A Study of Intrusion Prevention System against Collaborative and Inherited Attacks*

Tsung-Yi Tsai, Chia-Ming Sung, Wen-Nung Tsai
Department of Computer Science, National Chiao Tung University
{tytsai, chiaming, tsaiwn}@cs.nctu.edu.tw

ABSTRACT

Based on the system call interception technique, we develop a real-time intrusion prevention system to solve both the collaborative and inherited attacks. This system intercepts every system call invoked by application programs and tries to match any penetration patterns. Once there are evidences showing certain penetration is happening, the system can terminate the penetration process before it hurts the system. To improve detection accuracy, we build an inspection model analogous to the human-immunity concept that traces interactions among processes that constitute a villain family. With the help of these enhancements, our system can solve the collaborative attack problems, and can also select normal-behaved template genes to reduce false positive alarm rate.

1: INTRODUCTION

The intrusion detection system by far is the most important multi-function security tools. Based on the detection methods, IDS systems are divided into two types: the Anomaly IDS and the Misuse IDS. The Misuse IDS is more prevalence and uses known and observed attacking scenarios to build an intrusion characteristics database. Therefore, the Misuse IDS is also called the Signature-based IDS. When the monitored behavior matches an intrusion pattern, the behavior will be judged as an intrusion. In this manner, the Misuse IDS has the benefit of low false positive alarm rate, but suffers from the drawback of low detection rate to new kinds of intrusions, since no patterns of new attacking scenarios exist in the system's signature database.

Similar to the Misuse IDS, the Anomaly IDS has a database, too. The difference is that the database is used to store normal behaviors. The Anomaly IDS will tag a monitored action as abnormal when the action diverges significantly with normal models in the database. Although the Anomaly IDS can detect new attacking scenarios, it also has to bear high erroneous judgment rate. That is because it is quite hard and uncertain to exactly define what the normal behaviors are in this complicated computing world.

* This work was supported in part by National Science Council, Contract No. NSC 94-2213-E-009-006

In recent years, most researches on IDS are mainly focusing on improving detection rate and lowering erroneous judgment rate. However, some articles also tried to give warnings to the potential attacks of IDS [3], such as Mimicry Attacks [2].

In order to reduce the false alarm rate and enhance the capability of wrapper-based IDS, we proposed and implemented two improvements on the work of STBIPW [1]. First, when the user-defined intrusion template is used to monitor suspect processes, another wrapper will be spawned to supervise each new-born child process. Meanwhile, an extra family wrapper is going to watch out for stealthy interactions among this family. In this way, our work can prevent malicious processes from escaping inspection either by single child process or by collaboration of family process members. Second, we introduced the Human-Immunity concept into our system. We used negative selection mechanism to test user-defined templates and filter out improper ones to lower false alarm rate.

In this paper, we will first introduce several related works about IDS systems, and give an overview of human-immunity system. Then, we will discuss the mimicry attacks that give several challenges to signature-based IDS. In Section 3, we will describe how to conquer several issues on signature-based IDS, and follow that by the description of the system architecture of our approach. In Section 5, we use our experimental and evaluation results to show the detection capability and the system efficiency of our method. Finally, we give a brief discussion and conclusion.

2: Related Works

In this section, we first introduce several system call based IDSs and state-based IDSs that are related to this paper. Afterwards, we present the human immunity concept and introduce one intrusion detection system designed and implemented based on this concept. Finally, we will discuss some possible attacks to host-based IDSs and give ways to circumvent their detections.

2.1: System call based Detection Methods

In modern operating systems, user processes gain access to system resources through system calls. To provide security checking, it is quite practical to examine the system calls invoked by suspected

processes. Many IDS systems are designed with this concept, such as STBIPW [1] and N-Gram [8][9]. Furthermore, there are two ways to inspect system calls requested by applications. One way is to use user level system call interception mechanisms, such as the *ptrace* system call provided by OS to transfer execution from the monitored process to the monitoring process [23]. In this manner, security system programmers have the benefit of simplicity since modifying kernel code is not necessary. But high overhead is the fatal wound since each execution transfer requires two context switches. The other way is to intercept system calls via kernel level mechanisms [5][6][7]. This kind of security systems is resident mostly in the kernel and hijacks system calls directly from the system call table. Only jump instructions are needed to do this hijacking and thus little overhead penalty are produced compared to user level mechanisms.

