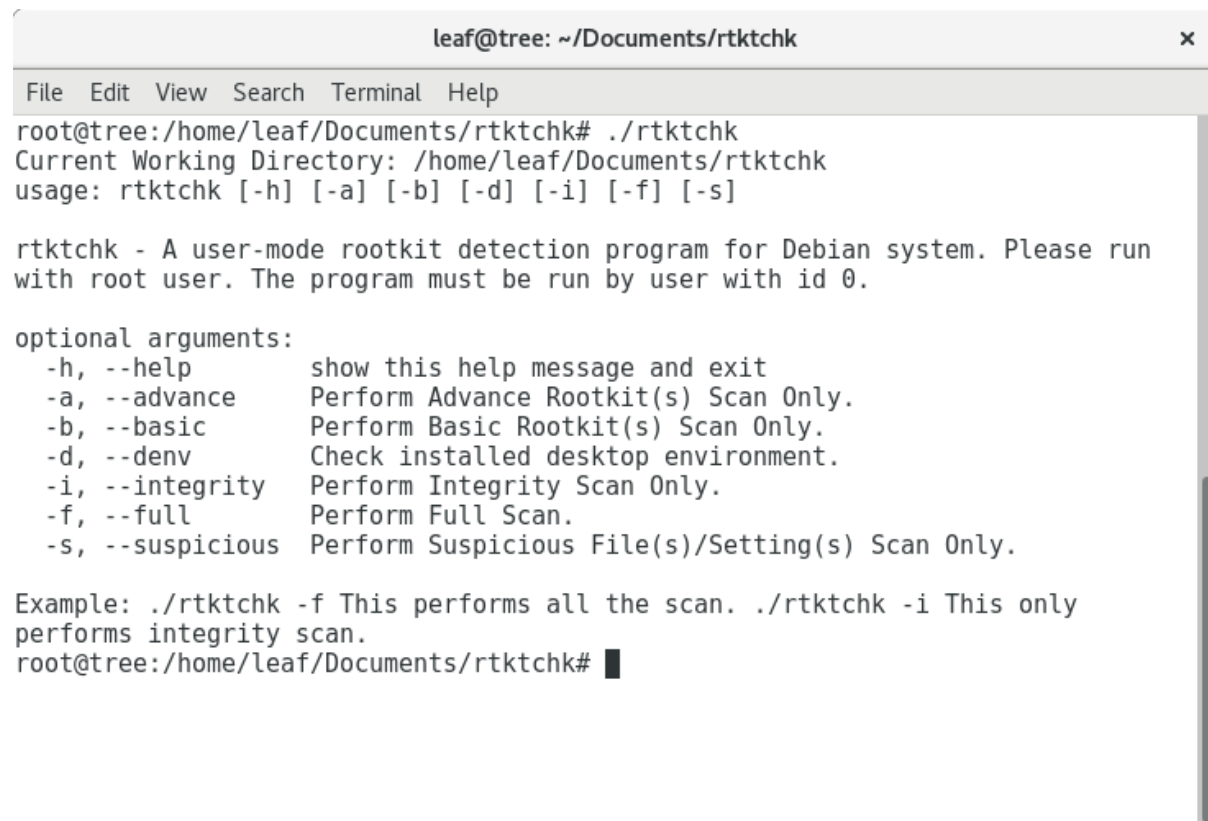
rtktchk - A user-mode rootkit detection program for Debian system. Please run
with root user. The program must be run by user with id 0.

optional arguments:
  -h, --help            show this help message and exit
  -a, --advance          Perform Advance Rootkit(s) Scan Only.
  -b, --basic            Perform Basic Rootkit(s) Scan Only.
  -d, --denv            Check installed desktop environment.
  -i, --integrity        Perform Integrity Scan Only.
  -f, --full            Perform Full Scan.
  -s, --suspicious       Perform Suspicious File(s)/Setting(s) Scan Only.

Example: ./rtktchk -f This performs all the scan. ./rtktchk -i This only
performs integrity scan.
leaf@tree:~/Documents/rtktchk$

```

Figure 24 Executing the script via a normal user.



The screenshot shows a terminal window titled 'leaf@tree: ~/Documents/rktchk'. The terminal content is as follows:

```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rktchk# ./rktchk
Current Working Directory: /home/leaf/Documents/rktchk
usage: rktchk [-h] [-a] [-b] [-d] [-i] [-f] [-s]

rktchk - A user-mode rootkit detection program for Debian system. Please run
with root user. The program must be run by user with id 0.

optional arguments:
  -h, --help            show this help message and exit
  -a, --advance          Perform Advance Rootkit(s) Scan Only.
  -b, --basic            Perform Basic Rootkit(s) Scan Only.
  -d, --denv             Check installed desktop environment.
  -i, --integrity        Perform Integrity Scan Only.
  -f, --full             Perform Full Scan.
  -s, --suspicious       Perform Suspicious File(s)/Setting(s) Scan Only.

Example: ./rktchk -f This performs all the scan. ./rktchk -i This only
performs integrity scan.
root@tree:/home/leaf/Documents/rktchk#
```

Figure 25 Executing the script via root user.

Table Number 8 – System Test Case 3

Objective	To perform full scan script
Expected Result	The script shall successfully execute performing all the scan scripts one by one and log the results
Actual Result	The script successfully executed performing all the scan scripts one by one and logged the results
Analysis	The script managed to fetch the required file with the correct permissions and executed them one by one
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help

root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -f
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.

!! Checking the existence of the file(s)... !!

{ Script for integrity scan found! }

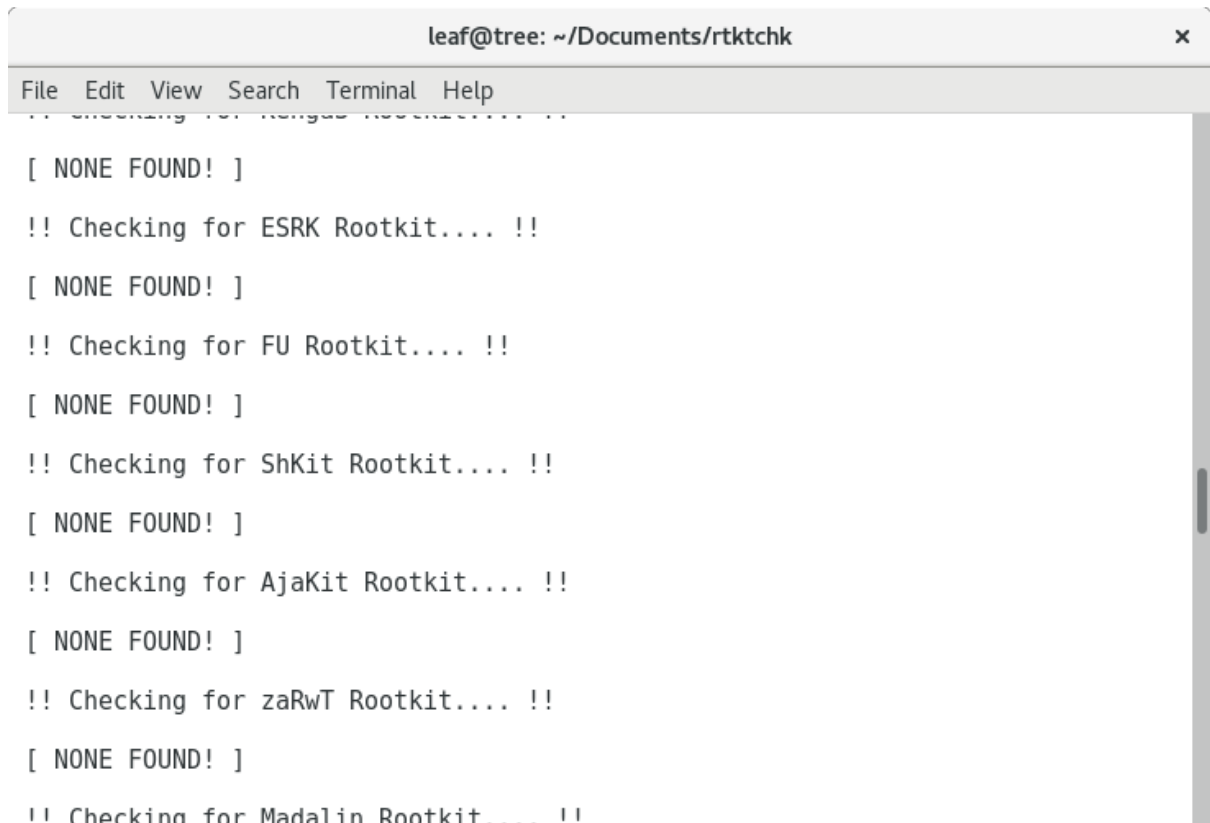
!! Performing integrity scan.... !!
Script started on 2019-04-08 at 16:10:47

{ Found host/ directory in the working directory. }
{ All the host hashes are saved inside this directory. }

Select Desktop Environment installed in your system:
1) CINNAMON
2) GNOME
3) KDE
4) LXDE
5) MATE
6) XFCE
7) CLI | WM

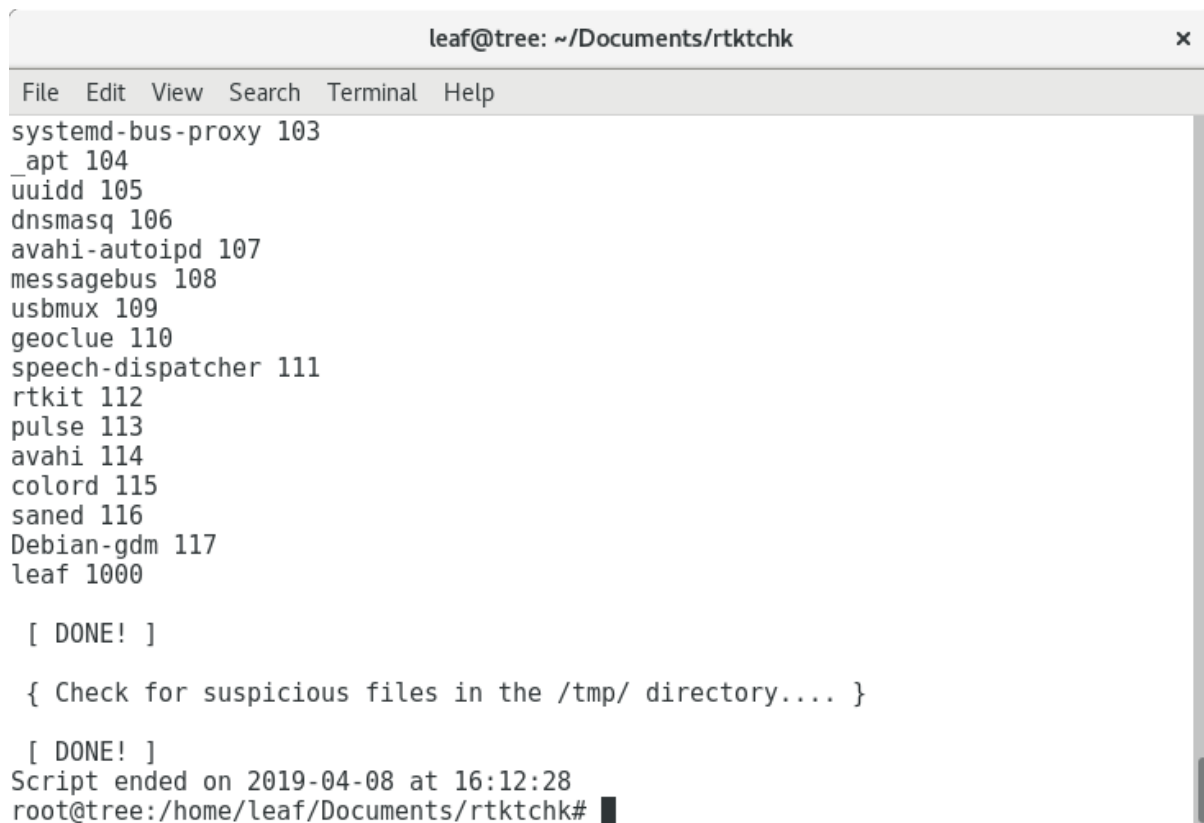
```

Figure 26 Executing the full scan script (PART 1).



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
!! Checking for Rengas Rootkit.... !!
[ NONE FOUND! ]
!! Checking for ESRK Rootkit.... !!
[ NONE FOUND! ]
!! Checking for FU Rootkit.... !!
[ NONE FOUND! ]
!! Checking for ShKit Rootkit.... !!
[ NONE FOUND! ]
!! Checking for AjaKit Rootkit.... !!
[ NONE FOUND! ]
!! Checking for zaRwT Rootkit.... !!
[ NONE FOUND! ]
!! Checking for Madalin Rootkit.... !!
```

Figure 27 Executing the full scan script (PART 2).

A terminal window titled 'leaf@tree: ~/Documents/rktchk' with a standard menu bar (File, Edit, View, Search, Terminal, Help). The terminal displays the output of a script, listing system services and their PIDs, followed by completion messages and a timestamp.

