A Linux Kernel Module for Locking Down Applications on Linux Clients

Noureldien A. Noureldien
dept. of Computer Science
University of Science and Technology
Khartoum, Sudan
noureldien@hotmail.com

Abubakr A. Abdulgadir
dept. of Computer Engineering
University of Gezira
Madani, Sudan
bakrysalih@gmail.com

***Abstract*—Preventing the installation and execution of unauthorized software should be a high priority for any organization. Allowing users to install and execute unauthorized software can expose an organization to a variety of security risks. In this paper we present a graylisting solution to control application execution on Linux clients using a loadable kernel module. Our developed kernel based solution, Locking Applications on Linux Clients or LALC is a new Linux subsystem which adds a graylisting application lockdown capability to Linux kernel. The restriction policy applied by LALC to specific client is based on the preconfigured security level of the client's group and on the application the client desire to execute or to install. LALC is flexible enough to support the business needs as well as new applications and new versions of existing applications. And it is so secure that no end user can circumvent its configuration.**

Keywords-Application Lockdown; Linux Kernel Module; Restriction Policy; Whitelisting; Blacklisting; Graylisting.

I. INTRODUCTION

The rising number of computer security incidents since 1988 [3][4] suggests that malware is an epidemic.

Malware is referred to by numerous names. Examples include malicious software, malicious code and malcode. Many definitions have been offered to describe malware. For instance, [7] describe a malware instance as a program whose objective is malevolent. Malicious codes defined in [6] as "any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system."

Nowadays, in many organizations, employees can peruse web sites, send and receive email, download software, and install applications whenever they want. On one hand, such openness helps business flow by empowering workers to use information freely; on the other, it can risk the security and integrity of both computers and data as it opens a wide window for malware and malicious attacks.

Often the first defensive step is to run an anti-virus and anti-malware protection software. These programs perform a thorough cleaning of existing virus and malware infections, returning the systems to a relatively stable state. However, they are typically just behind the hacker curve. Computers are vulnerable to newly released viruses or attacks until the malware code is identified and the anti-virus agents are updated on every machine.

Using these methods makes a "zero day attack" almost impossible to prevent using anti-virus software. And due to this failure of anti-malware, organizations take the choice of locking down their entire networking environments.

Locking down a network client can mean a lot of different things. In this paper we refer to a client as being locked down if it is configured in such a way that prevents unauthorized applications from being installed or executed.

It is obvious that locking down clients will stop users from installing or executing an application that contains spyware, a Trojan, a virus, or some other form of malware. This will result in a tremendous security improvement and business continuity.

Locking down client machines can be done using different methods. The problem with many of these methods, however, is that they are either impractical, costly or places a heavy burden on the network administrators.

In this paper, we develop a kernel based solution for Locking Application on Linux Clients (LALC) applying a graylisting approach. LALC uses a central server that controls applications running on clients. The server was configured to define client's security levels and their associate allowable and disallowable applications. Clients are configured to request server permission on executing an application. The server permits or denies client requests by comparing the hash value of the requested application to those pre-stored values. For flexibility and ease of use, the solution provides a Server Configuration Utility for managing clients groups, their security levels and their associate restriction lists.

This paper is organized as follows. In Section II, we revise the basic locking down approaches, and we discuss the design of LALC in Section III. In Section IV we show how we implement and test LALC and we conclude the paper in Section V.

II. LOCKING DOWN APPROCHES

Basically, there are three major approaches for locking down client applications; blacklisting, whitelisting and graylisting.

A. *Blacklisting Approach*

This approach applies the security premise "what is not expressly defined to be prohibited must be allowed". So in this approach only those applications that have been defined to be unwanted, the blacklist, will not be executed, all other applications will be allowed to run. Clearly this approach will not defend against malicious applications not previously identified in the blacklist.

B. *Whitelisting Approach*

This is the reverse approach to blacklisting, it applies the security premise "what is not expressly defined to be allowed must be prohibited". Application whitelisting is emerging as the security technology that gives a true defense-in-depth capability, filling in the gaps that anti-virus was never designed to cover. Application whitelisting is characterized by the ability to identify authorized executables and associated files and to treat as an attack any program or file that is not on the authorized whitelist. Recent advances in application whitelisting, including automatically approving files from trusted sources to reduce administrative overhead or allowing end-users to personalize their endpoint for greater user acceptance, has made application whitelisting an attractive choice.

Application whitelisting is a technique gathering momentum in commercial security systems. Most implement additional access controls within the operating system to stop unauthorized programs from running. Products from companies such as CoreTrace [5], SolidCore [10] and Bit9 [2] all use application whitelists to create a safer working environment.

C. *Graylisting Approach*

This approach combines the previous two approaches; it uses three lists, while, black and a gray. This approach works by focusing on valid whitelisting applications and allow only those applications to run. All the applications in the blacklist are not allowed to run. When an application is not in the white list or in the black list, it will be placed in the gray list for further justification. This approach uses software authentication to reduce the problem of malware and other unwanted software [9].

III. LOCKING APPLICATIONS ON LINUX CLIENTS (LALC)

LALC is a graylisting solution that restricts application execution on network Linux clients. The solution maintains three lists, a white list for applications that are authorized to run, a black list for applications that are solely prohibited and a gray list for applications that are neither white nor black.

LALC deploys client group restriction policy which allow establishment of different client groups that have different security levels. For system flexibility LALC implements three security levels, namely, Lockdown, Block-and-Ask and Monitor. In Lockdown level, only whitelisted applications are allowed to run. In Block-and-Ask a confirmation message for executing the application is sent to the user when the application is gray. In the Monitor level the gray applications are allowed to be executed without user confirmation. In all security levels, the gray applications are added to the gray list for later administrator analyses.

A. *LALC Components*

LALC is a client/server application. On the client side, we build two components, a Loadable Kernel Module (LKM) to intercept client attempts to execute applications, and an Agent program which was designed to calculate the hash value of the desired application file using MD5 algorithm and to communicate with the server. Although the Agent Module employs MD5 algorithm but any other hashing algorithm can be used instead.

On the server side we build a Server program to receive client's requests and to generate responses, and a Server Configuration Utility to allow administrators to manage client groups, security levels and application lists.

1) Client Components: Two components are deployed on each client; the Loadable Kernel Module (LKM) and the Agent.

a) The Loadable Kernel Module (LKM): The LKM is built based on the facts that; a loadable kernel module is a piece of code that can be dynamically loaded or unloaded from the Linux kernel, and once it loaded it becomes a part of the kernel [8]. And Linux kernel dedicates a specific system call, namely execve, to handle client request to the kernel for executing a program file [1].

LKM was designed to intercept client requests on behalf of the original execve, and to invoke the Agent. Based on the return value LKM may or may not allow original execve to handle the client application.

LKM comprises four functions; initialization(), custom_execev(), write() and read().

- Initialization() :When LKM is loaded into the kernel it executes the initialization(). This function redirects client calls from the original execve system call to the custom_execve function inside the LKM. Initialization() performs redirection by replacing the execve address in the kernel table by the address of the custom_execve(), and saving the original execve address. Also the initialization() prepares a communication channel to the Agent process via a /proc file. It creates a /proc file and connect its read/write operations with read() and write() inside the LKM. Also it creates two buffers to be used by LKM other functions, namely, Request Buffer and Response Buffer. Generally, /proc file system is a method used for communication between the kernel and user processes [9]. Fig. 1 shows how LKM initialization function works.

- custom_execve(): The purpose of this function is to replace the original execve system call, and therefore it will be executed whenever a client process desires to execute an application file. It saves the name of the application file to be executed in the Request Buffer and sets a flag to indicate that a request to execute an application file is pending (Request_Pending = 1). After that it wakes up the Agent to handle the pending request, and it renders itself in awaiting state. After custom_execve wakes up by the write(), it reads the Request Buffer and resets the pending flag. Based on the value in the buffer, custom_execve either allows the execution of the application or denies it. On allowing execution custom_execve executes the original execve system call, and on denying, it returns an error code on behalf of the original execve system call. Fig.2 shows how the custom execve function works.

