Fuzzy Intrusion Detection System via Data Mining Technique With Sequences of System Calls

Mohammad Akbarpour Sekeh
Department of Computer System and Communication
Faculty of Computer Science and Information, UTM
Skudai, Malaysia
Email: moh_akb_s@yahoo.com

Mohd. Aizaini bin Maarof
Department of Computer System and Communication
Faculty of Computer Science and Information, UTM
Skudai, Malaysia
Email: maarofma@fsksm.utm.my

***Abstract*: There are two main approaches for implementing IDS; host based and network based. While the former is implemented in the form of software deployed on a host, the latter, usually is built as a hardware product with its own hardware platform (IDS appliance). In this paper, a host based intrusion detection system, that uses the idea of tracing system calls, is introduced. As a program runs, it uses the services of the underlying operating system to do some system calls. This system does not exactly need to know the program codes of each process. Normal and intrusive behaviors are collected with gathering the sequences of system calls for each process. Analysis of data is done via data mining and fuzzy techniques. Data mining is used to extract the normal behavior. The proposed system is shown to improve the performance, and decrease size of database, time complexity, and the rate of false alarms.**

***Keywords**: Process-based Intrusion Detection, Data mining, Fuzzy, Operating system, system calls, kernel*

I. INTRODUCTION

Apart from anti viruses, firewalls and other security devices to provide complete security, Intrusion Detection Systems are also required. Some of the software vulnerability such as buffer over flow in UNIX, bugs in Microsoft internet explorer and operating system still are bothering the users of the modern software. When an intruder penetrates the firewall or the preventing security facilities, IDS can come in handy. Hence, Intrusion Detection Systems (IDS) can incrementally improve the security in our operating system and other software.

There are three approaches of intrusion detection, Network-based IDS, Host-based, and Process-based IDS. The last one appears to be one type of Host-based which has been concentrated in this paper.

Initially in 1996, Forrest [2] proposed process-based intrusion detection system. She traced the particular processes of UNIX for gathering sequences of system calls in normal mode and she also put the system in the unsafe mode and executed the intrusive processes for gathering intrusive behaviors. After that normal and abnormal behaviors are collected and can be analyzed to detect intrusion. All of the dataset are in http:/cs.unm.edu/~immsec. Wagner [3], Martin and others [5], and A.Hafmeyr [1] have continued her work.

In process-based Intrusion Detection System, because of detecting the new danger behavior, method of anomaly detection has been presented. Therefore, normal rules should be extracted from normal datasets and we will be able to detect behavior of the new processes (normal and abnormal).

All of used system calls in running process were saved one after another in a file. For example *open, fstat, ioctl, mmap, write, open* is the normal sequence of system calls which has been saved in a large database. We can consider the sequences of system calls and extract many unique rules from this. In a simple finding with a sliding window with size k=3, there will be following sequences:

1- open, fstat, ioctl	*2- fstat, ioctl, mmap*
3- ioctl, mmap, write	*4-mmap, write, open*

Most important problem of this method is size of database (the space complexity in this method is $S(n) \in \theta(nk)$ that n is number of executed system calls in process). Hence, at the first we intend to find a method to reduce it.

II. THE PROPOSED ANOMALY DETECTOR

In this paper, we propose a process-based Intrusion Detection System, which is implemented via an anomaly-based methodology. Some of the required datasets are downloaded from UNM and some sequences of system calls have been traced with *strace* command in Linux, too. We used these sequences of system calls for building normal and intrusive database. All of the normal sequences are entered as input data to our system "Fig. 1". In our proposed system, to detect intrusion the following steps have to be executed:

1. Gathering normal sequences of system calls from the particular running processes in safe mode (UNM dataset).
2. Gathering intrusive sequences of system calls while the intrusive process is running (UNM dataset).
3. Analyzing normal and abnormal database to extract some unique rules (Datamining).
4. Comparing the sequences of system calls of new running process with normal dataset (Fuzzy Anomaly Detector).
5. Finally, intrusion is detected if there are some differences and mismatches (Fuzzy output) and otherwise it can be normal sequence.