```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
systemd-bus-proxy 103
_apd 104
_uidd 105
dnsmasq 106
avahi-autoipd 107
messagebus 108
usbmux 109
geoclue 110
speech-dispatcher 111
rktkit 112
pulse 113
avahi 114
colord 115
saned 116
Debian-gdm 117
leaf 1000

[ DONE! ]

{ Check for suspicious files in the /tmp/ directory.... }

[ DONE! ]
Script ended on 2019-04-08 at 16:12:28
root@tree:/home/leaf/Documents/rktchk#
```

Figure 28 Executing the full scan script (PART 3).

Table Number 9 – System Test Case 4

Objective	To perform integrity scan script
Expected Result	The script shall execute the integrity scan script, compare the base hash with generated host hash, and finally log the output inside the log directory
Actual Result	The script successfully executed the integrity scan script, compared the base hash with generated host hash, and finally logged the output inside the log directory
Analysis	This script was possible by extracting the <i>sha256</i> hash from the host OS and compare them with the base <i>sha256</i> hash inside the base directory
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -i
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.
Script started on 2019-04-08 at 16:16:21

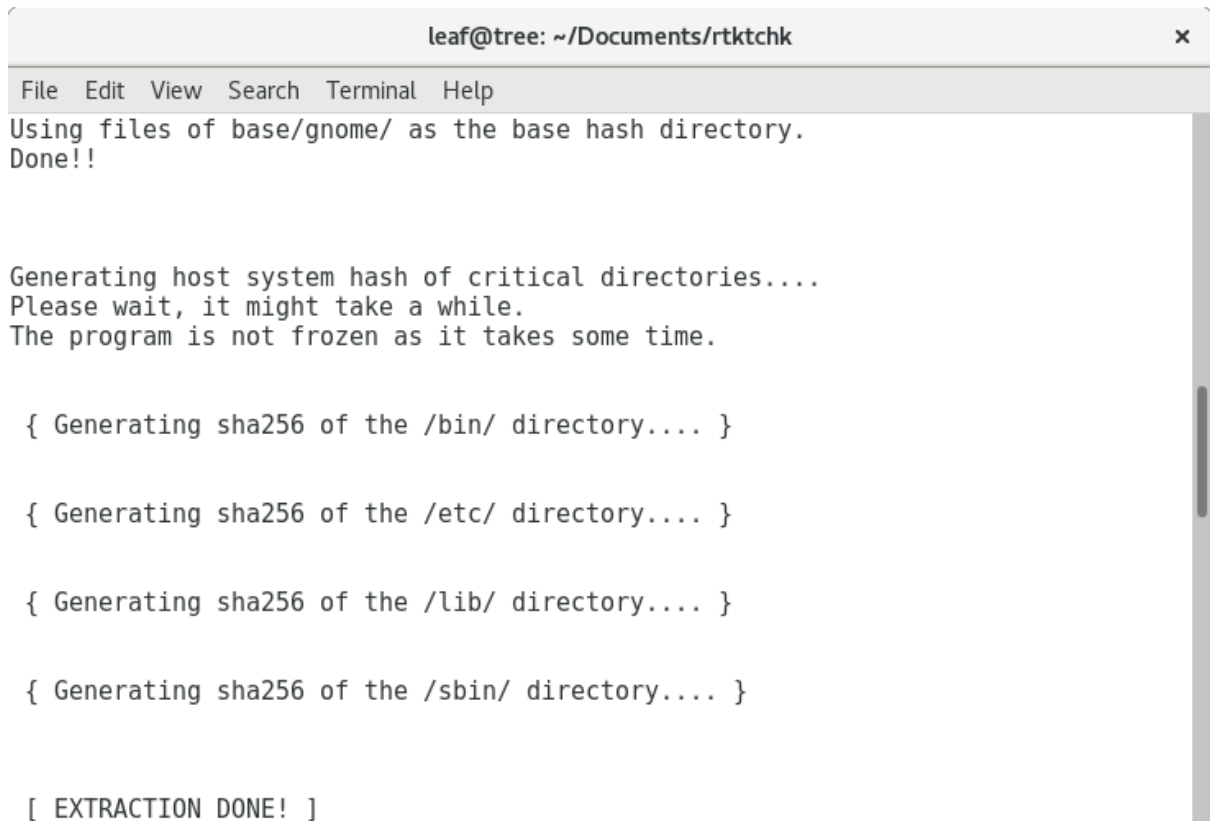
{ Found host/ directory in the working directory. }
{ All the host hashes are saved inside this directory. }

Select Desktop Environment installed in your system:
1) CINNAMON
2) GNOME
3) KDE
4) LXDE
5) MATE
6) XFCE
7) CLI | WM

Note: Please select the numeric value of the desktop environment.
For example; press 1 for CINNAMON. If you do not know what your desktop environm
ent is,
execute the "getDE" program with the normal user

```

Figure 29 Executing the integrity scan script (PART 1).



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
Using files of base/gnome/ as the base hash directory.
Done!!

Generating host system hash of critical directories....
Please wait, it might take a while.
The program is not frozen as it takes some time.

{ Generating sha256 of the /bin/ directory.... }

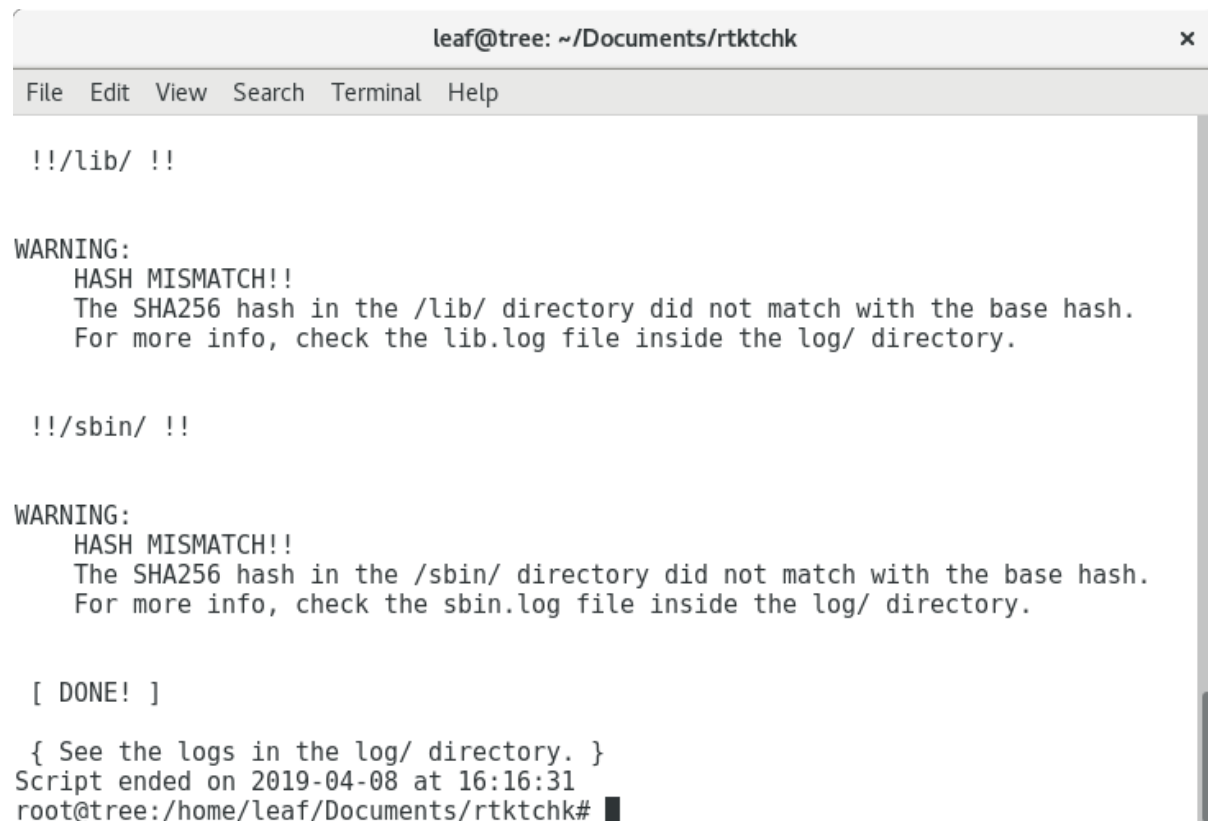
{ Generating sha256 of the /etc/ directory.... }

{ Generating sha256 of the /lib/ directory.... }

{ Generating sha256 of the /sbin/ directory.... }

[ EXTRACTION DONE! ]
```

Figure 30 Executing the integrity scan script (PART 2).



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help

!!/lib/ !!

WARNING:
HASH MISMATCH!!
The SHA256 hash in the /lib/ directory did not match with the base hash.
For more info, check the lib.log file inside the log/ directory.

!!/sbin/ !!

WARNING:
HASH MISMATCH!!
The SHA256 hash in the /sbin/ directory did not match with the base hash.
For more info, check the sbin.log file inside the log/ directory.

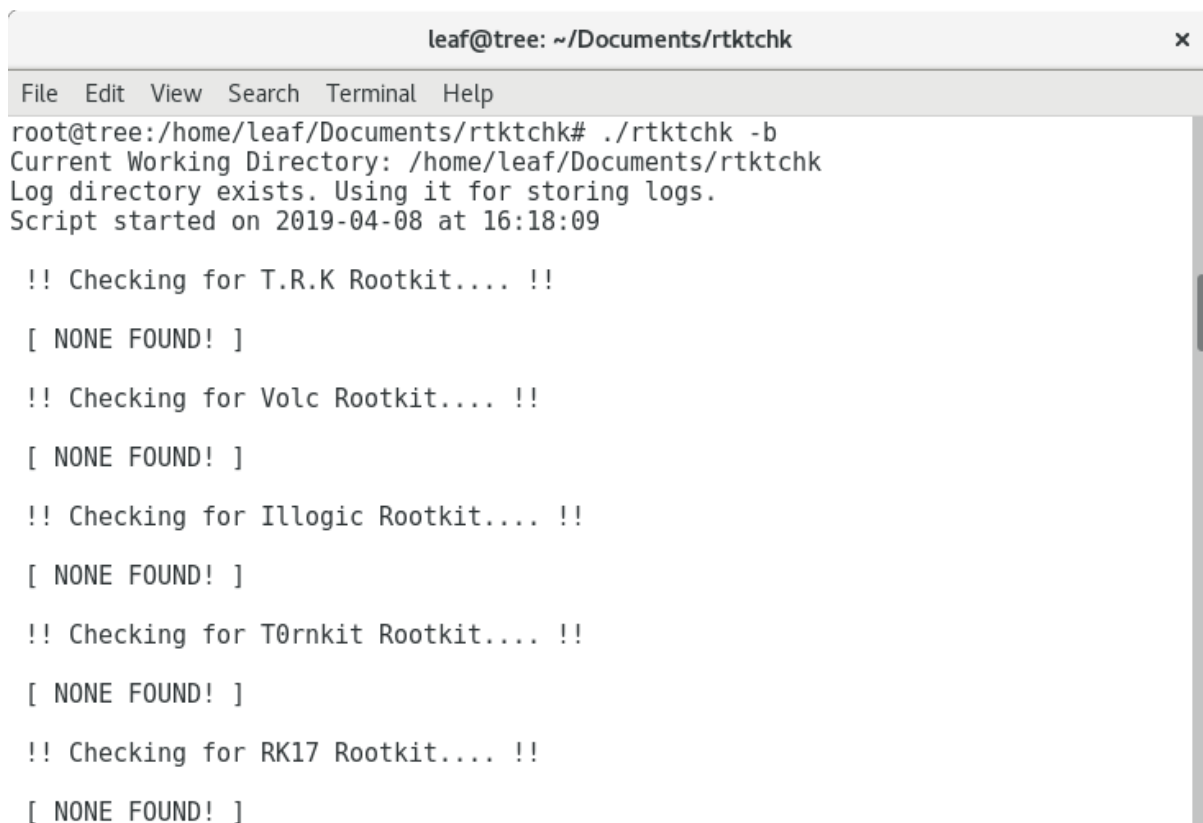
[ DONE! ]

{ See the logs in the log/ directory. }
Script ended on 2019-04-08 at 16:16:31
root@tree:/home/leaf/Documents/rktchk#
```

Figure 31 Executing the integrity scan script (PART 3).