N-gram used system call tracing technology to build the IDS system, and finally evolved into the pH-IDS in [10]. The pH-IDS verifies each system call invoked by the process and determine its status. The "N" in N-gram means that N continuous system calls are examined. Each fragment of system call sequence is matched against a database to detect intrusion. In this way, N-gram is simple and efficient, but the detection rate is dependent on the window size N. When N is large, the comparison result is more precise. However, the detection rate is also decreased. The other drawback of N-gram is that only system call sequences are checked, and system call parameters are not inspected. Some attacks can be carried out with valid system call sequences but harmful arguments to achieve the purpose of attacking. Furthermore, valid system call fragments can be chosen from normality database and inserted into intrusion sequence. In this way, the intrusion system calls are scattered within each fragment and beyond the scope of window size, thus successfully escaping the N-gram's check.

Another system call based IPS is the STBIPW [1]. The STBIPW was proposed and designed by the security research lab led by Wen-Nung Tsai in National Chiao Tung University. It is an Intrusion Prevention System (IPS) that uses the kernel level wrapper mechanisms to provide real-time prevention.

2.2: FSM-based Detection Methods

The STAT (State Transition Analysis Tool) [11][12][13] is one well-known IDS system that uses FSM to analyze intrusion behaviors. It has a modeling language that is used to describe penetration as a finite state machine. The STAT is a log investigation system, which uses FSM to describe attack scenarios and then feeds system logs into intrusion detection engine to determine whether the system has been attacked or not. However, attackers can stop being offensive and leave without being aware of.

There are many researches that use the concept of STAT. For example, in the work on STBIPW, a security platform is provided to allow users to define their own intrusion templates and to load them into system to do real-time detection. This system combines both the concepts of STAT and kernel level system call tracing mechanisms. It decomposes a penetration event into many states linked with critical system call transition.

2.3: Immunity-based Detection Methods

Human body is always being in contact with external materials. All these foreign materials may contain harmful invaders to human bodies. Fortunately, we have biological immune system that would detect and eliminate those foreign intruders. The major player in our immune system is the lymphocyte, which is more commonly known as the antibody.

To produce antibodies, the immune system first picks a random segment from the gene pool. However, not every gene segment is capable of being a lymphocyte. The gene segment is fitered with both positive selection and negative selection processes. The positive selection leaves behind those abnormal lymphocytes that cannot cooperate with other human cells. And the negative selection lets lymphocytes contact with human cells and filters out the active ones since they 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 are able to shoulder the important duty of epidemic prevention.

There are many researchers that proposed and designed IDS systems that embody the concept of human immunity [14][15][16]. The basic idea is to map certain computer characteristics as antibodies, and then cultivate random-chosen antibodies to be mature. For example, we can represent a network connection to be a byte stream with length L based on some basic information. We also define the sentries of IDS system as byte stream with length L. Those sentries correspond to lymphocytes in immune system and are responsible for detecting intrusions. When the byte stream representing certain network connection matches a detector, this connection might be dangerous.

2.4: Weaknesses of Signature-based IDS

To detect intrusions, signature-based IDS has to detect and match certain pattern exactly before it can judge the pattern as an attack. For this reason, attacking steps can be interleaved with normal patterns to accomplish invasion slowly and stealthily. These types of attacks are called Mimicry Attacks [2]. Another kind of mimicry attacks is called the collaborative attack. In order to escape detection, malicious process can fork another child process to carry out invasion. Most IDS systems will monitor related, forked actions and inspect child processes with equal emphasis to prevent such attacks. However, if both the parent and child processes finish parts of intrusion steps and exchange their results through Inter-Process Communication

(IPC), the attack will be undetectable by original methods. For example, if we define the following sequence of system calls as a simple intrusion pattern:

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

It will be undetectable by general IDS systems if we let the parent process do the open system call and let the child process finish the write.

