User-Space Authorization of Creat System Call in Linux

Rohith Krishnan, Hari Narayanan

Abstract--- Authorization is a hot topic for several decades' right from the time people started using information technology. It still continues to be. All the operating systems have some or other ways of authorizing user processes. Linux uses identity-based authorization by default which is not fine-grained. The architecture discussed in this paper provides an additional layer of authorization using security tickets over and above the userid and groupid based authorization. A Secure Daemon which is running in the user space is responsible for this second level of authorization. This new architecture is designed in such a way that all the critical system call invocations like creat are routed to the Secure Daemon which verifies the attached security ticket for authorization. This routing of the creat system call invocation is achieved by the use of wrapper function for C library function creat. This architecture ensures Principle of Least Privilege which is essential to prevent attacks from malicious programs. The processes are executed in a sandboxed environment to protect the system from potential attacks. Since the Secure Daemon is running in the user space itself, this is also portable across different Linux platforms.

Keywords--- Authorization, Principle of Least Privilege, Sandbox, System Calls, Secure Daemon, Wrapper Functions, Security Tickets.

I. INTRODUCTION

Authentication and authorization are parts of any access control mechanism implemented by an operating system. Authentication is the process of confirming the identity provided by an entity. The entity can be a person or a program. In an access control mechanism, access control policy has to be defined. Policies are actually rules. This can be an access control list or capability. When an entity tries to access the resource the access control policy is enforced. Authorization is the process of defining the access policies.

Two commonly used access control mechanisms used in operating systems are Mandatory Access Control (MAC) mechanism and Discretionary Access Control (DAC) mechanism. In MAC a subject can access an object based on some policies. The subject can be a process and the object can be files, ports etc. Here everything is centrally controlled by a policy administrator. An ordinary user cannot change the policies. In the case of DAC, the policies are defined by the subject itself. Here subject can pass the permissions to other subjects. DAC and MAC can coexist in a system but always the MAC policies override DAC policies. In the case of DAC, every object is owned by a subject.

The basic idea of DAC is identity-based authorization. Linux uses DAC traditionally and by default. If a process is being executed it has several identifiers like userid, groupid etc. When the process tries to access a system resource, the kernel verifies whether this processid and groupid have the sufficient permissions to access the resource. If the authorization is successful the process is given a resource handle by the kernel. The process can do the intended operations on the resource using this resource handle. If authorization fails then the process gets an error code. This type of access control is not fine-grained. A process may need only a few privileges to accomplish its task but the process is given all the privileges of the particular userid and the groupid. This violates the Principle of Least Privilege. So a malicious process can exploit its additional privileges and compromise the security of the system.

We have come up with a new architecture which enhances the security of the system. This is an additional layer of authorization over and above the identity-based authorization which is part of the Linux by default. Using this new architecture authorization can be fine-grained and we can enforce the Principle of Least Privilege. A process would not get all the privileges of a particular userid and groupid. It would be given privileges just enough to complete its intended task. The processes are made to run in a sandboxed environment. So a malicious process's ability to do illegal operations are restricted. This is a continuation of our research work. The architecture design [1] and the first implemented version [2] are published previously.

If a process wants to access a system resource like hard disk or network it invokes an API from an API library like glibc. In most of the cases, this API will invoke a system call with the same name. API functions are actually a wrapper function for the system calls. Invoking a system call will result in an interrupt and a context switch from the user mode to kernel mode. The system call number is saved in a register and the parameters are saved in kernel CALL stack so that the kernel code can access them. The system call handler will invoke the system call after authorizing the process based on its proccessid and the groupid.

The proposed architecture implements a wrapper function for the glibc APIs. When the API is invoked the wrapper function checks whether the process is running in the normal mode or secure mode. If the process is running in the secure mode, the function invocation is redirected to a Secure Daemon (Secd) which does an additional authorization using the security ticket provided by the process. If the authorization is successful, Secd will invoke the system call and the resource handle is returned to the wrapper function. The wrapper function will return the resource handle back to the original process.

