IPSdNA: an Intrusion Prevention System disputing No-op Attacks

預防繞道攻擊之入侵偵測防禦系統*

Tsung-Yi Tsai , Kuang-Hung Cheng, Wen-Nung Tsai
Department of Computer Science and Information Engineering,
National Chiao-Tung University
{ *tytsai, chengkh, tsaiwn*}*@csie.nctu.edu.tw*

Abstract

In this paper, we present a real-time Intrusion Prevention Ssystem named IPSdNA (an Intrusion Prevention System disputing No-op Attacks), which is based on system call interception technique. In this system, users can describe attacking models in forms of state machine through a well-designed GUI interface. This system intercepts every system call invoked by application programs and tries to match any penetration pattern. Once there is an evidence showing some penetration is undertaking, the system can terminate the penetration process before injury. To improve detection accuracy, we developed an inspection model based on automata theorem and human-immunity concept. With the help of this enhancement, IPSdNA can solve several kinds of mimicry issues that are destined for pattern-matching IDS.

Keywords: IPS, Wrapper, Mimicry Attacks, human-immunity

中文摘要

以攔截系統呼叫為基礎之入侵偵測系統提供即時的入侵防禦能力，在攻擊尚未造成損害之前即提供攔截制止以確保系統安全。但是這類藉由特徵比對方式來偵測攻擊的安全系統，在遭受刻意安排的攻擊模式時，常面臨準確度下降的問題。因此藉由攔截系統呼叫型之入侵系統的基礎，我們加強了繞道攻擊所可能帶來的問題，並且降低誤判率，進而提出本系統。根據自動機理論，我們將攻擊樣版經過轉換後，仍能提供相同特徵的防禦能力。除了原先所定義的攻擊行為外，還能偵測出相關的變種攻擊行為，這對於偵測入侵者的詭變攻擊方式有極大幫助。

關鍵字: 入侵防禦、擬態攻擊、人體免疫系統

1. Introduction

There are many security systems, such as firewalls, anti-virus engine and Intrusion Detection System (IDS), can be used to prevent the system from malicious attacks on network. Those tools are designed by experts and used to detect and/or to prevent intrusion behavior. However, no matter how powerful or complicated they are, the intruders can always sneak through and bypass the security checking after analyzing these tools from top to toe. Fortunately, the experts always try to push forward new defending technology as soon as possible to resist new attack methods.

IDS can be divided into two categories according to their detection methods, including the Anomaly IDS and the Misuse IDS. The Misuse IDS is the prevalence one and it will use given and occurred attacking scenarios to build up an intrusion characteristics database. So, the Misuse IDS is also called the Signature-based IDS. When the monitored behavior is compared and matched against some intrusion pattern, this action will be judged as an intrusion. In this manner, the Misuse IDS has the benefits of low false alarm rate. But it will also suffer from the drawbacks of low detection rate to new kinds of intrusions, since it does not have the patterns of new attacking scenario in its signature database.

The Anomaly IDS has a database which is used to store templates of normal program behaviors. The Anomaly IDS will compare the monitored action with normal models in database and identify it as abnormal when it has lots of divergence comparison result. Although the Anomaly IDS can detect new attacking methods, it also has to bear high erroneous judgments rate. That is because it is quite hard

* This work was supported in part by National Science Council, Contract No. NSC93-2213-E-009-034.

and uncertain to exactly define what the normal behaviors are in this complicated computing world.

Exactly as mentioned in the design goal and principals of STBIPW (State-Transition-Based Intrusion Prevention using Wrapper)[1] , only when the intruder have accessing privilege to system resource can he cause certain degree of damage to victim system, and the only possible way to access system resource is via system call interface provided by OS. Therefore, system call interface is an area of strategic importance. With the help of wrapper-based IPS, we can intercept and analyze the system call sequence invoked by process to detect intrusions.

In recent years, most of the researches regarding IDS are mainly focused on the improvements of detection rate and erroneous judgments rate. However, there are also some articles trying to give a warning to the potential attacks of IDS, such as Mimicry Attacks [5] .

In order to reduce the possibility of false positive rate and to enhance the capability of wrapper-based IDS, we proposed the IPSdNA (Intrusion Prevention System disputing No-op Attacks) based on the STBIPW which we built before. In STBIPW, user can define intrusion templates in the form of finite state machine (FSM) and those templates are similar to the patterns in traditional signature-based IDS, which can be used to detect intrusion actions and stop its execution before damage occurred. However, STBIPW is a signature-based IPS and it also has to resolve the problem of low detection rate caused by new kinds of attacking scenarios. Therefore, against STBIPW, there are two major improvements in IPSdNA. First, we analyze and transform user-defined intrusion templates according to Automaton theorems. After transformation, these modified templates will assist IPSdNA to detect not only original intrusion behaviors, but also the No-op attacks. Seconds, we adopted the concept of Human-Immunity in our system. We use negative selection mechanism to test user-defined templates and filter out improper ones to lower down false alarm rate.

In this paper, we will first introduce several related works regarding intrusion detection systems (IDS), and then give an overview of human-immunity system in section 2. Then, we will talk about mimicry attacks that give several challenges to signature-based IDS. In section 3, we describe how the IPSdNA conquers several issues that are destined in signature-based IDS, followed by the detail architecture of IPSdNA in section 4. In section 5, we use plenty of experiments results to show the intrusion detecting capability and to evaluate the system efficiency of IPSdNA. Afterwards, we will give a brief discussion and conclusion in the last section.