Table Number 10 – System Test Case 5

Objective	To perform rootkit scan script
Expected Result	The script shall check for all the scripted user-mode rootkits inside the defined directories and log the output inside the log directory
Actual Result	The script successfully checked for all the scripted user-mode rootkits inside the defined directories and logged the output inside the log directory
Analysis	The execution time is comparatively faster than any other rootkit because it checks for the traces rather than comparing the malicious hash with the intended file(s) in the respective directories
Conclusion	Successful Results



```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -b
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.
Script started on 2019-04-08 at 16:18:09

!! Checking for T.R.K Rootkit.... !!
[ NONE FOUND! ]

!! Checking for Volc Rootkit.... !!
[ NONE FOUND! ]

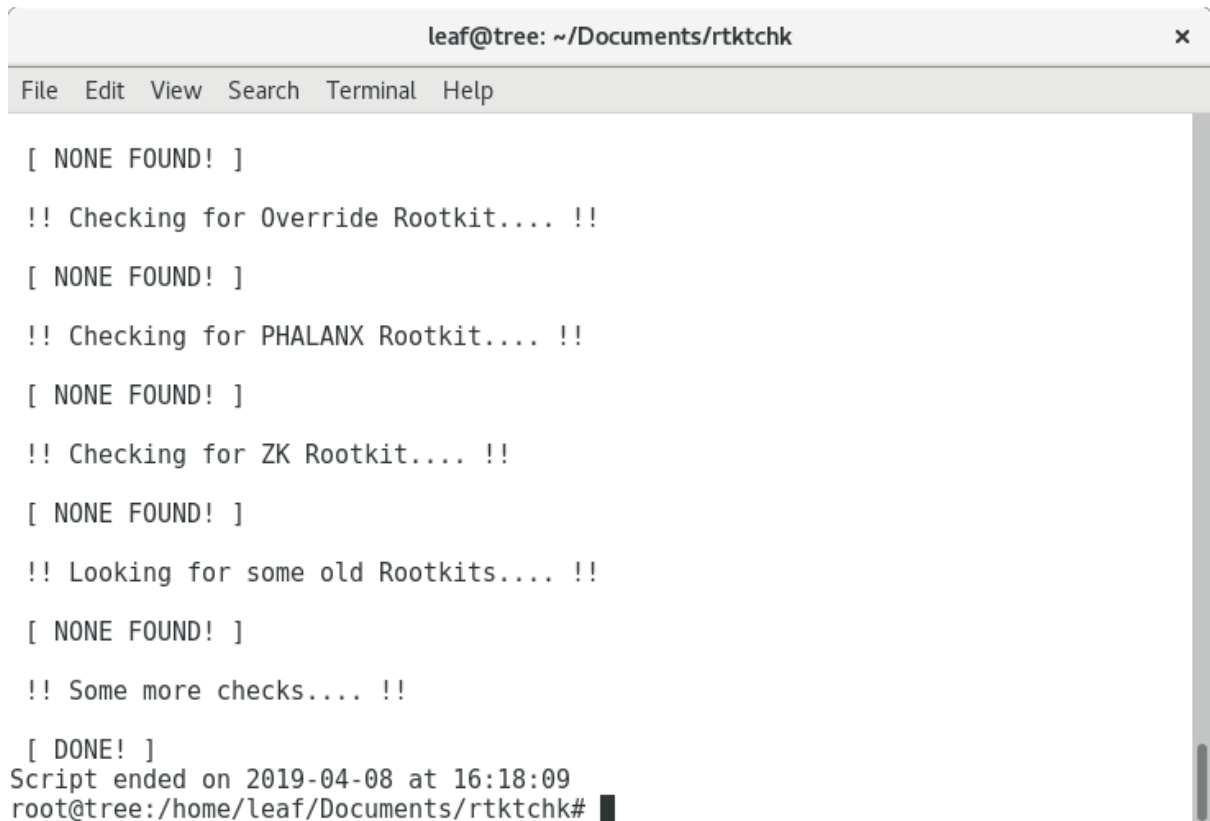
!! Checking for Illogic Rootkit.... !!
[ NONE FOUND! ]

!! Checking for T0rnkit Rootkit.... !!
[ NONE FOUND! ]

!! Checking for RK17 Rootkit.... !!
[ NONE FOUND! ]

```

Figure 32 Executing the basic rootkit scan script (PART I).



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help

[ NONE FOUND! ]
!! Checking for Override Rootkit.... !!
[ NONE FOUND! ]
!! Checking for PHALANX Rootkit.... !!
[ NONE FOUND! ]
!! Checking for ZK Rootkit.... !!
[ NONE FOUND! ]
!! Looking for some old Rootkits.... !!
[ NONE FOUND! ]
!! Some more checks.... !!

[ DONE! ]
Script ended on 2019-04-08 at 16:18:09
root@tree:/home/leaf/Documents/rktchk#
```

Figure 33 Executing the basic rootkit scan script (PART 2).

Table Number 11 – System Test Case 6

Objective	To perform rootkit scan in advanced mode script
Expected Result	The script shall check for all the scripted user-mode rootkits inside the root file system and log the output inside the log directory
Actual Result	The script successfully checked for all the scripted user-mode rootkits inside the root file system and logged the output inside the log directory
Analysis	The execution time is comparatively faster than any other rootkit because it checks for the traces rather than comparing the malicious hash with every file(s) in the root file system
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -a
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.
Script started on 2019-04-08 at 16:19:23

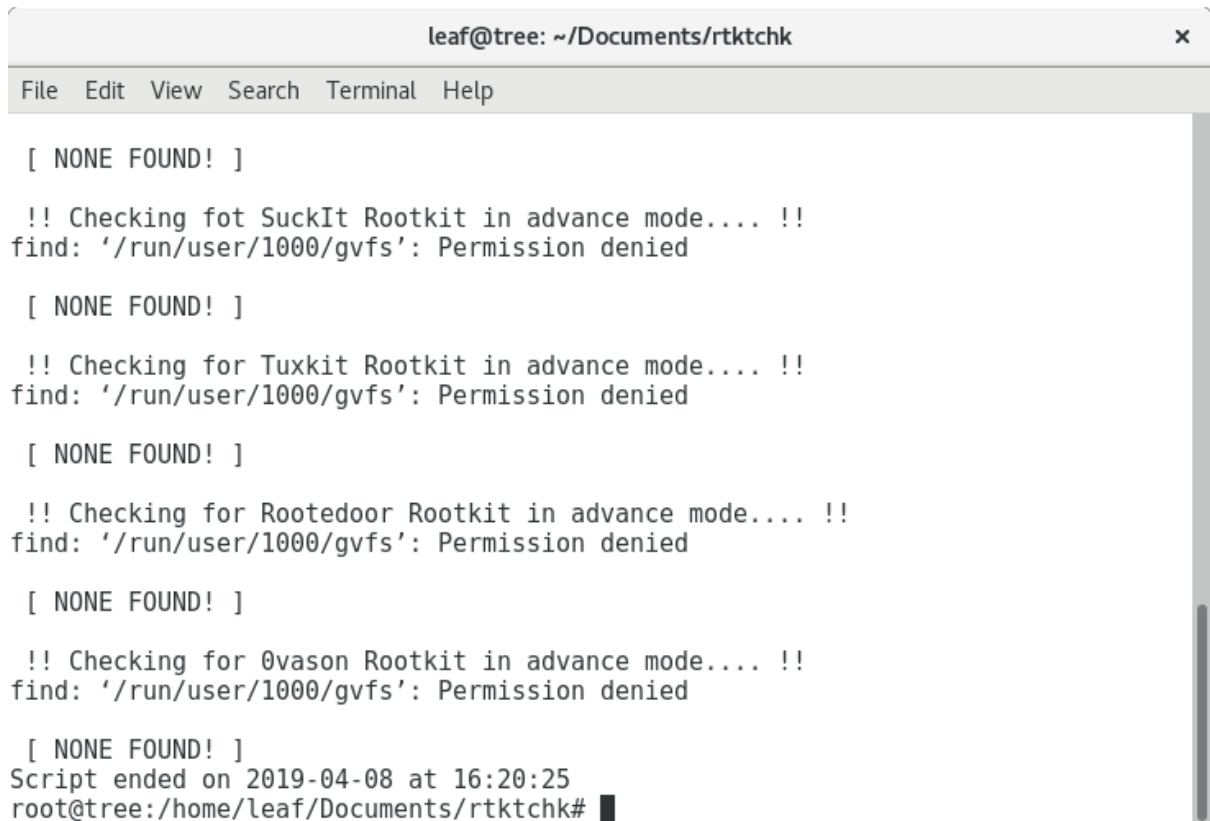
!! Checking for Illogic Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

[ NONE FOUND! ]

!! Checking for T0rnkit Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

```

Figure 34 Executing the advanced rootkit scan script (PART 1).



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help

[ NONE FOUND! ]

!! Checking fot SuckIt Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

[ NONE FOUND! ]

!! Checking for Tuxkit Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

[ NONE FOUND! ]

!! Checking for Rootedoor Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

[ NONE FOUND! ]

!! Checking for 0vason Rootkit in advance mode.... !!
find: '/run/user/1000/gvfs': Permission denied

[ NONE FOUND! ]
Script ended on 2019-04-08 at 16:20:25
root@tree:/home/leaf/Documents/rktchk#
```

Figure 35 Executing the advanced rootkit scan script (PART 2).

Table Number 12 – System Test Case 7

Objective	To check suspicious files and configurations script
Expected Result	The script shall search for suspicious files, configurations and display them and log the output inside the log directory
Actual Result	The script successfully searched for suspicious files, configurations and displayed them and logged the output inside the log directory
Analysis	Excluding the suspicious file(s), the configurations were manually deployed considering the survey results that help in detecting the newer and unknown version of user-mode rootkits
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rtktchk# ./rtktchk -s
Current Working Directory: /home/leaf/Documents/rtktchk
Log directory exists. Using it for storing logs.
Script started on 2019-04-08 at 16:21:06

!! Checking for suspicious files.... !!

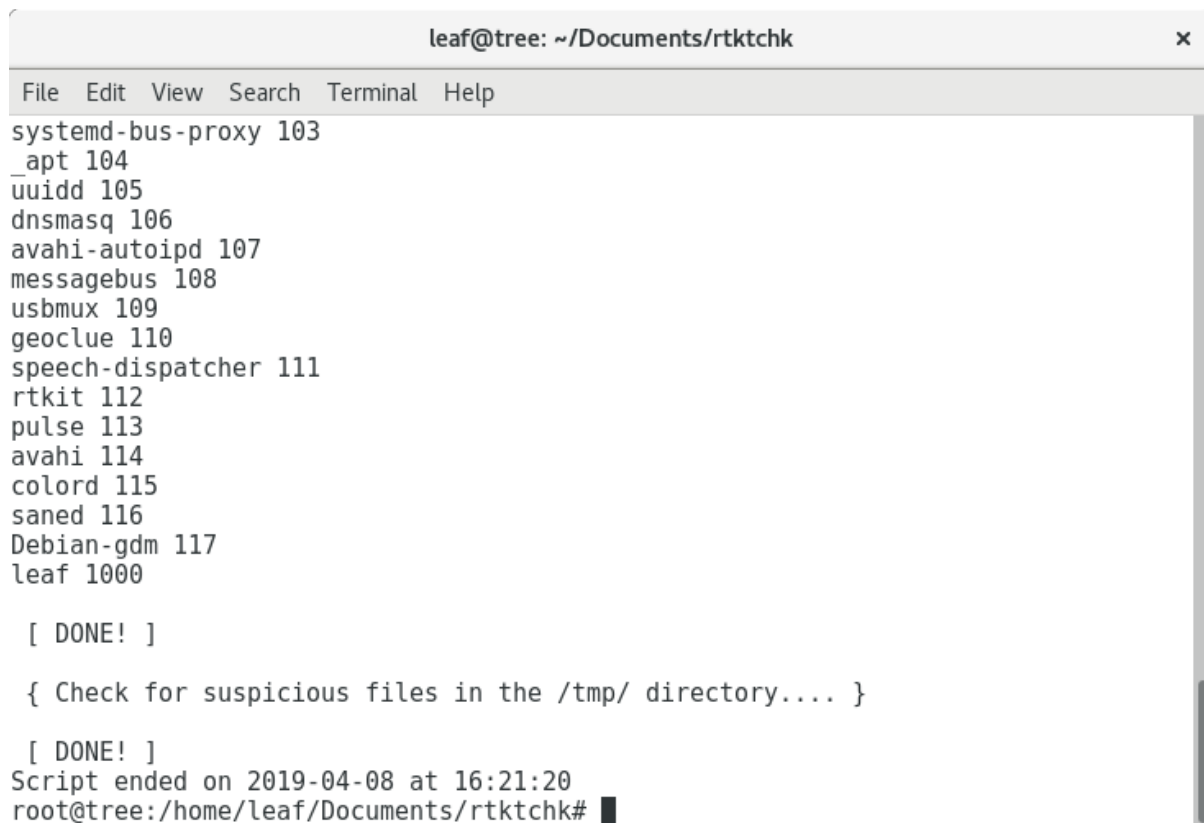
[ NO SUCH SUSPICIOUS FILE(S) DETECTED! ]

!! Checking the root file system (/) for other nasty rootkit files.... !!
!! This may take a while! Please be patience! !!

find: '/run/user/1000/gvfs': Permission denied

```

Figure 36 Executing the suspicious scan script (PART I).

A terminal window titled 'leaf@tree: ~/Documents/rktchk' with a standard menu bar (File, Edit, View, Search, Terminal, Help). The terminal output shows a list of system services and their PIDs, followed by a completion message and a check for suspicious files in the /tmp/ directory. The script ends with a timestamp and the user's prompt.

```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
systemd-bus-proxy 103
_apl 104
_uidd 105
dnsmasq 106
avahi-autoipd 107
messagebus 108
usbmux 109
geoclue 110
speech-dispatcher 111
rktkit 112
pulse 113
avahi 114
colord 115
saned 116
Debian-gdm 117
leaf 1000

[ DONE! ]

{ Check for suspicious files in the /tmp/ directory.... }

[ DONE! ]
Script ended on 2019-04-08 at 16:21:20
root@tree:/home/leaf/Documents/rktchk#
```

Figure 37 Executing the suspicious scan script (PART 2).

