Towards Preventing Password Thefts at Kernel Level using Terminate and Stay Resident Programs

N.G. Nageswari Amma[1] and N.G. Bhuvaneswari Amma[2]
[1]Department of Computer Science, Muslim Arts College, Thiruvithancode, Kanyakumari, Tamil Nadu, India.
[2]Department of Computer Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India.

Abstract

Terminate and Stay Resident (TSR) programs are application programs that always stay in the system once these are loaded and executed, until the system is shut down. TSR programs are highly used tools in password hacking techniques. For instance, a TSR can be designed as a screen lock program that prompts the user for entering the proper password for unlocking the screen which was locked for no activity on the system for a certain period of time. Such a TSR can be invoked automatically with malicious intention before the actual screen lock program runs. This malicious TSR captures and passes the password to any malicious program in the system or stores some where for further use. Presently no standard mechanism is available for preventing all such malicious TSR programs from stealing the passwords without the knowledge of the user. Albeit some third-party tools exist for this purpose these are at application level and cannot prevent all types of TSRs. Moreover, these were designed as particular TSR specific solutions. In this study, an algorithm is proposed that can be implemented in Operating System (OS) kernel for preventing all types of TSRs from stealing the system passwords. This solution needs no special software or hardware or any third-party tool. Furthermore, the user need not know any thing about this change in the OS. The proposed solution consists of modification of keyboard device driver, inclusion of new encryption and decryption modules in the kernel modules of which the later has been implemented in Linux kernel 5.3 of SUSE 15 Linux distribution.

Keywords: Hacker, Kernel, Password theft, TSR program

1. Introduction

As the deployment and usage of computer systems grow larger and larger, the security threats are also increasing. Many viruses, worms, trojans, and other computer threats are being found day by day [1]. At the same time effective solutions are also being developed. For instance, until now 12,000 different attacks were found on MS Windows where as it was only 30 on Linux due to its stringent access control policies on the users [2]. Albeit Linux offers many security features, all those features are dependent on the secrecy of the root password [3]. Because, once the root password is out, no security policy or feature can prevent any attack on the system and hence all the advantages are gone. In the same manner many security policies in the system are strongly dependent on the passwords or password related mechanisms. Hence, obviously strong and reliable password protection mechanisms are needed.

Many password stealing mechanisms exist in the literature. Terminate and Stay Resident (TSR) programs are one of the best and easiest methods of stealing the passwords [4]. TSR programs are application programs that always stay in the system once these are loaded and executed, until the system is shut down. A TSR can be easily created to mislead the legitimate user and to capture the certain system passwords. Presently no standard mechanism is available for preventing all such TSR programs from stealing the passwords. Albeit some third-party tools exist for this purpose at application level and cannot prevent all types of TSRs [5]. In this study, an algorithm is proposed that can be implemented in OS kernel for preventing all types of TSRs, which act as the screen lock programs and other programs in the same manner, from stealing the system passwords.

The rest of the paper is organized as follows: In section 2, the overview of the related past work is given. In section 3, the proposed algorithm and its working are described. In section 4, the pros and cons of the proposed mechanism are discussed. Finally, section 5 concludes the paper.

2. Related Works

The TSR programs mentioned in this study are basically a type of key loggers discussed in [6][7][8]. Key logger is a program which captures the user key strokes of the standard key board. The key strokes can be the password characters or some important user data. The key loggers are categorized as follows:

2.1 Hardware Key Loggers

Hardware key loggers are small inline devices placed in between the keyboard and the computer. The user of the computer cannot detect these loggers for long periods of time as the size of the device is small. However, these require physical access to the machine. These hardware devices have the power to capture hundreds of keystrokes including banking and email username and passwords. Nowadays attackers are working intelligently to hack all the confidential information.

2.2 Software Key Loggers

This type of logging is accomplished using the Windows function SetWindowsHookEx() that monitors all keystrokes. The spyware will typically come packaged as an executable file that initiates the hook function, plus a DLL file to handle the logging functions. An application that calls SetWindowsHookEx() is capable of capturing even auto complete passwords. During this pandemic period, almost 85% of the population is depending on the Internet that makes the attacker to easily hack the passwords and proceed further to perform all sorts of attacks.

