一个两层马尔可夫链异常入侵检测模型*

徐 明[1,2+], 陈 纯[1], 应 晶[1]

[1](浙江大学 计算机科学与技术学院,浙江 杭州 310027)

[2](杭州电子科技大学 计算机学院,浙江 杭州 310018)

A Two-Layer Markov Chain Anomaly Detection Model

XU Ming[1,2+], CHEN Chun[1], YING Jing[1]

[1](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[2](College of Computer, Hangzhou DIANZI University, Hangzhou 310018, China)

+ Corresponding author: Phn: +86-571-86919090, E-mail: xuming_903@tom.com, http://www.cs.zju.edu.cn/

Received 2002-12-17; Accepted 2003-11-07

Xu M, Chen C, Ying J. A two-layer Markov chain anomaly detection model. *Journal of Software*, 2005,16(2):276−285. http://www.jos.org.cn/1000-9825/16/276.htm

Abstract: On the basis of the current single layer Markov chain anomaly detection model, this paper proposes a new two-layer model. Two distinctly different processes, the different requests and the system call sequence in the same request section, are classified as two layers and dealt with by different Markov chains respectively. The two-layer frame can depict the dynamic activity of the protected process more exactly than the single layer frame, so that the two-layer detection model can promote the detection rate and degrade the false alarm rate. Furthermore, the detected anomaly will be limited in the corresponding request sections where anomaly happens. The new detection model is suitable for privileged processes, especially for those based on request-response.

Key words: Markov chain; system call; request; anomaly detection; intrusion detection

摘 要: 在现有的单层马尔科夫链异常检测模型基础上,提出一种崭新的两层模型.将性质上有较大差异的两个过程,不同的请求和同一请求内的系统调用序列,分为两层,分别用不同的马尔可夫链来处理.两层结构可以更准确地刻画被保护服务进程的动态行为,因而能较大地提高异常的识别率,降低误警报率.而且异常检测出的异常将被限制在相应的异常真正发生的请求区内.检测模型适合于针对特权进程(特别是基于请求—反应型的特权进程)的异常入侵检测.

关键词: 马尔可夫链;系统调用;请求;异常检测;入侵检测

中图法分类号: TP393 **文献标识码**: A

*Supported by the National High-Tech Research and Development Plan of China under Grant No.2003AA1Z2120 (国家高技术研究发展计划(863)); the Natural Science Foundation of Zhejiang Province of China under Grant No.Y104426 (浙江省自然科学基金); the Scientific Research Fund of Zhejiang Provincial Education Department of China under Grant No.20040457 (浙江省教育厅科研项目)

XU Ming was born in 1970. He received his Ph.D. degree from the College of Computer Science and Technology, Zhejiang University in 2004. His research areas are intrusion detection, network security and steganalysis. **CHEN Chun** was born in 1955. He is a professor and doctoral supervisor at the College of Computer Science and Technology, Zhejiang University. His research areas are CAD&CG, CAM, information security, AI and CSCW. **YING Jing** was born in 1971. He is a professor and doctoral supervisor at the College of Computer Science and Technology, Zhejiang University. His research areas include software engineering and CASE.

An intrusion detection system (IDS) is one that identifies intrusions, where an intrusion is defined as a misuse by an authorized user or as an unauthorized use by external adversaries[1,2]. An intrusion detection classification appears in Ref.[3]. There are two general methods for detecting intrusions: anomaly detection and pattern recognition. Our detection method belongs to the former. The audit data used in our detection model are system call traces. A system call trace is an ordered sequence of system calls, which a process performs during execution. The system calls in the trace depend on the execution path of the process. A process execution path depends on many factors such as the inputs and the states of the system. These factors determine what execution path a process takes at each passable branch point. Therefore, the inputs to process are important data for constructing normal profiles and detecting anomaly. The parameters and return values of call are often ignored in anomaly detection system that operates on system call traces[4−8]. In 1996, Forrest and others first introduced a simple intrusion detection method, which is based on monitoring the system calls of the privileged processes[6]. Her work shows that a normal process behavior could be characterized by local patterns in its traces, and deviations from these patterns could be used to identify security violations of an executing process. Over the past several years, many statistical learning techniques in IDS have been developed. Several of these methods have the potential for generating more accurate and (or) more compact models based on the system call trace data[4−8]. Warrender[7] wrote a survey on the intrusion detection methods based on system calls. Markov chain is a powerful tool of anomaly detection[9−15]. A high order Markov chain detection model appears in Ref.[10], but the computation and memory are too expensive.

