DESIGN OF INTRUSION DETECTION SYSTEM AT USER LEVEL WITH SYSTEM-CALL INTERPOSING

Toshihiro TABATA and Kouichi SAKURAI
Faculty of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

Keywords: Dynamic Linker, Intrusion detection system, Library function call, ELF

Abstract: As computers have become widely used, software vulnerability is now one of the most serious security threats. In particular, viruses and worms that use buffer overflow vulnerabilities are serious threats to computers. Therefore, techniques to detect the execution of malicious code are required when taking measures to prevent intrusion using such software vulnerabilities. An intrusion detection system is an example of such a defence mechanism against such attacks. The improvement in both false positive and false negative ratios, together with reduction of overhead are the problems to be overcome in an intrusion detection system. This paper presents the design of a user level intrusion detection system. This system can monitor the execution of target programs at both user and kernel levels. The access control function is divided between user and kernel. Access rights may also be checked with appropriate timing and with low overhead.

1 INTRODUCTION

Improvement in the performance of computers has lead to their widespread use. Computers connections to networks have also increased with widespread use of the Internet. In addition, with the spread of computers and the Internet the numbers of reports of software vulnerabilities increase year by year. In particular, viruses and worms using buffer overflow vulnerabilities pose serious threats to computers. For example, viruses and worms spread rapidly through the Internet and, as a result, many computers will be seriously damaged. Therefore, various defense mechanisms against such malicious programs have been actively studied.

Most current software possesses potential vulnerabilities that are represented by buffer overflow vulnerabilities. For example, much vulnerability has been found in most of open source software. Therefore, in order to prevent intrusion using such software vulnerabilities, techniques that can detect the execution of malicious code are required.

Intrusion detection systems (IDS) represent an example of such techniques. IDS monitor the behavior of programs and detect malicious behavior such as the abuse of software vulnerabilities. There are two types of intrusion detection systems. An anomaly detection system (Sekar et al., 2001) is based on a database containing knowledge of the normal behavior of the system being monitored. However, it is difficult to generate such knowledge of normal behavior for a system. One of the advantages, however, is that system does not require a database of attack signatures that must be kept up-to-date. One of the disadvantages is that the false positive ratio is significant. It alarms on detection of an unusual event. By contrast, a misuse detection system is based on the database of previously known attack signatures. One advantage is that the alarms contain diagnostic information about the cause. One of the disadvantages is that system cannot detect new attacks that are not contained in the database. In intrusion detection systems, the improvement in both false positive and false negative ratios, together with reduction of overhead, is the problems. As a consequence, monitoring the execution of the target program with appropriate timing and the reduction in verification time both are required to solve such problems.

Curry (Curry, 1994) developed a set of tools named Shared Library Interposer (SLI) that works for dynamic libraries. SLI has the loader resolve addresses as a wrapper of the library function. After an address resolution, SLI calls the real library function. One of the advantages is that an accurate trace of call se-

Tabata T. and Sakurai K. (2004).
DESIGN OF INTRUSION DETECTION SYSTEM AT USER LEVEL WITH SYSTEM-CALL INTERPOSING.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 263-268
DOI: 10.5220/0001397602630268

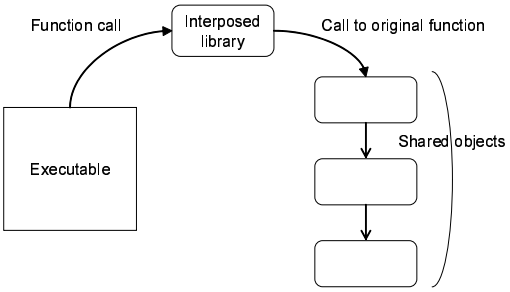

Figure 1: Interposed library.

quences can be logged.

Kuperman et al. (Kuperman and Spafford, 1998) applied Curry's technique to IDS for detecting buffer overflow. Figure 1 shows a sequence when a library call is encountered. The IDS acquires audit data at user level using an interposed library. Every library function is thus intercepted. The interposed library can acquire audit data at the user level without needing to modify either application or system libraries. The authors show the cost of audit on their approach is low, and report that this approach is suitable for buffer overflow detection.