2.3 Kernel Level Key Loggers

This type of key logger is at the kernel level and receives data directly from the input device such as keyboard. It replaces the core software for interpreting keystrokes. It can be programmed to be virtually undetectable by taking advantage of the fact that it is executed on boot, before any user-level applications start. Since the program runs at the kernel level, one inability of this approach is that it fails to capture auto complete passwords, as this information is passed in the application layer.

Generally different solutions such as anti-key loggers, virtual key boards, and firewalls exist to defend from those key loggers. Existing solutions and their drawbacks are as follows:

a) Software anti-key loggers are good defendable tools for software key loggers but these are not suitable for kernel level key loggers, which act directly on the key board registers.
b) Kernel level anti-key loggers are suitable for both the types of key loggers but these generally encrypt each and every character typed by the user by putting more processing burden on the processor.
c) Virtual key boards also called on-screen boards can be used for entering passwords but these are prone to screen grabbers, which can scan every action on the screen there by estimating the passwords [5].
d) Enabling a firewall does not stop key loggers from stealing the passwords, but can possibly prevent transmission of the logged material over the net [4].

This study deals with the software and kernel level key loggers as the solution is developed for execution in the kernel. But it does not deal with the hardware tampering as in hardware key loggers.

3. Proposed Solution

The proposed solution keeps the system passwords safe using symmetric encryption at the keyboard registers level even though the hacker captures them using some sort of TSR programs.

The solution works based on the following:
a) The mode of the operation (read/write) by the application on the password file, which stores the system passwords.
b) The data entered by the user (passwords/ ordinary data).

If the running program is screen lock program, the mode of operation is *read* and if the program is password changing application the mode is *write*. Instead of encrypting each and every character that user has typed a simple hot key mechanism is used to differentiate the critical data to ordinary data. The user has to use CTRL+ALT+P before entering the password and CTRL+ALT+Q after entering the password. A simple and less complex symmetric encryption in the key board device driver and a pair of symmetric encryption and decryption routines in the kernel with the same key is introduced. This encryption does not affect the actual encryption that takes place on the password files.
The solution can be described in two perspectives as follows:

3.1 Legitimate User's Perspective

The flow chart for the proposed mechanism in legitimate user perspective is depicted in Fig. 1. The shaded portion of the diagram is the actual flow of the proposed solution. Remaining portion is the conventional flow.

a) When the user is about to enter the passwords into screen lock or password changing programs first the user will give the hot key combination CTRL+ALT+P, which informs the keyboard driver to set a flag *keyreg* and *encrypt* the following data before putting into the registers.