Our detection model differs from the above research studies. On the one hand, above methods, which operate on system traces, only use call names as audit data, but ignore call parameters and return values. This factor may bring false alarm and false negation. In our detection model, requests/commands which are the kernel inputs to process and appear in the call parameters, are used to denote the states into which the process enters, and the call return value is also used. In fact, this means that the dynamic activity of process depends on not only code but also current input. On the other hand, above methods, which use Markov chain to detect anomaly regard the call trace as a whole. As illustrated in this paper, we view the call trace of process as being composed of two-layer Markov chains. This refined description to call trace will contribute to a more accurate intrusion detection.

The outline of the paper is as follow. The definition of intrusion detection that operates on system call trace is given in section 1. Section 2 provides a general outline of the traditional Markov chain anomaly detections. Section 3 describes two-layer Markov chain anomaly intrusion detections. We then describe a wu-ftpd detection prototype and experimental results in Section 4. Discussion is given in section 5. Future work and concluding remarks are provided in section 6.

1 Definition of Intrusion Detection System That Operates on Call Traces

An application can be viewed as a function f which takes the current application state s_t and input i_t as parameters, and returns the output o_t and next state s_{t+1}[16]: $f(i_t,s_t)=(o_t,s_{t+1})$. The state set S can be partitioned into two no overlapping subsets, i.e. safe state set S_s and compromised state set S_c: $S_s \cap S_c = \phi$. IDS that operates on system call traces can be defined as a function d which is to reconstruct the states of the application from the system call traces: $d(c_t,p_t,q_t,m_t)=(m_{t+1},n_t)$,where c_t is the system call name at time t; p_t and q_t are the call input and output at time t respectively; m_t and m_{t+1} are the states of the IDS at time t and t+1 respectively. n_t mirrors the state of the protected application at time t. IDS needs to remain synchronization with the protecting application (i.e. $n_t=s_t$) so that the IDS can decide if the application has entered into a safe or compromised state. Ideally, we expect $p_t=i_t$ and $q_t=o_t$. So using call name, call inputs and return value for anomaly detection is very reasonable.

2 Traditional Marko Chain Anomaly Detection Technique

The traditional first-order Markov model of event (call) transitions assumes that the next event depends on only the last event in the past. Let M_t be the value of a random variable or the state of a system at time t. A Markov chain is a stochastic process with the following assumptions[13,14]: $P(M_{t+1}=i_{t+1}|M_t=i_t,M_{t-1}=i_{t-1},\ldots,M_0=i_0)=P(M_{t+1}=i_{t+1}|M_t=i_t)$; $P(M_{t+1}=i_{t+1}|M_t=i_t)= P(M_{t+1}=j|M_t=i)=p_{ij}$. For all time t and all states, p_{ij} is the probability of the system in state i at time t and in state j at time t+1. The first equation denotes that the probability distribution of the state at time t+1 depends only on the state at time t, and does not depend on the previous states. The last equation specifies that a state transition from time t to time t+1 is independent of time t. If the system has a finite number of states 1,2,…,n, we can define a Markov chain model by a transition probability matrix $P=(p_{ij})_{n\mathrm{E}n}$ and an initial probability distribution vector $Q=[q_1,q_2,\ldots,q_n]$ [11,15,16], where q_i is the probability of the system in state i at time 0, and $\sum_{j=1}^{n} p_{ij}=1,(i=1,2,...,n)$, $\sum_{j=1}^{n} q_j=1$ .The probability of sequence $M_{t-k},M_{t-k+1},\ldots,M_t$ is computed as follows: $P(M_{t-k},...,M_t)=q_{M_{t-k}}\prod_{i=k}^{1} p_{M_{t-i}M_{t-i+1}}$. In the Markov chain detection model, the normal profiles are matrix P and vector Q. They can be learned from the historic data of the system's normal behavior: $p_{ij}=N_{ij}/N_i$, $q_i=N_i/N$. N_{ij} is the number of observation pairs M_t and M_{t+1} with M_t in state i and M_{t+1} in state j; N_i is the number of observation pairs M_t and M_{t+1} with M_t in state i and M_{t+1} in any state; N_i is the number of M_t in state i; N is the total state number of observations. After learning normal profiles, the anomaly detection is to observe the window of size k on the continuous steam of audit for viewing the last k audit states from the current time t ($M_{t-k},M_{t-k+1},\ldots,M_t$), and compute the probability of this sequence. The higher the probability we get, the more likely the sequence of states come from the normal activities. A sequence of the states from the intrusive activities are expected to receive a low probability. Two different measures to express the strength of anomalous signal are MC (mismatch count) and LFC (locality frame count)[17].