Table Number 13 – System Test Case 8

Objective	To get the installed desktop environment script with normal privileges
Expected Result	The script shall extract the installed desktop environment from the host system
Actual Result	The script successfully extracted the installed desktop environment from the host system
Analysis	The extraction was successful analyzing the environmental path which only had to be executed by the normal user as the root user is not intended to be logged in the system for several security purposes
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
base getDE integrityScan rootkitSignatureScan suspicious
leaf@tree:~/Documents/rtktchk$ ./rtktchk
Current Working Directory: /home/leaf/Documents/rtktchk
usage: rtkchk [-h] [-a] [-b] [-d] [-i] [-f] [-s]

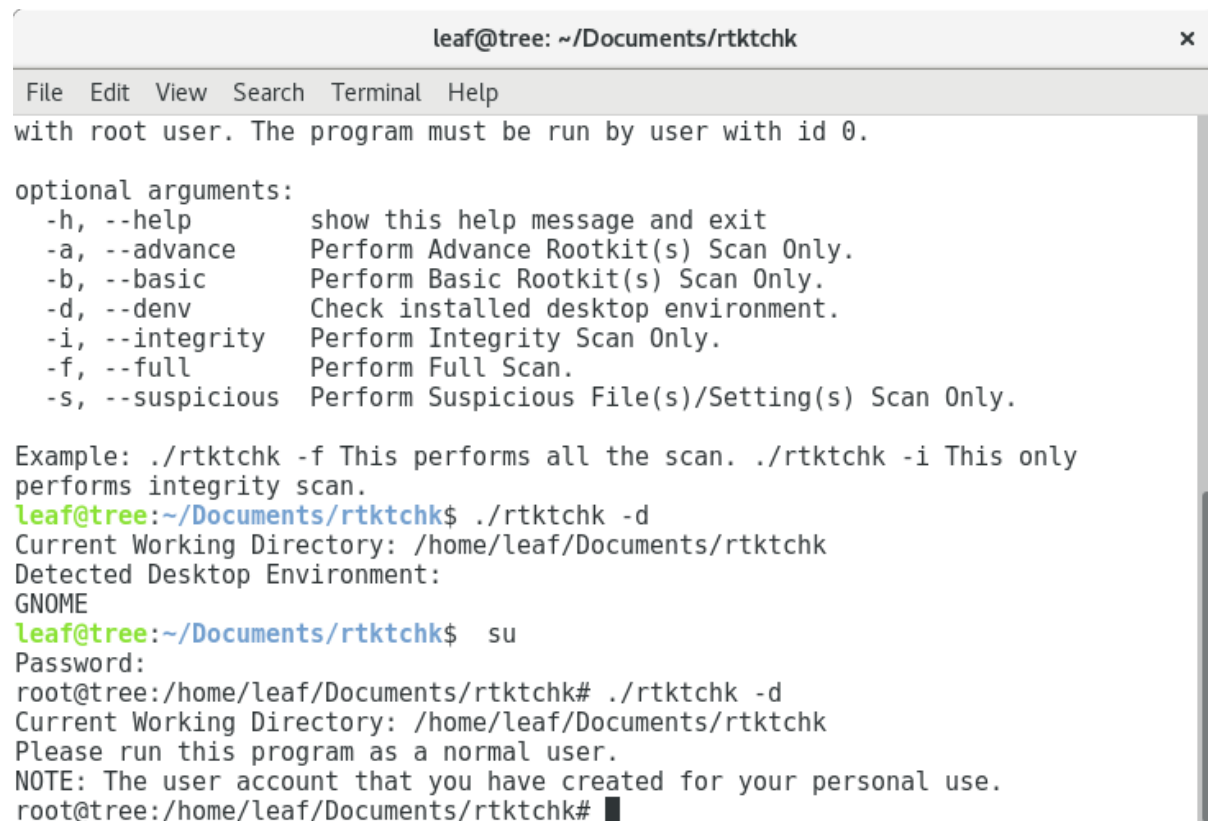
rtktchk - A user-mode rootkit detection program for Debian system. Please run
with root user. The program must be run by user with id 0.

optional arguments:
  -h, --help            show this help message and exit
  -a, --advance          Perform Advance Rootkit(s) Scan Only.
  -b, --basic            Perform Basic Rootkit(s) Scan Only.
  -d, --denv             Check installed desktop environment.
  -i, --integrity        Perform Integrity Scan Only.
  -f, --full             Perform Full Scan.
  -s, --suspicious       Perform Suspicious File(s)/Setting(s) Scan Only.

Example: ./rtktchk -f This performs all the scan. ./rtktchk -i This only
performs integrity scan.
leaf@tree:~/Documents/rtktchk$ ./rtktchk -d
Current Working Directory: /home/leaf/Documents/rtktchk
Detected Desktop Environment:
GNOME
leaf@tree:~/Documents/rtktchk$

```

Figure 38 Executing the getDE script by a normal user.



```
leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
with root user. The program must be run by user with id 0.

optional arguments:
  -h, --help            show this help message and exit
  -a, --advance          Perform Advance Rootkit(s) Scan Only.
  -b, --basic            Perform Basic Rootkit(s) Scan Only.
  -d, --denv             Check installed desktop environment.
  -i, --integrity        Perform Integrity Scan Only.
  -f, --full             Perform Full Scan.
  -s, --suspicious       Perform Suspicious File(s)/Setting(s) Scan Only.

Example: ./rktchk -f This performs all the scan. ./rktchk -i This only
performs integrity scan.
leaf@tree:~/Documents/rktchk$ ./rktchk -d
Current Working Directory: /home/leaf/Documents/rktchk
Detected Desktop Environment:
GNOME
leaf@tree:~/Documents/rktchk$ su
Password:
root@tree:/home/leaf/Documents/rktchk# ./rktchk -d
Current Working Directory: /home/leaf/Documents/rktchk
Please run this program as a normal user.
NOTE: The user account that you have created for your personal use.
root@tree:/home/leaf/Documents/rktchk#
```

Figure 39 Executing the getDE script by the root user.

Table Number 14 – System Test Case 9

Objective	To check the integrity scan log
Expected Result	The log file shall contain all the information that previously had been shown in the terminal while performing the integrity scan
Actual Result	The log file had all the information that previously had been shown in the terminal during the integrity scan
Analysis	The logs were possible with the help of output redirection techniques used in the script that stored the terminal output inside a file with <i>.log</i> extension
Conclusion	Successful Results

```

leaf@tree: ~/Documents/rktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rktchk# less logs/full.log
root@tree:/home/leaf/Documents/rktchk# less logs/etc.log
root@tree:/home/leaf/Documents/rktchk# cat logs/etc.log | head -n 10
/etc/adjtime 117d2db716ac59366d7481398873057ac0d04527fddf69ce | /etc/adjtime 33e
f60ac500c9e0d9608ecea6d0b75e928f39c89df308011
/etc/apparmor.d/usr.bin.thunderbird d1b66302d14f616c9b935e9d4 | /etc/apparmor.d/
usr.bin.thunderbird eca3f677e9d5d49eadb078165
/etc/apt/apt.conf.d/01autoremove-kernels ecb8bf19520e778e140 | /etc/apt/apt.conf
f.d/01autoremove-kernels 362faf565d2e624666b3
/etc/apt/sources.list 43f424ab6da859968875ffa3d5188b4d7bdc1d1 | /etc/apt/sources
.list bc27a234a1ab43aec9ff9feec5455349c806686
> /etc/apt/sources
.list.d/vscode.list dd581a11d40a169e315452436
> /etc/apt/trusted
.gpg.d/microsoft.gpg b4dcc2fb98c13b5fc96aee6c
> /etc/bash_comple
tion.d/git-prompt 4e3291fa48f43adebb522c3c91f
> /etc/cron.d/john
ele8e28b94d462b03ad8c45fa8033b342c7e3b30821d
> /etc/dpkg/shlibs
.default 52c64ebce4e9ac48a97d8c682510d0603e83
> /etc/dpkg/shlibs
.override f58f8dfe0a81ed9d3fb304158da178cac68
root@tree:/home/leaf/Documents/rktchk#

```

Figure 40 Displaying one of the logs (etc.log) among the integrity scan log files.

Table Number 15 – System Test Case 10

Objective	To check any of the rootkit scan logs
Expected Result	The log file shall contain all the information that previously had been shown in the terminal while performing the respective scan(s) along with the timestamp
Actual Result	The log file had all the information that previously had been shown in the terminal while performing the respective scan(s) along with the timestamp
Analysis	The logs were possible with the help of output redirection techniques used in the script that stored the terminal output inside a file with <i>.log</i> extension
Conclusion	Successful Results

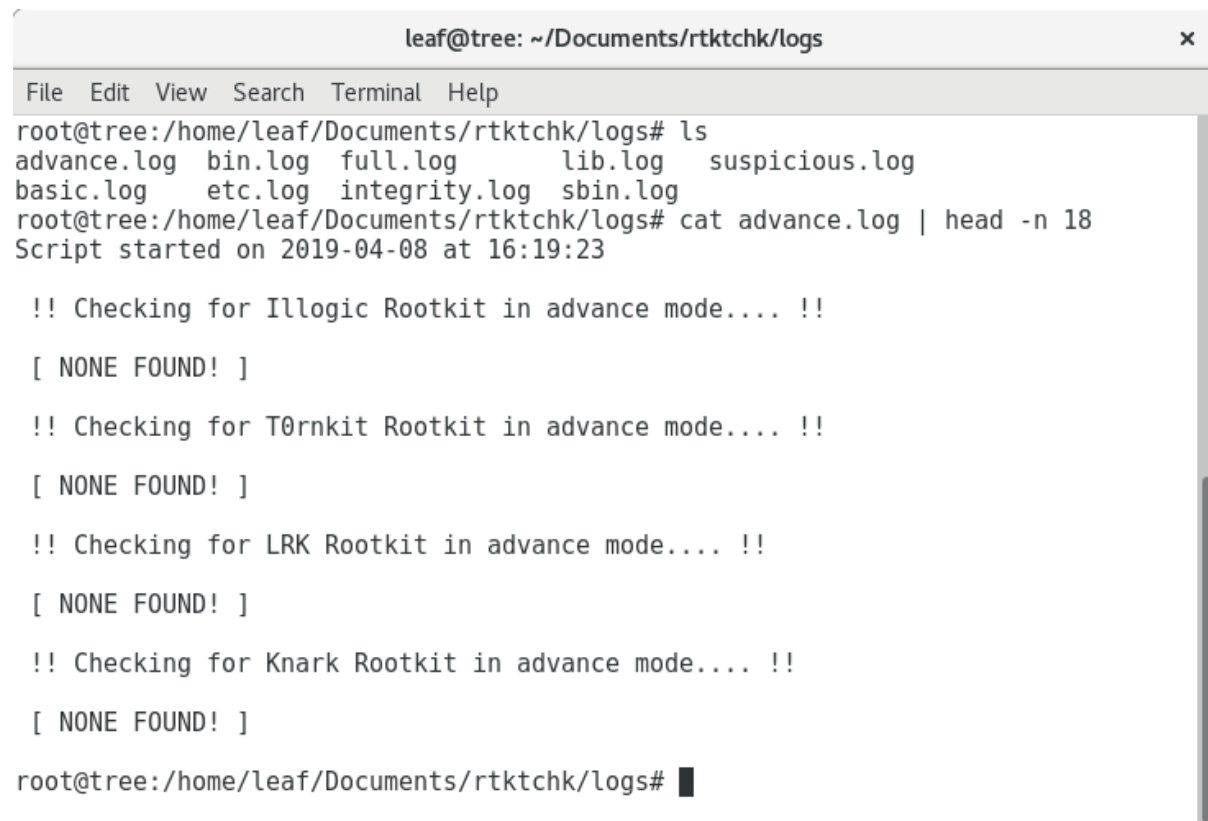
The screenshot shows a terminal window titled 'leaf@tree: ~/Documents/rtktchk'. The terminal displays the command 'ls -l logs/' and its output, which lists several log files with their permissions, sizes, and timestamps. The files are: advance.log (747 bytes, Apr 8 16:20), basic.log (2212 bytes, Apr 8 16:18), bin.log (2052 bytes, Apr 8 16:16), etc.log (4775 bytes, Apr 8 16:16), full.log (6665 bytes, Apr 8 16:12), integrity.log (1971 bytes, Apr 8 16:16), lib.log (231085 bytes, Apr 8 16:16), sbin.log (870 bytes, Apr 8 16:16), and suspicious.log (1339 bytes, Apr 8 16:21). The terminal prompt is 'root@tree:/home/leaf/Documents/rtktchk#'.

```

leaf@tree: ~/Documents/rtktchk
File Edit View Search Terminal Help
root@tree:/home/leaf/Documents/rtktchk# ls -l logs/
total 268
-rw-r--r-- 1 root root  747 Apr  8 16:20 advance.log
-rw-r--r-- 1 root root 2212 Apr  8 16:18 basic.log
-rw-r--r-- 1 root root 2052 Apr  8 16:16 bin.log
-rw-r--r-- 1 root root 4775 Apr  8 16:16 etc.log
-rw-r--r-- 1 root root 6665 Apr  8 16:12 full.log
-rw-r--r-- 1 root root 1971 Apr  8 16:16 integrity.log
-rw-r--r-- 1 root root 231085 Apr  8 16:16 lib.log
-rw-r--r-- 1 root root  870 Apr  8 16:16 sbin.log
-rw-r--r-- 1 root root 1339 Apr  8 16:21 suspicious.log
root@tree:/home/leaf/Documents/rtktchk#

```

Figure 41 All the logs inside the log directory.



A terminal window titled 'leaf@tree: ~/Documents/rktchk/logs' with a standard menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
root@tree:/home/leaf/Documents/rktchk/logs# ls
advance.log  bin.log  full.log      lib.log  suspicious.log
basic.log    etc.log  integrity.log sbin.log
root@tree:/home/leaf/Documents/rktchk/logs# cat advance.log | head -n 18
Script started on 2019-04-08 at 16:19:23

!! Checking for Illogic Rootkit in advance mode.... !!
[ NONE FOUND! ]

!! Checking for T0rnkit Rootkit in advance mode.... !!
[ NONE FOUND! ]

!! Checking for LRK Rootkit in advance mode.... !!
[ NONE FOUND! ]

!! Checking for Knark Rootkit in advance mode.... !!
[ NONE FOUND! ]

root@tree:/home/leaf/Documents/rktchk/logs#
```

Figure 42 Displaying one of the logs (advance.log) among the log files.