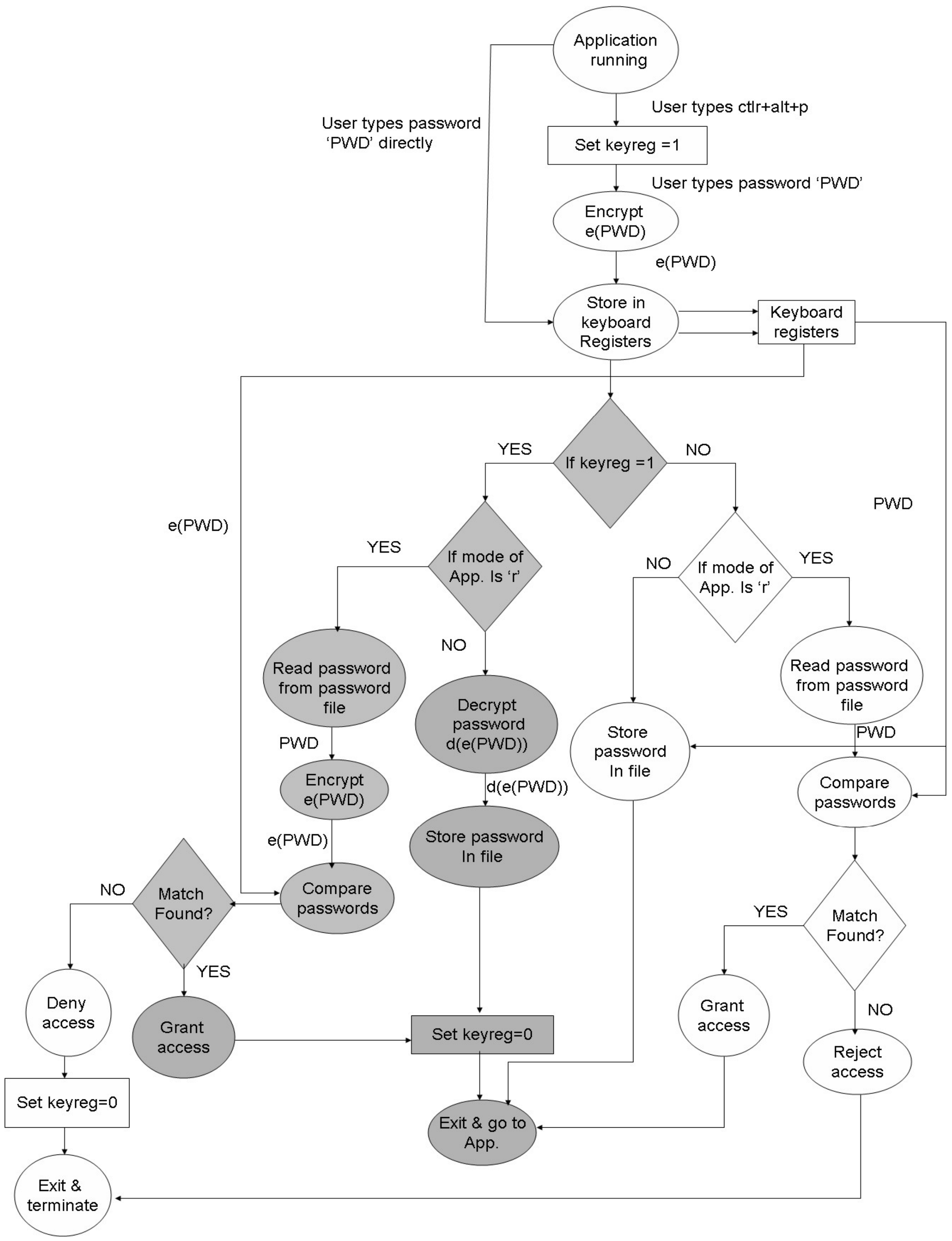

Fig. 1: Flow Chart of Legitimate User Perspective

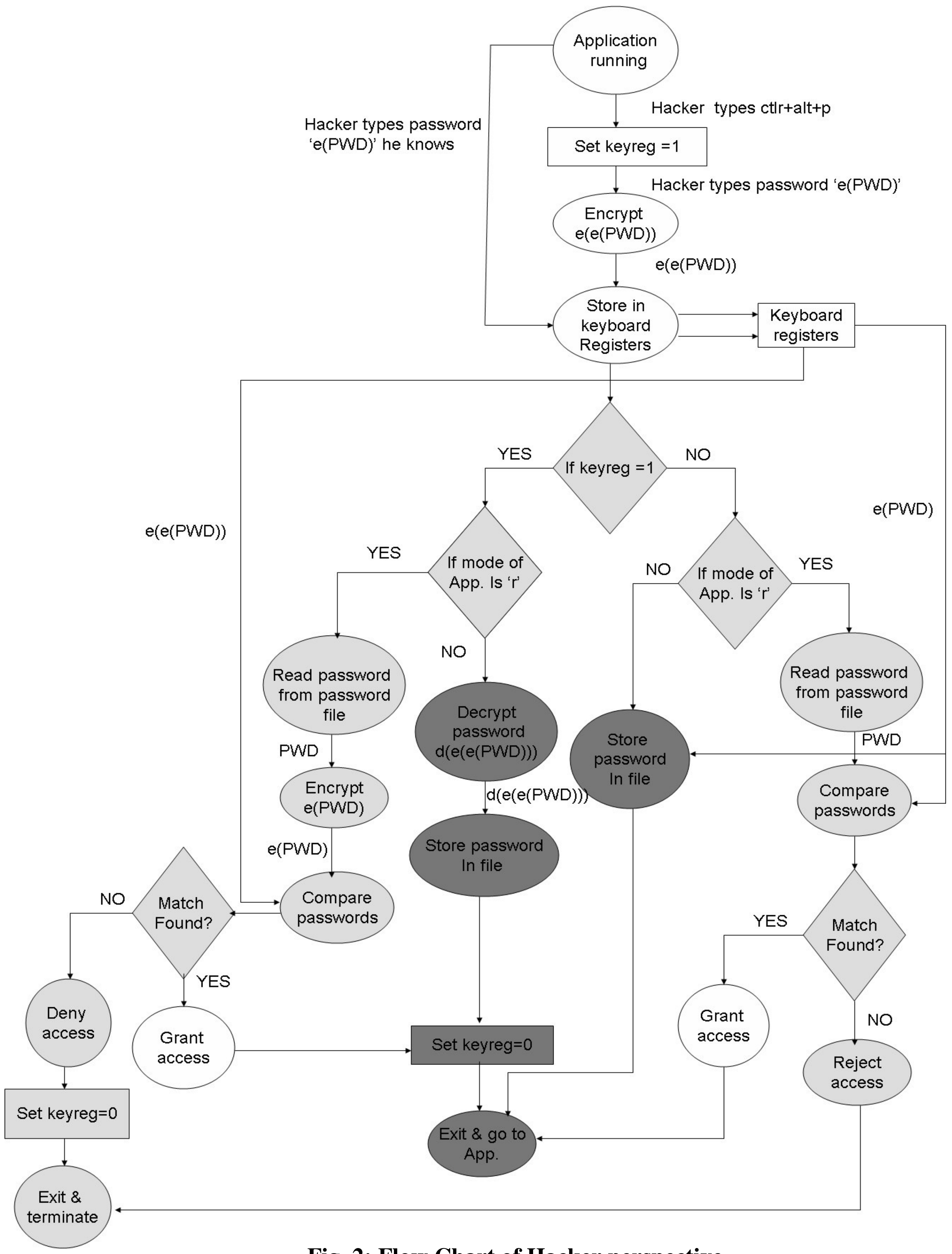

Fig. 2: Flow Chart of Hacker perspective

b) Now if the application uses the password file in *read* mode, i.e., screen lock program and the flag *keyreg* is set, the corresponding password contents will be read from the password file and encrypted before comparing with the password entered by the user. The comparison results in a match as both passwords are encrypted with the same symmetric key.

c) If the application uses the password file in *write* mode, i.e., password changing program and the flag *keyreg* is set, the entered and encrypted password will again be decrypted before writing into the file.

3.2 Hacker's Perspective

As the TSR program can read the passwords through the keyboard registers at its best that is through kernel level key logger, the hacker knows only the encrypted password. Further the hacker may or may not know the existence of the hot key mechanism. In both the cases the hacker is prevented from stealing the password. The flow chart for the proposed mechanism in hacker perspective is depicted in Fig. 2. The lightly shaded portion of the diagram will happen when the hacker tries to access the system. The strongly shaded portion of the diagram will not be happened as the hacker can not work with write access to the resources at that point of time.

a) If the hacker knows the hot key mechanism, he/she will enter the already encrypted password after hot keys and hence it is twice encrypted for comparison with password from password file, leading to denial of access.

b) If the hacker does not know the hot key mechanism or intentionally not using the mechanism, the flag *keyreg* will not be set. Therefore, even though the application is in *read* mode, as the flag is not set, the corresponding password contents of password file will not be encrypted for comparison with the entered not encrypted password leading to denial of access.

The following example makes the idea clear:

Suppose the password is *shrish* and for simplicity assume that the encryption is toggling the letters of even and odd positions.

a) The user entered the password after hot key combination. Now the keyboard registers will have the password as *hsirhs.* The hacker can read this only.

b) If the user wants to save the password into password file, the access will *write* and hence it will be decrypted into *shrish* before writing into the file.

c) If the user using password in the application like screen lock program, the access will be *read* and hence the password file contents, *shrish, w*ill be encrypted into *hsirhs,* which is used for comparison leading to granting access.

d) If the hacker enters the password as *hsirhs* without hot key combination it will be compared with the contents of the password file *shrish* without encryption leading to denial of access.

e) If the hacker enters the password with hot key mechanism, encryption will be done twice to make it *shrish*. It will be compared with *hsirhs* which is taken from encrypting the contents of the password file leading to denial of access.

