Two Sophisticated Techniques to Improve HMM-Based Intrusion Detection Systems

Sung-Bae Cho and Sang-Jun Han

Dept. of Computer Science, Yonsei University,
134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, Korea
{sbcho,sjhan}@cs.yonsei.ac.kr

Abstract. Hidden Markov model (HMM) has been successfully applied to anomlay detection as a technique to model normal behavior. Despite its good performance, there are some problems in applying it to real intrusion detection systems: it requires large amount of time to model normal behaviors and the false-positive error rate is relatively high. To remedy these problems, we have proposed two techniques: extracting privilege flows to reduce the normal behaviors and combining multiple models to reduce the false-positive error rate. Experimental results with real audit data show that the proposed method requires significantly shorter time to train HMM without loss of detection rate and significantly reduces the false-positive error rate.

Keywords: anomaly detection, hidden Markov model, privilege flow, combining multiple models

1 Introduction

Due to worldwide proliferation and rapid progress in networking, faster and more diversified services have become in reality. Because the reliance on computers gets extremely high, security of critical computers is very important. Especially, since important network infrastructures on finance, defense, and electric power are being intruded, intrusions should be detected to minimize the damage. While the demand and interest in intrusion detection have increased, the most active effort in this area was mainly on the development of system security mechanisms like firewalls. However, recently many intrusion detection systems (IDSs) have been developed.

An IDS detects unauthorized usage and misuses by a local user as well as modification of important data by analyzing system calls, system logs, activation time, and network packets [1]. Most server computer systems support C2 security auditing that allows an IDS to obtain detailed information on events generated within the system [2].

The rationale is that human-computer interaction has a cause and effect relation as mentioned by Lane and Brodley [3]. Causal relationship or usage pattern can be captured as temporal sequence of events. Various techniques

G. Vigna, E. Jonsson, and C. Kruegel (Eds.): RAID 2003, LNCS 2820, pp. 207–219, 2003.

have been employed to determine how far a temporal sequence of events deviates from the normal model. In [4], after normal behavior model is built based on user activities, event sequence vector similarity is evaluated using pattern matching techniques. [5] and [6] model each program's usage pattern. There are several works that evaluate collections of modeling and matching techniques. [5] compares pattern matching, BP neural network and Elman neural network. [6] compares frequency, data mining, and HMM. Results in [5] and [6] show that exploiting temporal sequence information of events leads to better performance and hidden Markov model (HMM) is a good technique for modeling sequence information.

IDS based on HMM was originally designed by S. Forrest at New Mexico University for modeling and evaluating invisible events based on system calls. HMM-based intrusion detection systems proposed by Arizona State University, RAID at New Mexico University, and Hong Kong University of Science and Technology showed substantial reduction of false-positive errors and increase in detection rate compared to other anomaly detection techniques [14]. Furthermore, desirable results have been produced in modeling normal behavioral events generated from BSM data with HMM when using system call related measures [7].

In spite of good performance of HMM-based IDS, the amount of time required to model normal behaviors and determine intrusions is huge and error rate is higher than that of misuse detection techniques. Thus, it is difficult to detect intrusions in real-time. Therefore, to improve the system performance, an efficient way of refining data for modeling normal behaviors is required.

In this paper, we propose two sophisticated techniques to overcome the drawbacks of conventional HMM-based IDS: modeling privilege flows and combining multiple models. The technique that models privilege flows reduces the time required for training HMM. By combining multiple HMMs, the false positive error rate can be reduced.

The rest of this paper is organized as follows. In Section 2, we give a brief overview of the related works. The overall design and detailed description of the proposed methods are presented in Section 3. Experimental results are shown in Section 4.

2 Hidden Markov Model

An HMM is a doubly stochastic process with an underlying stochastic process that is not observable, and can only be observed through another set of stochastic processes that produce the sequence of observed symbols [8]. HMM is a useful tool to model sequence information. This model can be thought of as a graph with N nodes called 'state' and edges representing transitions between those states. Each state node contains initial state distribution and observation probabilities at which a given symbol is to be observed. An edge maintains a transition probability with which a state transition from one state to another state is made. Fig. 1 shows a left-to-right HMM with 3 states.

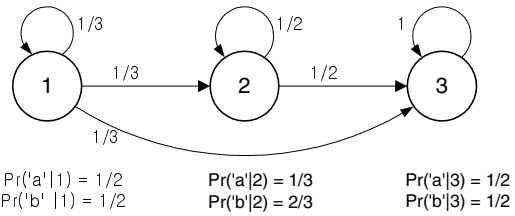

Fig. 1. An example of left-to-right HMM.