Manuscript received February 01, 2019

Rohith Krishnan, Center for Cyber Security Systems and Networks, Amritapuri Campus, Amrita Vishwa Vidyapeetham, Kollam, Kerala, India. (e-mail: rohithkrishnan@am.students.amrita.edu)

Hari Narayanan, Center for Cyber Security Systems and Networks, Amritapuri Campus, Amrita Vishwa Vidyapeetham, Kollam, Kerala, India. (e-mail: hari@am.amrita.edu)

The process can do the required operations on the resource using the resource handle. A user process cannot invoke sensitive system calls directly as is done in the normal mode. Redirection of the API invocation and return of the resource handle is done using Unix Domain Socket (UDS). The extra latency incurred is worth due to the enhanced security it provides. The research in this paper is based on the glibc system call creat.

The paper is further organized as given below. Section II presents the existing Linux architecture. Section III presents the proposed architecture. Section IV describes the modified LibC. Section V presents the Secure Daemon. Section VI presents results and analysis. Section VII presents the conclusion and section VIII describes the related work.

II. EXISTING LINUX ARCHITECTURE

Figure 1 depicts the authorization mechanism in conventional Linux.

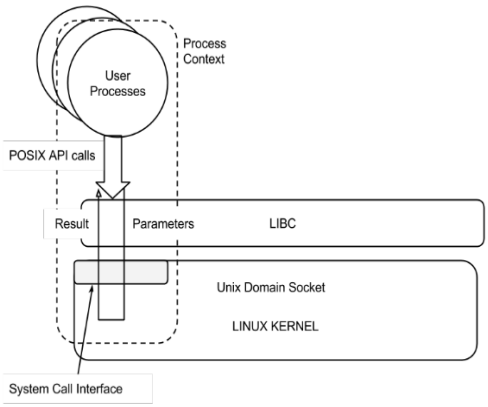

Fig. 1: Traditional Linux Architecture

Whenever a user process want an access to a particular system resource, the function defined in the LibC will be invoked. The request gets converted to a system call by LibC. There is a system call dispatcher at the interface that will invoke the system call handler which then executes the original system call. Theauthorization process is done by the system call handler by checking the effective userid and effective groupid. If the process is authorized to access the resource, theresource handle is returned to LibC and sends it back to the user function that invoked the LibC API.

III. PROPOSED ARCHITECTURE

The proposed architecture is presented in figure 2.

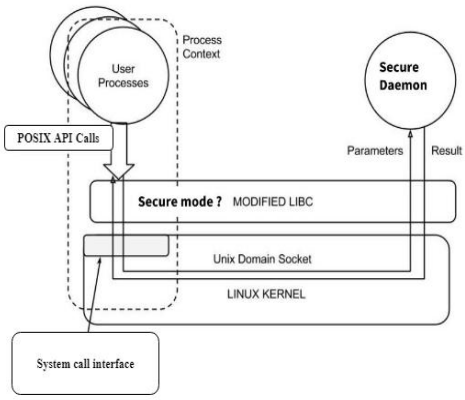

Fig. 2: Proposed Architecture

A process gets executed in either of the two modes (1) Normal mode and (2) Secure mode. The system will work like the traditional Linux system in Normal mode. Whereas in the capability mode, all the user invocations get trapped before invoking the actual function call in LibC and gets diverted to a Secure Daemon. This is done by modifying the LibC by writing a wrapper function for that function call. A wrapper is a function call written with the same name as the function. When a process invokes creat () operation, in Secure mode it will invoke the wrapper for the create operation and diverts the call to the Secure Daemon. Secure Daemon (Secd) is a background running process which runs as the system boots up. In the Secure mode, all the user invocations are redirected to the Secd via an IPC mechanism known as Unix Domain Socket (UDS), rather than getting converted into a system call. The Secd does the authorization, and if the process is authorized, Secd will convert the function call into a system call. That will result in switching to the kernel mode from the user mode. The Daemon does the job on behalf of the user process and returns the resource handle back to the calling function of user process via modified LibC. All the Inter-Process Communications are achieved by UDS. Once the process enters into the secure mode there is no switching back to normal mode, also all its child processes would run in the secure mode.