3 Two-Layer Markov Chain Anomaly Detection Model

3.1 Audit data

In a two-layer Markov chain detection model, request, call name, and call return value are used to detect anomaly. The request can be the application supporting interactive requests/commands, and also can be command line parameters of the simple application which doesn't support the interactive request or command. The key is that each request can provide an independent and simple function. Request determines mostly the current function and call sequences. Not all call parameters are used, otherwise the number of states will become much huger.

In the two-layer Markov chains detection model, server process activity is viewed as being composed of a high-level requests/commands layer. Within each request section, a series of system calls are to perform the request function. The request often appears in read call. For example, "read(0,"PWD\r\n",1024)=5" denotes that FTPD receives a PWD request from a user. Using this characteristic can identify the request from the call trace and then partition the trace into the request sections.

The low layer is the call status series in each particular request. Each call status is composed of a call name and its return value: $callStatus=\begin{cases} call\ name+'-1' & \text{the call return value is } -1 \\ call\ name & \text{other} \end{cases}$

Clearly, it is different between open("readme.txt,ORDONLY")=8 and open("readme.txt, ORDONLY")=−1.

The former denotes that the call open is a success and returns the file descriptor 8, but the latter denotes that the call open is a failure. Therefore, only using call name for detection can raise the false alarm and false negation.

3.2 Two-Layer Markov chains

Most of the complicated server daemons such as FTPD, HTTPD, and SENDMAIL etc. directly support some requests/commands. For example, FTPD daemon has USER, PASS, PORT and QUIT etc. Traditional Markov detection model regards the server application call trace as a whole. Each request of an application can provide an independent and simple function so that constructing a Markov chain for each request can provide a more exact detection model. Our detection model can be regarded as two-layer Markov chains. The high layer is a request Markov chain, and the low layer is a request corresponding to the system call status Markov chain (see Fig.1).

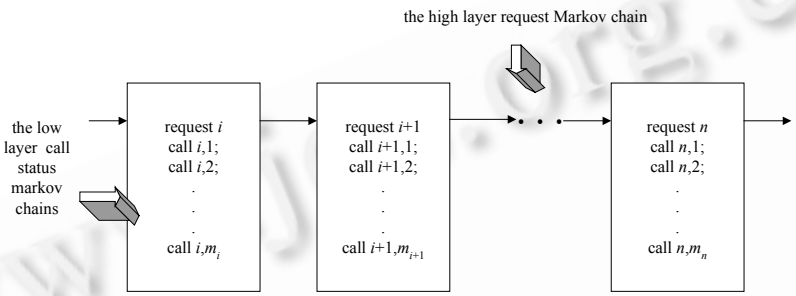

Fig.1 Two-Layer Markov chains

3.3 Profiling normal behavior

Profiling normal behaviors is to construct the transition probability matrix and initial probability distribution vector for each Markov chain from the normal trace data. To the high layer request Markov chain, the state is a request name. The size of the transition probability matrix $requestP(p_{ij})$ is $requestNum$E$requestNum$, and the size of the initial probability distribution vector $requestQ(q_i)$ is $requestNum$. $requestNum$ is the number of requests which appear in the normal training traces. To low-layer Markov chains, each request has a Markov chain. The state of the low layer Markov chains is *callStatus*. The size of $callStatusP_i(p_{jk})$ (transition probability matrix of $request_i$ corresponding to Markov chain) is $callStatusNum_i$E $callStatusNum_i$, and the size of $callStatusQ_i(q_j)$ (initial probability distribution vector of $request_i$ corresponding to Markov chain) is $callStatusNum_i$, where $callStatusNum_i$ is the total number of *callStatus* which appears in $request_i$ corresponding to the call traces in the training traces.