Given an input sequence $O = O_1, O_2, ..., O_T$, HMM can model this sequence with its own probability parameters using Markov process though state transition process cannot be seen outside. Once a model is built, the probability with which a given sequence is generated from the model can be evaluated. A model λ is described as $\lambda = (A, B, \pi)$ using its characteristic parameters. The parameters used in HMM are as follows:

T = length of the observation sequence

N = number of states in the model

M = number of observation symbols

$Q = \{q_1, q_2, ..., q_N\}$, states

$V = \{v_1, v_2, ..., v_M\}$, discrete set of possible symbol observations

$A = \{a_{ij} | a_{ij} = \Pr(q_j \text{ at } t+1 | q_i \text{ at } t)\}$, state transition probability distribution

$B = \{b_j(k) | b_j(k) = \Pr(v_k \text{ at } t | q_j \text{ at } t)\}$, observation probability distribution

$\pi = \{\pi_i | \pi_i = \Pr(q_i \text{ at } t = 1)\}$, initial state distribution

Suppose, for example, a sequence *aba* is observed in a model λ in Fig. 1 and initial state is 1, then the probability with which the given sequence is generated via state sequence 1-2-3 is calculated as follows:

$$\begin{aligned}\Pr(O = aba, q_1 = 1, q_2 = 2, q_3 = 3 | \lambda) \\ &= \pi_1 \cdot b_1(a) \cdot a_{12} \cdot b_2(b) \cdot a_{23} \cdot b_3(a) \\ &= 1 \cdot 1/2 \cdot 1/3 \cdot 1/2 \cdot 1/2 \cdot 1/2 \\ &= 1/48\end{aligned}$$

The probability with which the sequence is generated from the model can be calculated by summing the probabilities for all the possible state sequences. In practice, a more efficient method, known as forward-backward procedure, is used. Recognition can be done with sequences of measures of events as its input using well-established HMM procedures.

2.1 Anomaly Recognition

Anomaly recognition matches current behavior against the normal behavior models and calculates the probability with which it is generated out of each

model. Forward-backward procedure or Viterbi algorithm can be used for this purpose [9]. Each probability is passed to the recognition module for determining whether it is normal or not.

Forward-backward procedure calculates the probability $\Pr(O|\lambda)$ with which input sequence O is generated out of model λ using forward and backward variables. Forward variable $\alpha_t(i)$ denotes the probability at which a partial sequence $O = O_1, O_2, ..., O_t$ is observed and stays at state q_i.

$$\alpha_t(i) = \Pr(O = O_1, O_2, ..., O_t, q_t = S_i|\lambda)$$

According to the above definition, $\alpha_t(i)$ is the probability with which all the symbols in input sequence are observed in order and the current state is i. Summing up $\alpha_t(i)$ for all i yields the value $\Pr(O|\lambda)$. $\alpha_t(i)$ can be calculated by the following procedure.

- Initialization:
  $\alpha_1(i) = \pi_i b_i(O_1)$
- Induction:
  for $t = 1$ to $T - 1$
  $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1})$
- Termination:
  $\Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

Backward variable $\beta_t(i)$ is defined as follows:

$$\beta_t(i) = \Pr(O = O_{t+1}, O_{t+2}, ..., O_T, q_t = S_i|\lambda)$$

It can be calculated by the similar process to calculate α.

- Initialization:
  $\beta_T(i) = 1$
- Induction:
  for $t = T - 1$ to 1
  $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$

2.2 Normal Behavior Modeling

The determination of HMM parameters is to adjust $\lambda = (A, B, \pi)$ to maximize the probability $\Pr(O|\lambda)$. Because no analytic solution is known to do it, an iterative method called Baum-Welch reestimation is used [9]. This requires two more variables: $\xi_t(i, j)$ is defined as the probability with which it stays at state q_i at time t and stays at state q_j at time $t + 1$.

$$\begin{aligned} \xi_t(i, j) &= \Pr(i_t = q_i, i_{t+1} = q_j|O, \lambda) \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\Pr(O|\lambda)} \end{aligned}$$

$\gamma_t(i)$ is the probability with which it stays at state q_i at time t.

