Next generation system security measures based on the process behaviour and permissions in Linux

D Jayaprakash* and Nagoor Meeran
SRM Institute of Science and Technology, Chennai, India

*Corresponding author: jayaprakash1609@gmail.com

Abstract. Threats are increasing day by day all over the world so information assert is at major risk. Security can be provided with the help of security professionals by using various tools, software and techniques etc., but the weakest link in the security chain is END USER. So it’s essential to ensure that end user is aware of threats. If end user is aware of what is going on in his/her PC then most of the security related issues can be solved. The newly built application in this project, is user-friendly and user can easily navigate through the application since all these information will be in plain English and in GUI format. The objective of this project is to do behavioural analysis of the processes to create awareness to the end users about the processes running in the system is a legitimate process or not. The processes running on the Linux system will be running on behalf of the user who executes it. So, the process will be having the access permission as same as the user who executes it i.e., the process can access the file with what user is allowed to do. So in order to know the process permission of the file, we need to integrate user permission of the file and processes those are running under that user along with process description.

Apparently, some processes are legitimate to access the file but it’s not. So, we need to trace the process behaviour and if the behaviour doesn’t seem to be legitimate then by stating the behavioural activity of the process & its malicious activities, we have to suggest the user to take necessary action towards it. Also, there are more additional features included such as end user can kill non-legitimate processes or uninstall the application which runs those malicious processes. If these two features fail to work then user can contact any System Security professionals. This proposal is comparatively unique since it will create awareness for end user to secure his/her privacy. If this idea is implemented in all the systems then it can possibly enrich the knowledge of the end user to secure his/her private asserts in his/her system.

1. Introduction
In Linux system, every running process will execute on behalf of a given User. Thus, the process is allowed to do what this User is allowed to do. Accordingly, we can conclude that the process permission of the file is based on the file permission of the user who is running that process. In-order to find the process permission of the file, we need to integrate (Let’s Assume that the file name is AAAA) permission of all the users on the file AAAA (this includes owner, group users and other users permission), finding the no. of processes running on the file AAAA and finding the processes running under which user. We can fetch those data from Inode table, File table, User file descriptor table and Process Descriptor table in Linux. Inode table can be used to get the Meta data of the file. File table will keep track of user’s next read or write bye and access rights of opening process.
User file descriptor table will identify all the open files for a process. Process Descriptor table is used to find resources used by the process. Each process is uniquely identified by process identifier (pid).

In Linux system, there are many algorithms for file access. There is a low level algorithm called Namei which comes under File System algorithm and namei consist of 3 sub algorithms namely iget, iput and bmap. Iget Algorithm will return the previously identified inode. Bmap will set the kernel parameter for accessing the file. Ipput will release the inode. Understanding of these 3 sub algorithms is very important to secure a file without being accessed by any process.

| Namei       | Alloc free | Ialloc ifree |
|-------------|------------|--------------|
| iget        |            |              |
| iput        |            |              |
| bmap        |            |              |
| getblk      | brelse     | breada       |
|             | bwrite     |              |

Buffer allocation algorithm

| Figure 1. Lower Level File System Algorithm of Linux Operating System |

2. Proposed System

2.1 Problem Statement

Human is the weakest link in the security chain. Malicious process can access the user files illegally even without user’s knowledge. So, the integrity and confidentiality of the files in our systems can be compromised. End-user will not have any idea about his file’s secrecy and the necessary actions needs to be taken if the file is not secured. So, we can conclude that End-user is not aware about his/her privacy on his/her PC to secure his file being accessed by malicious processes. Unless the end-user is aware of the malicious processes running on his/her PC, security breach cannot be solved completely.

2.2 Proposed Solution

The End User is the one who works with the computer to perform various task. Some processes are legitimate to access the file but it’s not. So, we need to trace the process behavior and if the behavior doesn’t seem to be legitimate then by stating the behavioral activity of the process & its malicious activities, we have to suggest the user to take necessary action towards it. Application is nothing but a GUI which can perform the task specified to it on the computer. Here end-user can start our application to perform two operations. In, first operation user can scan the process behavior with the help of process descriptor in order to find the malicious process. Second operation is securing the file by keeping it in secure mode so that no process can access that file.