4.3 Comparative Testing

The author's version of the program is compared with the existing tools, i.e. *chkrootkit* and *rkhunter*. For accurate findings, the log files were analyzed and compared with the other one because, from a security point of view, log files are important (Ray & Nath, 2016). For this purpose, *full.log* from the author's script (*rtktchk*), *rkhunter.log* from the *rkhunter* script, and *chkrootkit.log* for the *chkrootkit* script are analyzed (See appendix for the complete log).

```
tree@house:~/logs$ ls -l
total 132
-rwxrwxrwx 1 tree tree 5733 Apr 9 14:55 chkrootkit.log
-rwxrwxrwx 1 tree tree 6828 Apr 9 14:55 full.log
-rwxrwxrwx 1 tree tree 116633 Apr 9 14:55 rkhunter.log
tree@house:~/logs$
```

Figure 43 Logs to be analyzed for comparison.

```
tree@house:~/logs$ cat chkrootkit.log | grep -i 'rootkit install'
Searching for t0rn's default files and dirs... Possible t0rn rootkit installed
Searching for Volc rootkit... Warning: Possible Volc rootkit installed
Searching for Gold2 rootkit... Warning: Possible Gold2 rootkit installed
tree@house:~/logs$ cat chkrootkit.log | grep -i 'warning'
Searching for Volc rootkit... Warning: Possible Volc rootkit installed
Searching for Gold2 rootkit... Warning: Possible Gold2 rootkit installed
tree@house:~/logs$ cat chkrootkit.log | grep -i 'promisc'
Checking 'sniffer'... enp0s3: PROMISC PF_PACKET(/sbin/dhclient)
tree@house:~/logs$
```

Figure 44 Key points from the *chkrootkit.log* file.

```
tree@house:~/logs$ cat rkhunter.log | grep -i 'warning'
[02:41:10] Info: No mail-on-warning address configured [ Warning ]
[02:41:15] Warning: Checking for prerequisites [ Warning ]
[02:41:15] Warning: WARNING! It is the users responsibility to ensure that when the '--propupd' option [ Warning ]
[02:41:16] /usr/sbin/adduser [ Warning ]
[02:41:16] Warning: The command '/usr/sbin/adduser' has been replaced by a script: /usr/sbin/adduser: Perl script text e
xecutable
[02:41:19] /usr/bin/ldd [ Warning ]
[02:41:19] Warning: The command '/usr/bin/ldd' has been replaced by a script: /usr/bin/ldd: Bourne-Again shell script, A
SCII text executable
[02:41:22] /usr/bin/lwp-request [ Warning ]
[02:41:22] Warning: The command '/usr/bin/lwp-request' has been replaced by a script: /usr/bin/lwp-request: Perl script
text executable
[02:41:25] /bin/egrep [ Warning ]
[02:41:25] Warning: The command '/bin/egrep' has been replaced by a script: /bin/egrep: POSIX shell script, ASCII text e
xecutable
[02:41:25] /bin/fgrep [ Warning ]
[02:41:25] Warning: The command '/bin/fgrep' has been replaced by a script: /bin/fgrep: POSIX shell script, ASCII text e
xecutable
[02:41:27] /bin/which [ Warning ]
[02:41:27] Warning: The command '/bin/which' has been replaced by a script: /bin/which: POSIX shell script, ASCII text e
xecutable
[02:42:00] Warning: T0rn Rootkit [ Warning ]
[02:42:04] Warning: Volc Rootkit [ Warning ]
[02:42:39] Checking for promiscuous interfaces [ Warning ]
[02:42:39] Warning: Possible promiscuous interfaces:
[02:42:41] Checking for passwd file changes [ Warning ]
[02:42:41] Warning: User 'vboxadd' has been added to the passwd file. [ Warning ]
[02:42:41] Checking for group file changes [ Warning ]
[02:42:41] Warning: Group 'vboxsf' has been added to the group file. [ Warning ]
[02:42:44] Checking for hidden files and directories [ Warning ]
[02:42:44] Warning: Hidden directory found: /etc/.java
tree@house:~/logs$
```

Figure 45 Key points from the rkhunter.log file.

```
tree@house:~/logs$ cat full.log | grep -i 'rootkit found'
[ Illogic Rootkit Found! ]
[ T0rnkit Rootkit Found! ]
[ LKR Rootkit Found! ]
[ Knark Rootkit Found! ]
[ Bobkit Rootkit Found! ]
[ SuckIT Rootkit Found! ]
[ Tuxkit Rootkit Found! ]
[ Rootedoor Rootkit Found! ]
[ Ovason Rootkit Found! ]
tree@house:~/logs$ cat full.log | grep -i 'suspicious'
{ Script for suspicious file/settings scan found! }
!! Performing suspicious file/setting scan.... !!
!! Checking for suspicious files.... !!
[ SUSPICIOUS FILE DETECTED! ]
[ SUSPICIOUS FILE DETECTED. ]
{ Check for suspicious files in the /tmp/ directory.... }
tree@house:~/logs$ cat full.log | grep -i 'promisc'
{ Checking ethernet for promiscuous mode.... }
2: enp0s3: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
[ Promiscuous Mode Enabled! ]
tree@house:~/logs$
```

Figure 46 Key points from the full.log file from the author's script.

After analyzing all the log files, being one different from another, the *chkrootkit* script displayed the lease information, followed by the *rktchk*, and *rkhunter* with broad information. As the comparison scenario was designed by the author, only *rktchk* script found user-mode rootkits and their suspicious file(s) precisely. However, both *chkrootkit* and *rkhunter* are

overall scanners performing other types of scans. But, being able to script a similar system to the known good ones within a limited time frame is one of the positive signs showing the improvement in the project. A comparison table below shows why user-mode rootkits were focused on the project.

Table Number 16 - Critical Evaluation with Similar Systems

	<i>chkrootkit</i>	<i>rkhunter</i>	<i>rtktchk</i>
Is the program solely for user-mode rootkits?	No	No	Yes
Can a user perform a custom scan?	No	No	Yes
Can it be executed in a live environment?	Yes	No	Yes
Can the base hash for integrity scan be user defined?	Yes	No	Yes
Is it focused on the Debian system?	No	No	Yes
Does it check for suspicious system configurations rather than just checking for a promiscuous mode in the NIC?	No	No	Yes

4.4 Critical Evaluation

Within a limited circle, the project feedback form has been distributed with the help of *Google Forms* among the users who tested the project in their version of the Debian 9.8 system (Refer to appendix for full survey results). The major findings from the evaluation of this project have according to the survey were:

- ✓ 75.5% of the users rated the simplicity of the project as “4”, 24.5% of the users rated “3”, while 0% of the users rated “1”, “2”, and “5” respectively on a scale from 1 (complex) to 5 (simple)
- ✓ 60.4% of the users rated the overall project as “4”, 39.6% of the users rated “3”, whereas 0% of the users rated “1”, “2”, and “5” respectively on a scale from 1 (worse) to 5 (best)
- ✓ 56.6% of the users rated the potential in growth of the project as “4”, 43.4% of the users rated “3”, whereas 0% of the users rated “1”, “2”, and “5” respectively on a scale from 1 (least) to 5 (most)
- ✓ Among around 19% of the users who gave feedback/suggestions, 100% of the users reacted positively liking the features in the project

Additionally; a pre-survey conducted beforehand the initial development (during the feasibility study of the project); 70% of responders suggested the project to be targeted for educational use, 62.2% of responders suggested to be targeted for personal use, and 0% suggested being targeted for enterprise use.

Chapter 5: Conclusion

An application-level rootkit detection program for Linux system, specifically targeted for the Debian system as the initial release of the project, contributes in the open-source community as a security tool. Being this project focused for both personal and educational use, this FYP draws attention towards the detection of user-mode rootkits in the Linux systems. Because no such in-depth research had been going on during the development of the project. Besides, the agility of the scripts helps students, researchers, and users tweak according to their personal necessity being under the software license agreement. Also, the users find the project as one of the inspirations to work and contribute because of the potential of the project in the future minimizing the threats posed by user-mode, aka application-level rootkits.

5.1 Advantages

Being only focused on user-mode rootkits in the Debian system, this project is the lightest and efficient for detecting application-level rootkits. The signatures are scripted according to the results from the sandboxed environment. Being not focused on the file hash of the malicious contents, it is one of the most effective approaches as everything is a file in Linux (Singh & Singh, 2016) and even a slight change in a file changes the whole integrity of the file. Besides, being all the signature viewable in the scripts, it contributes to the educational field as well as getting people into the field of rootkits. Because of its flexibility, it can perform an integrity scan of the operating system with the help of custom binaries, locations, and all sort of wonderful tasks being under the license terms of the project.

5.2 Limitations

Since the project is done by the author from scratch, there might be various pieces of stuff that are missing. The project might be completely unaware of the fact that other mysterious pieces of information that should be analyzed besides the generic rootkit behavior. Secondly, the signatures are all based upon the results from a sandboxed environment, thus, might not be a hundred percent effective as there are some rootkits that disable themselves when a virtualized and or sandboxed environment is detected. And lastly, the signature scripts must be manually updated as it does not rely on the malicious file hash for rootkit detection, which might be a hectic job keeping an eye on every changelog in the source repository.

5.3 Future Work

Being under the paradigm of FOSS, all the works shall be completely based upon the feedback and survey done among the community. However, to note some of the author's critical evaluation about the future works, they shall be:

1. Support for other major Linux distribution based on Debian
2. Support for the future user-mode rootkits
3. Signature scans with hooks and system calls
4. Detection via background services and process
5. Support for suspicious files, socket status, system configurations, and rootkits that evolve in the future

Chapter 6: Legal, Ethical, and Social Issues

6.1 Legal Issues

Illustration from one of the major rootkit incident from the past, the security flaws inherited from the Sony BMG DRM rootkit in late 2005; allowing a malicious user to inject arbitrary codes in the system (Mulligan & Perzanowski, 2007), rootkit detection had been one of the controversial propaganda among the security researchers and the proprietary software distributors.

However, during the time of this project, rootkit detection has been one of the major areas to be aware in the digital world and is marked as a legal activity to perform, supporting personal rights of a citizen. Moreover, expertise in rootkits recommends end users to perform a full scan of their system on a regular basis to be in the safe zone. Because of the loss that a party may bare because of the ability to subvert the software intended to find, rootkit detection methods are legitimately measured as a legal and authorized gesture.

6.2 Ethical Issues

The user manner is the space in the memory where a user processes run (U, et al., 2016), a perfect place for an application-level rootkit, aka user-mode rootkits to be executed. As they are executed within user space, a user has full rights to know what is running in their system.

As rootkits are a malicious script that executes itself as a background service, some parties consider it as an unethical act for hunting all the scripts and services used by a script or a program within a user's personal space. However, it is an individual's decision whether to perform such activities to check for the integrity of their computer system. If the concept of

malware detection was considered as unethical behavior, there shall be no existence of companies like Avast, Clam Antivirus, Kaspersky, Symantec, and other serving anti-malware solutions to the end users. Because many users feel comfortable for running the programs and scripts that protect their privacy in the digital world, this project strictly believes that it is an ethical move to check for the integrity of a system. Moreover, in the end, all that matter is being safe from the cyberspace posing critical security threats.

6.3 Social Issues

As the project is host specific and doesn't require any sort of internet connection at the time of execution, no data is transmitted elsewhere to any of the third parties. Being under the paradigm of FOSS and its source code freely available on the internet, no harm is intended to any of the individual using this project. Since people are pretty much conscious about their privacy and security in the digital world, the project does nothing else than checking for user-mode rootkits for their signatures. Although 90% of the scripts require root permissions to execute, no such malicious scripts are hooked within the script itself that creates havoc among the users and or generates any information that a user may find offensive under any circumstances. Moreover, even the logs that the script generates does not reveal any of the critical information like user passwords, critical system information, and memory dumps. Hence, this project creates no such issues that lead to a violation in the community.

Chapter 7: References

- Alhassan, J. & Misra, S., 2016. Evaluating Capabilities of Rootkits Tools. *International Journal of Advanced Multidisciplinary Research and Studies*, 1(1), pp. 27-30.
- Alonso, A., 2009. *Rootkits & Honeypots*. s.l., s.n.
- Alsalam, J., Banerjee, S., Musick, G. & Saftoiu, R., 2005. *Computer Security and Rootkits*, s.l.: s.n.
- Bajaj, G., 2006. *Taxonomy of Rootkits*, Patiala: s.n.
- Beazley, D. & Jones, B. K., 2013. *Python Cookbook*. 3 ed. s.l.:O'Reilly Media.
- Beck, D. R. & Wang, Y.-M., 2011. *DETECTING USER-MODE ROOTKITS*. United States, Patent No. US 8,661,541 B2.
- Beegle, L. E., 2007. Rootkits and Their Effects on Information Security. *Information Systems Security*, 16(3), pp. 164-176.
- Bellasi, P., 2010. *The Unix and GNU/Linux command line*. s.l., Free Electrons.
- BerkeleyTech. L.J., 2006. Sony-BMG Copy Protection Rootkit. *Berkeley Technology Law Journal*, 21(1), pp. 551-555.
- Bhardwaj, S. & Pranjal, K., 2016. Detection Techniques of Rootkits. *Imperial Journal of Interdisciplinary Research*, 2(5), pp. 2454-1362.
- Blunden, B., 2011. The Final Chapter. In: B. Blunden & B. G. Labs, eds. *The Rootkit Arsenal*. s.l.:Jones & Bartlett Learning, pp. 2-21.
- Bowman, M., Brown, H. D. & Pitt, P., 2007. *An undergraduate rootkit research project: How available? How hard? How dangerous?*, s.l.: s.n.