IV. MODIFIED LIBC

When a user process invokes an operation like creat(), the shared library that has the wrapper for the creat gets executed. Wrapper is a function intended to check the mode of operation of the user process and it diverts the call to secure daemon for authorization. For this to happen, the wrapper has to be loaded prior to the original LibC function call. For that, a Linux shell environment variable called LD_PRELOAD is being used.

Functions of the wrapper:

- Check if the process is executing in the secure mode
- If it is in secure mode, divert the call to the securedaemon (Secd).

Normal Mode

An unused POSIX capability bit, CAP_AUDIT_CONTROL is used for setting the secmode flag.

If the POSIX capability bit is set, i.e. CAP_AUDIT_CONTROL = 1, the process enters into the Secure mode and the arguments of creat function are passed to Secure Daemon via UDS as shown in figure 3. UDS operates in a way similar to RPCs (Remote Procedure Calls). The major difference with other IPC mechanisms is that, by using UDS file descriptors and process credentials (process id, user id, group id) can be shared among the processes.

On the other hand, if the POSIX capability bit is set as 0, the system will work like traditional Linux system or in Normal mode.

So as to call the original glibc function, a function call called dlsym () which is defined in #include <dlfcn.h>is used. dlsym() is used to obtain the address of the original function.

real_creat = dlsym(RTLD_NEXT, "creat");

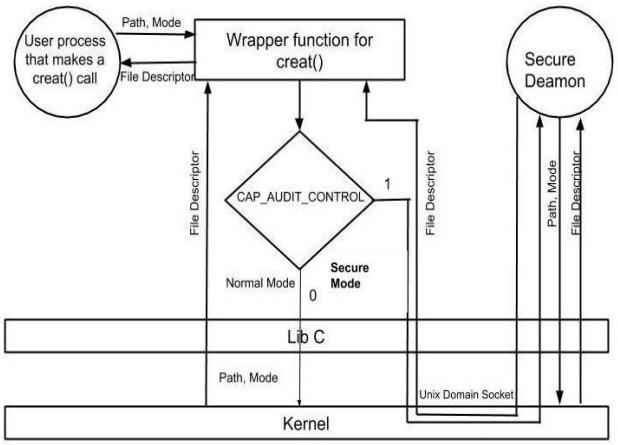

Fig. 3: The Wrapping Mechanism

Secure Mode

In secure mode, Modified LibC will request for a security ticket creation which is the first and foremost step in the communication. LibC will then send the function call i.e. creat (), along with resource specification and operation specification obtained from the Security Descriptor file to the Secure daemon for authorization. Security Descriptor file is an XML file which has the entries like the pathname or resource which is to be accessed, the permissions with which the process access the resource. Security descriptor file is shown as figure 4.

```
<?xml version="1.0"?>
<resource>
   <res id="res111">
      <title>./*</title>
      <permission>rw-r--r--</permission>
   </res>
</resource>
```

Fig. 4: Security Descriptor file

V. SECURE DAEMON (SECD)

The SecD is a user level background running process designed to run autonomously, with little or no user intervention. It runs when the system boots up. SecD is responsible for issuing security tickets and authorization of resource access requests. In order to protect the SecD from any attack, it functions with a unique user id and groupid.

SecD receives the security object creation request from modified LibC through UDS which initiates the communication between the two. Security object creation is a one-time process in the lifetime of a process. The security object has the resource name, permissions with which the process gets executed, UUID and a reference count, which indicates the number of processes accessing the resource. The process which has this security ticket can only enter into the secure mode. After creating the security object, the daemon does a three-step mapping.

Mapping

So as to ensure unforgeability of the security object, three arrays are used such that index of the first array is the security ticket. The three arrays used are per process security array, global security store and global security object array respectively. The Global security object array has the security object. The per process security array point to global security store, which has the index of global security object array as its content. Security objects are immutable, which can be shared by two processes.