The two-layer Markov chain model equates to the traditional Markov chain model in computing complexity of profiling normal behaviors, but the size of the needed memory to store parameters is greater than that of the traditional Markov chain model. Because each request performs an independent and simple function of the server daemon, our profiling method can describe more exactly the normal activities of the server daemon.

3.4 Anomaly detection

Measures to express the strength of anomalous signal can use MC or LFC in every Markov chain in a trace, and then use the maximum of MC or LFC values to determine anomaly. But we can also define another measure: AS (anomaly signal). In a trace T, we define $S_A(T)=\max\{SH_A(T), SL_A(T)\}$, where $SH_A(T)$ is a MC or LFC value of the high layer Markov chain in trace T, $SL_A(T)=\max\{N_{A_{S_i}}/N_{S_i}, S_i\in T, i=1,2,3,...\}$, and N_{S_i} and $N_{A_{S_i}}$ denote to total number of call status sequences and the number of mismatch call status sequences in the i^{th} request section in traces T, respectively. We note that every value, $N_{A_{S_i}}/N_{S_i}$, maybe computed from different low layer Markov chains. $S_A(T)$

expresses how much T deviates from the normal profiles. The value of the normalized anomaly signal, $S_A(T)$, is between 0 and 1. The two-layer Markov chain model equates to the traditional Markov chain model in computing complexity of anomaly detection.

3.5 Classification error

In IDS, there are two types of classification errors: false alarms (false positives) and false negatives. A false alarm occurs when a call trace generated by legitimate behavior is classified as anomalous, and a false negative occurs when a call trace generated by an intrusion is classified as normal.

In the two-layer Markov chains detection model, each Markov chain can choose different thresholds of the mismatch probability and state sequence length. Therefore our detection model can provide a higher anomaly detection rate and a lower false alarm than the tradition Markov chain anomaly detection model. Let $M(T)=\{m_1,m_2,...,m_{l_T}\}$ be the set of all Markov chains in trace T; ε_i and k_i are the thresholds of the mismatch probability and state sequence length of m_i respectively; $S_{m_i}(t)$ is the set of all the overlapping state sequences of length k_i in m_i of the trace T; $P_{\min_i}(t)=\min\{p(s)\mid s\in S_{m_i}(t)\}$, where $p(s)$ denotes the probability of the sequence s. In order to detect an intrusion trace I, at least one request or *callStatus* sequence generated by the intrusion must be classified as mismatch, i.e. $\exists i\in[1..l_T],\varepsilon_i>P_{\min_i}(I)$. When $\forall i\in[1..l_T],\ \varepsilon_i\le P_{\min_i}(I)$, the IDS will produce false negatives. To a normal trace N, none of the request or *callStatus* sequence generated by the normal activity should be classified as anomalous, i.e. $\forall i\in[1..l_T],\varepsilon_i\le P_{\min_i}(N)$. When $\exists i\in[1..l_T],\varepsilon_i>P_{\min_i}(N)$, the IDS will produce a false alarm.

Although we would like to minimize both kinds of the errors, we are more willing to tolerate false negatives than false positives. False negatives can be reduced by adding layers of defense, whereas layering will not reduce the overall false positive rates[6]. Therefore, to the given normal traces N, we can choose threshold ε_i of the mismatch probability and state sequence length k_i by $P_{\min_i}(N)$ and the toleration degree to false positives for Markov chain m_i. On the condition $\varepsilon_i=P_{\min_i}(N)$, to enhance detection anomaly sensitivity, we expect that $P_{\min_i}(N)$ and k_i would be as great as possible. But $P_{\min_i}(N)$ and k_i restrict one another. Therefore, we can expect $\log P_{\min_i}(N)/k_i$ would be as great as possible.

To high layer request Markov chain, choose $k=\arg\max_{k\in[1,2,...,l]}\{\log PR_{\min}(N,k)/k\}$, where $PR_{\min}(N,k)$ is the minimal probability of length k request sequence in N and l is the minimal request sequence length of likely normal trace to ensure that every trace has at least one length k request sequence. Then choose ε by toleration degree to false positives. In a general way, the ε can be $\varepsilon=PR_{\min}(N,k)$ to zero false alarm of request anomaly for the given normal traces N. To each low layer call status Markov chain, choose $k=\arg\max_{k\in[1,2,...,l]}\{\log PC_{\min}(r,N,k)/k\}$, where $PC_{\min}(r,N,k)$ is the minimal probability of length k call status sequence in request r in N and l is the minimal call status sequence length of request r in N to ensure that each trace included in request r has at least one length k call status sequence. Then choose ε by toleration degree to false positives. In a general way, the ε can be $\varepsilon=PC_{\min}(r,N,k)$ to zero false alarm of call status anomaly in request r for the given normal traces N.