However, in their system there are two problems. The first problem is that malicious attackers can easily bypass the interposed library, because only the library function is checked. If the attacker calls any system-call directly, the system-call will not be checked by their system. The secondly, every library function is checked every time. As a result, the interposed library adds an overhead to every library function.

This paper discusses an intrusion detection system with interposed system-calls and a user level access control mechanism using a dynamic linker. This intrusion detection system resolves the first problem described above. The system is based on the access control mechanism at the user level. It operates in conjunction with the interposing of system-calls. As a result, our intrusion detection system can detect system-calls that bypass the library interposition.

The proposed access control mechanism can resolve the second problem described above. The mechanism monitors the execution of target programs at appropriate timing using dynamic linker. Dynamic linking is employed, such that a program and a particular library referenced by the program are not combined together by the linker at link-time. In the proposed mechanism, a dynamic linker resolves a reference to call the access control module. The module runs at the user level and decides to grant or deny the library function after the check of access rights. If the module denies the access, the library function is not called. In addition, the checkpoint of library functions can be selected arbitrarily. Important library functions that call system-calls, can be controlled by the mechanism with a little overhead. The reference of non-important routines that do not call system-calls such as math functions, are resolved to call the module. As a result, the library functions are called directly without overhead. Generally, library functions that cause security problems invoke system-calls. Therefore, it is believed that this approach is reasonable.

2 RELATED WORK

IDS are an example of a defense mechanism against attacks that abuse software vulnerabilities (Wagner and Dean, 2001) (Hofmeyr et al., 1998) (Wagner and Soto, 2002). IDS observe program execution and detect malicious behavior of the program. As mentioned previously, because an anomaly detection system is based on a normal behavior database, the disadvantages is that the false positive ratio is significant. Therefore, main issue of intrusion detection systems is reduction of overhead and false positive and false negative ratios.

Sekar et al. (Sekar et al., 2001) reported an overhead due to the execution of learning and/or detection code of between 3% and 4%. Sekar et al. also reported that the overhead due to system-call interposition is between 100% and 250%. In addition, Ooyama et al. (Oyama et al., 2003) reported that the run-time of httpd increased 47%. These IDS are based on the kernel. By contrast, the IDS proposed in this paper is based on both the user and the kernel. Kuperman et al. reports that the overhead is between 3.2%. Thus, IDS based on user level possess overhead advantages. However, Kuperman reports the overhead of 57.2% in the worst case.

Jain et al. (Jain and Sekar, 2000) presented an approach to developing a user-level infrastructure for system-call interception and extension. This approach requires a process switch to intercept system-calls. As a result, there is a high overhead. The mechanism proposed here does not involve process switching to intercept library functions and system-calls.

3 REQUIREMENT OF INTRUSION DETECTION SYSTEM

3.1 Problem

Most of the existing kernel based IDS detect intrusion using history of system-call sequence. In the IDS, the false positive ratio is a crucial problem. The reduction in the false positive ratio facilitates their introduction

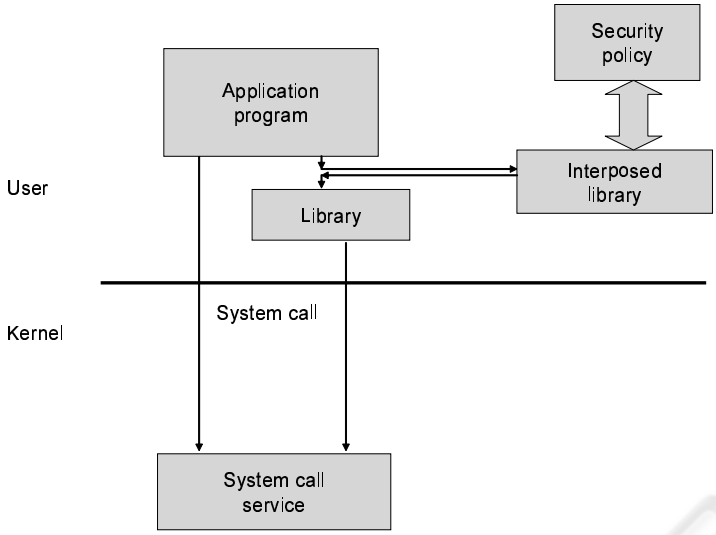

Figure 2: An intrusion detection system based on Curry's approach.