2. Related Works

In this section, we first introduce two intrusion detection systems related to this paper. One is the system-call-based IDS that intercepts and monitors the sequence to detection intrusion. And the other is the state-based IDS that use states and state transitions mechanisms to find out the evidence of intrusions. Afterwards, we present the human immunity concepts and introduce one intrusion detection system designed and implements based on this concept. Finally, we talk about some possible attacks to host-based IDS and ways to pass by its detection.

2.1 System-call based Detection Methods

In modern operating systems, user process has to use system calls to access system resource through kernel. In this way, kernel can schedule each request and ensure fairness among multi processes. To provide security checking, it is quite practical to do examination on system calls invoked by suspected process. There are many IDS systems designed with this concept, such as STBIPW[1] and N-Gram [18] [23] .

N-gram was proposed by Stephanie Forrest and other team members in 1996. They used system call tracing technology to build their IDS system and finally presented the pH-IDS[2] (process Homeostasis IDS) in year 2000. pH-IDS will verify each system call invoked by process and determine its status. It builds a “self database” for each privileged process which is used to represent this process’s normal behavior under specific hardware architecture, software version and configuration.

The “N” in N-gram represents that it will examine N continuous system calls. For example, when N = 3, N-gram will observe each system call and check their relationship with the following three system calls. It will compare each fragment of system call sequence against database to detect intrusion. In this way, N-gram is simple and efficient, but the detection rate will depend on the window size N. When window size N is larger, it will be more precise to the comparison result since it has more evidence to prove it is an attack. However, it will decrease the detection rate with too large N. Therefore, choosing N is an important and tough job when using N-gram since every process has different adapted window size. The other drawback of N-gram is that it checks only system call sequence. It does not inspect the system call parameters. There are some attacks that are carried out with valid system call sequence but harmful arguments to achieve the purpose of attacking. Furthermore, we can choose some valid system call fragment from normal database and insert it into intrusion sequence. In this way, the intrusion system calls are scattered into each fragments and beyond the scope of window size, thus successfully escaping the N-gram's check.

2.2 FSM based Detection Methods

The finite state machine (FSM) is composed of states and transitions. Each state is used to record status of certain task and related to each other with transitions. Each transition will be triggered to make switch when some event happened. We can view each intrusion behavior as a sequence of states that each state represents some key action has been done. For example, Figure 1 represents the behavior of virus infection.

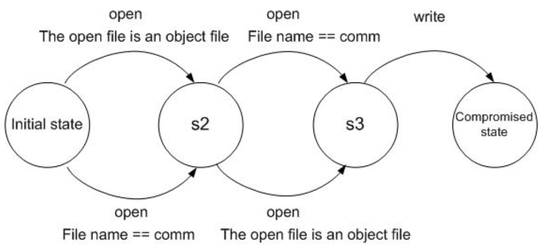


Figure 1 FSM of Virus Infection

The STAT (State Transition Analysis Tool) [15] [20] is one well-known IDS system that uses FSM to analyze intrusion. STAT is a log investigation system. It uses FSM to describe attack scenarios and then feeds system logs into intrusion detection engine to find whether the system has been attacked. However, attackers can make a detour to go on the offensive and leave without being aware of.

There are many researches that use the concept of STAT, such as STBIPW[1] . In that work, it decomposes a penetration into many states linked with critical system call transition. It uses graphical interface to describe intrusion behaviors. It would be more intuitive and easy to depict attacks, especially for complicated rules. Therefore, the STBIPW can provide real-time intrusion prevention, and the graphical models will not only decrease the complexity of rule maintenance but also raise the accuracy of detection rate.

2.3 Immunity based Detection Methods

Human body is always being in touch with external materials, such as water and air. All these outside materials may contain harmfulness invaders to human bodies. It might be bacteria, virus, even the parasite. Fortunately, we have biological immune system that would detect and eliminate those foreign intruders. The major character of immune system is the lymphocyte, which is known as antibody. It is like a sentry that patrols around and ferrets out pernicious materials.

To produce antibodies, the immune system will first pick up a random segment from gene pool. This randomness is the main reason why human bodies can resist unknown diseases. However, not every gene segment is able to be used as lymphocyte. They have to be tested with both positive selection and negative selection. The positive selection will leave behind those abnormal lymphocytes that can not cooperate with other human cells. On the other hand, negative selection will let lymphocytes contact with human cells and filters out those active ones since these lymphocytes misjudge normal cells as enemies, producing the phenomenon of autoimmunity. The lymphocytes that pass both positive and negative selections are said to be mature and be able to shoulder the important duty of epidemic prevention.

There are many researches that proposed and designed IDS systems with the concept of human immunity [24] [25] [26] . The basic idea is to map certain computer characteristics as antibody, and then cultivate random-chosen antibody to be a mature one. For example, we can represent a network connection to be a byte stream in length L based on some basic information, such as IP addresses and port numbers. We also define the sentries of IDS system as byte stream with length L. Those sentries are corresponding to lymphocytes in immune system and responsible for the jobs of intrusion detection. When the representative byte stream of certain network connection matches some detectors, it means that this connection might be a dangerous one.

Another kind of immunity detector is described as a state machine, such as the work in IGSTAM [26] . In IGSTAM, it represents each kind of intrusion as sequence of states alternated with transitions. Consequently, it defines the antibody as state machine and calls it vaccine in the following format:

$$Vac = (S1, A1, S2, A2, \ldots, Sn).$$

Those vaccines also have to pass negative selection. They will be trained against normal database to see whether they are active to normal behaviors. If there is no match, those vaccines are said to be mature and can be spread to detect whether system is being attacked.