4 A wu-ftpd Detection Prototype and Experimental Result

On a Red Hat Linux system, we use Perl to implement a wu-ftpd daemon detection prototype. We use the wu-ftpd2.6.0 as an anomaly detection application, because wu-ftpd is a widely deployed software package to provide File Transfer Protocol (FTP) services on Unix and Linux systems, and exists many intrusion scripts on the Internet.

4.1 Training and testing trace data

We use command *strace* ("strace –p pid –f –o output.file") to get trace data. *Strace* is a system call trace, i.e. a debugging tool that prints out a trace of all the system calls made by another process. The program to be traced need not to be recompiled. Strace can also trace child processes, which are created by the currently traced processes as a

result of the fork or vfork system call at the recent Linux kernel. Because Linux uses multi processes rather than multithreading to implement multitasking, we can use *strace* to trace multitasking. Each line in the trace contains the pid, system call name, followed by its arguments in parentheses and its return value. All of the training data are normal traces, while the testing data include both the normal traces and intrusion trace data. The data used in this study is described in Table 1. The normal training trace data come from a live wu-ftpd daemon, with a careful checking by Snort[18] and other experts to keep no anomaly activity in normal training traces. The intrusion scripts are downloaded from Internet (http://www.hack.co.za and http://packetstormsecurity.nl) (in Table 2). In fact, only three scripts (bobek.c, wu-lnx.c and wuftpd2600.c) can triumphantly penetrate into system and get a root shell.

Table 1 The training and testing trace data

	Number of trace	Number of call	Number of request
Training traces	32	728,138	17,787
Normal traces	59	842,295	13,084
Intrusion testing traces	20	47,365	679

Table 2 Intrusion scripts from Internet

Script names	Description
ADMwuftpd.c,w00f.c,wu-ftpd.pl	wu-ftp beta 18 remote overflow
beroftpd.c	beroftpd 1.3.4(1) site exec format strings exploit
bobek.c, wuftpd2600.c, wu-lnx.c	wuftpd 2.6.0 SITE EXEC vulnerability
Ftpexp.c	getwd() overflow
ftp-ozone.c	layer violation
ftpspy.c	vulnerability of passive ftp connection
ftpwarez.c	wu-ftpd beta17 remote root overflow
Glob.sh	Glob vulnerability
ifafoffuffoffaf.c	wuftpd 2.5.0 heap overflow
own-proftpd.c	proftpd pre4 remote overflow
pasvagg.pl	pasv dumps core with root passwd
proftp-ppc.c, proftpX.c	proftpd-pre1,2,3 buffer overflow
wuftp25.tar.gz	wuftpd 2.5.0 heap overflow
crash_ftpd.c	crash proftpd 1.2.0pre4 servers
ftpsed.pl	proftpd dos vulnerability

4.2 Request recognization

In our prototype, the ftpd is managed by the inetd daemon. Wu-ftpd2.6.0 has 47 directional support requests, but in which 10 requests are not implemented. We add START and OTHER request into the request set. The START denotes the ftpd start process, and OTHER denotes the unidentifiable request. We use process ID to identify calls belonging to the same process in the trace data. The request can simply use “read (0,...)” to identify.

4.3 Learning wu-ftpd normal profile

The process of learning wu-ftpd normal profiles is to construct the transition probability matrix and initial probability distribution vector for each Markov chain from wu-ftpd training trace data. After learning, the total nonzero parameters in all of the probability state transaction matrixes are 1574; total nonzero parameters in all of the initial probability distribution vectors are 581. To the traditional Markov chain model, total nonzero parameters in the probability state transaction matrixes are 356; total nonzero parameters in the initial probability distribution vectors are 68.In each probability state transaction matrix, many values of items do not exist because the corresponding call transactions do not appear in the training trace data at all. Let these values of items be chosen as a little probability constant number 1e-5 according to 0.01.