A security object has a bunch of name value pairs. For file systems the name would be a file name or a pattern and value would be the permissions like read/write/execute. After this mapping process, the Modified LibC will receive the index of per process security array from the SecD and appends it along with the parameters of creat system call and return it to SecD for authorization. After successful authorization, Modified LibC will receive the file descriptor and returns back to the user process.

Authorization

Rather than permitting the user process to access the resources directly, the proposed architecture does a second level of fine-grained authorization. Fine-grained authorization of function calls is achieved by security tickets, which cannot be forged.

When a creat function call is invoked by the user, the wrapper function for creat() will check whether the process is running in the secure mode or in the normal mode. In secure mode the call gets diverted to Secure Daemon. The wrapper will send a "sec_creat" request for security object creation, followed by the resource specification and operation specification. In order to create a file, permissions of the parent directory have to be considered. The minimum permissions required to create and read that file in a directory is 744.

Daemon obtains the permissions of the parent directory (ambient permission) and checks whether the permissions (operation specification) in security descriptor file is a subset of the ambient permission. If it is a subset, then the daemon will create a security object and returns the index of the per-process security array (sec_id) back to the user process.

Secure Daemon would receive <creat(), sec_id> from the Modified LibC and authorizes the call using the received sec_id. The Secure daemon (Secd) would then create the requested file and returns back the file descriptor to the user process.

VI. RESULTS AND ANALYSIS

Figures 5 shows the output obtained while running the Modified LibC prior to glibc.

It shows that after switching to secure mode, the call get diverted to daemon via UDS. Security object creation and authorization happens at the daemon which is shown in figure 6. After proper authorization, the user intended file i.e. file.txt is created by the daemon on behalf of the user and returns the file descriptor back to the user process.

```
$ LD_PRELOAD=./preload.so ./creat file.txt

Switching the mode
LibC : Wrapping creat()..
File path : file.txt
File mode : 744
Modified LibC: Secure mode... Diverting the request to SecD via
UDS

Security Object Creation
Modified LibC: Sending Security create request : sec_create
Modified LibC: Sending creat request : creat
Modified LibC: Sending resource specification from xml file : ./*
Modified LibC: Sending operation specification from xml file :
rw-r--r--

sec_id : 0
Invocation of CREAT
Modified LibC: Sending the function name and capoid : creat 0
Modified LibC: Sending file path : file.txt
Modified LibC: Sending file path : rw-r--r--
LibC : Got control_message from SecD...!!
LibC : File Descriptor received : 5

fd received is 5
```

Fig. 5: Preloading the wrapper function for creat()

```
$ ./sec_daemon

client() accepts socket_desc from wrapper: 4

Request received : sec_create

Call is found in the array at position 0 sec_create

Resource description : ./*
Resource permission : rw-r--r--

Current working dir: /home/rohith/Downloads/operating-system-
security-project-master

PARENT DIRECTORY is :/operating-system-security-project-master

Ambient Permissions: rwxrwxr-x
Minimum Privilege: rw-r--r--

Security Object creation

Resource name : ./*
Permission : rw-r--r--
UUID : 993592
Reference count : 1

Got the Operation spec .. server back the sec_id 0

sec_id is : 0
First message from Client: creat 0
File to be created: file.txt
Permissions with which file is created: rw-r--r--
CapD : Going to create the file: file.txt !!!!!

File Descriptor(fd) :5
```

Fig. 6: Execution of the Secure Daemon

A real-time comparison of time taken for execution of creat() operation in normal Linux and the proposed system is done as shown in figure 7. It is inferred that there is some additional overhead incurred, which can be neglected due to the enhanced security the system is providing. The graph depicts that the latency in Normal mode of proposed architecture is exactly similar to the latency incurred for the conventional linux. Time latency includes three metrics namely real, user and sys time. The wall clock time evaluated from the time of execution till the end of the execution of a function call is known as real metric. The CPU time spent for the program execution in user mode is called User metric whereas the CPU time spend during execution in kernel space is called the Sys metric.