and maintenance. Ooyama et al. (Oyama et al., 2003) presented a mechanism to analyze functions for the reduction in the false positive ratio in the kernel. It achieves the capture of the behavior of the program in detail. However, the cost of analysis of the behavior is not sufficiently low to allow detection in real-time. The reason is as follows. The mechanism searches the stack recursively when a system-call is called. This adds to the cost penalty of the search. In other words, the reason for the time penalty is the need to search the stack to analyze the sequence of library functions.

3.2 Requirement

It is proposed that low overhead detection and fine access control are required for intrusion detection systems and access control mechanisms.

Low overhead is necessary for real-time detection and real-time access control. Fine access control implies that detailed information of program behavior is effective in intrusion detection systems or other access control systems.

4 PROPOSED INTRUSION DETECTION SYSTEM

4.1 Overview

It is proposed that the inspection of library functions is effective and convenient, because functions calls are too small to audit and system-calls are too large to audit appropriately. As mentioned above, existing kernel based systems search a stack to detect library functions recursively. These systems involve searches of all functions and result in an increase in audit cost. These systems also audit library functions in non-real-time, because the searches are delayed until a system-call is made. Therefore, an access control mechanism at the user level is required to audit library functions in real-time.

An access control mechanism at user level is proposed using a dynamic linker. The dynamic linker resolves a reference at user level when a function is first called. The proposed mechanism uses the resolution of reference. It can decide whether it should interpose into the library function. In the proposed mechanism, the access control module is called before a routine is called. This module can decide whether to grant or deny the routine call.

As mentioned previously, Curry's approach uses library interposition. Figure 2 shows an intrusion detection system based on Curry's approach. In this paper, an intrusion detection system at user level is also proposed to be combined with interposition of system-calls. The system is based on an access control mechanism using dynamic linker which cooperates with a system-call access control mechanism that exists in the kernel. This approach also uses a dynamic linker run-time program. The system prevents an attacker from bypassing library interposition. Figure 3 shows the proposed intrusion detection system.

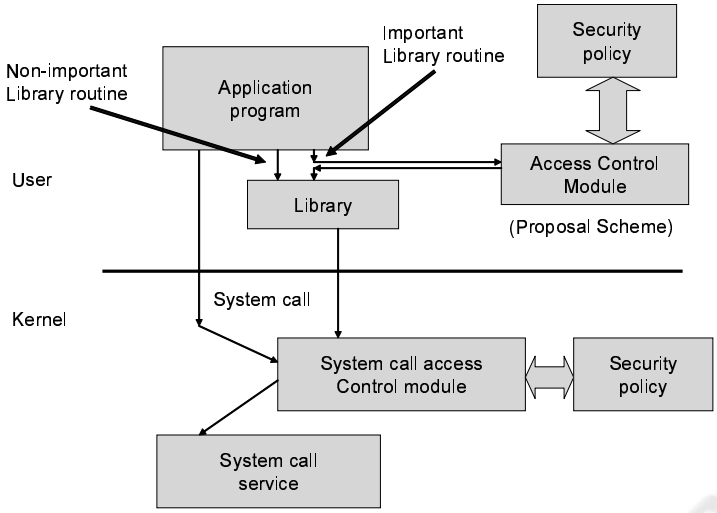

Figure 3: Overview of the proposed intrusion detection system.

4.2 Processing of Dynamic Linker on Program Execution

There follows a brief description of dynamic linker. Most current operating systems support Executable and Linking Format (ELF) (Levine, 2000) as the object file format and dynamic linking to delay the processing of linking by program execution. In operating systems supporting dynamic linking, most of references are not resolved before programs start. When unresolved references are referred, the dynamic linker places information into the executable that informs the loader of the code for shared object module. In contrast, a run-time linker finds and binds the references at run-time before the program starts.

ELF is supported in current UNIX and Linux as an object file format. In program execution environments that support ELF and dynamic linking, a run-time linker is called to resolve undefined references, when a library function is first called. After the resolution, the run-time linker is not called, and the function is called directly.

4.3 Address Resolution

In this subsection, the processing of address resolution is described. Figure 4 shows address resolution in general operating systems. In general, operating system such as Linux, the run-time dynamic linker program is called when a library function is first called. The run-time program finds the symbol of the library function and stores the address of the symbol into Global Offset Table (GOT) entry that corresponds to the Procedure Linkage Table (PLT) entry. After the address resolution, the library function is called directly without executing the dynamic linker run-time program.

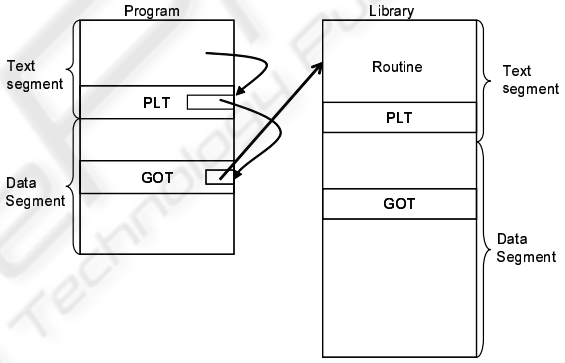

Figure 4: Address resolution in general operating systems.

Figure 5 shows the processing of the proposed address resolution. In the first address resolution, the run-time program of dynamic linker is called in the same way. Next, the run-time program finds the symbol of the library function and stores the address of access control module into GOT entry that corresponds to the PLT entry. Then the run-time program stores the address of the symbol into local table in the dynamic linker. After the address resolution, the access control module is called before the library function is called. This module can gather the information of the library function call and decide whether to grant or deny the library function call. The module calls library function, when the library function call is granted.

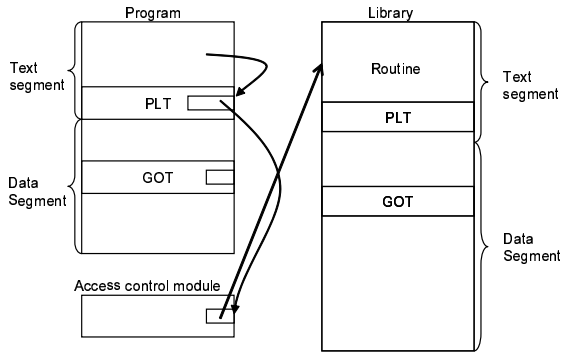

Figure 5: The proposed address resolution.

4.4 Selection of Address Resolution

Access control involves some overhead. Thus, it is desirable that the overhead of access control is kept to a minimum. Curry's approach interposes every library function call. Here, selectable interposition is presented, using dynamic linker.

The operating systems provide many functions as system-calls. Thus, system-calls can have an effect on the security of computer systems. On the other hand, not every library function executes system-call. It is important to control library functions involving the execution of a system-call. The library functions involving the execution of system-call are named important library routines in figure 3. Non-important library functions do not involve the execution of system-calls.

When an important library function is called first, the address is resolved. When a non-important library is called first, the address is resolved by a general approach. The run-time dynamic linker program decides whether use the proposed technique or a general approach. In the proposal, there is no overhead when non-important routine is called after the address resolution. Therefore, this proposal is able to reduce the overhead of access control.

4.5 Access Control Module

The access control module exists within the dynamic linker. This module is called before the library function calls and can gather the state of library function calls. The states consist of the name of the caller routine, the name of the callee routine and the string of arguments of the callee routine. In addition, users can write the rules of access control. The module decides whether to grant or deny a library function call by following it. Anomaly detection or misuse detection can both be implemented in the proposed system.