3: System Requirements and Design Issues

In this section, we first introduce how to use negative selection mechanism to filter out unsuitable detectors, and then propose solutions to dispute collaborative and inherited attacks, which are of mimicry attack type.

3.1: Inspection of Improper Templates

In order to lower 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 the training phase, we collect system call sequences of normal actions to build the Normal Database. This is much like the method used in Anomaly IDS. However, the normal database in Anomaly IDS is used for intrusion detection. Contrarily, we use normal database to inspect user-defined patterns, as shown in **Figure 1**.

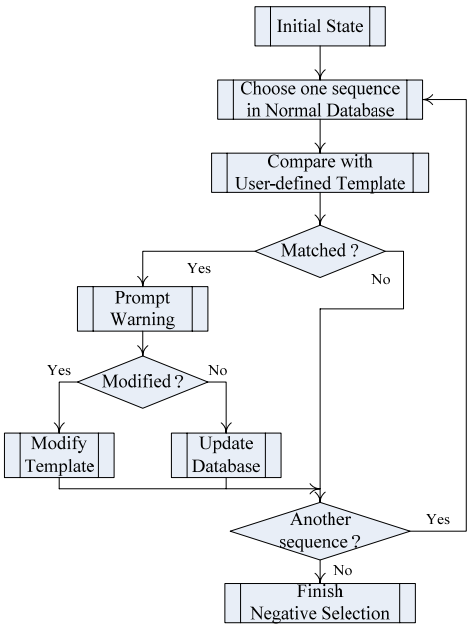

Figure 1. The negative selection procedure.

In the testing phase, if the comparison gets positive reactions, it means that this template might cause normal behaviors to be misinterpreted as abnormal ones. In this case, the system will warn users of this event. With the testing procedure inspired from human immune system, we can diminish the loss caused by improper template detectors.

3.2: Prevention of Collaborative and Inherited Attacks

When a process is being monitored, one FSM instance is created for each possible intrusion path to trace the process's status. All these related instances will be updated according to their execution situations. If this process issues a fork system call, its set of instances is duplicated to monitor its child. These newly created instances would all be in the initial state and the execution of child processes are traced independently. To prevent collaborative attack, our system will create another instance called the family instance. Each family instance will be in the same state the parent is in at the time the fork system call is issued. An example isshown in **Figure 2**.

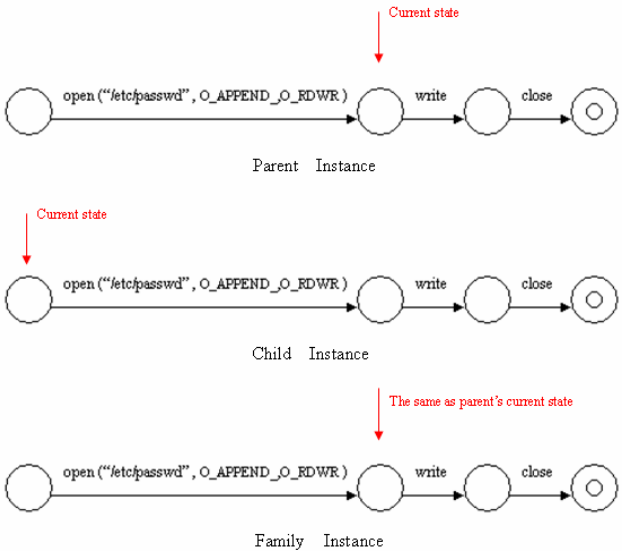

Figure 2. Example of family instance

After the fork system call, both the parent and child processes will have an individual set of monitoring instances to prevent them from launching independent attacks. In addition, the family instance is used instead to prevent collaborative attack. The instance will be updated when either the parent or the child requests a system call. In this way, no matter how this family divides their intrusion steps, the intrusion will be detected by our system.