4.4 How much is the normal training trace enough?

To build reasonable probability state transaction matrixes and initial probability distribution vectors of the Markov chains, how much are the training trace data needed? We use one training trace as one step; use subscript i to denote the parameters built from the previous i training trace data; use R to denote the request set, and let C

denote the call status set. Then we apply the result values of the following equations to evaluate the probability state transaction matrixes and initial probability distribution vectors.

$$hdp_i = \sum_{r_1 \in R} \sum_{r_2 \in R} (requestP_i[r_1, r_2] - requestP_{i+1}[r_1, r_2])^2 / f^2(requestP_i - requestP_{i+1});$$

$$hdq_i = \sum_{r \in R} (requestQ_i[r] - requestQ_{i+1}[r])^2 / f^2(requestQ_i - requestQ_{i+1});$$

$$ldp_i = \sum_{r \in R} \sum_{c_1 \in C} \sum_{c_2 \in C} (callStatusP_i[r, c_1, c_2] - callStatusP_{i+1}[r, c_1, c_2])^2 / f^2(callStatusP_i - callStatusP_{i+1});$$

$$ldq_i = \sum_{r \in R} \sum_{c \in C} (callStatusQ_i[r, c] - callStatusQ_{i+1}[r, c])^2 / f^2(callStatusQ_i - callStatusQ_{i+1}).$$

The high layer Markov chain parameter *requestP* matrix and *requestQ* vector use the first and second equations respectively, while the low layer *callStatusP* and *callStatusQ* use the third and last equations respectively. Here, using $request_i(r_1,r_2)$ to denote the transaction probability from request r_1 to r_2, and $requestQ_i(r)$ to denote the initial probability of request r; using $callStatusP_i(r,c_1,c_2)$ to denote the transaction probability from call status c_1 to c_2 in request r corresponding to the low layer Markov chain, and $callStatusQ(r,c)$ to denote the initial probability of call status c in request r corresponding to the low layer Markov chain; function $f(x)$ is to get the number of the nonzero items in matrix or vector x. The values are smaller; the parameters are more stabile and reasonable. If the parameters are stable enough, the training data are enough. During computing, the value of the item is regarded as zero if it does not appear after training. Using the training trace data, the results are described in Fig.2. From Fig.2, we know our training trace data are enough to create the stable and reasonable Markov parameters.

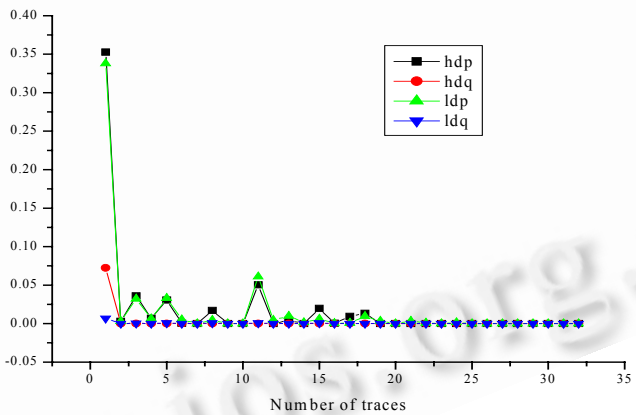

Fig.2 The changing of hdp, hdq, ldp and ldq

4.5 Finding ε and k

To the given normal training traces N (see Table 1), using the method in subsection 3.5, we choose the best parameter values of threshold of the mismatch probability and state sequence length for high layer request Markov chain and each low layer call status Markov chain. The threshold of mismatch probability is chosen to become zero false alarms for these normal training traces. We use 80 as the maximum length because the maximum of the minimum length of call status sequence in each request in training traces is 80, i.e. $\max_{r \in R}\{\min\{length(r, N)\}\} = 80$.

The results of the sequence length to each Markov chain are given in Table 3 (the threshold of mismatch probability is not given here).