Brumley, D., 2015. Invisible Intruders: Rootkits in Practice. *History*, 40(2), pp. 69-71.

Bunten, A., 2004. *UNIX and Linux based Rootkits Techniques and Countermeasures*, Hamburg: DFN-CERT Services GmbH.

Butler, J., 2004. *Black Hat Windows 2004 - DKOM (Direct Kernel Object Manipulation)*. s.l., HBGary.

Carnegie Mellon University, 2014. *Linux Antivirus*, s.l.: Software Engineering Institute .

Case, A., 2012. *Volatility Labs: MoVP 1.5 KBeast Rootkit, Detecting Hidden Modules, and sysfs*. [Online]

Available at: <https://volatility-labs.blogspot.com/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html>

[Accessed 19 November 2018].

CERN Computer Security Team, 2017. *CERN Computer Security Information*. [Online]

Available at: <https://security.web.cern.ch/security/venom.shtml>

[Accessed 27 December 2018].

Check Point Research, 2017. *MID-YEAR REPORT*, s.l.: Check Point.

Choi, J. et al., 2008. *Live Forensic Analysis of a Compromised Linux System Using LECT (Linux Evidence Collection Tool)*. Busan, s.n.

Chuvakin, A., 2003. *An Overview of Unix Rootkits*, Chantilly: iDEFENSE Inc..

CollabNet , 2018. *What Is Kanban? An Introduction to Kanban Methodology*. [Online]

Available at: <https://resources.collab.net/agile-101/what-is-kanban>

[Accessed 29 December 2018].

Davis, M. A., Bodmer, S. M. & LeMasters, A., 2009. *Malware and Rootkits: Malware & Rootkits Secrets & Solutions*. 1st ed. s.l.:McGraw-Hill Education.

Digité, 2018. *What is Kanban? Comprehensive Overview of the Kanban Method*. [Online]
Available at: <https://www.digite.com/kanban/what-is-kanban/>
[Accessed 28 December 2018].

Dillard, K., 2005. *What are user-mode vs. kernel-mode rootkits?*. [Online]
Available at: <https://searchenterprisedesktop.techtarget.com/news/1086469/What-are-user-mode-vs-kernel-mode-rootkits>
[Accessed 18 March 2019].

Dobre, C., 2013. *antivirus - How do antiviruses scan for thousands of malware signatures in a short time? - Information Security Stack Exchange*. [Online]
Available at: <https://security.stackexchange.com/questions/30362/how-do-antiviruses-scan-for-thousands-of-malware-signatures-in-a-short-time>
[Accessed 4 March 2019].

Donovan, F., 2016. *Top 5 Rootkit Threats and How to Root Them out*. [Online]
Available at: <https://www.esecurityplanet.com/network-security/top-5-rootkit-threats-and-how-to-root-them-out.html>
[Accessed 2 February 2019].

Dumont, R., M.Léveillé, M.-E. & Porcher, H., 2018. *The Dark Side of the ForSSH // A landscape of OpenSSH backdoors*, s.l.: ESET.

Dunn, J. E. & Magee, T., 2018. *Linux malware threats you should know about | Gallery | Computerworld UK*. [Online]
Available at: <https://www.computerworlduk.com/galleries/security/10-linux-malware-threats->

[bots-backdoors-trojans-malicious-apps-3634006/](#)

[Accessed 28 January 2019].

ElProCus, 2014. *Know all about Linux Operating System with Applications*. [Online]

Available at: <https://www.elprocus.com/linux-operating-system/>

[Accessed 19 October 2018].

EVOKNOW, Inc, 2005. DEALING WITH ROOTKIT ATTACKS ON LINUX. In:

DEALING WITH ROOTKIT ATTACKS ON LINUX. s.l.:The Evolving Knowledge Company, pp. 1-29.

Fehlmann, T. M., 2004. *Six Sigma for Software* , Zurich: s.n.

Fortinet, 2018. *Threat Landscape Report Q3 2018*, Sunnyvale: Fortinet.

F-Secure Labs, 2017. *Rootkit Description* | *F-Secure Labs*. [Online]

Available at: <https://www.f-secure.com/v-descs/rootkit.shtml>

[Accessed 19 January 2019].

Gray, M., 2019. *Shell Scripting for Reconnaissance and Incident Response*, s.l.: SANS.

Hannel, J., 2003. *Linux RootKits For Beginners – From Prevention to Removal*, s.l.: SANS Institute Reading Room.

Harley, D. & Lee, A., 2009. *The Root of All Evil? - Rootkits Revealed*, San Diego: ESET, LLC.

Harley, D. & Lee, A., 2009. *The Root of All Evil? - Rootkits Revealed*, s.l.: ESET.

Heasman, J., 2006. Rootkit threats. *Network Security*, 2006(1), pp. 1-20.

Hensing, R., 2004. *Rootkit Technologies*. s.l., s.n.

Hertzog, R. & Mas, R., 2015. *The Debian Administrator's Handbook*. 1 ed. s.l.:Freexian .

Hoffman, C., 2016. *The Linux Directory Structure, Explained*. [Online]

Available at: <https://www.howtogeek.com/117435/htg-explains-the-linux-directory-structure-explained/>

[Accessed 2 January 2019].

Hoglund, G. & Butler, J., 2005. *Rootkits: Subverting the Windows Kernel*. s.l.:Addison Wesley Professional.

Holt, T. J., 2005. *Hacks, Cracks, and Crime: An Examination of the Subculture and Social Organization of Computer Hackers*, s.l.: Dissertation.

Hosch, W. L., 2018. *Rootkit | malware | Britannica.com*. [Online]

Available at: <https://www.britannica.com/technology/rootkit>

[Accessed 15 January 2019].

Infosec, 2017. *Rootkits: User Mode*. [Online]

Available at: <https://resources.infosecinstitute.com/rootkits-user-mode-kernel-mode-part-1/>

[Accessed 19 March 2019].

IT PRO, 2018. *The cyber security threat in six charts | IT PRO*. [Online]

Available at: <https://www.itpro.co.uk/security/29224/the-cyber-security-threat-in-charts>

[Accessed 2 January 2019].

ITU, 2017. *Cybersecurity - Facts and Figures*. [Online]

Available at: <https://www.itu.int/en/ITU-D/Partners/Pages/Call4Partners/CYBLDCStats.aspx>

[Accessed 28 January 2019].

Kanban Tool, 2018. *Kanban Presentation - Kanban in four easy steps | Kanban Tool*.

[Online]

Available at: Kanban Tool

[Accessed 30 December 2018].

Kapoor, A. & Mathur, R., 2011. *PREDICTING THE FUTURE OF STEALTH ATTACKS*, Beaverton: McAfee Inc..

Kaspersky Lab, 2013. *What is a rootkit and how to remove it* | *Kaspersky Lab official blog*.
[Online]

Available at: <https://www.kaspersky.com/blog/rootkit/1508/>

[Accessed 21 March 2019].

Kassner, M., 2008. *News, Tips, and Advice for Technology Professionals - TechRepublic*.
[Online]

Available at: <https://www.techrepublic.com/blog/10-things/10-plus-things-you-should-know-about-rootkits/>

[Accessed 28 January 2019].

Kiger, D., 2017. *The advantages and disadvantages of Six Sigma Methodology - David Kiger's Blog*. [Online]

Available at: <https://davidkigerinfo.wordpress.com/2017/06/13/the-advantages-and-disadvantages-of-six-sigma-methodology/>

[Accessed 9 December 2018].

Kili, A., 2018. *5 Tools to Scan a Linux Server for Malware and Rootkits*. [Online]

Available at: <https://www.tecmint.com/scan-linux-for-malware-and-rootkits/>

[Accessed 17 March 2019].

Koch, M., 2015. *An Introduction to Linux-based malware*, s.l.: SANS Institute.

Komur, Y. S. & Ayfer, C. U., 2004. *Detecting and Removing Rootkits*. [Online]

Available at:

http://cayfer.bilkent.edu.tr/~cayfer/linux/Detecting_and_Removing_Rootkits.html

[Accessed 15 March 2019].

Kornbrust, A., 2005. *Database Rootkits*. Neunkirchen, Red-Database-Security GmbH.

Leibowitz, M., 2016. *Horse Pill A New Type Of Linux Rootkit*. Las Vegas, Blackhat USA 2016.

leontranter, 2016. *Alternatives to scrum - Extreme Uncertainty - practical agile*. [Online]

Available at: <https://www.extremeuncertainty.com/alternatives-to-scrum/>

[Accessed 21 December 2018].

Leontranter, 2016. *Alternatives to scrum - Extreme Uncertainty - practical agile*. [Online]

Available at: <https://www.extremeuncertainty.com/alternatives-to-scrum/>

[Accessed 6 January 2019].

Lessard, P., 2003. *Linux Process Containment – A practical look at chroot and User Mode Linux*, s.l.: SANS.

Lewinson, M., 2012. *How to Write a Feasibility Study Report (FSR)*. [Online]

Available at: <https://mymanagementguide.com/feasibility-study-reporting-steps-to-writing-a-feasibility-study-report-fsr/>

[Accessed 3 November 2018].

Leybourn, E., 2015. *stages of delivery - an example Kanban - The Agile Director*. [Online]

Available at: <https://theagiledirector.com/article/2015/04/21/7-stages-of-delivery-an-example-kanban/>

[Accessed 6 December 2018].

Liu, L. et al., 2012. Research and Design of Rootkit Detection Method. *Physics Procedia*, Volume 33, pp. 852-857.

Luz-Romero, P. A., 2003. *Secure OS Environments for Linux*, s.l.: SANS.

M.Léveillé, M.-E., 2014. *An In-depth Analysis of Linux/Ebury* | *WeLiveSecurity*. [Online]
Available at: [Marc-Etienne M.Léveillé](#)
[Accessed 30 December 2018].

Marlabs, 2018. *Threat Briefing – October 01*. [Online]
Available at: <https://www.marlabs.com/threat-briefing-october-01/>
[Accessed 30 December 2018].

Martin, C. D. & Martin, D. H., 1990. Professional Codes of Conduct and Computer Ethics Education. *Social Science Computer Review*, 8(1), pp. 96-108.

Mashevsky, Y., Sapronov, K. & Monastyrsky, A., 2005. *Rootkits and how to combat them* | *Securelist*. [Online]
Available at: <https://securelist.com/rootkits-and-how-to-combat-them/36055/>
[Accessed 4 January 2019].

Mathur, K. & Hiranwal, S., 2013. A Survey on Techniques in Detection and Analyzing Malware Executables. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4), pp. 422-428.

McAfee, 2006. Rootkits, Part 1 of 3: The Growing Threat. In: California: McAfee, pp. 1-8.

Miller, J. V., 2003. *SHV4 Rootkit Analysis*, s.l.: Symantec.

Miller, K., 2003. *Fight crime. Unravel incidents... one byte at a time.*, s.l.: SANS.

Mukund, 2018. *Why a Feasibility Study is Important in Project Management*. [Online]

Available at: <https://www.simplilearn.com/feasibility-study-article>

[Accessed 9 December 2018].

Mulligan, D. K. & Perzanowski, A. K., 2007. The Magnificence of the Disaster:

Reconstructing the Sony BMG Rootkit Incident. *Berkeley Technology Law Journal*, 22(3), pp. 1158-1218.

Muslihat, D., 2018. *7 Popular Project Management Methodologies And What They're Best Suited For* | Zenkit. [Online]

Available at: <https://zenkit.com/en/blog/7-popular-project-management-methodologies-and-what-theyre-best-suited-for/>

[Accessed 19 December 2018].

Naugolnyi, M., 2015. *CloudLinux OS Blog - iDRAC vulnerability - hacked servers*. [Online]

Available at: <https://cloudlinux.com/cloudlinux-os-blog/entry/idrac-vulnerability-hacked-servers>

[Accessed 3 December 2018].

Network Computing, 2006. *Rootkit Detection* | *IT Infrastructure Advice, Discussion, Community - Network Computing*. [Online]

Available at: <https://www.networkcomputing.com/careers-and-certifications/rootkit-detection/page/0/1>

[Accessed 29 January 2019].

New Horizon College, 2018. *Software prototyping*. [Online]

Available at: http://newhorizonindia.edu/nhc_kasturinagar/wp-content/uploads/2018/10/SOFTWARE-PROTOTYPING.pdf

[Accessed 7 January 2019].

Newhorizonindia.edu, 2018. *Software prototyping*. [Online]
Available at: http://newhorizonindia.edu/nhc_kasturinagar/wp-content/uploads/2018/10/SOFTWARE-PROTOTYPING.pdf
[Accessed 28 December 2018].

Newman, L. H., 2017. *The Biggest Cybersecurity Disasters of 2017 So Far* | WIRED.
[Online]
Available at: <https://www.wired.com/story/2017-biggest-hacks-so-far/>
[Accessed 29 December 2018].