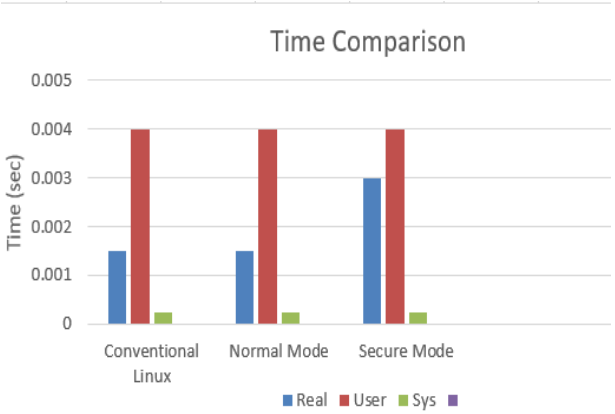

Fig. 7: Latency Measurement

VII. RELATED WORKS

We had come up with an architecture to secure the Linux operating system [1]. The proposed architecture provides fine-grained authorization for a process to perform critical operations based on the principle of least privilege. Then we presented a top-down modular design for the proposed architecture and also implemented the authorization of two critical system calls open and fopen [2].

Security of Linux has been the key point of all security researchers for the past decades. MAC which does the security by enforcing system-wide policies was introduced to improve the security. Michael Wikberg [3] came up with a new architecture SELinux (Security Enhanced Linux) that does security by applying policies. The FLASK architecture is used in SELinux.

Schreuders et al. [4] compared some security models that implements MAC, like SELinux, AppArmor, TOMOYO, and FBAC-LSM. In AppArmor profiles are created for the applications that need to be protected.

System-wide policies had some issues with its complexity and difficulty in use. Ian Goldberg et al. [5] came up with a different approach to system call interposition and capability based authorizations which are used in Janus, a sandboxing system. It acts like a firewall that sits between the application and the operating system regulating which system calls are allowed to pass.

Robert N. M. Watson et al. [6], [7] presented Capsicum, that implements per-process capability sandboxing. Capsicum was introduced in FreeBSD 9 and with a lots of modifications it is now implemented in FreeBSD 10.3. Capsicum now has a daemon process i.e. Casperd that does the sensitive jobs on behalf of the user [8]. William R. Harris et al. [9] introduced a method using safety games to find the sensitive tasks in Capsicum. It also discusses way of capsicumizing the applications.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Another system which implements MAC architecture is TOMOYO [10], which implements automatic policy generation technique. It has a CUI based policy editor that doesn't require any labeling. Tomoyo discard unnecessary privileges of the existing programs voluntarily.

As a solution for changing policies on the flyas per user preference, a system known as Pluggable User space Linux Security Environment (PULSE) [11] was introduced. Unlike existing security architectures, PULSE is more flexible and user friendly due to the use of user-space plugins.

So as to confine the attack vectors decomposition of operating system kernel is being done. In Lightweight capability domains [12] presented by Charles Jacobsen et al. has developed certain mechanisms based on the existing Linux kernel that could further led to OS level kernel decomposition. Adwitiya Mukhopadhyay et al. [13] came up with a firewall that incorporates Netfilter framework and IP tables which communicates the policies with the kernel. Firefox OS [14] uses Linux Kernel in which the process runs in sandboxed environment. So as to enhance security, Sandboxing, content security policies and a permission management system [14] has been proposed.

D. Nidhin et al. [15] proposes a Role Based Encryption scheme using higher dimensional vector decomposition problem (VDP) in which the integrity of cloud provider is not trusted. So as to avoid unauthorized access to the data the data is stored in encrypted manner.

With the vast growing tech trends in IoT, security is the major concern. So as to address this issue I.Ray et al. [16] proposes a framework for Remote Healthcare Monitoring and proposed the use of the NIST Next Generation Access Control (NGAC) framework for specifying and applying access control policies.

