Mechanisms for Attack Protection on a Prevention Framework

Joaquín García, Sergio Castillo, Guillermo Navarro, and Joan Borrell

Dept. of Information and Communications Engineering,
Autonomous University of Barcelona,
Edifici Q, 08193 Bellaterra (Spain)
Email: {jgarcia,scastillo,gnavarro,jborrell}@ccd.uab.es

Abstract

Current research in Intrusion Detection Systems *(IDSs), targeted towards preventing computer attacks, is mainly focused on improving detection and reaction mechanisms, without preserving the protection of the system itself. This way, if an attacker compromises the security of the detection system, she may be able to disarm the detection or reaction mechanisms, as well as delete log entries that may reveal her actions. Given this scenario, we introduce in this paper the use of an access control mechanism, embedded into the operating system's kernel, to handle the protection of the system itself once it has been compromised by an attacker.*

1 Introduction

Network attacks often benefit from distributed and coordinated techniques, since it opens the possibility to perform more complex tasks, such as distributed denial of service or coordinated port scans. These techniques are also useful to make their detection more difficult and, normally, these attacks will not be detected by exclusively considering information from isolated sources of the network. Likewise, Network Intrusion Detection and Response Systems also benefit from a distributed implementation. Different components of the system may look for different attack evidences, in order to detect this new kind of attacks.

We are currently working on a decentralized approach for achieving this distribution. Our solution uses a tuple space to communicate the different components within and with each other. Middle-level analyzers place alerts into a distributed tuple space, and higher-level analyzers consume those tuples to perform a detection and reaction process [3].

One of our main objectives is to obtain a system able to fulfill attack protection to its components. This way, the system itself must maintain acceptable, though possibly degraded, service despite attacks in parts of the system, be them at network, application or system level. In order to achieve this goal, we started out doing research on protection mechanisms to handle the security and strength of our prevention system's components. As a result of our current work, we present in this paper the development of a protection module integrated into the kernel of our research prototype's operating system, *GNU/Linux*, and implemented over the *Linux Security Modules* (LSM) framework [9].

The protection mechanism consists of building a complementary kernel access control scheme, to handle the protection of the system itself once it has been compromised by an attacker. To do this, it intercepts and cancels unlawful system calls launched by the attacker. Thus, even if the attacker gains administrator permissions, she will not achieve her purpose. This security enhancement is solved without having to recompile the kernel, and with a high degree of flexibility and portability when compared to other proposals for the *GNU/Linux* kernel, such as [5] and [6].

1.1 Paper Organization

The rest of this paper is organized as follows. Section 2 shows a brief overview of our attack prevention system, as well as the main components to protect it through the protection scheme proposed in Section 3. We also present in this section a first implementation of such mechanism on our research prototype, through the *Linux Security Modules* (LSM) framework, as a kernel based access control mechanism. Finally, Section 4 closes the paper with a list of conclusions and future work.

2 System Overview

The main purpose of our prevention system is to detect and react to coordinated or distributed attacks. By means of a set of cooperative entities which are lodged inside the network, the system avoids the use of network resources to perform coordinated attacks against third party networks. The aim of this system is not only to detect incoming attacks against these entities, but also to detect when these nodes are the source of one of the different steps of a coordinated attack to avoid it.

Our approach is based on gathering and correlating information held by multiple sources. We use a decentralized scheme based on message passing to share alerts in a secure communication infrastructure. The information exchange between peers is intended to manage a more complete view of the whole system. Once achieved, one can detect and react on the different actions of the corresponding attack [3].

Each node of the architecture is made up of a set of analyzers (with their respective detection units or sensors), a set of alert managers (to perform alert processing and manipulation functions), and a set of local reaction units (or effectors). These components, and the interactions between them, are briefly described below:

Analyzers – They process, in user space, the information gathered by associated sensors, implemented as operating system's kernel modules, to infer possible alerts. Their task is to identify occurrences which are relevant for the execution of the different steps of an attack and pass this information to an alert correlation manager. The interesting occurrences are local alerts. Each local alert is detected in a sensor's input stream and exchanged to the set of managers through the use of a publish/subscribe system.

On the other hand, the communication between sensors and analyzers is solved via *netlink sockets*, a Linux specific mechanism that allows us to perform communication between kernel modules and user space processes via the well known primitives from the socket treatment.

Managers – In order to facilitate the management of alerts, our architecture provides a set of clients to manage both both local alerts (i.e., messages provided by the node's analyzers) and external messages (i.e., the information received from other collaborating nodes). Just like for the communication between the analyzer and the correlation manager, the communication between managers is also implemented through the use of a publish/subscribe system.