2.4 Weakness of Host-based IDS

To detect intrusions, signature-based IDS has to detect and match certain pattern exactly, and then it can judge it as an attack. For that reason, we can insert some useless operations (no-op) into penetration sequence to confuse the detectors. This kind of attack is called "no-op attack". Take N-gram discussed above as an example, N-gram will inspect several continuous system calls to detect intrusions. If we know the trained database it uses and insert some normal system calls into detector pattern purposely, we can get away from detection since the attack sequence has been out of the window size. In the same way, we can disperse attacking steps to normal patters and accomplish invasion slowly and stealthily. These kinds of attacks are called Mimicry Attacks [5] and the no-op attack is one typical type of them.

Another kind of mimicry attacks is called the collaborative attack. In order to escape from examination, malicious process can fork another child process to carry out invasion. Most IDS systems will monitor fork related actions and apply equal inspections for child processes to prevent such attacks. However, if both the parent and child processes finished parts of intrusion steps and exchange their results through IPC, it will be undetectable by original methods. For example, if we define the following sequence of system calls as a simple intrusion patter:

open("/etc/passwd");
write("/etc/passwd");
close( );

It will be undetectable by general IDS systems if we let parent process do the *open* system call and let child process finish the *write*, as shown in Figure 2. Currently we will focus on detecting the No-operation attacks. To disputing the collaborative attacks would be our future work.

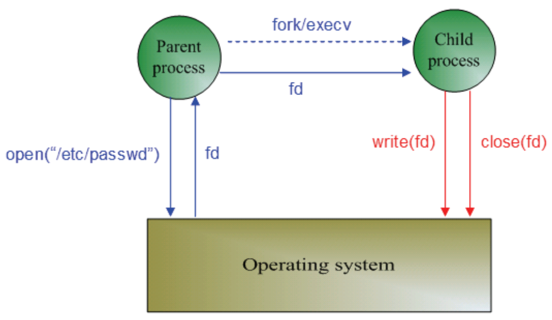


Figure 2 Collaborate Attacks

3. IPSdNA System Requirements and Design Issues

Detection rate and false positive rate have been challenges to intrusion detection systems all the time. Therefore, a well-designed IDS system should pay attention to both requirements. In this section, we first introduce how to use negative selection mechanism to inspire from immune system to filter out unsuitable detectors, and thus reduces the chance of false positive. In additions, we proposed solutions to against mimicry attacks.

3.1 Examination of Improper templates

In this paper, the system architecture we proposed can be classified as a Misuse IPS. It can let users customize their own penetration templates so that it can be used to detect intrusions as required. In order to lower down the false positive rate, we collect normal system call sequences on a clean system and use this data to examine user-defined templates. There are two phases in this procedure, one is the training phase and the other is the testing phase.

In training phase, we collect system call sequences of normal actions to build the Normal Database. During this phase, it should be guaranteed that the testing environment is clear and there would be no any intrusion at all. This is much like the way used in Anomaly IDS. However, the normal database built in Anomaly IDS is used for intrusion detection. On the contrast, normal database is used to inspect user-defined patterns (as shown in Figure 3), just as lymphocytes and negative selection mechanism in the immune system.

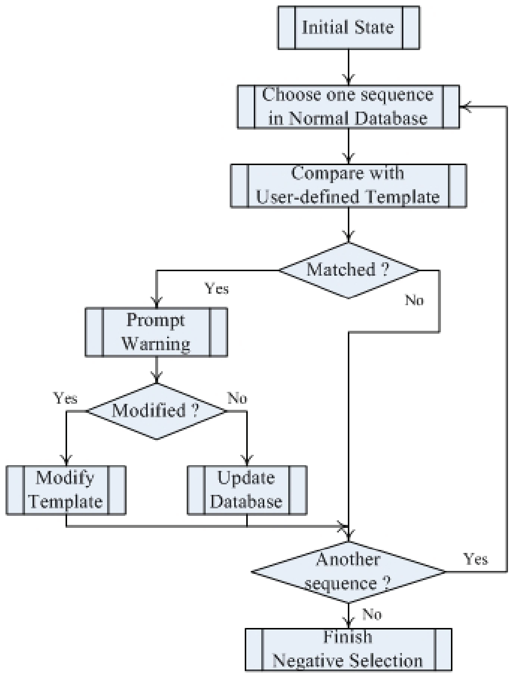


Figure 3 Procedure of Negative Selection

In testing phase, if the comparison gets positive reactions, it means that this template might misinterpret normal behavior as abnormal one. The system will make a warning prompt to user. In this way, users will have the opportunity to modify just-defined templates or let it be used in system even though it got positive result. Based on the testing procedure inspired from immune system, we can diminish lost caused by improper template detectors.

3.2 Prevention the No-op attacks

In traditional signature-based IDS, it is easy to circumvent intrusion examination by inserting lots of useless system calls. However, those systems that specify intrusion as state machines have certain degrees of resistance against no-op attacks in nature. This is because the finite state machine would stay remained when it faces an unrelated event. However, it is not enough to defend no-op attacks with only such inborn gifts.

For example, we can define an intrusion template as shown in Figure 4 and suppose all those labels on arcs are system call numbers. This machine is a DFA (Deterministic Finite Automaton) and thus it must make a state transition when the current state and current system call satisfied its definition of transition function. To attack this template, we can make a system call sequence “a, c, e” and this sequence won’t be detected by this machine. That is because the attacker can escape the inspection of “c, e” sequence by executing “a” first. We call this kind of attack as Evasion attack. It is one kind of no-op attacks since the system call “a” behaves like a no-op operation used to disturb detection.