4: System Architecture

Extending from 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 act as a bridge between these two segments. On one side, the driver passes commands and data to the underlying core engine for user configuration. On the other side, it also returns execution information back to users.

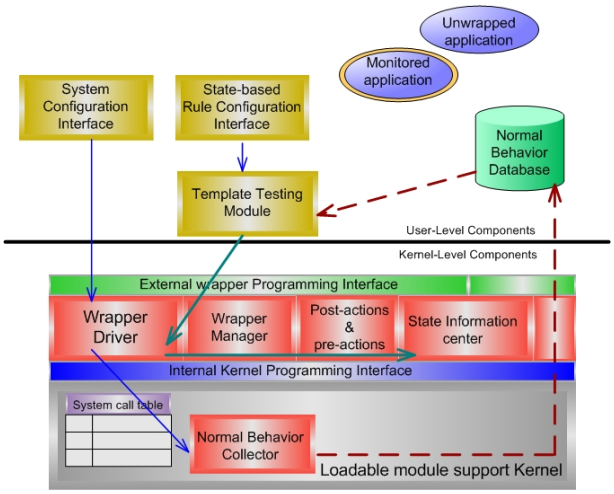

Figure 3. System Architecture.

4.1: User Level Components

The main user level module other than STBIPW is the Template Testing Module. The purpose is to make sure that no inappropriate attack templates are defined by the State-based Rule Configuration Interface that may cause a high false positive alarm rate.

The Template Testing Module first selects the normal system call sequence from the Normal Behavior Database, and then tests 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 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.

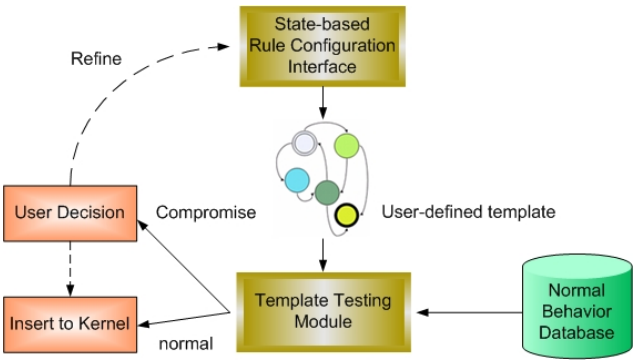

Figure 4. Template Testing Modules.

4.2: Kernel-Level Components

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

The Normal Behavior Database is constructed by the Normal Behavior Collector. This collector records the system call sequence requested by a normal user program in a secure environment. Via the Wrapper Driver, a user could set up capturing constraints in the Wrapper Manager to record normal system calls. The Wrapper Manager could then demand the Normal Behavior Collector to train an exclusive Normal Behavior Database for any application program according to the constraints. The user could decide both the duration of training and the size of the database. The longer the duration, or the larger the size, the more accurate the attack template test can be, resulting in reduced false positive alarm rate.

When the Wrapper Manager intercepts an exec system call and compares it with the supervising characteristics of the attack template, the manager can decide whether this process should be monitored or not. If the answer is positive, the Wrapper Manager will generate an FSM instance to monitor this process. Whenever the wrapped process requests the fork system call, the anti-collaborative attack module will generate an FSM instance for this newly created child process, and also generate an FSM family instance when this leading process forms a family. Afterwards, once the system intercepts a system call invoked by any process in this family, the collaborative attack module will verify the FSM instance for that specific process, as well as the family instance.

5: Experimental Result

In this section, experimental results are used to illustrate the efficiency and practicability of our system.

5.1: Runtime Overhead

The runtime overhead of our system is mainly due to state-transition. The first testing program opens a text file and copies it to another. All the read and write system calls will cause state-transitions in the FSM objects of the supervising program. The result is shown in **Figure 5**. The state-transition time is a stable constant around 1300μs. Therefore, the larger the file, the longer the time used for I/O operations; and the smaller the system's runtime overhead relatively, as shown in **Table 1**.

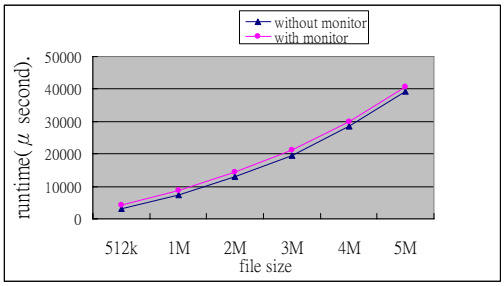

Figure 5. File size vs. Elapsed time.

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

Table 1. File size vs. System overhead.

However, the penalty increases if every system calls invoked by processes make FSM instances alter their states. In general, when the user customizes attack templates to detect intrusions, not every invoked system call of suspected process should make the FSM do transition. The critical number of key events that would change the status of the FSM instance when running in real situation should be relatively small compared with the total amount of invoked system calls of the monitored process. Therefore, in the second evaluation experiment, the program behaves the same as in the first experiment, but some extra behaviors are added to reduce the fraction of the key system calls that cause the state-transition in attack templates. **Figure 6** shows the result, which allows us to claim that the runtime overhead is approximately 0 while the ratio of the system calls causing the state-transition in attack templates is kept under 8%.

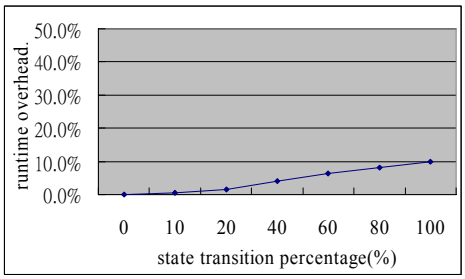

Figure 6. Percentage of critical system calls vs. overhead.

Next, the file-copying program is used again but we increased the number of monitoring templates to measure the variation of the runtime overhead against the number of monitoring templates. As shown in **Figure 7**, the runtime overhead remains roughly constant as the number of FSM instances increases. This is because even though the program is being monitored by many monitoring FSM instances, not many instances transit states simultaneously if the program is normally executed since different templates describe different intrusion behaviors and thus constitutes different system call sequences.

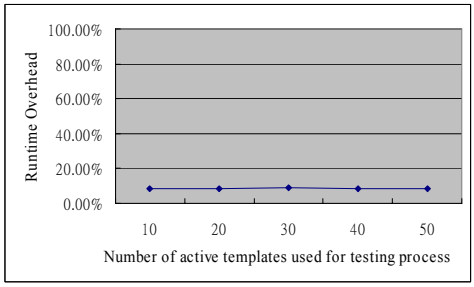

Figure 7. Number of monitored templates vs. overhead.

In the third experiment, we study the overhead variation due to the number of concurrently running processes and give the result in **Figure 8**. Each process is monitored by a single FSM instance derived from an identical template. We can see that while many processes are running simultaneously, the runtime overhead can also remain stable since only one system call is invoked by one process at any given time and the ratio of critical system calls for certain monitored process is fixed. Therefore, the overall runtime overhead is nearly fixed.

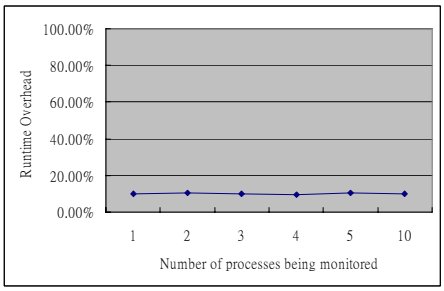

Figure 8. Number of concurrent monitored processes vs. overhead.

5.2: Experiments on Intrusion Detection of Collaborative Attacks

To detect collaborative attack, we assume that one simple attack template is defined as the figure below.

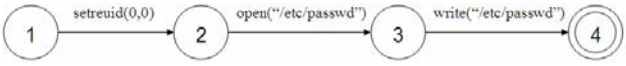

Figure 9. Original experiment template on collaborative attack.

We use one forged program to launch the following attacking system call sequence intended to test the anti-collaborative module.