3 Protection Mechanisms

As described out in [3], the entities of our prevention platform cooperate to detect if the resources, where they are lodged, are taking an active part of a coordinate attack. As it happens with any other traditional detection system, if an attacker is able to manipulate the processes associated to each node, she could bypass the detection mechanisms. Thus, the intruder could make its way to hide the local part of the attack from the node. This problem leads to the need for introducing a protection mechanism on the different components of each node, keeping with their protection and mitigating or even eliminating any attempt to attack or compromise the platform and its operation. This way, even if an attacker compromises the security of the system, she would not be able to disarm the detection and reaction mechanisms.

Given the inherent characteristics of our prevention platform's design, and according to [5], we consider two possible protection mechanisms: the *auto-protection* carried by each node's elements, and the protection of the elements carried by the operating system's kernel. In the first case, each component is responsible for its own protection, using mechanisms such hiding its processes, or cryptographic techniques associated to the system logs. Most of the current proposals in this field, e.g. [2, 7], are inefficient against some attacks. For example, when the intruder is in a privileged position, she may interact without restriction with the components through the underlying operating system.

Therefore, the cancellation of processes associated with the detection system, or the deletion of logs, show the problem of these methodologies. The problem relies in two facts: the existence of privileged users in most of the current operating systems, that can freely interact with the system; and the delegation of part of the protection to the operating system access control mechanism, which does not consider that an attacker could gain privileged user permissions from a bug or security failure.

In the second case, the kernel of the operating system provides the proper protection mechanisms, detached from the detection and reaction system. Protection is achieved by incorporating an access control mechanism into the kernel system calls. This way, one may allow or deny a system call based on several criteria such as the identifier of the process making the call, parameters of the call, etc. The kernel's access control allows to eliminate the notion of trust associated to privileged users, delegating the authorization for the execution of a given system call to the internal access control mechanisms. In addition, and contrary to the previous *auto-protection* mechanisms, it provides a unified solution, avoiding the implementation of different specific mechanisms for each component.

3.1 Proposed Scheme

In order to protect the components of our platform we propose an access control mechanism integrated into the kernel of the operating system. This way, even if an attacker gains administrator permissions, she will not be able to generate actions attempting on the node. Each unlawful system call to the components will be intercepted and cancelled by the access control. This methodology also allows us to provide a second level of protection. The mechanism provided by the kernel and the modularity based on components allows to enforce the compartimentalization principle [8]. This principle is based in the segmentation of a system, so several components can be protected independently one from another. This ensures that even if one of the components is compromised, the rest of them can operate in a trusted way.

In our case, several components from the node can be executed as processes. By specifying the proper permission based on the process ID, we can limit the interaction between elements of the node. If an intruder takes control of a process associated to a given component (through a buffer overflow, for example), she will be limited to make the system call for this given process. In our proposal the compartimentalization principle is used as follows. In each node we assume that the execution of the analyzers is isolated from the cooperation manager. The protection at kernel level avoids that potentially dangerous system calls (such as *killing* a process) could be produced from one component against another one.

Even so, it is not always possible to achieve a complete independence between the components. There is a need to determine which system calls may be considered as a threat when launched against an element from the node. This requires a meticulous study of each one of the system calls provided by the kernel, and how can they be misused. On the other hand, we have to define the access control rules for each one of these system calls. Despite its complexity, we consider the following three protection levels to classify the system calls: Critical process protection (e.g. execution of a new application already in memory), Communication mechanisms protection (e.g. messages interchanged between the node elements), and Protection of files associated to the components (e.g. configuration or log files).

3.2 Implementation

In this section we outline a first implementation of our protection scheme. In accordance with Section 3.1, it consists of a kernel based access control mechanism, and its development has been done over the *Linux Security Modules* (LSM) framework for *GNU/Linux* systems [9].

The LSM framework does not consist of a single specific access control mechanism; instead it provides a generic framework, which can accommodate several approaches. There are several hooks (i.e. interception points) across the kernel that can be used to implement different access control strategies. Such hooks are: *Task hooks, Program Loading Hooks, Filesystems Hooks* and *Network hooks*.

These LSM hooks, can be used to provide protection at the three levels commented in the previous section. Furthermore, LSM adds a set of benefits to our implementation. First, it introduces a minimum load to the system when comparing it to kernels without LSM, and does not interfere with the detection and reaction processes. In the second place, the access control mechanism can be composed in the system as a module, without having to recompile the kernel. And third, it provides a high degree of flexibility and portability to our implementation when compared to other proposals for the Linux kernel, such as [5] and [6], where the implementation requires the modification of some features of the original kernel 2.6.x.