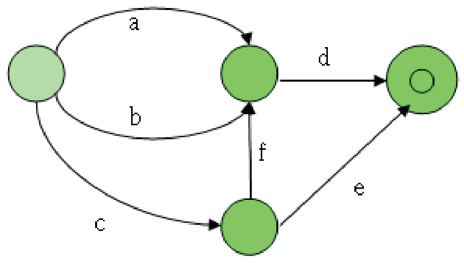


Figure 4 Original Template

In order to solve evasion problem, we have to reorganize the FSM. First, we decompose the complex parts of original FSM into several sub-FSMs as shown in Figure 5. Every two crotched paths that join before final state would be separated. Formally, we will do DFS search on original graph and take apart the graph into individual subgraphs represented by the paths from initial state to final state when we find a back edge.

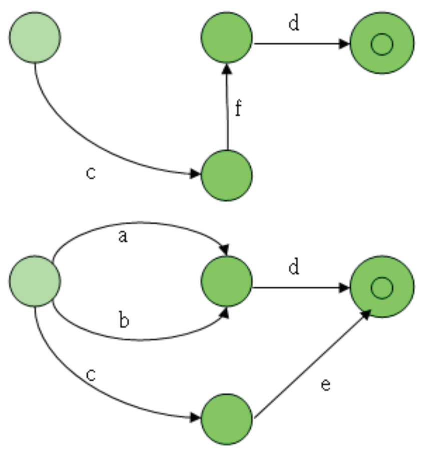


Figure 5 Template after first transformation

In this way, attackers can not insert evasion operation (unrelated but influenced) to escape inspection. For example, it would be undetectable for sequence “c, f, e” and become a suspect one after transformation. This algorithm is listed as follows:

```
G:graph (FSM)
V[G]:nodes in G
U[G]:edges in G
P[G]:path from initial node to final node in G

DFS(G)
    for each node u ∈ V[G]
        do  color[u] <--WHITE
            $\pi$[u] <--NIL
    for each node u ∈ V[G]
        do  if  color[u] = WHITE
                then DFS-VISIT(u)
DFS-VISIT(u)
   color[u] <--GRAY
   for each v ∈ Adj[u]
       do  if  color[v]=WHITE
               then  $\pi$[v] <--u
                     DFS-VISIT(v)
           if  color[v]=BLACK  and  v!=final  and  $\pi$[v] != u
               then for each  p ∈ P[G] that include (u,v)
                    do New a sub-FSM
                  discard (u,v) from G
   color[u] <--BLACK
   if  Adj[u] = NIL  and  u is not final
       then discard ($\pi$[u],u) from G
           discard u from G
```

In second step, we add an ε-transition for each non-final state back to previous fork state that has more than two possible transitions. It will become an ε-NFA (Nondeterministic Finite Automaton), as shown in Figure 6. In this way, no matter how far this machine has been to, it can go back to previous fork state and trace another path to the destination.

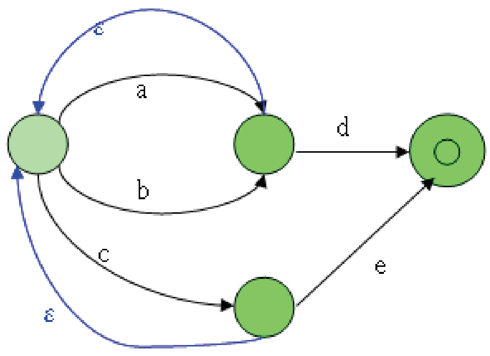


Figure 6 Template after second transformation

After this transformation, the enhanced FSM can detect any kind of no-op attacks, no matter how much the useless operations are and how dispersed those key events are. This algorithm is listed as follows:

```
G:graph (FSM)
V[G]:nodes in G
U[G]:edges in G
D[G] : crotched nodes in G

DFS(G)
    D[G]=NULL
    for each node u ∈ V[G]
        do  color[u] <--WHITE
            π[u] <--NIL
    for each node u ∈ V[G]
        do  if  color[u] = WHITE
                then DFS-VISIT(u)
DFS-VISIT(u)
    color[u] <--GRAY
    if  Num(Adj[u]) > 1
        then  Add  u  to D[G]
    for each v ∈ Adj[u]
        do  if  color[v]=WHITE  and  v!=final
                then  π[v] <--u
                      for each d ∈ D[G]
                          do  Add (d,v) labeled ε
                      DFS-VISIT(v)
    color[u] <--BLACK
    if  u ∈ D[G]
        then discard u from D[G]
```

Finally, we transform the ε-NFA into an equivalent DFA as shown in Figure 7. We do this transformation for the reason of simplicity and efficiency. In DFA, the system only has to pay attention to relative system calls that are critical to current state. However, in ε-NFA, it needs to guest all possible paths against happened events and search whether there is one path that reaches the final state. Therefore, it would be much simpler and efficient in real-time execution when we use DFA to specify the intrusion. Even so, the transformed FSM does not lose its detection capability since they are equivalent according to automaton theorem. It can detect several variants of no-op attacks, including evasion attack.

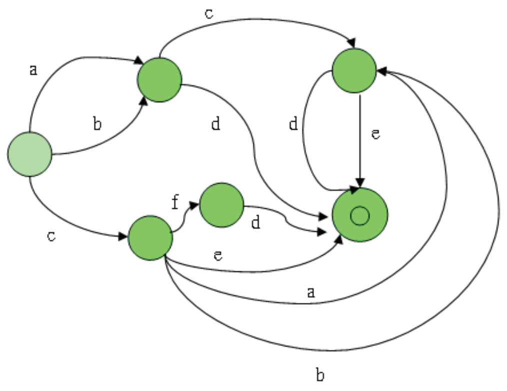


Figure 7 Final Template

4. IPSdNA System Architecture

Based on STBIPW, our system is divided into two segments; one is the user level components and the other is the kernel-level components. We use a device driver to connect these two parts. The device driver acts as a bridge between these two segments. In one side, it will help to pass commands and data to underlying core engine for user configuration. And in the other side, it will also help to return execution information back to users. The architecture of the complete system is shown in Figure 8. We use the Kernel-Level Wrapper technique to build all the kernel components in the form of LKM (Loadable Kernel Module) and plug it into the Linux kernel in the run time.