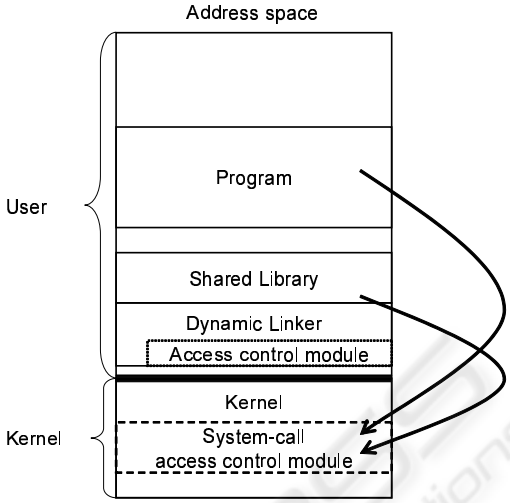

Figure 6: A system-call access control module.

4.6 System-call Access Control Module

The interposition mechanism at user level can be bypassed by executing a system-call directly. To prevent an attacker from bypassing the interposition mechanism at the user level, a system-call access control function is required in the kernel. Introducing the interposition mechanism at the user level renders the implementation of system-call access control module a simple task.

Recently, most attackers abuse buffer overflow vulnerabilities. A program that has some vulnerability can be easily hijacked. Using the buffer overflow vulnerabilities, the attacker sends a malicious code. The code may include a direct system-call.

For this reason, Figure 6 describes a system-call access control module. This module traps the address of the system-call. If the address is included within shared library text segment, the module knows that the system-call request is safe. All of the important library function calls have already been checked by the access control module in the dynamic linker. If the address is included within the stack segment or heap segment of a program, the request of the system-call is suspicious. As described above, the system-call access control module checks the address of the code executing the system-call. Thus, the overhead due to execution of the codes of system-call access control module is small.

4.7 Advantage of Proposed Mechanism

1. Library function calls may be intercepted with low overhead.
2. User level access control mechanisms are selectable. Audit is only important for library function calls with low overhead. A non-important library function is called with no overhead after address resolution.
3. An intrusion detection system is combined with user and kernel modules. Using access control module at user level simplifies the structure of system-call access control module at kernel.

5 CONCLUSION

This paper has addressed the design of an intrusion detection system at the user level with system-call interposition. An access control mechanism has also been proposed which uses a dynamic linker.

The proposed intrusion detection system uses this access control mechanism. The overhead of the access control process at the user level is low. However, it can be seen that the access control mechanism at user level can be easily bypassed. Thus, an intrusion detection system at user level has been combined with system-call interposition. As a result, proposed IDS can prevent attacker from bypassing the interposition mechanism. Proposed intrusion detection system possesses both low overhead and fine access control.

The feature of proposed access control mechanism is selectable interposition using a dynamic linker. Thus, it can be decided which library function should be controlled.

ACKNOWLEDGMENT

This research was partly supported by the 21st Century COE Program 'Reconstruction of Social Infrastructure Related to Information Science and Electrical Engineering'.

REFERENCES

Curry, T. W. (1994). Profiling and tracing dynamic library usage via interposition. In *USENIX Summer 1994 Technical Conference*.

Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion detection using sequences of system calls. In *Journal of Computer Security, Vol.6, No.3*.

Jain, K. and Sekar, R. (2000). User-level infrastructure for system call interposition: A platform for intrusion detection and confinement. In *In ISOC Network and Distributed System Security*.

Kuperman, B. A. and Spafford, E. (1998). Generation of application level audit data via library interposition. In *CERIAS TR 99-11, COAST Laboratory, Purdue University, West Lafayette*.

Levine, J. (2000). *Linkers and Loaders*. Morgan Kaufmann.

Oyama, Y., Wei, W., and Kato, K. (2003). Modularizing normal behavior databases in anomaly detection systems. In *IPSJ Transactions on Advanced Computing Systems, Vol.44CNo.SIG 10(ACS 2)*.

Sekar, R., Bendre, M., Bollineni, P., and Dhurjati, D. (2001). A fast automaton-based method for detecting anomalous program behaviors. In *IEEE Symposium on Security and Privacy*.

Wagner, D. and Dean, D. (2001). Intrusion detection via static analysis. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*.

Wagner, D. and Soto, P. (2002). Mimicry attacks on host based intrusion detection systems. In *Proc. of Ninth ACM Conference on Computer and Communications Security*.