Newstrom, H., 2002. *Using Linux Scripts to Monitor Security*, s.l.: SANS.

Norton, 2018. *Rootkit Malware*. [Online]
Available at: <https://www.nortonsecurityonline.com/security-center/rootkit.html>
[Accessed 28 December 2018].

Open Source Initiative, 2018. *The MIT License* | *Open Source Initiative*. [Online]
Available at: <https://opensource.org/licenses/MIT>
[Accessed 3 January 2019].

Panda Security, 2011. *ROOTKITS - Information - Types of Malware- PANDA SECURITY*.
[Online]
Available at: <https://www.pandasecurity.com/en/security-info/types-malware/rootkit/>
[Accessed 21 January 2019].

Passeri, P., 2018. *2017 Cyber Attacks Statistics - HACKMAGEDDON*. [Online]
Available at: <https://www.hackmageddon.com/2018/01/17/2017-cyber-attacks-statistics/>
[Accessed 2 October 2018].

Pearson, S., 2018. *Six Sigma Software: Definition, Types and Uses - Tallyfy*. [Online]

Available at: <https://tallyfy.com/six-sigma-software/>

[Accessed 7 January 2019].

Pearson, S., 2018. *Six Sigma Software: Definition, Types and Uses - Tallyfy*. [Online]

Available at: <https://tallyfy.com/six-sigma-software/>

[Accessed 4 December 2018].

Pérez, A. P., 2011. *Linux rootkits & TTY Hijacking*. Lyon, CERN Computer Security Team.

Planview, 2016. *What is Kanban? | Planview LeanKit*. [Online]

Available at: <https://www.planview.com/resources/articles/what-is-kanban/>

[Accessed 14 December 2018].

QASymphony, 2018. *Agile Methodology Guide: Understanding Agile Testing -*

QASymphony. [Online]

Available at: <https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/>

[Accessed 21 November 2018].

Rajnic, S., 2002. *An Introduction to the NSA's Security-Enhanced Linux: SELinux*, s.l.:

SANS.

Ray, A. & Nath, D. A., 2016. Introduction to Malware and Malware Analysis: A brief overview. *International Journal of Advance Research in Computer Science and Management Studies*, 4(10), pp. 22-29.

Schultz, E. E. & Ray, E., 2008. *Rootkits: The Ultimate Malware Threat*. [Online]

Available at: <http://www.infosectoday.com/Articles/Rootkits.htm>

[Accessed 15 October 2018].

Singh, P. & Singh, A., 2016. Computer Forensics: An Analysis on Windows and Unix from data recovery perspective. *International Research Journal of Engineering and Technology*, 3(4), pp. 586-591.

Singh, P. & Singh, A., 2016. Computer Forensics: An Analysis on Windows and Unix from data recovery perspective. *International Research Journal of Engineering and Technology*, 3(4), pp. 586-591.

Sokol, P., Host, J. & Vaško, M., 2015. Data control in virtual honeynets based on operating system-level virtualization. *International Journal of advanced studies in Computer Science and Engineering*, 4(9), pp. 14-20.

Sokol, P., Host, J. & Vaško, M., 2015. Data control in virtual honeynets based on operating system-level virtualization. *International Journal of advanced studies in Computer Science and Engineering*, 4(9), pp. 14-20.

Stiawan, D., Idris, M. Y., Salam, M. S. H. & Abdullah, A. H., 2012. Intrusion threat detection from insider attack using learning behavior-based. *International Journal of the Physical Sciences*, 7(4), pp. 624-637.

Su, H., 2008. *Rootkit-Resistant Disks*. Pennsylvania, s.n.

Thilak, 2018. *Rootkit: What is Rootkit & it's types? How to Detect it*. [Online]
Available at: <https://antivirus.comodo.com/blog/computer-safety/what-is-rootkit/>
[Accessed 19 March 2019].

Tzu, S., 1971. *SUN TZU on the ART OF WAR*. s.l.:The Winds of Japan Shop.

U, N., B, N. & D, S., 2016. An Analysis of Linux Operating System. *International Journal of Trend in Research and Development*, 3(1), pp. 32-35.

University of Southern California, 2018. 6. *The Methodology - Organizing Your Social Sciences Research Paper - Research Guides at University of Southern California*. [Online]

Available at: <http://libguides.usc.edu/writingguide/methodology>

[Accessed 31 December 2018].

Uppal, D., Mehra, V. & Verma, V., 2014. Basic survey on Malware Analysis, Tools and Techniques. *International Journal on Computational Sciences & Applications*, 4(1), pp. 103-112.

Wampler, D. R., 2007. *Methods for detecting kernel rootkits*, s.l.: Electronic Theses and Dissertations.

Zacks, A., 2018. *Antivirus Facts, Trends and Statistics for 2019*. [Online]

Available at: <https://www.safetydetective.com/blog/antivirus-statistics/>

[Accessed 16 February 2019].

Digité (2018). Kanban Board. [image] Available at:

<https://d30s2hykpf82zu.cloudfront.net/wp-content/uploads/2018/11/Kanban-Board.png>

[Accessed 12 Nov. 2018].

The prototyping model of Software development lifecycle. (2017). [image] Available at:

<https://www.daaminotes.com/wp-content/uploads/2017/10/prototyping.jpg> [Accessed 13

Dec. 2018].

Ygraph (2012). The Six Sigma Flow. [image] Available at:

<http://ygraph.com/graphs/sixsigma-20120605T014853-etvesmw.jpeg> [Accessed 15 Dec.

2018].

Chapter 8: Bibliography

Alhassan, J. & Misra, S., 2016. Evaluating Capabilities of Rootkits Tools. *International Journal of Advanced Multidisciplinary Research and Studies*, 1(1), pp. 27-30.

Alonso, A., 2009. *Rootkits & Honeypots*. s.l., s.n.

Alsalam, J., Banerjee, S., Musick, G. & Saftoiu, R., 2005. *Computer Security and Rootkits*, s.l.: s.n.

Bajaj, G., 2006. *Taxonomy of Rootkits*, Patiala: s.n.

Beazley, D. & Jones, B. K., 2013. *Python Cookbook*. 3 ed. s.l.:O'Reilly Media.

Beck, D. R. & Wang, Y.-M., 2011. *DETECTING USER-MODE ROOTKITS*. United States, Patent No. US 8,661,541 B2.

Beegle, L. E., 2007. Rootkits and Their Effects on Information Security. *Information Systems Security*, 16(3), pp. 164-176.

Bellasi, P., 2010. *The Unix and GNU/Linux command line*. s.l., Free Electrons.

BerkeleyTech. L.J., 2006. Sony-BMG Copy Protection Rootkit. *Berkeley Technology Law Journal*, 21(1), pp. 551-555.

Bhardwaj, S. & Pranjal, K., 2016. Detection Techniques of Rootkits. *Imperial Journal of Interdisciplinary Research*, 2(5), pp. 2454-1362.

Blunden, B., 2011. The Final Chapter. In: B. Blunden & B. G. Labs, eds. *The Rootkit Arsenal*. s.l.:Jones & Bartlett Learning, pp. 2-21.

Bowman, M., Brown, H. D. & Pitt, P., 2007. *An undergraduate rootkit research project: How available? How hard? How dangerous?*, s.l.: s.n.

Brumley, D., 2015. Invisible Intruders: Rootkits in Practice. *History*, 40(2), pp. 69-71.

Bunten, A., 2004. *UNIX and Linux based Rootkits Techniques and Countermeasures*, Hamburg: DFN-CERT Services GmbH.

Butler, J., 2004. *Black Hat Windows 2004 - DKOM (Direct Kernel Object Manipulation)*. s.l., HBGary.

Carnegie Mellon University, 2014. *Linux Antivirus*, s.l.: Software Engineering Institute .

Case, A., 2012. *Volatility Labs: MoVP 1.5 KBeast Rootkit, Detecting Hidden Modules, and sysfs*. [Online]

Available at: <https://volatility-labs.blogspot.com/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html>

[Accessed 19 November 2018].

CERN Computer Security Team, 2017. *CERN Computer Security Information*. [Online]

Available at: <https://security.web.cern.ch/security/venom.shtml>

[Accessed 27 December 2018].

Check Point Research, 2017. *MID-YEAR REPORT*, s.l.: Check Point.

Choi, J. et al., 2008. *Live Forensic Analysis of a Compromised Linux System Using LECT (Linux Evidence Collection Tool)*. Busan, s.n.

Chuvakin, A., 2003. *An Overview of Unix Rootkits*, Chantilly: iDEFENSE Inc..

CollabNet , 2018. *What Is Kanban? An Introduction to Kanban Methodology*. [Online]

Available at: <https://resources.collab.net/agile-101/what-is-kanban>

[Accessed 29 December 2018].

Davis, M. A., Bodmer, S. M. & LeMasters, A., 2009. *Malware and Rootkits: Malware & Rootkits Secrets & Solutions*. 1st ed. s.l.:McGraw-Hill Education.

Digité, 2018. *What is Kanban? Comprehensive Overview of the Kanban Method*. [Online]
Available at: <https://www.digite.com/kanban/what-is-kanban/>
[Accessed 28 December 2018].

Dillard, K., 2005. *What are user-mode vs. kernel-mode rootkits?*. [Online]
Available at: <https://searchenterprisedesktop.techtarget.com/news/1086469/What-are-user-mode-vs-kernel-mode-rootkits>
[Accessed 18 March 2019].

Dobre, C., 2013. *antivirus - How do antiviruses scan for thousands of malware signatures in a short time? - Information Security Stack Exchange*. [Online]
Available at: <https://security.stackexchange.com/questions/30362/how-do-antiviruses-scan-for-thousands-of-malware-signatures-in-a-short-time>
[Accessed 4 March 2019].

Donovan, F., 2016. *Top 5 Rootkit Threats and How to Root Them out*. [Online]
Available at: <https://www.esecurityplanet.com/network-security/top-5-rootkit-threats-and-how-to-root-them-out.html>
[Accessed 2 February 2019].

Dumont, R., M.Léveillé, M.-E. & Porcher, H., 2018. *The Dark Side of the ForSSH // A landscape of OpenSSH backdoors*, s.l.: ESET.

Dunn, J. E. & Magee, T., 2018. *Linux malware threats you should know about | Gallery | Computerworld UK*. [Online]
Available at: <https://www.computerworlduk.com/galleries/security/10-linux-malware-threats->

[bots-backdoors-trojans-malicious-apps-3634006/](#)

[Accessed 28 January 2019].

ElProCus, 2014. *Know all about Linux Operating System with Applications*. [Online]

Available at: <https://www.elprocus.com/linux-operating-system/>

[Accessed 19 October 2018].

EVOKNOW, Inc, 2005. DEALING WITH ROOTKIT ATTACKS ON LINUX. In:

DEALING WITH ROOTKIT ATTACKS ON LINUX. s.l.:The Evolving Knowledge Company, pp. 1-29.

Fehlmann, T. M., 2004. *Six Sigma for Software*, Zurich: s.n.

Fortinet, 2018. *Threat Landscape Report Q3 2018*, Sunnyvale: Fortinet.

F-Secure Labs, 2017. *Rootkit Description* | *F-Secure Labs*. [Online]

Available at: <https://www.f-secure.com/v-descs/rootkit.shtml>

[Accessed 19 January 2019].

Gray, M., 2019. *Shell Scripting for Reconnaissance and Incident Response*, s.l.: SANS.

Hannel, J., 2003. *Linux RootKits For Beginners – From Prevention to Removal*, s.l.: SANS Institute Reading Room.

Harley, D. & Lee, A., 2009. *The Root of All Evil? - Rootkits Revealed*, San Diego: ESET, LLC.

Harley, D. & Lee, A., 2009. *The Root of All Evil? - Rootkits Revealed*, s.l.: ESET.

Heasman, J., 2006. Rootkit threats. *Network Security*, 2006(1), pp. 1-20.

Hensing, R., 2004. *Rootkit Technologies*. s.l., s.n.

Hertzog, R. & Mas, R., 2015. *The Debian Administrator's Handbook*. 1 ed. s.l.:Freexian .

Hoffman, C., 2016. *The Linux Directory Structure, Explained*. [Online]

Available at: <https://www.howtogeek.com/117435/htg-explains-the-linux-directory-structure-explained/>

[Accessed 2 January 2019].

Hoglund, G. & Butler, J., 2005. *Rootkits: Subverting the Windows Kernel*. s.l.:Addison Wesley Professional.

Holt, T. J., 2005. *Hacks, Cracks, and Crime: An Examination of the Subculture and Social Organization of Computer Hackers*, s.l.: Dissertation.

Hosch, W. L., 2018. *Rootkit | malware | Britannica.com*. [Online]

Available at: <https://www.britannica.com/technology/rootkit>

[Accessed 15 January 2019].

Infosec, 2017. *Rootkits: User Mode*. [Online]

Available at: <https://resources.infosecinstitute.com/rootkits-user-mode-kernel-mode-part-1/>

[Accessed 19 March 2019].

IT PRO, 2018. *The cyber security threat in six charts | IT PRO*. [Online]

Available at: <https://www.itpro.co.uk/security/29224/the-cyber-security-threat-in-charts>

[Accessed 2 January 2019].

ITU, 2017. *Cybersecurity - Facts and Figures*. [Online]

Available at: <https://www.itu.int/en/ITU-D/Partners/Pages/Call4Partners/CYBLDCStats.aspx>

[Accessed 28 January 2019].