The LSM interface provides an abstraction, which allows the modules to mediate between the users and the internal objects from the operating system kernel. To this effect, after accessing the internal object, the hook calls the function provided by the module and which will be responsible to allow or deny the access. There, for example, a module registers the function to make a check over the *inodes* of the filesystem.

At the same time, LSM allows to keep the discretionary access control (DAC) provided by Linux by standing between the discretionary control and the object itself. This way, if a user does not have permissions in relation to a given file, the DAC of the operating system will not allow the access and no call to the function registered by the LSM will be made. This architecture reduces the load of the system when compared to an access control check centralized in the operating system call interface, which always gets used for all the system calls.

The node components will be allowed to make operations only permitted to the system administrator (such as packet filtering, process or application cancellation, etc.). This implies that the system processes associated to the components will be executed by the root user. On the contrary, if we associate the processes to a non privileged user, the discretionary access control of Linux will not allow the execution of some specific calls.

The internal access control mechanisms at the kernel is based in the process identifier (PID) that makes the system call, which will be associated to a specific component. Each function registered by an LSM module, determines which component is making the call from the PID of the associated process. It then, applies the access control constraints taking also into account the parameters of the system call. So, for example, a given component can access its own configuration files but not configuration files from other components.

An important issue in the implementation is the administration of the access control mechanisms and the management of each one of the nodes. As described in the previous section, the administrators should not be able to throw a system call, which may suppose a threat to the node.

This prevents an intruder to do any harm to the node even if she could scale its privileges to the administrator ones. This contrasts with the administration of the node, if an administrator cannot interact with the components of the node, she will not be able to carry on any management or configuration process and activities.

To solve this problem, we have introduced a temporal authentication process based on a cryptographic USB token [1]. While the device is connected to the system, the administrator will be able to hold the indispensable privileges to manipulate the node. When the device is retired, the access control enforcement will come to its normal operation.

4 Conclusions

In this paper we have presented a protection mechanism specially suited for a distributed prevention and reaction system. The distributed system is made up of several components such as sensors, analyzers, managers, etc. that may be distributed in a network. We provide a solution for the protection of the components by making use of the *Linux Security Modules* (LSM) framework in the Linux kernel.

The proposed mechanism works by providing and enforcing access control rules at system calls. It is based on a protection module integrated into the operating system's kernel, providing a high degree of modularity and independence between components.

Our scheme offers a good degree of transparency to the administrator in charge, since the access control is integrated inside the kernel of the operating system, and it does not interfere directly with user space's processes. At the same time, a complementary token based authentication for administration purposes, provides a transparent and secure management of the platform.

As future work we are considering to continue our study about attack and intrusion tolerant mechanisms, to address the security of our proposed architecture from a wider and more global point of view.

Acknowledgments

Part of this work has been founded by the Spanish Government Commission CICYT, through its grant TIC2001-0633-C03-01, and the Catalan Government Department DURSI, with its grant 2003FI-126.

References

[1] Aladdin Knowledge Systems. eToken – USB Token Authentication Device. `http://www.aladdin.com/etoken/`, 2005.

[2] S. Bhattacharya and N. Ye. Design of robust, survivable intrusion detection agent. In *1st Asia-Pacific Conference on Intelligent Agent Technology*, Hong Kong, December 1999.

[3] J. García, F. Autrel, J. Borrell, S. Castillo, F. Cuppens, and G. Navarro. Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation. In *6th International Conference on Information and Communications Security*, October 2004.

[4] L. McVoy, and C. Staelin. LMbench – Tools for Performance Analysis `http://www.bitmover.com/lmbench/`, 1998.

[5] T. Onabuta, T. Inoue, and M. Asaka. A Protection Mechanism for an Intrusion Detection System Based on Mandatory Access Control. In *13th Annual Computer Security Incident Handling Conference*), Toulouse, France, June 2001.

[6] A. Ott. The Role Compatibility Security Model. In *7th Nordic Workshop on Secure IT Systems*, Karlstad, Sweden, November 2002.

[7] B. Schneier, and J. Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In *7th USENIX Security Symposium Proceedings*, San Antonio, Texas, January 1998.

[8] J. Viega, and G. McGraw. *Building Secure Software - How to Avoid Security Problems the Right Way*. Addison-Wesley, September 2002.

[9] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *11th USENIX Security Symposium*, San Francisco, California, August 2002.