Table 3 The sequence length k for every Markov chain

Request	CDUP	CWD	DELE	HELP	LIST	MKD	NLST	NOOP	OTHER
k	21	8	14	15	23	30	80	5	8
Request	PASS	PASV	PORT	PWD	QUIT	REST	RETR	RMD	RNFR
k	19	19	6	8	23	8	17	43	9
Request	RNTO	SITE	STOR	SYST	TYPE	USER	XPWD	etc.	High level
k	20	14	15	8	8	16	8	1	2

4.6 Comparison of detection performance comparing

We compare the detection performance of the two-layer Markov chain model with that of the traditional Markov chain model by ROC curve[13]. Each point in an ROC curve indicates a pair of the hit rate and the false alarm rate for a signal threshold. If an intrusive trace in the testing data is classified as anomaly, this is a hit. If a normal trace in the testing data is classified as anomaly, this is a false alarm. The hit rate is computed from the dividing of the total number of hits by the total number of intrusion traces in the testing data. The false alarm rate is computed from the dividing of the total number of false alarms by the total number of normal traces in the testing data. By varying the value of the signal threshold, we obtain an ROC curve. The closer the ROC is to the top-left corner (representing 100% hit rate and 0% false alarm rate) of the chart, the better the detection performance in the intrusion detection technique yields.

In Fig.3, we provide three ROC curves. The ROC (1) is for the traditional Markov chain model, in which the sequence length is 17(the average value of sequence lengths in Table 3). The determination of anomaly or normal uses the LFC method, and the ROC curve is created by changing the sequence mismatch threshold value from 0 to 1. The ROC (2) is for the tow-layer Markov chain model, but in which each Markov chain uses the same sequence length 17 and sequence mismatch threshold as in ROC (1), determining anomaly or normal uses the LFC method as in ROC (1) too, and the ROC curve is created by changing the sequence mismatch threshold from 0 to 1. The ROC (3) is also for the tow-layer Markov chain model, in which each Markov chain uses the sequence length in Table 3 and corresponding to different sequence mismatch thresholds, determining anomaly or normal uses the AS method, and the ROC curve is created by changing the AS threshold value from 0 to1; For the LFC method in experiments, we arbitrarily choose 20 as a (reasonable) size for the locality frame, and take 10 mismatches in a local frame as the anomaly threshold. From Fig.3, despite using the same sequence length, the LFC method and the method to create ROC, the detection performance of the two-layer Markov chain model is better than the traditional Markov chain model's. Using different sequence lengths, different sequence thresholds for each Markov and AS method, the two-layer Markov chain model can get the best detection performance. In experiment, we try other LFC frame lengths (30, 50 and100 etc.) and many others mismatch anomaly thresholds, but the results are similar to those of Fig.3.

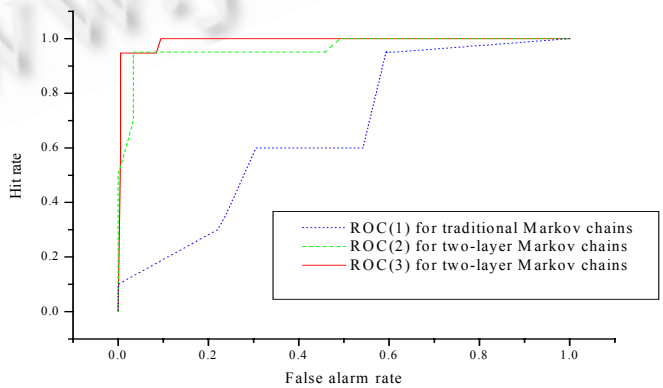

Fig.3 The ROC curves

If an anomaly is detected, the two-layer Markov chain model can provide other useful information: request name in which the anomaly occurs. For example, the model can accurately denote the maximum anomaly signal location in the SITE request section for the three scripts (bobek.c, wu-lnx.c and wuftpd2600.c) which just use SITE EXEC vulnerability in the wuftp daemon (CERT Advisory CA-2000-13, http://www.cert.org/advisories/CA-2000-13.html) to triumphantly penetrate into systems and get a root shell. Although intrusion scripts used in the experiments are old and known in real-life world, they are new and unknown to the detection model because they are not used to train the model at all. Therefore it is reasonable to expect that the two-layer Markov chain model is equally effective for detecting the really and truly new and unknown intrusions to those used in the experiments.

5 Discussion

5.1 Extensibilities

When following the following three steps, we can easily extend our model to protect other applications, especially for the privileged daemon processes such as SENDMAIL, HTTPD, POPD, LPD and NAMED etc. (1) Getting the application request set. For most daemon applications, this is easy. The key of request is simple and independent of the function. (2) Finding the way to identify request from the living system call trace. (3) Using our model frame to create the needed anomaly detection model. We can also use other anomaly detection techniques in our model, such as expert system, petri net, and neural network etc.