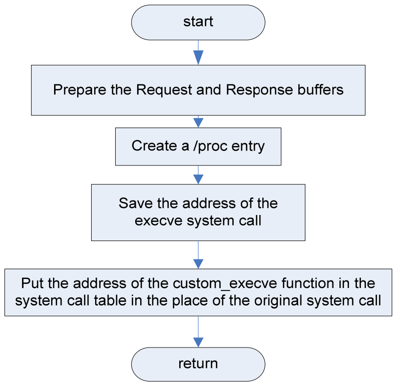
start
Prepare the Request and Response buffers
Create a /proc entry
Save the address of the execve system call
Put the address of the custom_execve function in the system call table in the place of the original system call
return

Figure 1. KLM Initialization Function

- read(): When the Agent tries to read the /proc file this function is executed. It waits until the variable Request_Pending is set. Once the variable is set, it returns the contents of the Request Buffer - which is the application file name- to the Agent module.
- write(): When the Agent tries to write to the /proc file this function is executed. The purpose of write() is to write to Response Buffer the message that the Agent desire to write to the /proc file and then it call upon custom_execve function.

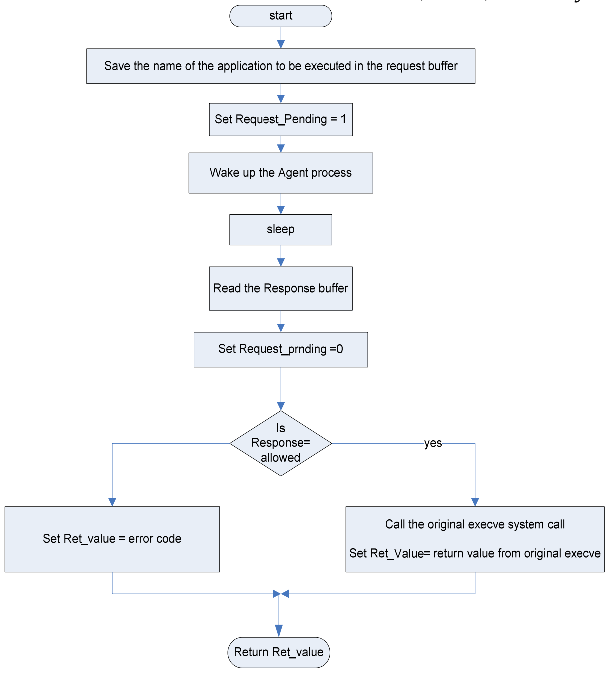
start
Save the name of the application to be executed in the request buffer
Set Request_Pending = 1
Wake up the Agent process
sleep
Read the Response buffer
Set Request_prnding =0
Is Response= allowed
yes
Set Ret_value = error code
Call the original execve system call
Set Ret_Value= return value from original execve
Return Ret_value

Figure 2. LKM custom_execve function

b) The Agent: The Agent program is a user level program that runs in the client machine. Its purpose is to calculate the hash value for the application file content, and to forward it to the server combined with the requesting client hostname and the application file name. Later, the Agent has to forward back the server's response to the LKM custom_execve function through writing to /proc file. Fig.3 shows how Agent works.

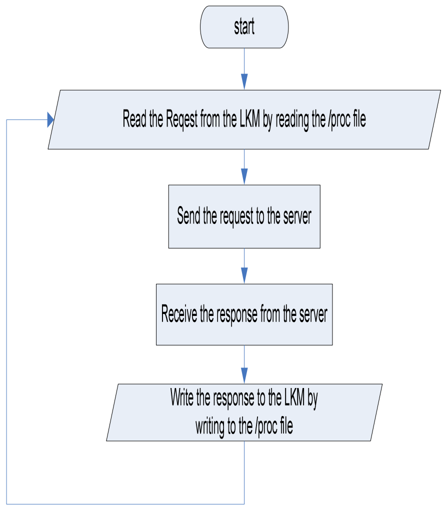
start
Read the Reqest from the LKM by reading the /proc file
Send the request to the server
Receive the response from the server
Write the response to the LKM by writing to the /proc file

Figure 3. Agent program main loop

2) Server Components: Two components are deployed on the server side; the Server program and the Server Configuration Utility.

a) Server Program: The main task of the Server program is to receive client requests via Agent programs and to respond to those requests. The request's hash value and the requested client host name are used by the server to generate the permission response, and it uses the application file name to identify the client in its log file.