Kanban Tool, 2018. *Kanban Presentation - Kanban in four easy steps | Kanban Tool*.

[Online]

Available at: Kanban Tool

[Accessed 30 December 2018].

Kapoor, A. & Mathur, R., 2011. *PREDICTING THE FUTURE OF STEALTH ATTACKS*, Beaverton: McAfee Inc..

Kaspersky Lab, 2013. *What is a rootkit and how to remove it* | *Kaspersky Lab official blog*.
[Online]

Available at: <https://www.kaspersky.com/blog/rootkit/1508/>

[Accessed 21 March 2019].

Kassner, M., 2008. *News, Tips, and Advice for Technology Professionals - TechRepublic*.
[Online]

Available at: <https://www.techrepublic.com/blog/10-things/10-plus-things-you-should-know-about-rootkits/>

[Accessed 28 January 2019].

Kiger, D., 2017. *The advantages and disadvantages of Six Sigma Methodology - David Kiger's Blog*. [Online]

Available at: <https://davidkigerinfo.wordpress.com/2017/06/13/the-advantages-and-disadvantages-of-six-sigma-methodology/>

[Accessed 9 December 2018].

Kili, A., 2018. *5 Tools to Scan a Linux Server for Malware and Rootkits*. [Online]

Available at: <https://www.tecmint.com/scan-linux-for-malware-and-rootkits/>

[Accessed 17 March 2019].

Koch, M., 2015. *An Introduction to Linux-based malware*, s.l.: SANS Institute.

Komur, Y. S. & Ayfer, C. U., 2004. *Detecting and Removing Rootkits*. [Online]

Available at:

http://cayfer.bilkent.edu.tr/~cayfer/linux/Detecting_and_Removing_Rootkits.html

[Accessed 15 March 2019].

Kornbrust, A., 2005. *Database Rootkits*. Neunkirchen, Red-Database-Security GmbH.

Leibowitz, M., 2016. *Horse Pill A New Type Of Linux Rootkit*. Las Vegas, Blackhat USA 2016.

leontranter, 2016. *Alternatives to scrum - Extreme Uncertainty - practical agile*. [Online]

Available at: <https://www.extremeuncertainty.com/alternatives-to-scrum/>

[Accessed 21 December 2018].

Leontranter, 2016. *Alternatives to scrum - Extreme Uncertainty - practical agile*. [Online]

Available at: <https://www.extremeuncertainty.com/alternatives-to-scrum/>

[Accessed 6 January 2019].

Lessard, P., 2003. *Linux Process Containment – A practical look at chroot and User Mode Linux*, s.l.: SANS.

Lewinson, M., 2012. *How to Write a Feasibility Study Report (FSR)*. [Online]

Available at: <https://mymanagementguide.com/feasibility-study-reporting-steps-to-writing-a-feasibility-study-report-fsr/>

[Accessed 3 November 2018].

Leybourn, E., 2015. *stages of delivery - an example Kanban - The Agile Director*. [Online]

Available at: <https://theagiledirector.com/article/2015/04/21/7-stages-of-delivery-an-example-kanban/>

[Accessed 6 December 2018].

Liu, L. et al., 2012. Research and Design of Rootkit Detection Method. *Physics Procedia*, Volume 33, pp. 852-857.

Luz-Romero, P. A., 2003. *Secure OS Environments for Linux*, s.l.: SANS.

M.Léveillé, M.-E., 2014. *An In-depth Analysis of Linux/Ebury* | *WeLiveSecurity*. [Online]
Available at: [Marc-Etienne M.Léveillé](#)
[Accessed 30 December 2018].

Marlabs, 2018. *Threat Briefing – October 01*. [Online]
Available at: <https://www.marlabs.com/threat-briefing-october-01/>
[Accessed 30 December 2018].

Martin, C. D. & Martin, D. H., 1990. Professional Codes of Conduct and Computer Ethics Education. *Social Science Computer Review*, 8(1), pp. 96-108.

Mashevsky, Y., Sapronov, K. & Monastyrsky, A., 2005. *Rootkits and how to combat them* | *Securelist*. [Online]
Available at: <https://securelist.com/rootkits-and-how-to-combat-them/36055/>
[Accessed 4 January 2019].

Mathur, K. & Hiranwal, S., 2013. A Survey on Techniques in Detection and Analyzing Malware Executables. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4), pp. 422-428.

McAfee, 2006. Rootkits, Part 1 of 3: The Growing Threat. In: California: McAfee, pp. 1-8.

Miller, J. V., 2003. *SHV4 Rootkit Analysis*, s.l.: Symantec.

Miller, K., 2003. *Fight crime. Unravel incidents... one byte at a time.*, s.l.: SANS.

Mukund, 2018. *Why a Feasibility Study is Important in Project Management*. [Online]

Available at: <https://www.simplilearn.com/feasibility-study-article>

[Accessed 9 December 2018].

Mulligan, D. K. & Perzanowski, A. K., 2007. The Magnificence of the Disaster:

Reconstructing the Sony BMG Rootkit Incident. *Berkeley Technology Law Journal*, 22(3), pp. 1158-1218.

Muslihat, D., 2018. *7 Popular Project Management Methodologies And What They're Best Suited For* | Zenkit. [Online]

Available at: <https://zenkit.com/en/blog/7-popular-project-management-methodologies-and-what-theyre-best-suited-for/>

[Accessed 19 December 2018].

Naugolnyi, M., 2015. *CloudLinux OS Blog - iDRAC vulnerability - hacked servers*. [Online]

Available at: <https://cloudlinux.com/cloudlinux-os-blog/entry/idrac-vulnerability-hacked-servers>

[Accessed 3 December 2018].

Network Computing, 2006. *Rootkit Detection* | *IT Infrastructure Advice, Discussion, Community - Network Computing*. [Online]

Available at: <https://www.networkcomputing.com/careers-and-certifications/rootkit-detection/page/0/1>

[Accessed 29 January 2019].

New Horizon College, 2018. *Software prototyping*. [Online]

Available at: http://newhorizonindia.edu/nhc_kasturinagar/wp-content/uploads/2018/10/SOFTWARE-PROTOTYPING.pdf

[Accessed 7 January 2019].

Newhorizonindia.edu, 2018. *Software prototyping*. [Online]
Available at: http://newhorizonindia.edu/nhc_kasturinagar/wp-content/uploads/2018/10/SOFTWARE-PROTOTYPING.pdf
[Accessed 28 December 2018].

Newman, L. H., 2017. *The Biggest Cybersecurity Disasters of 2017 So Far* | WIRED.
[Online]
Available at: <https://www.wired.com/story/2017-biggest-hacks-so-far/>
[Accessed 29 December 2018].

Newstrom, H., 2002. *Using Linux Scripts to Monitor Security*, s.l.: SANS.

Norton, 2018. *Rootkit Malware*. [Online]
Available at: <https://www.nortonsecurityonline.com/security-center/rootkit.html>
[Accessed 28 December 2018].

Open Source Initiative, 2018. *The MIT License* | Open Source Initiative. [Online]
Available at: <https://opensource.org/licenses/MIT>
[Accessed 3 January 2019].

Panda Security, 2011. *ROOTKITS - Information - Types of Malware- PANDA SECURITY*.
[Online]
Available at: <https://www.pandasecurity.com/en/security-info/types-malware/rootkit/>
[Accessed 21 January 2019].

Passeri, P., 2018. *2017 Cyber Attacks Statistics - HACKMAGEDDON*. [Online]
Available at: <https://www.hackmageddon.com/2018/01/17/2017-cyber-attacks-statistics/>
[Accessed 2 October 2018].

Pearson, S., 2018. *Six Sigma Software: Definition, Types and Uses - Tallyfy*. [Online]

Available at: <https://tallyfy.com/six-sigma-software/>

[Accessed 7 January 2019].

Pearson, S., 2018. *Six Sigma Software: Definition, Types and Uses - Tallyfy*. [Online]

Available at: <https://tallyfy.com/six-sigma-software/>

[Accessed 4 December 2018].

Pérez, A. P., 2011. *Linux rootkits & TTY Hijacking*. Lyon, CERN Computer Security Team.

Planview, 2016. *What is Kanban? | Planview LeanKit*. [Online]

Available at: <https://www.planview.com/resources/articles/what-is-kanban/>

[Accessed 14 December 2018].

QASymphony, 2018. *Agile Methodology Guide: Understanding Agile Testing -*

QASymphony. [Online]

Available at: <https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/>

[Accessed 21 November 2018].

Rajnic, S., 2002. *An Introduction to the NSA's Security-Enhanced Linux: SELinux*, s.l.:

SANS.

Ray, A. & Nath, D. A., 2016. Introduction to Malware and Malware Analysis: A brief overview. *International Journal of Advance Research in Computer Science and Management Studies*, 4(10), pp. 22-29.

Schultz, E. E. & Ray, E., 2008. *Rootkits: The Ultimate Malware Threat*. [Online]

Available at: <http://www.infosectoday.com/Articles/Rootkits.htm>

[Accessed 15 October 2018].

Singh, P. & Singh, A., 2016. Computer Forensics: An Analysis on Windows and Unix from data recovery perspective. *International Research Journal of Engineering and Technology*, 3(4), pp. 586-591.

Singh, P. & Singh, A., 2016. Computer Forensics: An Analysis on Windows and Unix from data recovery perspective. *International Research Journal of Engineering and Technology*, 3(4), pp. 586-591.

Sokol, P., Host, J. & Vaško, M., 2015. Data control in virtual honeynets based on operating system-level virtualization. *International Journal of advanced studies in Computer Science and Engineering*, 4(9), pp. 14-20.

Sokol, P., Host, J. & Vaško, M., 2015. Data control in virtual honeynets based on operating system-level virtualization. *International Journal of advanced studies in Computer Science and Engineering*, 4(9), pp. 14-20.

Stiawan, D., Idris, M. Y., Salam, M. S. H. & Abdullah, A. H., 2012. Intrusion threat detection from insider attack using learning behavior-based. *International Journal of the Physical Sciences*, 7(4), pp. 624-637.

Su, H., 2008. *Rootkit-Resistant Disks*. Pennsylvania, s.n.

Thilak, 2018. *Rootkit: What is Rootkit & it's types? How to Detect it*. [Online]
Available at: <https://antivirus.comodo.com/blog/computer-safety/what-is-rootkit/>
[Accessed 19 March 2019].

Tzu, S., 1971. *SUN TZU on the ART OF WAR*. s.l.:The Winds of Japan Shop.

U, N., B, N. & D, S., 2016. An Analysis of Linux Operating System. *International Journal of Trend in Research and Development*, 3(1), pp. 32-35.

University of Southern California, 2018. 6. *The Methodology - Organizing Your Social Sciences Research Paper - Research Guides at University of Southern California*. [Online]

Available at: <http://libguides.usc.edu/writingguide/methodology>

[Accessed 31 December 2018].

Uppal, D., Mehra, V. & Verma, V., 2014. Basic survey on Malware Analysis, Tools and Techniques. *International Journal on Computational Sciences & Applications*, 4(1), pp. 103-112.

Wampler, D. R., 2007. *Methods for detecting kernel rootkits*, s.l.: Electronic Theses and Dissertations.

Zacks, A., 2018. *Antivirus Facts, Trends and Statistics for 2019*. [Online]

Available at: <https://www.safetydetective.com/blog/antivirus-statistics/>

[Accessed 16 February 2019].

Sourcefire, 2013. *Rootkits (Part 7): Signature-Based Defense*. [Video File]

Available at: <https://www.youtube.com/watch?v=pEkm03Npf7U>

[Accessed 21 March 2019]

Mark Shead, 2018. *What is Agile Methodology?* [Video File]

Available at: https://www.youtube.com/watch?v=ZZ_vnqvW4DQ

[Accessed 21 March 2019]

Axosoft, 2013. *Intro to Kanban in Under 5 Minutes (What is Kanban, Learn Kanban)*. [Video File]

Available at: <https://www.youtube.com/watch?v=R8dYLBjITUE>

[Accessed 21 November 2018]

Mark Shead, 2016. *What is Agile?* [Video File]

Available at: <https://www.youtube.com/watch?v=Z9QbYZh1YXY>

[Accessed 20 November 2018]

Education 4u, 2018. *prototype model in software engineering*. [Video File]

Available at: <https://www.youtube.com/watch?v=JhHkb7z5GzY>

[Accessed 21 November 2018]

Siddharth Agarwal Classes, 2017. *Six Sigma*. [Video File]

Available at: <https://www.youtube.com/watch?v=Mpg1LnqdZS8>

[Accessed 21 November 2018]

Development That Pays, 2017. *Scrum vs Kanban - Two Agile Teams Go Head-to-Head + FREE CHEAT SHEET*. [Video File]

Available at: https://www.youtube.com/watch?v=HNd1_irOL5k

[Accessed 21 November 2018]

Android Authority, 2017. *Why GPL violations are bad - Gary explains*. [Video File]

Available at: <https://www.youtube.com/watch?v=-BhGJz2yPt4>

[Accessed 19 November 2018]

Sinn, 2017. *How to Create a Malware Analysis Lab – VirtualBox*. [Video File]

Available at: <https://www.youtube.com/watch?v=proAnW3TvSs>

[Accessed 29 December 2018]

Myron Estibeiro, 2014. *automated malware analysis with cuckoo*. [Video File]

Application-level Rootkit Detection Program for Linux

Available at: <https://www.youtube.com/watch?v=OzNMkR6yaJ0>

[Accessed 29 December 2018]