The details about the process can be fetched from the kernel with the help of user file descriptor, file table, inode table and process descriptor. Also we can secure the file by enabling the secure mode of file so that even if process has permission to access the file, the process should again ask for permission to the user before it starts accessing the files in the secure mode. System architecture of the proposed solution is given below,
2.3 Algorithm for securing the file
   - Add files in our custom mode called “Secure mode”
   - Find inode number of the files in secure mode
   - Add the inode number of the files in secure mode to an array
   - Compare the array with inode table in the structure called inode
   - If inode number of secure mode files present in inode table
     - Return error
   - Else
     - Return inode

2.4 Data Structures and implementation of kernel
2.4.1 Struct inode.
This structure will contain all the Meta data about the file such as user id, group id, inode number, access time, creation time, modification time, block size and so on. So this structure will be useful to pull out all the details about the file and especially inode number. This structure will be as follows,

```
struct inode {
    __fs16 ui_mode;
    __fs16 ui_nlink;
    union {
        struct {
```
2.4.2. Struct inode *ufs_iget.
This structure will return the inode value for accessing the file by the process. So, this structure can be used to compare the inode values of the files in secure mode and inode table and if inode number is present then returning inode value can be blocked.

3. Experimental Results
At initial stage, the experimental results are produced based on the basic shell scripts and basic level implementation on kernel is done but yet need to enhance the implementation on kernel level. The experimental results on shell scripts is success to understand the resources used by the process and its permissions but failed to figure out whether the process is legitimate or not. But, kernel level implementations are even more feasible in finding non-legitimate processes and securing the file is successfully done in kernel level.

3.1. Shell script level Outputs

![Image showing shell script outputs](image)

**Figure 3.** Implementation results which shows the list of processes and it’s permissions on File.
The figure 2 clearly explains that our script “test2” determines the process permission of the file “a.out” with the help of user-file permissions.

3.2 Application Level Outputs:

![Figure 4](image)

**Figure 4.** Start-up page of custom application which has been built.
Figure 5. Selecting the file to be secured from the malicious processes.

Figure 6. –rwxrwxrwx Clearly explains that all the users have read, write
and execute permission on the file called fork.c

Figure 7. Even though the processes has access on that file, still the file couldn’t be opened hence it’s a Secured File
Figure 8. List of Inode number of the files which has to be secured

The Files can be secured with the help of inode number of that file so in order to secure the file we need to get the inode number of the file so once if we select the file and confirm to secure it then the inode number of that file will be saved in a file as list as given in fig 5.4. Others are some of the screenshots of the application which we created to secure our files with help of inode numbers of those files and also to perform process scan.
Figure 9. The process scan result which helped to find the process is legitimate and the file that created the process.

Figure 9 shows that the process called python3 originates from the file called myapp.py and then myapp.py is been analyzed and the result says that the myapp.py file is legitimate and trustable. So, it necessary to check whether malicious process and its file getting caught or not. So, again checked for all the suspicious process and finally found a suspicious process and its file. The process name is EICAR and the file that initiates this process is EICAR.COM as shown in the Figure 10.

Other than EICAR, few more malicious processes are also been found. So, this behavior based malicious process detection is successful to find any malicious process that runs on the system without the End-user’s knowledge.
4. Background and Related Work

The monitoring will collect the information of the processes which are initially created at the starting and will put into the white list. After that if any new process going to get created and if it is found that the process is in the process white list, it will be normally created and searched into the active process queue waiting to be scheduled; However if the process is not present in the process white list, then immediately report will be generated and given to the user through the terminal (including the process ID, the process name, process user ID, process group ID, parent process and so on) and wait for the user to authenticate the process [1]. The drawback of this paper is processes that are already been in whitelist may be malicious process.

TOMOYO module will be acting like an operation watchdog, restricts each process to only the behaviors and resources, set by the administrator. So, no processes can run without administrator’s permission [2]. The drawback of this paper is the need of administrator’s help to secure the file and also setting limit of every process is impossible.

The Linux kernel manages system processes with the help of process descriptor task_struct structure, which relates closely kernel process view. From the value of ESP, we can get the content of thread_info [3].

The task structure in the kernel of Linux operating system will contain information about every action performed and resources used by a process. The study thus far has identified 15 features from

*Figure 10. The process scan result which helped to find the malicious process and the file that created the process.*
118 features which can effectively distinguish between genuine and malicious processes [4]. The drawback of this paper is some malicious processes may not have the listed feature and this fact allows malicious processes to be undetectable.