The server generates the response by manipulating a database which stores information about client groups, group's security levels and application lists. The server waits for Agents connections on a specific TCP port, and when an Agent connects to that port, the server receives the request and sends back a response. Fig.4 shows how the server works.

b) Server Configuration Utility: The Server Configuration Utility is a friendly graphical user interface for enterprise administrators to configure the Server to enforce enterprise restriction policy. They can use it to manage clients, clients groups, group's security levels and application lists.

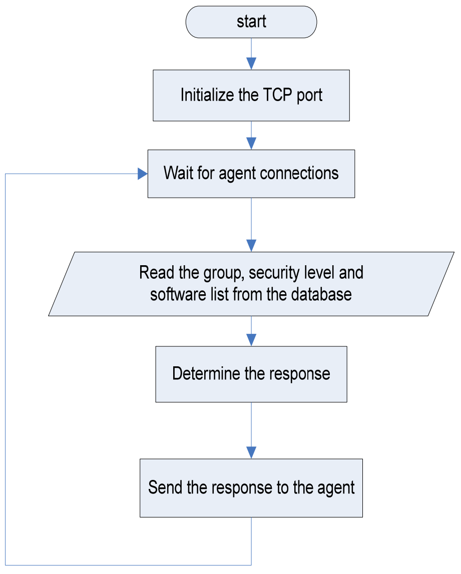
start
Initialize the TCP port
Wait for agent connections
Read the group, security level and software list from the database
Determine the response
Send the response to the agent

Figure 4. Server program loop

The database manipulated by the configuration utility consists of three tables that stores information about clients, client groups, and restriction rules.

The clients table contains information about each client, which includes; the client host name and its corresponding group ID. The client groups table is where group information is stored, which includes; group ID, group-name and the group security level. The restriction rules table stores information about rules applied to each group. A rule specifies the applied list (white or black) to a specific application for a particular group.

IV. IMPLEMENTATION AND TESTING

A. Implementation

Many tools have been used to implement the system. Open source tools have been chosen for implementation. Linux ubuntu 7.04 have been chosen as an operating system for client and server machines. The LKM is written in C language. The Agent, Server and the Server Configuration Utility are written in C++ with Qt4 library. Qt is a library that helps in building GUI C++ programs. The database management system used was SQLite. SQLite is a self-contained, serverless SQL database engine. The hashlib++ library was used to generate the hash of executable files in the agent program.

B. Testing

To test LALC, LKM and the Agent program have been compiled in the client side. A shell script has been written to load the LKM and to run the Agent at startup. When the client machine comes up the LKM and the Agent are ready.

The Server and the Server Configuration Utility have been compiled in the server machine and the Server was started. Groups have been added using the Server Configuration Utility and clients have been added to each group. The lock-down security level has been chosen for the group and applications have been added to the whitelist.

We test the system by attempting to launch two programs form the client machine, one is a white listed and the other is not. The system performs exactly as expected; the whitelisted program is executed while the other one is prohibited.

V. CONCLUSIONS

LALC brings an easy-to-use, kernel integrated solution for locking applications on Linux clients. Its simplicity makes extending it fairly easy, while its integration into Linux kernel allows it to improve Linux security features that support enterprise needs.

REFERENCES

[1] Andrew S. Tanenbaum, Modern Operating Systems, Prentice hall, 2nd ed , 2001.
[2] Bit9 global software registry (website) (April 2010).
[3] Bit9 global software registry (website) (April 2010). URL http://www.bit9.com/products/gsr.php
[4] CERT/CC, Carnegie Mellon University. http: // www.cert.org/ present/cert-overview-trends/ module-4. pdf , May 2003.
[5] CoreTrace: Application Whitelisting For Enterprise Endpoint Control (Website) (April 2010). URL http://www.coretrace.com/
[6] G. McGraw and G. Morrisett. Attacking malicious code: A report to the infosec research council. IEEE Software, 17(5):33–44, 2000.
[7] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, "Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy," pp 32–46, 2005.
[8] Peter Jay Salzman, Ori Pomerantz, "The Linux Kernel Module Programming Guide", ver 2.4.0, 2001.
[9] Robin Bloor, Partner, "Antivirus is Dead", Hurwitz & Associates, 2006
[10] Solidcore (Website) (April 2010). URL http://www.solidcore.com