In this paper, the particular used processes are *stide, named, xlock, ps,* and *login* in Linux platform which the processes were executed several times and used system calls were saved in the file. UNM University has collected normal and abnormal behavior for these processes and these dataset are downloadable from the site. Considered intrusive processes are *lprcp*, *buffer overflow*, and *sunsendmailcp*. Since intrusive processes do not follow a normal sequence, we can compare new system calls with normal dataset to detect mismatches and intrusions [1].

III. IMPLEMENTATION

As shown in "Fig. 1", the following steps should be executed to detect intrusion:

1. In the 'Normal traces' step, all of the normal system calls are collected. We have used UNM dataset for this step.
2. In the 'Data mining' step, the *association rule* method of Data mining is used and all of the normal rules are extracted.
3. In the 'FSM' step, all of the extracted rules are saved in finite state machine with a special format. The rules will be used in 'Anomaly Detector' step.

All of the above steps are executed off-line and they will take large time, But the remaining steps will be ran in online.

4. In the 'Strace' step, the kernel of operating system extracts all of the system calls, which they are called via running process. These sequences go to Anomaly Detector as Input data.
5. In the 'Anomaly Detector' step, there is a Fuzzy system which computes fuzzy inputs (Fuzzy sets) and creates the fuzzy output as alarm to administrator. In this step, new sequences are compared with FSM normal rules and if number of mismatches were larger than a threshold, it is recognized as a dubious sequence.

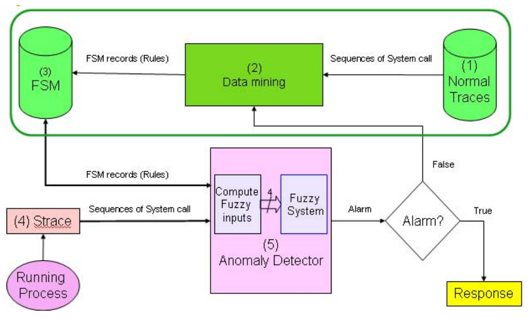
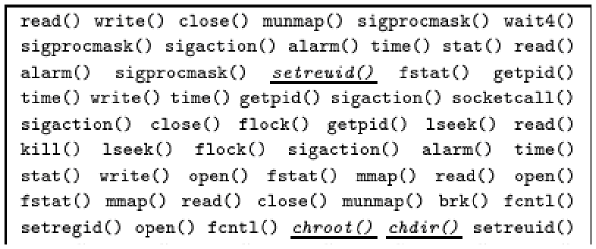


Figure 1. The proposed model for detect intrusion

A. Data mining

While a process is running, some system calls are executed. The "Fig. 2" shows a trace of sequences of system calls for *sendmail* process. We have used Data mining and *Association rules* method for extracting invisible rules in the sequences of system calls. At the first, the *support* and *confidence* degree were computed for each itemset. (Example of itemset is sequences of system call *open, read, getrlimit, mmap, and close*.).



```
read() write() close() munmap() sigprocmask() wait4()
sigprocmask() sigaction() alarm() time() stat() read()
alarm() sigprocmask() setreuid() fstat() getpid()
time() write() time() getpid() sigaction() socketcall()
sigaction() close() flock() getpid() lseek() read()
kill() lseek() flock() sigaction() alarm() time()
stat() write() open() fstat() mmap() read() open()
fstat() mmap() read() close() munmap() brk() fcntl()
setregid() open() fcntl() chroot() chdir() setreuid()
```

Figure 2. Sequences of system calls for a particular process

The following formula is used for computing *support* and *confidence*:

$$S = \textit{support}\,(x \cup y)$$

$$\textit{confidence} = c = \frac{\text{support}\,(x \cup y)}{\text{support}\,(x)}$$

Here are some simple examples of association rules:

open → *read* , c, s : 5 → 3 , c, s
open → *getrlimit* , c, s : 5 → 76 , c, s
mmap → *close* , c, s : 90 → 6 , c, s

A unique number has been assigned in each of the system calls as identifier of system calls (open= 5, read= 3, getrlimit= 76, mmap= 90, close= 6). In the example just *read* and *getrlimit* come after *open* system calls and thus if another system calls come after *open*, the system will recognize it as mismatch sequence. Therefore, data mining technique extract all of the normal rules and they should be saved in the finite state machine "Fig. 3".

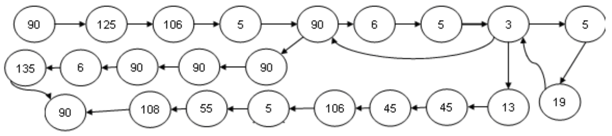


Figure 3. Saving sequences of system calls of process in FSM

After running the process several times, unique sequences of system calls were recognized which they frequently were being executed. In this section, our goal is finding all of the unique sequences.

Note: all of the mismatch sequence is not intrusive behavior, and it may be a part of error code and legal sequence, thus distinguishing between legal sequence and intrusive sequence is not easy.

B. Fuzzy System

In this paper, we have used Fuzzy to simulate and emulation of human thought for detecting intrusion. We would like to create a system, which acts like an expert human. One of the earliest methods is the rule-based system method. In the method, "*If* . . . *Then* . . ." rules are used to represent the expert's reasoning knowledge (if the data meet certain specified conditions, then take appropriate actions).

We consider four fuzzy parameters which each of them can improve and increase system flexibility and decrease the rate of false alarms. The fuzzy parameters are:

1. rate of mismatch rules
2. confidence average
3. dangerous score of mismatch rules
4. rule count in the process

All of the fuzzy parameters can be *Small*, *Medium*, or *Large*. There is also a fuzzy output (*Decision* alarm). As "Fig. 4" shows, it can be *Green* (normal sequence), *Blue* (dubious sequence), or *Red* (intrusive sequence).

Numerical input values can be easily translated into descriptive words such as Small, Medium, or Large.

The used fuzzy system has the following features and details:

1. Four fuzzy set of input variable (see table 1 for details)
2. A fuzzy set of output variable (see table 1 for details)
3. Fuzzifier: *singleton* (it translates numerical input values into descriptive words such as small, medium, or large.)
4. Defuzzifier: *center average*
5. Fuzzy inference engine: *product*
6. Fuzzy rulebase: *searching table*

TABLE 1. DETAILS OF FUZZY SETS (VALUES OF A, B, AND C)

	Fuzzy set	*A*	*B*	*C*
Input	*Mismatch*	*0*	*1*	*2*
	Confidence	*50*	*80*	*90*
	Rule_count	*200*	*700*	*1300*
	Score_mismatch	*50*	*55*	*60*
Output	*Decision*	*0.2*	*0.6*	*0.8*

We should use the following formula to compute fuzzy output (computing **y** in the product inference engine):

$$\mu_{B'}(y) = Max_{l=1}^{M}\left[\sup(\mu_{A'}(x).\prod_{i=1}^{n}\mu_{A_i^l}(x_i)\mu_{B^l}(y))\right]$$

$$f(x) = y^* = \frac{\sum_{l=1}^{M}\bar{y}^l(\prod_{i=1}^{n}\mu_{A_i^l}(x_i))}{\sum_{l=1}^{M}(\prod_{i=1}^{n}\mu_{A_i^l}(x_i))}$$

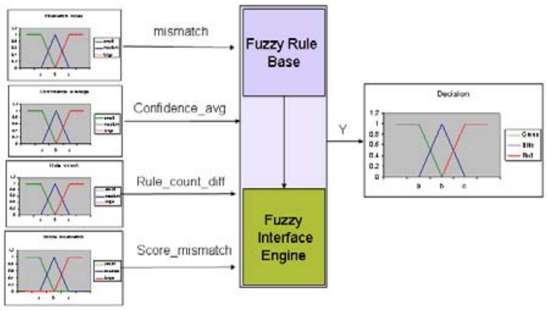


Figure 4. The proposed Fuzzy system in details

At first, all of the fuzzy variable should be computed by fuzzifier part. After that, Fuzzy system and fuzzy inference engine compute Decision variable via Fuzzy rulebase as an output alarm. Examples of our rules in the fuzzy rulebase are:

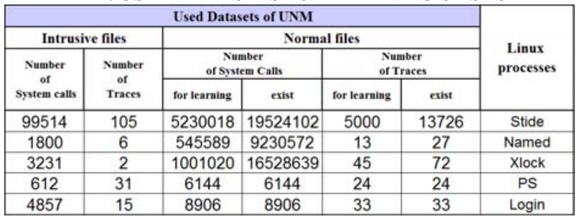
1. **IF** Mismatch= M and Confidence_avg=M and Rule_count_dif= M and Score_mismatch= L **THEN** decision= Blue
2. **IF** mismatch= M and confidence_avg=M and Rule_count_dif= M and Score_mismatch= L **THEN** decision= Blue
3. **IF** mismatch= L and confidence_avg= L and Rule_count_dif= S and Score_mismatch= L **THEN** normal= Blue
4. **IF** mismatch= L and confidence_avg= S and Rule_count_dif= L and Score_mismatch= L **THEN** decision= Red

The system will able to detect dangerous behavior and intrusive process and create the output as an alarm.

IV. THE RESULTS

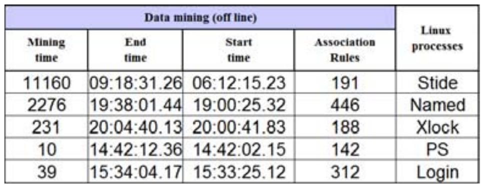
We executed the implemented system several times on the processes *Login*, *PS*, *Xlock*, *Named*, and *Stide.* The table 2 shows used datasets for learning the system and the table 3 shows results of offline stages in which have been extracted unique rules in each processes. For example, number of unique rules in *Stide* is 191. As the table 3 shows, executing the Data mining stage wasted large time and it was very slow. As a sample, Stide process need to 11160 second for extracting the 191 unique normal rules. But this stage is executed just one time as offline.

TABLE 2. USED DATASETS FOR LEARNING OF SYSTEM



Used Datasets of UNM						Linux processes
Intrusive files		Normal files				
Number of System calls	Number of Traces	Number of System Calls		Number of Traces		
		for learning	exist	for learning	exist	
99514	105	5230018	19524102	5000	13726	Stide
1800	6	545589	9230572	13	27	Named
3231	2	1001020	16528639	45	72	Xlock
612	31	6144	6144	24	24	PS
4857	15	8906	8906	33	33	Login

TABLE 3. RESULTS OF DATA MINING STAGE



Data mining (off line)				Linux processes
Mining time	End time	Start time	Association Rules	
11160	09:18:31.26	06:12:15.23	191	Stide
2276	19:38:01.44	19:00:25.32	446	Named
231	20:04:40.13	20:00:41.83	188	Xlock
10	14:42:12.36	14:42:02.15	142	PS
39	15:34:04.17	15:33:25.12	312	Login

The table 4 shows the final results of our system in which all of the intrusions approximately have been detected. The output can be Red, Blue, or Green as fuzzy decision.

TABLE 4. FUZZY OUTPUTS (RATE OF FALSE ALARMS)

make a Decision (real time)				Number of Intrusion	Linux processes
False Alram	Output Of Fuzzy Detection				
	Green	Blue	Red		
1	1	43	61	105	Stide
0	0	2	4	6	Named
0	0	0	2	2	Xlock
0	0	0	31	31	PS
4	4	1	10	15	Login

As we have accomplished our work, 61 instance of the existed intrusive process in Stide have been recognized as red and 43 instances as blue and one remaining as green that the last one is first false positive [table 4].

One of the problems in the proposed method is detecting of intrusion in login process. We could not extract a normal behavior for the process. According the "Fig. 5", as number of sequences of system call in *login* was increasing, number of rules also gradually was growing and it was not remained steady ever. Hence, the login process behavior was not commuted easily to some unique rules. However, other processes were stabilized at 188 rules (Xlock), 191 rules (Stide), 142 rules (PS), and 446 rules (Named) as shown in "Fig. 6 and 7".