VIII. CONCLUSION

We had come up with a top-down modular design of our security architecture earlier in our previous paper [2]. In this paper, we made some progress in the implementation and analysis. SecLib library is expanded by adding some APIs to authorize the creat API of glibc. glibc is modified by creating a wrapper function for the creat system call. When the process running in the secure mode invokes creat API of the glibc, the wrapper function will take over and redirect the invocation to the Secure Daemon (secd) via Unix domain socket. Secd will authorize the operation by checking the security ticket provided by the process. Upon success, Secd performs the operation on behalf of the process and returns the file descriptor back to the wrapper function and back to the initial process. We have performed time latency measurement to show that it is feasible for implementation. We have achieved enhanced security for the system by providing fine-grained authorization. This is at the cost of a marginal degradation in the performance but it is worth. The main drawback in this design is the single point of failure as everything depends on the proper execution of the Secure Daemon (SecD). Our future research is to implement all the modules shown in the top-down design and provide additional authorization for all the sensitive system calls. Also, we would work to protect the system from the single point of failure.

REFERENCES

1. Hari Narayanan, Vivek Radhakrishnan, Shiju Sathyadevan, and Jayaraj Poroor. Architectural design for a secure linux operating system,accepted at International Conference on Wireless Communications Signal Processing and Networking (WISPNET), Chennai, 2017.
2. Vivek Radhakrishnan, Hari Narayanan, and Shiju Sathyadevan.System Call Authorization in Linux by a Secure Daemon, accepted at International Conference on Operating system security (ICOSS'17).
3. Michael Wikberg. Secure computing: Selinux. http://www.tml.tkk./Publications/C/25/papers/Wikberg nal. pdf, 2007
4. Schreuders, Z. Cliffe, Tanya McGill, and Christian Payne. "Empowering end users to confine their own applications: The results of a usability study comparing SELinux, AppArmor, and FBAC-LSM." ACM Transactions on Information and System Security (TISSEC) 14(2)19.2011
5. Garnkel, Tal. "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools." NDSS. Vol. 3. 2003.
6. Robert NM Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway. Capsicum: Practical capabilities for unix. In USENIX Security Symposium, volume 46, 2010.
7. Farley, Benjamin. "Analyzing Capsicum for Usability and Performance." (2010).
8. Pawel Jakub Dawidek and Mariusz Zaborski.Sandboxing with Capsicum,www.usenix.org VOL. 39, NO. 6 DECEMBER 2014
9. William R. Harris∗, Benjamin Farley∗, Somesh Jha∗, Thomas Reps∗†∗Computer Sciences Department; University of Wisconsin–Madison; Madison, WI, USA. Programming for a Capability System via Safety Games 2011.
10. Harada, T. Horie, and K. Tanaka, "Task oriented management obviates your onus on linux," in Linux Conference, vol. 3, 2004.
11. A P Murray and Duncan A Grove. Pulse: A pluggable user-space linux security environment. In Proceedings of the sixth Australasian conference on Information security-Volume 81, pages 19-25. Australian Computer Society, Inc., 2008.
12. Charles Jacobsen, Muktesh Khole, Sarah Spall, Scotty Bauer, and Anton Burtsev. Lightweight capability domains: towards decomposing the linux kernel. ACM SIGOPS Operating Systems Review, 49(2):44-50, 2016.
13. Adwitiya Mukhopadhyay, V Srinidhi Skanda, and CJ Vignesh. An analytical study on the versatility of a linux based firewall from a security perspective. International Journal of Applied Engineering Research, 10(10): 26777-26788, 2015.
14. S. P and K. P. Jevitha. Static analysis of firefox os privileged applications to detect permission policy violations. International Journal of Control Theory and Applications, 9(7):3085-3093, 2016.
15. D. Nidhin, Praveen, I., and Praveen, K., "Role-Based Access Control for Encrypted Data Using Vector Decomposition", in Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015, Volume 2, P. L. Suresh and Panigrahi, K. Bijaya New Delhi: Springer India, 2016, pp. 123–131.
16. I. Ray, Alangot, B., Nair, S., and Dr. Krishnashree Achuthan, "Using Attribute-Based Access Control for Remote Healthcare Monitoring", in 2017 4th International Conference on Software Defined Systems, SDS 2017, 2017, pp. 137-142.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication