A Useful Anomaly Intrusion Detection Method Using Variable-length Patterns and Average Hamming Distance

Ye Du
Department of Computer Engineering, School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
Email: ydu@bjtu.edu.cn

Ruhui Zhang
Department of Computer Engineering, School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
Email: zhangruhui1978@gmail.com

Youyan Guo
Information Management Center, Beijing Anzhen Hospital, Capital Medical University, Beijing, China
Email: youyanguo@sohu.com

***Abstract*—Intrusion detection techniques at the level of system processes are discussed, and a new method named V-AHD (Variable-length Average Hamming Distance) is presented, which can be used to monitor and calculate deviation to discriminate between normal and abnormal sequences of system calls. For the reason that fixed-length patterns can not describe the system behavior correctly and its inability to represent long meaningful substrings, the V-AHD method use Teiresias to get variable-length patterns and construct the normal set. Then the algorithm for detection is described in detail, and the pseudocode is also given. The method has some advantages, such as algorithm simplicity, low overhead of time, high accuracy and real-time detection. The prototype experiments with four attacks prove the validation of the method, which has high True Positive Rate and low False Positive Rate.**

***Index Terms*—anomaly intrusion detection, variable-length patterns, average hamming distance, system call**

I. INTRODUCTION

In recent years, the number of information attacks is increasing and malicious activity becomes more sophisticated. It is very important that the security mechanisms of a system are designed so as to prevent unauthorized access to system resources and data. Intrusion detection has been an active topic for many years, starting in 1980 with the publication of John Anderson's "Computer Security Threat Monitoring and Surveillance"[1]. In that paper, an intrusion attempt is the potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable. Since then, several techniques for detecting intrusions have been studied. In general, intrusion detection techniques can be categorized into anomaly detection and misuse detection. Here we focus on the former one.

The first approach, called anomaly detection, assumes that all intrusive activities are necessarily anomalous. The method defines and characterizes correct static form or acceptable dynamic behavior of the system, and then detects wrongful changes or wrongful behavior. It relies on being able to define desired form or behavior of the system and then to distinguish between that and undesired or anomalous behavior[2]. For example, if a user only uses the computer from his office between 9 am and 5 pm, an activity on his account late in the night is anomalous and hence, might be an intrusion. The main advantage of anomaly detection is that it does not require prior knowledge of intrusion and can thus detect new intrusions. By defining what is normal, it can identify any violation, whether is part of the threat model or not.

The second approach, called misuse detection, involves characterizing known ways to penetrate a system. It uses patterns of well-known attacks or weak spots of the system to match and identify known intrusions. The patterns may be static bit strings or describe suspect sets or sequences of actions. The method represents signature about suspicious behavior and seeks to detect it directly, as opposed to anomaly intrusion detection, which seeks to profile overall normal behaviors. The main advantage of misuse detection is that it can accurately and efficiently detect instances of known attacks.

In the work reported here, an anomaly intrusion detection method named V-AHD using variable-length patterns and average hamming distance (AHD) is brought forward to detect abnormal events.

The AHD is used to build a normal profile from the training data of normal activities, and then it is employed to detect anomalous activities from testing data of both

doi:10.4304/jcp.5.8.1219-1226

normal and intrusive activities on the computer for intrusion detection. Sequences of system calls are represented as a series of event transition in models. The detection process relies on the assumption that when an attack exploits vulnerabilities in the code, new subsequences of system calls will appear. For the reason that fixed-length patterns can not describe the system behavior correctly, we use variable-length patterns to construct the normal set.

The rest of the paper is organized as follows: Section 2 briefly reviews related work. Section 3 presents the way to get normal behavior patterns and AHD algorithm in our approach. Section 4 describes the details of our experiments, and analysis of results. Section 5 makes some conclusions and outlines some open issues for future work.

II. RELATED WORK

Some previous work on anomaly intrusion detection collects profiles of user behavior, generated by audit logs. Intrusions are detected when a user behaves out of character, such as IDES[4], NSTAT[5] or neural networks[6]. IDES[4] builds statistical profiles of users, though the entities monitored can also be workstations, network of workstations, remote hosts, groups of users, or application programs. IDES uses statistically unusual behavior to detect an intruder masquerading as a legitimate user. NSTAT (Network State Transition Analysis Tool)[5] collects the audit data from multiple hosts and combines the data into a single, chronological audit trail to be analyzed by a modified version of USTAT. To chronologically maintain the audit trail, each component sends a sync message periodically to make sure that the clocks are synchronized within some threshold. Tan[6] introduces an algorithm based on neural network, which uses information, such as command sets, CPU usage, login host addresses, to distinguish between normal and abnormal behavior. In a dynamic environment, it is nearly impossible to create user profile that determines normal behaviors.

An alternative approach is process-based intrusion detection, which is introduced by Forrest[7], identifies anomalous sequence of system calls executed by programs. Self-Nonself addresses the problem of assuring that static strings do not change. Again, some unchanging portion of code and data is defined to be the static string to be protected. The Self-Nonself signatures are for unwanted string values, that is, strings that might result if an unwanted change were to be made to the static system state. This algorithm uses fixed-length pattern which is called lookahead pairs. Next generation of modeling system call traces on process-based intrusion detection involves moving beyond lookahead pairs to Finite State automata (FSA).[8] A system call trace is the ordered sequence of system calls that a process performs during its execution. Three major detection approaches were proposed on previous research.[9] First, signature-based approach. In Ref.[5], a state-machine technique is applied to model the state of each process according to its current privileges, and violations of the system's security policy are identified as "forbidden" transitions. Second, it is specification-based approach that uses pattern language to detect. In Ref.[10], a high-level specifica- tion language called Auditing Specification Language (ASL) is developed for specifying normal and abnormal behaviors. These two approaches both are misuse detection, which have the same shortcomings lie in hard to create compact models of attacks and vulnerable to novel attacks. The third one is anomaly- based approach that was first proposed by Ref.[7]. Their work uses lookahead pairs to build normal usage of system call executed by program. For next research, they try to use other algorithm to build better normal profile. Ref.[10] uses data-mining technique, called RIPPER, to generate associate rule of normal profile. Ref.[16] builds a classifier with recurrent neural networks for system calls dataset. It is a good source on the comparison of recurrent neural networks and Multi-layered feed forward (MLFF) networks in terms of network structure, computational complexity, and classification performance. Artificial intelligence technique, like Hidden Markov Models (HMM)[11], is used to build more generalize normal system call behavior. HMM is well known as a good probabilistic method to model temporal sequences, and is largely exploited in the speech recognition field, but it requires huge amounts of time to model normal behaviors and detect intrusions. Unlike most researchers who concentrated on building individual program profiles, Asaka[12] introduces a method based on discriminate analysis. Without examining all system calls, an intrusion detection decision was made by analyzing only 11 system calls in a running program. Due to its small size of sample data, however, the feasibility of it still needs to be established[13]. A k-Nearest Neighbor (kNN) classifier based approach is introduced by Liao[13] to classify program behavior as normal or intrusive. Program behavior, in turn, is represented by frequencies of system calls. Each system call is treated as a word and the collection of system calls over each program execution as a document. These documents are then classified using kNN classifier, a popular method in text categorization. Ye[19] attempts to compare the intrusion detection performance of methods that used system call frequencies and those that used the ordering of system calls. The names of system calls were extracted from the audit data of both normal and intrusive runs, and labeled as normal and intrusive respectively. Since both the frequencies and the ordering of system calls are program dependent, this over simplification limits the impact of their work[13]. The technique we proposed here is an anomaly detection method, which is primarily focuses on the accuracy of detection, while not on building normal profile

III. METHOD

The course of our method can be divided into two stages: building up normal variable-length pattern database and detecting in real environment. In the first stage, we build normal profile by collecting system calls of some interested process and extracting patterns with Teiresias[14], an algorithm initially developed for

discovering rigid patterns in unaligned biological sequences. It is a phase of training. In the second stage, we use that profile to monitor system behavior, calculate the average hamming distance (AHD) between them, which determines the strength of an anomalous signal. If it exceeds the threshold value T that was set before, a suspicious event happens.

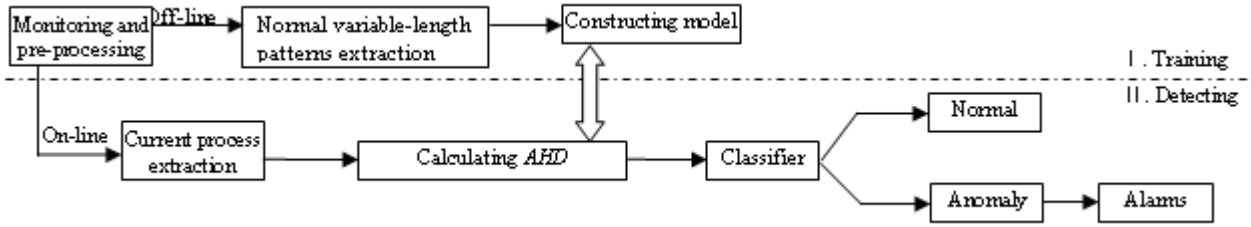
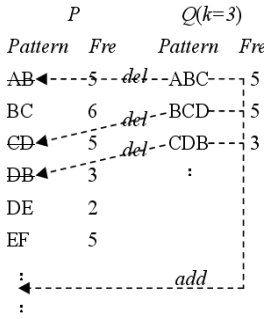


Figure 1. Flow of detection.

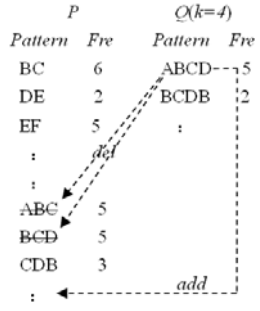
A. Normal Variable-length Patterns Database

Suppose we observe the following system call $S=\{S_A,S_B,S_C,S_D,S_B,S_C,S_D,S_E,S_F,S_G,\ldots\ldots\}$, S_I means one system call such as *lseek*, *read* etc. Initialize sliding window length k equals 2, and take out the iterant sequences, we get some patterns whose length is 2. Recodes the frequency of each pattern and add it into normal database P. We use P to denote the patterns set, and Teiresias method is used here.

If k is smaller than n (n is the length of S or a predefined value which is bigger than 2), initializes sliding window length k equals k+1, and repeats the extraction steps. Here, we use Q to denote the patterns set. $\forall q_i \in Q$, $\exists(p_j \in P, p_j \in q_i)$, which means p_j is the substring of q_i. If the sequence of p_j is smaller than or equal to the sequence of q_i, then we delete p_j from the normal database, and add q_i in it. Repeats the steps until all the patterns in Q are included. The extraction (A stand for S_A, B stand for S_B, etc.) is shown in Fig.2. Only the processes of k=3 and k=4 are described, while others are omitted.



(a) Extraction k=2 and k=3



(b) Extraction k=2 and k=4

P

Pattern	Fre
BC	6
DE	2
EF	5
:	
:	
CDB	3
ABCD	5
BCDB	2
:	

(c) Result

Figure 2. The process of building up normal variable-length database.

B. Algorithm

Suppose there are m variable-length patterns stored in normal database P, that is $|P|=m$. One sequence $S=\{S_1,S_2,S_3,\ldots\}$ is to be checked for suspicious or not, S_i is a system call such as *write* or *open* etc.

$\forall p \in P, \max(|p|) = l$, that is the number of system calls one pattern in normal database contains can be up to l. $p_i=\{S_i,S_{i+1},\ldots,S_{i+m-1}\}$. Our method can be described as the following steps.

(1) Initialize x equals to 1. x is a counter which points to the place in detected sequence.

(2) Initialize n equals to 1, and D equals to 0. n is a counter that points to the place in normal database. D is the cumulative result of *AHD*.

(3) If the length of pending tested sequence is smaller than l, delay.

(4) For $|p_n|=w$ ($w \le l$), we get w system calls from the place x, $p_n'=\{S_x,S_{x+1},\ldots,S_{x+w-1}\}$.

(5) Caculate the distance of p_n and p_n' . The formula of $dis(p_n, p_n')$ is shown in below.

(6) If $dis(p_n, p_n')$ equals to 0, that means pattern p_n and detected sequence p_n' is matched, and p_n' is a permitted system behavior. x equals to x plus w, goto(2).

(7) If $dis(p_n, p_n')$ is not equal to 0, which demonstrates p_n' is malicious. Then we plus D and $dis(p_n, p_n')$, and assign the result to D. n equals to n plus 1.

(8) If $n \leq m$, goto (4), else goto (9).
(9) We calculate the average distance *AHD*. If *AHD* is bigger than the threshold we set before, alarm may be generated.
(10) x is set to x plus 1. Then goto (2).

For the purpose of understanding above method fully, the pseudocode of our method is written as follows.

(1) set x=1;
(2) set n=1, D=0;
(3) if getlength(*detected sequence*) <l {wait( ); goto (3)};
(4) w=getlength(p_n);
p_n'= getsystemcalls(x,w); // p_n'= {S_x,S_{x+1},…, S_{x+w-1}}
(5) d= *dis*(p_n, p_n')
(6) if d=0 {x=x+w; goto (4)}; else goto(7);
(7) D=D+d; n=n+1;
(8) if $n \leq m$, goto (4), else goto (9)
(9) A=avedistance(x,D);
If $A \geq T$, alert( );
(10) x=x+1; goto (2);

Functions used in the method are as bellows.

Function *dis*(p_n, p_n'):

$$dis(p_n, p_n') = \frac{\sum_{i=1}^{w} H(p_{ni}, p_{ni}')}{w} \quad (1)$$

$$= \frac{1}{w} (number\ of\ positions\ p_{ni} \ and\ p_{ni}'\ are\ differ)$$

Function avedistance(x,D):

$$\overline{D} = \frac{D}{m} = \frac{\sum_{i=1}^{w} H(p_{ni}, p_{ni}')}{wm} \quad (2)$$

$$avedistance( ) = \frac{1}{x} \sum_{i=1}^{x} \overline{D} \quad (3)$$

C. Benefits of Using Privileged Process and Variable-Length Patterns

The easiest way to collect data for detection is to use system log files. In UNIX systems, user log information such as wtmp, utmp, and sulog files is in /var/log/message or /var/adm/ directory. The basic log files can be easily obtained without any significant effort[25]. However, the type of data source can only be used for offline extraction and analysis, and the successful and careful breaker may even erase his footprints logged in system. Because of the above shortcomings, system call level audit data are used.

Privileged processes are running programs that perform services (such as sending or receiving mail), which require access to system resources that are inaccessible to the ordinary user. To enable these processes to perform their jobs, they are given privileges over and above those of an ordinary user (even though they can be invoked by ordinary users)[20]. In operating systems such as Unix, the privileged processes that only the superuser is authorized to execute make use of system calls to access privileged system resources or services. Such privileged processes are often a major target for intruders[3]. For example, privileged programs include telnetd and logind, function as servers that control access into the system. Corruption of these servers can allow an intruder to access the system remotely. So, system calls corresponding to these processes are adopted as data sets.

- The way for most intrusive activities to penetrate the system security is by wrecking the normal mode of privileged processes. In other words, privileged process is one of main channels for attacker to gain certain authority.
- The behavior space of privileged process is numbered, which helps to describe the relation between superficies and inherence. On the other hand, the behavior space of user activity is hard to bewrite, for the activity modes change greatly with time.
- Intuitively one system call sequence is correspondence to special procedure function. Privileged processes are all usually focus on given and finite purposes, so the activity mode is more stable than user programs. By putting a numerical value to each system call, we can look on the system call sequences as time series.

To describe the normal behavior of a process, variable-length patterns appear to be more naturally suitable than fixed-length patterns. When using fixed-length patterns, the "fixed" length needs to be chosen first.

- Long patterns are expected to be more process-specified than short patterns. However with the increasing of pattern length, the size of normal database is also increasing rapidly, which makes the overhead of calculation huge.
- Short pattern may help us reduce the amount of computation. But the capability of tracking system behaviors is also diminished. For example, if the length equals to 2, most system call sequences are viewed as normal.

Using variable-length patterns can deal with above apparently contradictory constraints.

D. Value of Threshold

There are always two ways to define the threshold. One is using empirical value and then gradually corrects it with self-learning algorithm such as artificial intelligence or machine learning methods; the other is based on experimental results which were done in advance. In our method, the former way is adopted. We first pre-out part of the training data and divide it into 30 sample sets. After calculating the V-AHD between them and normal database, 10 random values were taken out to compute the average value. Then we set it as the threshold.

IV. EXPERIMENT RESULTS AND ANALYSIS

There are two prerequisites in adoption of our method: 1. When a process runs normally, the sequences of system calls are stable. That means the behavior database, after training of a period of time, ought to contain all possible normal short sequences, and the size of it cannot be large in order to save the cost of time and space. 2. When vulnerabilities are exploited, abnormal system call sequences come out. The *AHD* value between them and normal database must be great. Then, we can assume this fact is a sign of an abnormal incident. In the following section, we will prove these two conditions.

Two methods can be used for choosing the normal behavior that is used to define the normal database:

- Generating a "synthetic" normal by exercising a program in as many normal modes as possible and tracing its behavior;
- generating a "real" normal by tracing the normal behavior of a program in a live user environment

Here, we choose publicly available system call sequences from the University of New Mexico (UNM) as our experimental data sets. Forrest[20] studied normal behavior for three different programs in UNIX: sendmail, lpr and wu.ftpd. Sendmail is a program that sends and receives mail, lpr is a program that enables users to print documents on a printer, and ftpd is a program for the transfer of files between local and remote hosts.

Compared with other processes, system calls of sendmail is more complicated. So the data sets tested are "synthetic sendmail" and "synthetic sendmail CERT". Synthetic data were collected at UNM on Sun SPARC stations running unpatched SunOS 4.1.1 and 4.1.4 with the included sendmail. UNM considered variations in message length, number of messages, message content (text, binary, encoded, encrypted), message subject line, who sent the mail, who received the mail, and mailers. In addition, UNM looked at the effects of forwarding, bounced mail and queuing. Lastly, they also considered the effects of the origin of all these variations in the cases of remote and local delivery.

- Normal traces: traces of sendmail daemon.
- Abnormal traces: traces of syslog-remote intrusion, syslog-local intrusion, decode intrusion, and sm565a intrusion.

With the increase of normal trace quantity in real time, the number of variable-length patterns in normal database is shown in table 1.

In the beginning of collecting, the number of patterns in database increases quickly. When the number of system calls is bigger than 1500, the number of patterns increases slowly or no increasing. In table 1, the max value of system calls is 5000, while in experiment, the test value of C is 15000, and only after it is bigger than 9300, the patterns size adds 2 positions. The test indicates that although amount of system calls keep extending constantly, the patterns in database do not increase limitlessly. On the contrary, it is almost certainly convergent.

TABLE I. NUMBER OF VARIABLE-LENGTH PATTERNS IN NORMAL DATABASE

	Number of system calls in real time										
	100	200	400	600	800	1000	1200	1500	2000	3000	5000
Number of patterns	36	116	241	282	301	314	363	403	404	405	405

To examine the validity, we first see if it can distinguish self-process *sendmail* from other processes, then detect four kinds of attacks.

TABLE II. DISTINGUISHING SENDMAIL FROM OTHER PROCESSES

Process	sendmail	lpr	ftp	inetd
AHD	0	0.926719	0.890161	0.989235

From the table, we can see that the result of *AHD* can distinguish self and non-self processes accurately, for the differ between them is huge. *AHD* between sendmail normal database and other processes are great, while equals to 0 with sendmail process. It is because all system calls of sendmail process are included in database.

The following experiments are detecting *syslog*, *sm5x*, *Decode* intrusions. *Syslog* attack can be run either locally (*syslog-local*) or remotely (*syslog-remote*); we have tested both modes. Results are shown in Fig.3 and Fig. 4.

The x axis indicates the number of system calls traced, with unit of which is 100; the y axis indicates the value of *AHD*. We collected 1500 system calls. From figures, it is clear to see the changing process of intrusions. Based on the threshold established before, attacks can be detected. In Fig.3 and Fig.4, we separately set threshold T_1=0.75, T_2=0.83, these four kinds of intrusions were all detected. In experiments, we still test *lprcp*, *wu-ftp* attacks, and results are distinct.

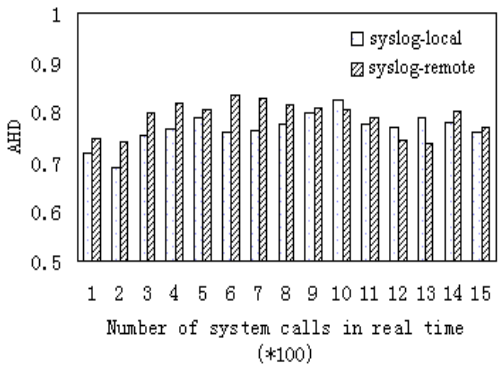


Figure 3. Detecting syslog attack.