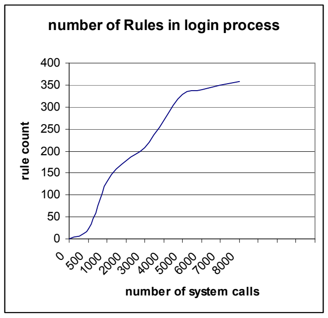


Figure 5. Growth of Extracted rules in login process

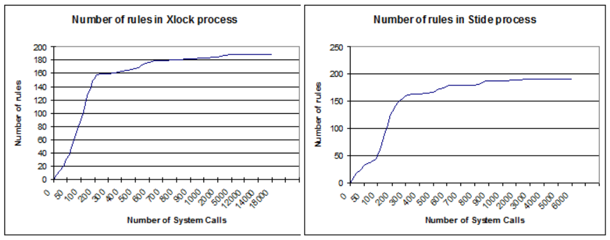


Figure 6. Growth of Extracted rules in Xlock and Stide process

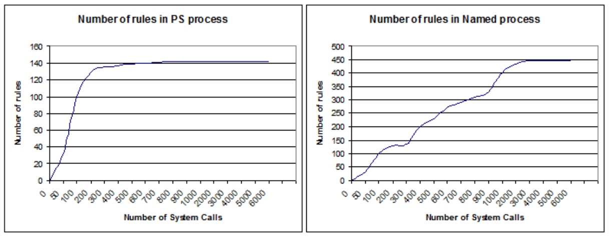


Figure 7. Growth of Extracted rules in Ps and Named process

V. CONCLUSION

It can be clearly seen that the proposed method will be able to detect most of the intrusive behaviors of each process, but the process *Login* doesn't imitate in a specially path like other processes. One of the most important causes for the no imitation is that *Login* directly depends on behavior of human and it is unpredictable.

In conclusion, according to findings, there are following advantages in the proposed process-based system:

1. Ability to detect intrusion in real time (so the size of normal database has decreased).
2. Rate of false alarm has been decreased.
3. Visibility is high. All of the intruder activities are visible.
4. Rate of information assurance in this method is higher than other methods (such as network-based IDS).
5. The proposed method is operating system independent.
6. The proposed method can detect the intrusions which other IDSs can not detect (such as buffer overflow, lprcp, *etc.*).
7. Pattern and rule creation in this method is easier than other methods, because number of system calls is specific and low.

REFERENCES

[1] Stephanie Forrest, Steven A.Hofmeyr "*Intrusion Detection using Sequences of System Calls" 2007*

[2] Stephanie Forrest, Steven A.Hofmeyr *"A Sense of Self for Unix Processes", New Mexico IEEE 1996*

[3] David Wagner *"Mimicry Attacks on Host-Based Intrusion Detection Systems", University of California, 2002*

[4] Xuan Dau Hoang *"A Multi-layer Model for Anomaly Intrusion Detection Using Program Sequences of System Calls", University of Melbourne, Australia IEEE 2003*

[5] Cheryl Martin, Alexander Liu *"A Comparison of System Call Feature Representations for Insider Threat Detection" , University of Texas at Austin, United States Military Academy IEEE 2005*

[6] Nam Nguyen *"Detecting Insider Threats by Monitoring System Call Activity", University Of California, Los Angeles United States Military Academy Proceedings of the 2003 IEEE*

[7] Gaurav Tandon *"Learning Rules from System Call Arguments and Sequences for Anomaly Detection", ICDM 2003*

[8] Eleazar Eskin & Wenke Lee *"Modeling System Calls for Intrusion Detection with Dynamic Window Sizes", Computer Science Department Columbia University 2002*

[9] Davide Libenzi *"Guarded Memory Move (GMM) Buffer Overflow Detection and Analysis", January 17, 2004*