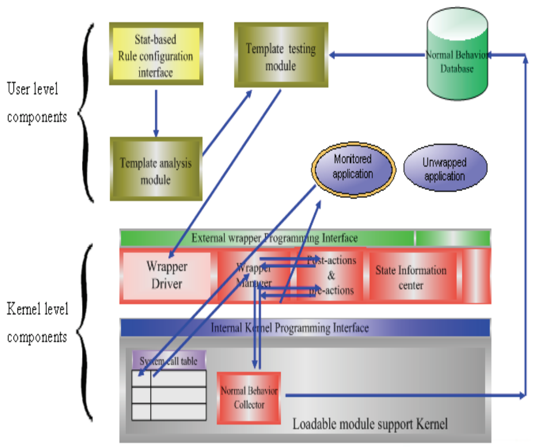


Figure 8 System Architecture of IPSdNA

4.1 User Level Modules

The main function of the user level module is to pre-set the attack templates. This includes providing the interface for users to formulate attack templates, and analyzing the attack templates the users have defined. This can not only prevent no-op attacks, but can also verify the user-defined templates. The purpose is to make sure there are no inappropriate attack template monitoring programs that may cause a high false positive alarm rate.

First, we must analyze and verify the attack template defined by the user. Therefore, we might alert the user when he/she defines an inappropriate attack template to eliminate recognition errors, and also prevent hackers from inserting no-op system calls to avoid detection. These functions are accomplished by the Template Analysis Module and the Template Testing Module.

The Template Analysis Module first analyzes the original attack templates, divides them into several sub-FSMs according to the condition, transforms each sub-FSM into a ε-NFA, and last, transforms them into real attack templates that can actually be used to detect attacks. As in Figure 9, the upper part reveals the original attack template that the user has defined; after going through analysis and transformation via the Template Analysis Module, it will become the final attack template as revealed in the lower part.

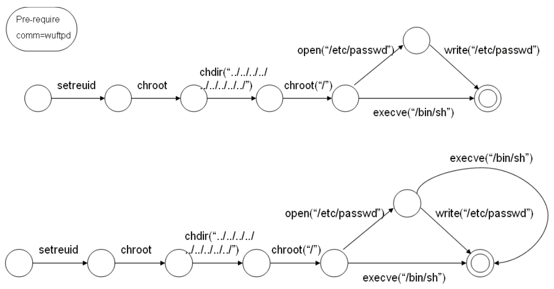


Figure 9 Template Transformation

After analyzed, the attack templates must be tested before being inserted into the kernel. The Template Testing Module will first select the normal system call sequence from the Normal Behavior Database, and then test to see if it could transform the attack template to the template's ultimate condition. The flow is shown in the figure below.

The template will be inserted into the Wrapper Driver after it is verified. Then the kernel level modules will execute the monitoring program according to the user defined template, and detect attack behaviors which match with those ones that the users have defined.

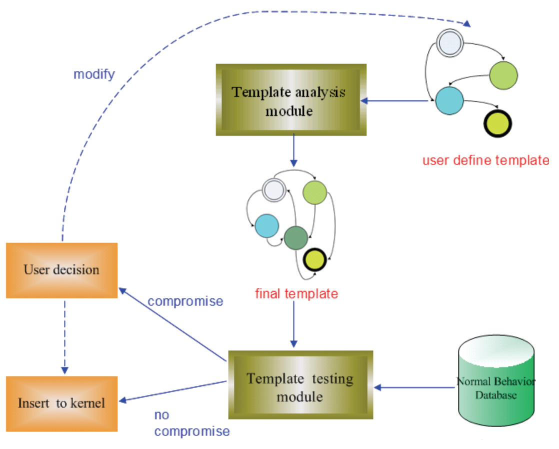


Figure 10 Template Analysis and Verification module

4.2 Kernel Level Modules

The main function of the Kernel level module is to detect real-time attacks. This includes generating an FSM object, intercepting system calls, and executing the state-transition of the attack FSM object. Training the Normal Behavior Database is also a function of the Kernel level module.

The Normal Behavior Database used by the Template Testing Modules is constructed by the Normal Behavior Collector. The Normal Behavior Collector records the system call sequence while normally and securely used by a user. This training method could be used on any application program. Via the Wrapper Driver, a user could set up the requirements in the Wrapper Manager to record normal system calls. The Wrapper Manager could then call the Normal Behavior Collector according to the requirements, and train an exclusive Normal Behavior Database for any application program. The user could decide both the time of training and the size of the database. The longer the time, or the lager the size, the more accurate the attack template test would be, resulting in reduced false positive alarm rate.

5. Experimental Results

In this section, we will use the experimental results to illustrate the efficiency and practicability of the IPSdNA system we proposed. First, we will discuss the run-time performance overhead of our system; then we will show the results of the experiment on detecting improper attack templates by implementing the user-trained Normal Behavior Database. We use attack behaviors that can purposely avoid detection for the mimicry attacks to verify that our system can catch these attack behaviors, and detect the attacks more accurately.

5.1 Runtime Overhead

The runtime overhead of our system is mainly caused by state-transition. We designed the first testing program as a copying program, which opens a text file and copies it to another file. All the read and write system calls will cause state-transition in the FSM object of the supervising program while it is executed. The result is shown in Figure 11. No matter how large the size of the copied file is, the state-transition time is a stable constant around 1300 μ s. Therefore, the lager the file is, the longer the time is used on the I/O operation, and relatively, the smaller the system's runtime overhead will be. This is shown in Table 1.