5.2 Strace command

When using the *strace* command to trace application, the application performance may lose about 100%–200%. This is the main flaw of our detection model. An amelioration way can be the integration of the trace function into the operation system kernel. For Linux, it is a feasible way.

6 Conclusions and Future Work

In this paper, we present a new two-layer Markov chain anomaly detection model, a general framework for our model, and a wu-ftpd anomaly detection prototype. The experimental results clearly demonstrate that the detection performance of the two-layer Markov chain detection model is better than the first Markov chain model's. Moreover our model can easily be extended to other applications and anomaly detection techniques. Using our detection model, the anomaly can be localized in the place where happens, i.e. it will be limited in the corresponding request sections, but not in the entire trace. For future work, we will use more call recorder information. We are also interested in extending our model to a mixed detection model (anomaly and pattern matching). We are currently pursuing these directions.

References:

[1] Mukherjee B, Heberlein LT, Levitt KN. Network intrusion detection. IEEE Network, 1994,8(3):26−41.

[2] Denning DE. An intrusion-detection model. IEEE Trans. on Software Engineering, 1987,13(2):222−232.

[3] Ilgun K, Kemmerer RA, Porras PA. State transition analysis: A rule-based intrusion detection approach. IEEE Trans. on Software Engineering, 1995,21(3):181−199.

[4] Lee W, Stolfo SJ, Chan Pk. Learning patterns from UNIX process execution traces for intrusion detection. In: Proc. of the AAAI97 Workshop on AI Methods in Fraud and Risk Management. Menlo Park: AAAI Press, 1997. 50−56.

[5] Helmer GG, Wong JSK, Honavar V, Miller L. Intelligent agents for intrusion detection. In: Proc. of the IEEE Information Technology Conf. Syracuse: IEEE Computer Society Press, 1998. 121−124.

[6] Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for UNIX processes. In: Proc. of the 1996 IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society Press, 1996. 120−128.

[7] Warrender C, Forrest S, Pearlmutter B. Detecting intrusions using system calls: Alternative data models. In: Proc. of the 1999 IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society Press, 1999. 133−145.

[8] Okazaki Y, Sato I, Goto S. A new intrusion detection method based on process profiling. In: Proc. of the Symp. on Applications and the Internet. Nara: IEEE Computer Society Press, 2002. 82−90.

[9] DuMouchel W. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. Technical Report, TR91, National Institute of Statistical Sciences, 1999. http://www.niss.org

[10] Ju WH, Vardi Y. A hybrid high-order Markov chain model for computer intrusion detection. Technical Report, TR92, National Institute of Statistical Sciences, 1999. http://www.niss.org

[11] Schonlau M, DuMouchel W, Ju WH. Computer intrusion: Detecting masquerades. Technical Report, TR95, National Institute of Statistical Sciences, 1999. http://www.niss.org

[12] Scott SL. Detecting network intrusion using a Markov modulated nonhomogeneous Poisson process. http://www-rcf.usc.edu/~sls/fraud.ps

[13] Ye N, Li XY, Chen Q, Emran SM, Xu M. Probabilistic techniques for intrusion detection based on computer audit data. IEEE Trans. on Systems, Man, and Cybernetics—Part A: Systems and Humans, 2001,31(4):266−274.

[14] Jha S, Tan K, Maxion RA. Markov chains, classifiers, and intrusion detection. In: Proc. of the14th IEEE Computer Security Foundations Workshop. Cape Breton: IEEE Computer Society Press, 2001. 206−219.

[15] Welz M, Hutchison A. Interfacing trusted applications with intrusion detection systems. In: Lee W, Mé L, Wespi A, eds. Proc. of the Recent Advances in Intrusion Detection 2001. LNCS 2212, New York, Heidelberg: Springer-Verlag, 2001. 37−53.

[16] Ye N. A Markov chain model of temporal behavior for anomaly detection. In: Proc. of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop. New York: IEEE Computer Society Press, 2000. 171−174.

[17] Jones AK, Lin Y. Application intrusion detection using language library calls. In: Proc. of the 17th Annual Computer Security Applications Conf. New Orleans: IEEE Computer Society Press, 2001.

[18] Marty R. Snort – lightweight intrusion detection for networks. In: Proc. of the 13th Conf. on Systems Administration. Washington: USENIX, 1999. 229−238.