Before fork：
    setreuid(0,0);
After fork：
    parent process：
        open("/etc/passwd");
    child proces：
        setreuid(0,0)
        write("/etc/passwd");

In this experiment, after the parent process executes the *setreuid* system call, the family FSM instance transits to state-2. Then, the *open* request of the parent process makes the instance to move to state-3. Finally, the *write* system call issued from the child process will update the family instance to move to the final state of "being compromised".

5.3: Precaution Test of Improper Templates

In this section, we use the sftp-server program as an example to demonstrate the capability of our proposed system in preventing improper user-defined templates from violating the usage patterns in a normal environment. In order to inspect the sftp-server program,

the Normal Behavior Collector is used to collect two days worth of normal behavior training session into a 2MB Normal Behavior Database. Then, we define an attack template, as shown in **Figure 10**, to prevent attackers from gaining the root authority and installing malicious Trojan horse programs by exploiting the loophole of the sftp-server program.

The attack template we defined is intended to prevent attackers from using loopholes to get the root authority. Attackers may create new file folders, modify privileges, and write invading programs, etc. However, after passing through Template Testing Modules, the FSM may still move to the final state under normal use. This shows that the attack template is not accurate enough and may sometimes treat a normal behavior as an assault, contributing to the rise of false positives.

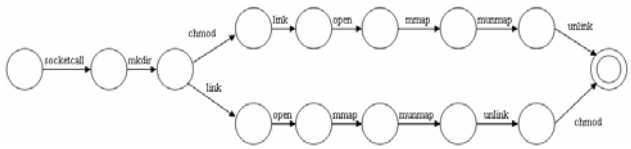

Figure 10. Improper sftp-server intrusion template.

6: Discussions and Conclusions

In this paper, continuing on the work on STBIPW, we come up with an intrusion prevention system which has the following two advantages over STBIPW:

(1) The reduction on the false positive rate. We use the negative selection mechanism to train a Normal Behavior Database to discriminate inappropriate attack templates defined by users.

(2) The discovery of collaborative attacks. Our system analyzes the original attack templates and constructs a monitoring FSM family instance that enables the detection of collaborative attacks.

REFERENCES

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

[3] 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.

[4] 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.

[5] Timothy Fraser, LeeBadger, Mark Feldman, "Hardening COTS Software with Generic Software Wrappers," in *Proceeding of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[6] Calvin Ko, Timothy Fraser, LeeBadger, Doublas Kilpatrick, "Detecting and Countering System Intrusions Using Software Wrapper," in *Proceedings of the 9th Usenix Security Symposium*, 2000.

[7] Mitchem, T., Lu R., O'Brien R., "Linux Kernel Loadable Wrappers," In *Proceedings of the DARPA In-formation Survivability Conference and Exposition*, 2000.

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

[9] S. Forrest , S. A. Hofmeyr , and A. Somayaji, "A sense of self for unix processes," in *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, 1996, pp.120-128.

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

[11] 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.

[12] Giovanni Vigna and Richard A. Kemmerer. "NetSTAT: A Network-based Intrusion Detection System", J*ournal of Computer Security*, 1999.

[13] Giovanni Vigna, Steve T. Eckmann, Richard A. Kemmerer. "The STAT Tool Suite," in *Proceedings of DISCEX 2000*, 2000.

[14] 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.

[15] 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*, vol.1, pp.28-31, 2002.

[16] 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.

[17] Yan Qiao, Xie Weixin,"A Network IDS with low false positive rate," in *Proceedings of the 2002 Congress on Evolutionary Computation*, vol.2, pp.1121-1126, 2002.

[18] 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.

[19] Massimo Bernaschi, Emanuele Gabrielli, Luigi V. Mancini. "REMUS: A Security-Enhan-ced Operating System". *ACM Transactions on Information and System Security*, 2002.

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

[21] 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.

[22] 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*, pp.146-161, 1999.

[23] Ian Goldberg, David Wanger, Randi Thomas, "A Secure Environment for Untrusted Helper Application," in *Proceedings of the 6th Usenix Security Symposium*, 1996.