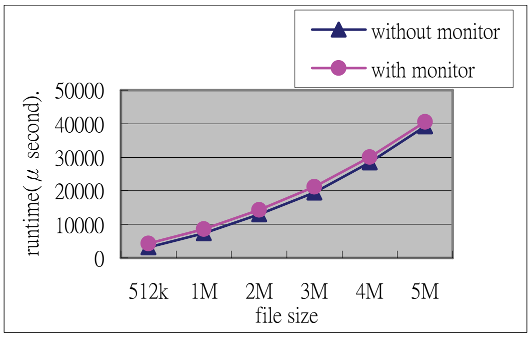


Figure 11 Program size vs. elapsed time

File Size (MB)	0.5	1	2	3	4	5
Overhead Rate	36%	18%	11%	8%	6%	4%

Table 1 Program size vs. System overheads

However, the combination of the first program and the attack template results in the worst state. That is, when a user formulates an attack template, the key system calls on the template seldom occurs. This is because comparing to the rate of normal behaviors, the rate of attack behaviors is much lower while executing a program. Therefore, in the second testing program, our program copies a 2M file in the experiment, modifies its behavior conditions and attack templates, and reduces the rate of the system calls that causes the state-transition in attack templates. The result is shown in Figure 12. From the figure, we can claim that the runtime overhead is approximately 0 while the rate of the system calls that causes the state-transition in attack templates is below 8%.

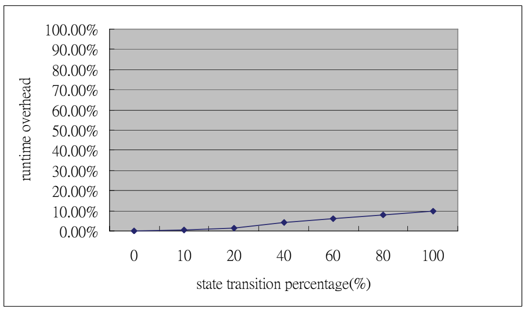


Figure 12 Overhead in State Transition

The experiences above merely show the state of a single attack template monitored by a single program. Next, we will use the program that copies a 2M file to experience the condition with attack templates and several monitor programs executed concurrently. Figure 13 reveals the relationship chart of one program being monitored by different numbers of attack templates. From the chart we can see that even if the program meets the monitoring conditions of several different attack templates, the executing efficiency will not reduce as long as the program is still normally running. Therefore, the runtime overhead can remain in a stable status. This is because even though the program is affected by being monitored under many monitoring conditions, it will not cause a lot of templates to do state-transitions constantly if normally executed. Hence, it will be able to stably retain a low runtime overhead.

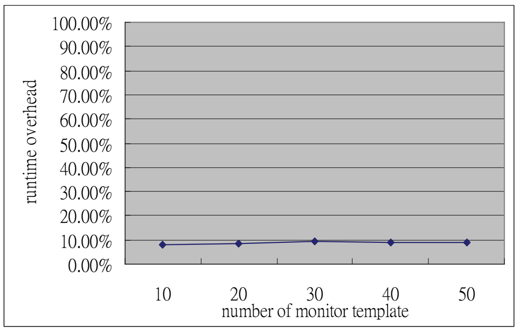


Figure 13 Number of Monitored Templates vs. Overheads

Figure 14 shows the overhead when many programs that require being monitored are executing. We can see that while running many monitored programs at a time, the runtime overhead can also retain stable. This is because while a system call of a program is intercepted, the hash method is used to search for the FSM object of the monitoring programs. Therefore, running many programs concurrently would only increase the searching time, which is relatively little. This can retain a stable runtime overhead.

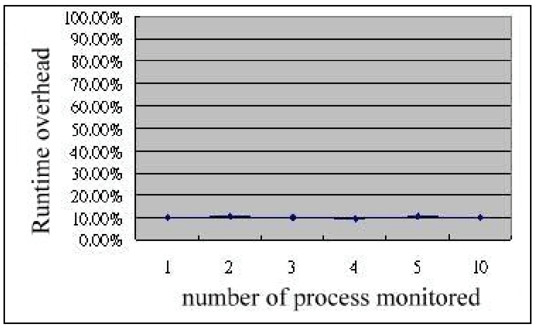


Figure 14 Number of Process vs. Overheads

Afterwards, we experienced running multiple programs at once, each program being monitored by many attack templates, as shown in Figure 15. In the figure, each line represents the number of programs that are executed at the same time. This shows that even under such a sophisticated monitoring condition, our system can still retain a low stable runtime overhead, and not be affected by the increased numbers of programs and templates.

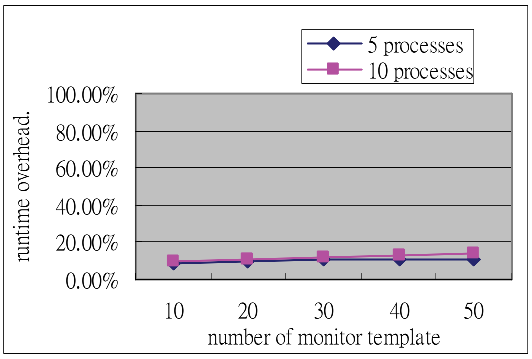


Figure 15 Runtime overheads

5.2 The Attack-Detection Expeirments

First, we assume that the sequence of an attacking system call is as follows:

setreuid(0,0)
open("/etc/passwd")
write("/etc/passwd")

In the pH-IDS detection method, say that the window size is 3; we would only have to insert a no-op system call, such as read, to make the sequence of the attack system call as below:

setreuid(0,0)
open("/etc/passwd")
read("/etc/passwd")
write("/etc/passwd")

in order to avoid the pH-DS detection.

Based on the characteristic of STAT, our system assumes that the attacker already knows the attack template the user will define, as in Figure 16. However, our FSM would only do state-transition when a key system call occurs. Therefore, inserting unrelated no-op system calls would not affect our detection. As in the example above, while going through the *setreuid* system call and the *open* system call, the FSM will stay in state 3; if read, an unrelated system call, is inserted at this moment, it will not affect our detection. Our FSM will remain in state 3 until write, another system call, is intercepted; it then moves to the final state and the attack is caught.

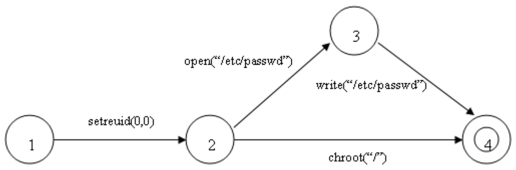


Figure 16 Original testing template

Besides, experienced attackers may use the weaknesses of STAT and try to insert onto the other system calls of the attack template to find another route to attack. The attacking system calls are as below:

setreuid(0,0)
open("/etc/passwd")
chroot("\")

The "*open*" is a no-op system call which attempts to use open as other-routed attacks. It first executes system calls *setreuid* and *open* to put the FSM in state 3; because FSM could not intercept chroot in state 3, it could avoid the detection. Yet, in our system, our Template Analysis Modules will first transform the original attack templates into no-op attack-evading attack templates, as shown in the figure below.

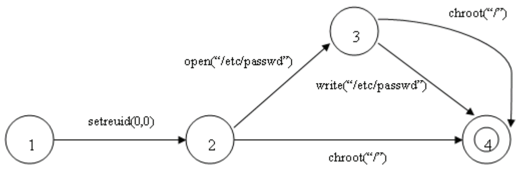
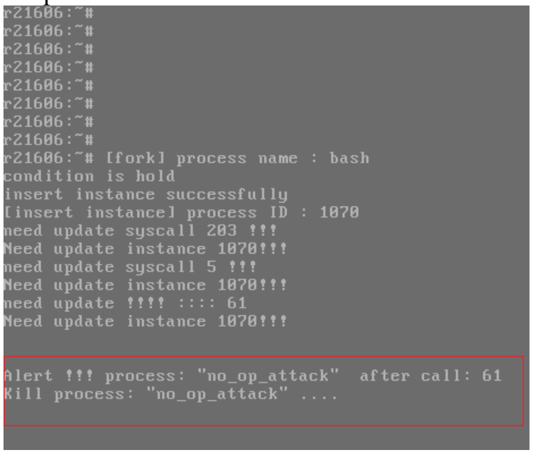


Figure 17 Testing template after transformed

We can detect the inserted No-op system call's rerouted attack, as shown in Figure 18. No matter how many no-op system calls the attacker inserts into the original attack template, or how many attacking routes it tries, we will be able to successfully detect it.

5.3 Warning Test of Improper Templates

If we want to monitor the sftp-server program, we use the Normal Behavior Collector for two days of normal behavior trainings to generate a 2M sized Normal Behavior Database. This can prevent errors from occurring in the future, and also prevent high false positive rate templates.



```
r21606:~#
r21606:~#
r21606:~#
r21606:~#
r21606:~#
r21606:~#
r21606:~#
r21606:~#
r21606:~# [fork] process name : bash
condition is hold
insert instance successfully
[insert instance] process ID : 1070
need update syscall 203 !!!
Need update instance 1070!!!
need update syscall 5 !!!
Need update instance 1070!!!
need update !!!! :::: 61
Need update instance 1070!!!

Alert !!! process: "no_op_attack"  after call: 61
Kill process: "no_op_attack" ....
```

Figure 18 No-op attacks detected

During the training phase, we only need to use the intranet to login and operate, in case of attack behaviors. Then we try to define an attack template, as in Figure 19, to prevent attackers from using the loophole of the sftp-server to get the root authority and leave some programs that would cause to the system.

The attack template we defined is to prevent attackers from using loopholes to get the root authority. They may create new file folders, modify authorities, write invading programs, etc. However, after going through Template Testing Modules, it will still move to the final state under a normally used condition. Because the system calls sequence shown in Figure 19, it may still happen under another file with the same authority. This shows that the attack template is not accurate enough, casing false positive rate to enhance. Thus our system tests the templates first once they are defined, and report them to the users.

6. Discussion and Conclusion

In this paper, based on the result of STBIPW[1] [1] we developed an intrusion prevention system, IPSdNA, which has the following four advantages:

(1) It has the advantage of a kernel wrapper, and the monitoring job is done in the kernel. The runtime overhead is relative low. Our system can intercept all the system calls. Any application programs that requests services from the system will be monitored. Thus, before the system being harmed, malicious behaviors will be blocked in real-time.

(2) It has the state-transition statement advantage of STAT. IPSdNA can use state-transition statement to generate graphical statement interface, so that the user may use direct sense to understand sophisticated behaviors.

(3) Low false positive rate. We use the negative selection inspired from the concept of human immunity system to train a Normal Behavior Database to detect attack templates, in order to prevent users from defining inappropriate, easily mistaken attack templates. Therefore, the false positive rate will reduce.

(4) It is able to go through the tricks of purposely avoiding IDS detects. To dispute the No-op attacks, we first analyze the original attack template that the user has defined, and then add an ε-transition for each non-final state that has more than two possible transitions. After that, we use the FSM transformation algorithm to transform the ε-NFA (Nondeterministic Finite Automaton) to an equivalent DFA so that every path can be traced, and thus can prevent no-op attacks.

In the near future, we will try to improve the IPSdNA to fight the collaborative attacks.

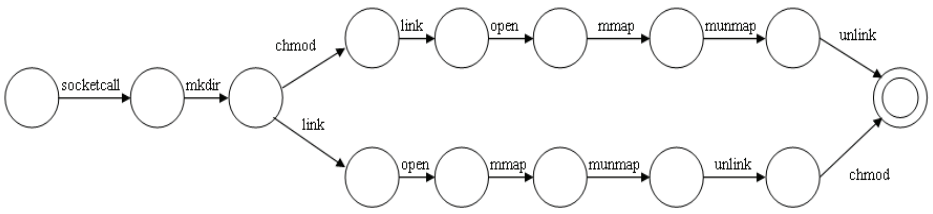


Figure 19 improper sftp-server attack template

Reference

[1] Tsung-Yi Tsai, Kuang-Hung Cheng, Chi-Hung Chen, Wen-Nung Tsai, "An Intrusion Prevention System using Wrapper,"in *Proceedings of International Computer Symposium*, pp.1218-1223, 2004.

[2] A. Somayaji, S. Forrest, "Automated Response Using System-Call Delays," in *Proceding of 9th Usenix Security Symposium*, pp.185, 2000.

[3] Bai, Y., Kobayashi, H., "Intrusion Detection Systems: technology and development," in *Proceding of 17th International Conference*, pp. 710-715, 2003.

[4] Caberera, J.B.D., Ravichandran, B., Mehra, R.K., and Sci. Syst. Co., Woburn, "Statistical traffic modeling for network intrusion detection," in *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 466-473, 2003.

[5] D. Wagner and P. Soto ., " Mimicry Attacks on Host-Based Intrusion Detection Systems," in *Proceding of the ACM Conference on Computer and Communications Security*, pp. 255-264, 2002.

[6] Dozier, G.,Brown, D., Hurley, J., Cain, K, "Vulnerability analysis of AIS-based intrusion detection systems via genetic and particle swarm red teams," *Evolutionary Computation*, Vol1, pp. 111-116, 2004.

[7] Eskin, E., Wenke Lee, Stolfo, S.J., "Modeling system calls for intrusion detection with dynamic window sizes," in *Proceding of DARPA Information Survivability Conference & Exposition II*, Vol 1, pp.165-175, 2001.

[8] F. Besson, T. Jensen, D. L. Metayer, and T. Thorn., "Model checking security properties of control flow graphs," *Journal of Computer Security*, pp.217-250, 2001.

[9] F Gonzalez and D Dasgupta, "Anomaly detection using real-valued negative selection," *Journal of Genetic Programming and Evolvable Machines*, p.383-403, 2003.

[10] Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee,W., Gong, W., "Anomaly Detection Using Call Stack Information," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy,Berkeley*, pp. 62, 2003.

[11] Ghosh, A.K., Wanken, J., Charron, F., " Detecting anomalous and unknown intrusions against programs, " in *Proceedings of the 14th Annual Computer Security Applications Conference*, pp. 259-267, 1998.

[12] Iguchi, M., Goto, S., " Network surveillance for detecting intrusions, " *Internet Workshop ISW99*, pp.99-106, 1999.

[13] Joseph, M. McAlerne and Stuart Staniford, James A. Hoagland, "Practical Automated Detection of Stealthy Portscans," Silicon Defense Publications, http://downloads.securityfocus.com/library/spice-ccs2000.pdf

[14] K.M.C. Tan, K.S. Killourhy, R.A. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits," to appear at *RAID* 2002 pp.54-73, 2002.

[15] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach,"*IEEE Transaction on Software Engineering*, Vol.21 No.3, pp.181-199, 1995.

[16] Phillip A. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)*,"in *Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, California*, pp.146-161, 1999.

[17] Rapaka, A., Novokhodko, A., Wunsch, D.,"Intrusion detection using radial basis function network on sequences of system calls," in *Proceedings of the International Joint Conference*, Vol3, pp. 1820-1825, 2003.

[18] S. A. Hofmeyr , S. Forrest , and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, pp.151-180, 1998.

[19] Snort Homepage. http://www.snort.org/

[20] STAT Homepage. http://www.cs.ucsb.edu/~rsg/STAT/

[21] T. Garfinkel, "Traps and pitfalls: Practical problems in system call interposition based security tools," in *Proceedings of Network and Distributed Systems Security Symposium*, pp.163-176, 2003.

[22] Tal Garfinkel, Ben Pfaff, Mendel Rosenblum, “Ostia: A Delegating Architecture for Secure System Call Interposition,” in *Proceedings of the Internet Society's 2004 Symposium on Network and Distributed System Security*, pp.187-201, 2004.

[23] Warrender, C., Forrest, S., Pearlmutter, B.,“Detecting intrusions using system calls: alternative data models,”in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pp.133-145, 1999.

[24] Zhang Yanchao , Que Xirong , Wang Wendong , Cheng Shiduan , “ An immunity-based model for network intrusion detection,” in *Proceedings of ICII 2001 - Beijing*, Vol 5, pp.24-29, 2001.

[25] Zhao Junzhong, Huang Houkuan, “An evolving intrusion detection system based on natural immune system,” in *Proceedings of 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, Vol1, pp.28-31, 2002.

[26] Zhou-Jun Xu , Ji-Zhou Sun , Xiao-Jun Wu , “An immune genetic model in rule-based state action IDS,” *in Proceedings of International Conference on Machine Learning and Cybernetics*, Vol4, pp.2472-2475, 2003.