3.3 Algorithms

3.3.1 Algorithm 1: Part of Modified Keyboard Driver

```
ReadScanCodes (scancode);
if (scancode = = ‘CTRL + P’)
{
        keyreg = true;
}
else
{
        if (keyreg)
        {
                tempkeycode = Encrypt (scancode);
        }
        SetKeyRegisters (tempkeycode);
}
```

3.3.2 Algorithm 2: Accessing the Password File

```
if (mode = = ‘r’)
{
        if (CheckPermission (processid, fileid, ‘r’))
        {
        temppassword = Read (passwordid, fileid);
        if (keyreg = = true)
        {
        temppassword = Encrypt (temppassword);
        }
        }
        if (Compare (temppassword, userpassword))
        {
                PermitAccess();
        }
        else DenyAccess();
        keyreg = false;
}
if (mode = = ‘w’)
{
        temppassword = userpassword;
        if (CheckPermission (processed, fileid, ‘w’))
        {
        if(keyreg = = true)
        temppassword = Decrypt (userpassword);
        Write (passwordid, fileid, temppassword);
        }
        keyreg = false;
}
```

4. Pros and Cons of the Proposed Solution

The positive side of the solution is, it requires no special hardware or software or third-party tools for designing this solution. Even the end user need not know any thing about this change except the usage of hot key combination. As the solution can be uploaded into the kernel as Dynamically Loadable Modules (DLM) [9] it is backward compatible as those modules can be loaded and removed as and when needed. By implementing this solution into open-source kernel such as Linux distributions, the strength and other security features of such operating systems remains unaltered.

Another side, this solution is designed only for a specific class of password thefts. Hundreds of methods exist in the literature for password thefts. And this solution can not be ported into other Operating Systems that are not open source [10]. This method seems to be increasing the load on the processor due to another layer of encryption. But care has been taken so that encryption will be used only on critical data such as passwords which may not leave burden on the processor. The main drawback of this approach is it does not work with the web related passwords as they will be stored somewhere in the internet and those servers are not accessible to the host system.

5. Conclusion

A kernel level mechanism for preventing system password thefts by TSR programs has been proposed. The key idea is using one more level of encryption on the passwords at the keyboard registers to mislead the TSRs on the content of the passwords. Care has been taken to reduce the load of the encryption by using some hot key combination. Based upon the access mode of the password file and hot key combination an intelligent escape from the TSR to capture the password has been discussed. More over this approach does not require any hardware or software or any third-party tool. Even the user need not know this change in the kernel.

References

[1] Keyloggers, http://www.keylogger.org
[2] U. Shafique and S.B. Zahur, “Towards Protection Against a USB Device whose Firmware has been Compromised or Turned as ‘BadUSB’”, In Proceedings of Future of Information and Communication Conference, pp. 975-987, 2019.
[3] S. McClure, J. Scambray, and G. Kurtz, “Hacking Exposed”, McAfee, 5th edition, 2005.
[4] Hacking novell local area networks, Fairfax Country, http://www.textfiles.com/hacking
[5] Young, M. Yung, "Deniable Password Snatching: On the Possibility of Evasive Electronic Espionage," IEEE Symposium on Security & Privacy, pages 224-235, May 4-7, 1997.
[6] Complete details of key logging, http://en.wikipedia.org/wiki/Keystroke_logging

[7] White paper on key loggers by sachin setty, http://www.securityfocus.com
[8] Cormac Herley and Dinei Florencio, "How to Login From an Internet Cafe Without Worrying About Keyloggers", Microsoft Research, Redmond.
[9] Daniel P. Bovet, Marco Cesat, "Understanding Linux kernel", O'Reilly (3rd edition), 2010.
[10] Peter G. Smith, Charles, "Linux network security" by River Media, 2005.