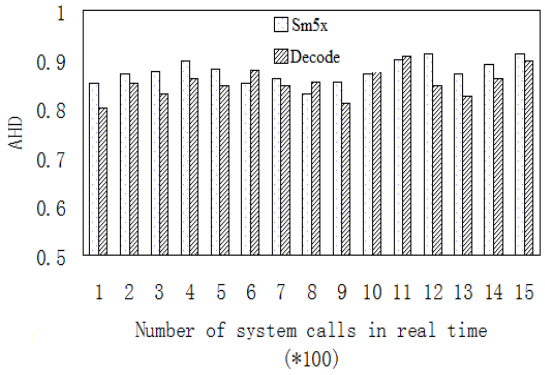


Figure 4. Detecting Sm5x attack and Decode attack.

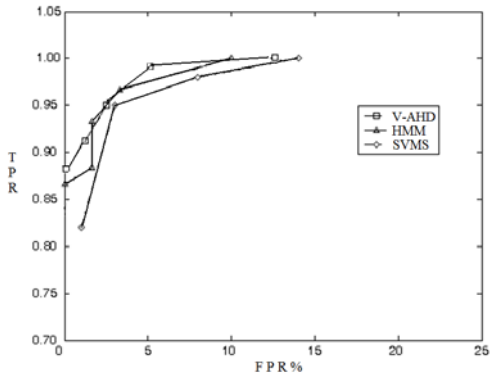


Figure 5. Performance of V-AHD，HMM and SVMS Methods Expressed in ROC Curve.

When evaluating an algorithm for intrusion detection, four indicators are always used, which include True negative (TN), False positive (FP), False negative (FN), and True positive (TP). True negatives as well as true positives correspond to a correct operation of the method; that is, events are successfully labeled as normal and attacks, respectively. False positives refer to normal events being predicted as attacks; false negatives are attack events incorrectly predicted as normal events. Based on the above indicators, a numerical evaluation can apply the following measures to quantify the performance of an algorithm[23]:

- True Positive Rate (TPR) = TP/(TP+FN), the probability that an intrusion attempt in the environment is detected.
- False Positive Rate (FPR) = FP/(TN+FP), the probability of an alarm given no intrusion.
- False Negative Rate (FNR) = FN/(TP+FN), the probability of no alarm given an actual intrusion.

TABLE III. TPR & FPR & FNR

Attack	TPR(%)	FPR(%)	FNR(%)
syslog-local	97.9	1.6	2.1
syslog-remote	98.2	2.1	1.8
sm5x	98.7	1.9	1.3
Decode	98.1	2.2	1.9

Experiment data show that the method can find these four attacks precisely, with the lowest TPR equals to 97.9% and the highest FPR equals to 2.2%. Compared to other privileged process, *sendmail* is more complicated. So it can be used as examples to illustrate the usability when detecting.

ROC curves are often used to identify the quality of an automated system where some criteria can be changed in order to affect the quality of the system. In assessing an anomaly intrusion detection method, we are usually interested in looking at how change in the ruleset affect the detection probability (TPR) and the probability for a false alarm (FPR).

To V-AHD, when the FPR is 0, the TPR is 87.56%. Then the threshold value is change to be lower, and when the TPR is 100%, the FPR is 12.9%. It is clear to see that the detection effect of our method V-AHD is higher than SVMS, while similar to HMM. However, the major drawback of HMM is the time consumption of training and detection. For example, to the lpr process with about 700,000 system calls, it will cost HMM method 5 days to train and extraction. While it just cost V-AHD method about 90 minutes to do the same thing.

The performance comparison between V-AHD and some common used methods is shown in the following table, which includes CTBIPS[22] and RSVMs[23]. We can see that the True Positive Rate of V-AHD is higher than other methods, and the False Positive Rate is similar to them. To RSVMs method, the False Positive Rate is very low but the True Positive Rate is also very low. So, it is not suitable.

TABLE IV. COMPARISON OF V-AHD AND OTHER METHODS

Method	The Best TPR%	The Best FPR%
V-AHD	98.7	1.3
HMM	96.67	1.67
CTBIPS	97.37	3.59
RSVMs	81.8	1

Experiments show that, in attainable attack databases, V-AHD method has pleased detection achievements without safety vulnerabilities. Compared with other processes, system calls of *sendmail* is more complicated. Thus, by tests on that process, V-AHD can have high accuracy to detect other processes and unknown attacks. Also, we will do more experiments to validate it. While on-line detecting, V-AHD finds attacks at the very beginning of they work, which avoids the disadvantages of afterwards detection and allows administrators to take actions to eliminate damages to system.

V. CONCLUSION

This paper presents a process level based anomaly intrusion detection method V-AHD, which monitors short sequences of system calls in processes and calculates deviation. For solving the shortcomings of fixed-length patterns, V-AHD uses variable-length patterns to form the normal database. We have explained the limitation of the fixed-length pattern approach, namely its inability to represent long meaningful substrings. For the purpose of understanding the process of V-AHD, we have written the pseudocode of it.

The algorithm is simple, feasible, low overhead of time and meets the requirements of detection in real-time. Detection results between self-process and other processes or between normal processes and intrusive processes are different obviously, which indicates high accuracy. The method, through experiments, is not only available in theory, but also able to construct a lightweight intrusion detection system. Further work will concentrate on enhancing the algorithm for variable-length generation of patterns in order to reduce the risk of obtaining false positive alarms while keeping the good results regarding the true positive rate. In addition, we need to establish normal behavior databases of more processes, expand the range of detection, and change this method into a real applicable facility.

APPENDIX A

This appendix describes detailed information for intrusions that Forrest selected at UNM and we used as data source.

syslogd intrusion: syslogd provides a kind of logging mechanism to issue warnings that will be displayed. Every logged message contains at least a time and a hostname field, normally a program name field. There is the potential for the syslogd daemon to be utilized as a way to actualize the denial of service attack. A vicious program could very easily flood the syslogd daemon with large numbers of messages which depletes all the remaining resources on the filesystem. The syslogd attack here uses the syslog interface to overflow a buffer in sendmail. A message is sent to the sendmail on the victim machine, causing it to log a very long, specially created error message. The log entry overflows a buffer in sendmail, replacing part of sendmail's running image with the attacker's machine code. This attack can be run either locally or remotely.

decode intrusion: a common configuration for older mail transfer agents is to include an alias for the decode user. In older versions of sendmail, the uuencode and uudecode programs (used to decode and encode email messages respectively) are referenced in the /etc/aliases file. This file is used by sendmail to either properly route email messages to mail accounts or perform certain actions when mail is sent to a particular email address. uudecode respects absolute filenames, so if a file "bar.uu" says that the original file is "/home/foo/.rhosts" then when uudecode is given "bar.uu", it will attempt to create foo's .rhosts file. A remote attacker can send mail to the decode alias that is present on some systems to create or overwrite files on the remote host. This allows an attacker to gain remote access to the system.

ACKNOWLEDGMENT

This work is supported by National High Technology Research and Development Program of China (project No. 2007AA01Z410, 2007AA01Z177), National Key Basic Research Program (project No. 2007CB307101), Program for Changjiang Scholars and Innovative Research Team in University, National Nature Science Foundation of China (project No. K09A300150). Some of this work was performed whilst the author was at Harbin Engineering University, where a very helpful support staff provided a good environment for research and experimentation. Many people have contributed important ideas and suggestions for this paper, including Huiqiang Wang, Yonggang Pang, Tong Wang, Zhonglan Yuan, Meihong Li, Yongzhong He, The authors are grateful for the anonymous reviewers who made constructive comments.

REFERENCES

[1] J. P. Anderson, “Computer security threat monitoring and surveillance,” James P. Anderson Co., Fort Washington, 1980

[2] Anita K. Jones, Robert S. Sielken, “computer system intrusion detection: a survey,” http://www.cs.virginia.edu/~jones/IDS.../jones-sielken-survey-v11.pdf

[3] Dit-Yan Yeung, Yuxin Ding, “Host-based intrusion detection using dynamic and static behavioral models,” Pattern Recognition, vol. 36, no.1, pp. 229-243, 2003

[4] Lunt T F, Tamaru A, and Gilham F, “A real-time intrusion detection expert system (IDES),” Computer Science Laboratory(SRI International, Menlo Park, Calif.):Final Technical Report SRI-CSL-92-05, 1992

[5] Kemmerer, R.A, “NSTAT: a model-based real-time network intrusion detection system,” University of California-Santa Barbara, Technical Report TRCS97-18, November 1997.

[6] K. Tan, “The application of neural networks to unix computer security,” Proceedings of IEEE International Conference on Neural Networks, vol. 1, pp. 476–481, Nov.1995

[7] S. Forrest, S. A. Hofmeyr, and A. Somayaji, “A sense of self for unix process,” Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, Orkland California, pp. 120-128, 1996

[8] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, “A fast automaton-based method for detecting anomalous program behaviors,” IEEE Symposium on Security and Privacy, pp. 144-155, 2001

[9] chowalit Tinnagonsutibout, pirawat watanapongse, “A novel approach to process-based intrusion detection system using read-sequence finite state automata with inbound byte profiler,” http:// pindex.ku.ac.th/file_research/ Full2.pdf

[10] N. Nuansri, “A process state-transition analysis and its application to intrusion detection,” Proceedings of the 15th Annual Computer Security Applications Conference, Arizona, USA, 1999

[11] R. Sekar, T. Bowen, and M. Segal, “On preventing intrusions by process behavior monitoring,” Proceedings of the Workshop on Intrusion Detection and Network Monitoring, California, USA, 1999

[12] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa and S. Goto, “A new intrusion detection method based on discriminant analysis,” IEEE trans. inf. & syst., Vol. E84-D, pp. 570-577, 2001

[13] Y. Liao, V. R. Vemuri, “Use of K-Nearest neighbor classifier for intrusion detection,” Computer & Security, vol. 21, pp. 439-448, 2002

[14] Andreas Wespi, Marc Dacier, and Herve Debar, “An intrusion detection system based on the teiresias pattern discovery algorithm,” Proceedings of EICAR, 1999

[15] Rune Hammersland, "ROC in Assessing IDS Quality," http:// rune.hammersland.net/tekst/roc.pdf
[16] M. Al-Subaie, M. Zulkernine, "The power of temporal pattern processing in anomaly intrusion detection," Proceedings of IEEE International Conference on Communications, Glasgow, Scotland, pp. 1391-1398, June 2007
[17] Calvin Ko, George Fink, and Karl Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring," Proceedings of the 10th Annual Computer Security Applications Conference, Los Alamitos, CA, vol.8, pp. 134-144, 1994.
[18] Sandeep Kumar, Eugene H. Spafford, "A pattern matching model for misuse intrusion detection," Proceedings of the 17th National Computer Security Conference, Baltimore MD, USA, pp.11–21, 1994
[19] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," IEEE Trans. SMC-A, vol.31, pp. 266-274, 2001
[20] Hofmeyer, S.A., S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," Revised: December 17, 1997. http://www.cs.unm.edu/~steveah/ publications/ids.ps.gz.
[21] Forrest, S., L. Allen, A.S. Perelson, and R. Cherukuri, "Self-Nonself discrimination in a computer," Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, 1994.
[22] Lihong Yao, Xiaochao Zi, "Research of system call based intrusion detection," Chinese Journal of Electronics, vol.31, pp. 1134-1137, 2003
[23] Shelly Xiaonan Wu, Wolfgang Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," Applied Soft Computing, vol. 10, pp. 1-35, 2010
[24] UNM synthethic sendmail data, http://www.cs.unm.edu/ ~immsec/data/synth-sm.html
[25] Sung-Bae Cho, Hyuk-Jang Park. "Efficient anomaly detection by modeling privilege flows using hidden Markov model," Computers & Security, Vol.22, pp.45-55, Jan. 2003
[26] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," Computers & Security, vol.28, pp. 18-28, 2009

Ye Du, Hebei, China, 1978. He is a lecture and master supervisor of Beijing Jiaotong University, Beijing, China. He received his Ph.D. degree in computer science and technology from Harbin Engineering University, Harbin, China.

He studied network and system security, distributed computing, parallel computer architecture, etc. And his current research interests focus on intrusion detection and response, distributed systems security, fault tolerant, and disaster recovery. He has published a book named Network Attack and Defense: Principles and Practice (Wuhan, China: Wuhan University Press, 2008). In recent years, he has published over thirty papers in international journals and conferences.

Ruhui Zhang, Hebei, China, 1978. She is a postdoctoral fellow in Beijing Jiaotong University. She received her PH.D. degree in Information Systems Engineering from Kochi University of Technology, Kochi, Japan.

She studied network security, distributed systems, etc. And her current research interests focus on network security system design and evaluation, intrusion response. She has published many papers in international journals and conferences.

Youyan Guo, Beijing, China, 1969. She is an associate professor and master supervisor of Information Management Center, Beijing Anzhen Hospital. She received her master degree in computer science from Beijing Jiaotong University, Beijing, China.

She studied information security, artificial intelligence, etc. And her current research interests focus on network security, distributed object computing. She has published many papers in computer science.