There are some programs which are critical and will maintain the security of the system and accomplish user’s business logic. Detection module will check if these critical tasks exist in the monitored virtual machine’s running process queue reconstructed by the semantic reconstruction module. If one or more critical processes are missing then it means that the monitored guest OS is affected by malware and these critical programs are terminated [5]. The drawback of this paper is some alerts may be false negative.

All the process running on a Linux system will be executed on behalf of a given User. Thus, the process can do only what this User is allowed to do. So, the process may access the Files only if user has access on the file [10].

A file which is owned by the root Linux user, part of the root Linux group, and that the owner (root), group (root) and other users can read it (the r bit is set). Only if the owner (root Linux user) has write access to the file (the w bit is set). So access-wise, a process that runs as a non-root user will be able to read the file, but not to write [11].

5. Conclusion and Future Work

Usually the concept of anti-virus or any other security mechanisms will work basically based on two mechanisms such as signature based malware detection and anomaly based malware detection. The problems with signature based detection is it can’t find zero day exploits but even though anomaly based detection can find zero days, the chances of false positive is more and it can’t find the zero day exploit which doesn’t create any anomaly activity.

So, If this idea of process based detection is been implemented then it will be a great contribution to the system security. The result where each and every end users can have enough knowledge to secure his/her private information asserts in his/her system.

The application which we built is for Linux but this application can be built in Windows based system as well. This application can be built as default Linux application which runs as a Kernel application. All processes in the system can be hashed in order to find the process injection based malware.

6. References

[1] Kehi W, Yueguang G, Wei C, Tong Z 2012 Research and implementation of the Linux process real-time monitoring technology. In: Fourth IEEE International Conference on Computational and Information Sciences (ICCIS). IEEE, Beijing, pp 1046-1049. doi 10.1109/ICCIS.2012.342
[2] Shukla H., Singh V, Choi YH, Kwon J, Hahm CH 2013 Enhance OS security by restricting privileges of vulnerable application. In: IEEE 2nd Global Conference on Consumer Electronics (GCCE). IEEE, Tokyo, pp. 207-211. doi 10.1109/GCCE.2013.6664800
[3] Kong WL., Zeng, GY Zhou DY 2015 Research on Technology of Process Hiding based on VMM. In: International Conference on Computer Science and Applications (CSA), IEEE, Wuhan, pp. 339-344. doi:10.1109/CSA.2015.26
[4] Gohel HA, Upadhyay H, Pons, A, Lagos LE, 2017 Design of Virtualization Framework to Detect Cyber Threats in Linux Environment. In: IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud).IEEE, New York pp. 316-320. doi:10.1109/CSCloud.2017.18
[5] Hua Q, Zhang Y, 2015.Detecting malware and rootkit via memory forensics. In:
International Conference on Computer Science and Mechanical Automation (CSMA), IEEE, Hangzhou, pp. 92-96, doi: 10.1109/CSMA.2015.25

[6] Gaoshou Zhai, Yaodong Li 2008. Analysis and Study of Security Mechanisms inside Linux Kernel. In: International Conference on Security Technology, IEEE, Hainan Island, pp. 58 – 61. doi: 10.1109/SecTech.2008.17

[7] Ligh MH, Case A, Levy J, Walters A, 2014. The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory. John Wiley & Sons

[8] Upadhyay H, Gohel HA, 2017. Design of Advanced Cyber Threat Analysis Framework for Memory Forensics. In: International Journal of Innovative Research in Computer and Communication Engineering, ISSN (Online): 2320-9801, ISSN (Print): 2320-9798, Vol 5, Special Issue 2, pp. 132-137.

[9] Raghavan S, 2013. Digital forensic research in current state of the art. In: CSI Transactions on ICT, 1(1), pp. 91-114.

[10] Stephen Smalley, Chris Vance, Wayne Salamon, 2006. Implementing SELinux as a Linux Security Module, NAI Labs Report #01-043.

[11] haifux.org available on http://haifux.org/lectures/84-sil/users-processes-files-and-permissions/users-perms-lec.html

[12] Gentoo linux available on https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_SELinux_controls_file_and_directory_accesses