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j)$$

Summing up the two variables over time t respectively, we can get the expectation that state i will transit to state j and the expectation that it will stay at state i. Given the above variables calculated, a new model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ can be adjusted using the following equations:

$$\begin{aligned}
\bar{\pi}_i &= \text{expected frequency (number of times) in state } S_i \text{ at time } t = 1 \\
&= \gamma_1(i) \\
\bar{a}_{ij} &= \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\bar{b}_j(k) &= \frac{\text{expected number of times in state } S_j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \\
&= \frac{\sum_{\substack{t=1 \\ \text{s.t.} O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}
\end{aligned}$$

After $\bar{\lambda}$ is adjusted from sequence O, $\Pr(O|\bar{\lambda})$ is compared against $\Pr(O|\lambda)$. If $\Pr(O|\lambda)$ is greater than $\Pr(O|\bar{\lambda})$, it implies that a critical point in likelihood has been reached, thus finishing the reestimation. Otherwise, $\Pr(O|\lambda)$ is replaced by $\Pr(O|\bar{\lambda})$ and reestimation continues.

3 Improvement of HMM-Based IDS

The intrusion detection system developed in this paper is composed of a preprocessing module and an anomaly detection module (Fig. 2). The former is in charge of data reduction using self-organizing map (SOM) and privilege flow detection. The latter is responsible for the normal behavior modeling and anomaly detection. The normal behavior modeling module generates HMM-based normal behavior models from collected normal audit data. The intrusion detection module determines whether an intrusion has occurred by comparing accumulated target audit data with the normal behavior model. The core technology in this proposed IDS lies in filtering of audit data from abstracted information around the change of privilege flows to reduce the required time to build normal behavior model and combining multiple HMM to reduce false-positive errors.

3.1 Extracting Privilege Flow

The ultimate way of an intrusion is to acquire root privilege and the most prevalent attack is known as buffer overflow. This attack breaks into the system with unused user account or an account without any password and acquires root privilege with advanced attack techniques. Here, we analyze host-based attack types that aim to acquire root privilege. Most of user-to-root intrusions attack the files which have SETUID privilege that gives the attacker the opportunity to execute

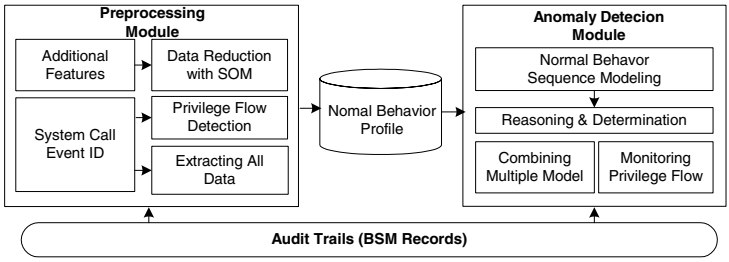

Fig. 2. The overview of intrusion detection system.

arbitrary program with the root privilege. Typical buffer overflow attacks acquire root privilege by attacking SETUID files that does not thoroughly examine the length of inputted parameter. The core difference between a normal SETUID file and an exploited SETUID file in execution is in the process of transition of EUID privilege. Generally after the privilege flow, EUID changes to local user with the call to exit event, but an exploited file executes root shell with execve call and maintains root privilege.

Thus, we can realize that the majority of host-based attacks exploit system bugs or misuse of a local user to acquire root privilege. Therefore, if we model information on the acquisition of normal root privilege to detect any indication of an attack, we can monitor more than 90% of illegal privilege flow attacks excluding denial of service attacks and this indicates that majority of host-based attacks can be detected. In addition, if we effectively minimize the number of targets of intrusion detection, we can also minimize the consumption of system resources which makes it possible for practical operation in real world.

In UNIX system, several users share fixed amount of disk and memory in a single system so that each user must have its own privilege. However, the problem of acquiring super user privilege caused by program error or misuse of a system is always possible. A normal privilege flow happens when an administrator logs in through local user account and acquires root privilege using SU command or by the local user who executes temporarily a file whose SETUID is set to super user. SETUID is a rule applied to the file that temporarily changes the privilege of the user, and anyone who executes that file will execute the file with the SETUID of the file [10].

Most host attacks exploit vulnerabilities described above. Fig. 3 is an example of privilege flow when buffer overflow attack is in action. Normally if a user executes fdformat command from a normal state (Q0), the command is executed temporarily with super user privilege (Q1), and on termination of the task, it returns to its original state (Q0). However if buffer overflow attack occurs during the transaction, it does not return to its original state (Q0) but retains super user privilege (Q2) and ultimately the user can access the system with super user privilege. But if security module knows the sequence of normal privilege flow related to fdformat command, it can identify the transition to an abnormal

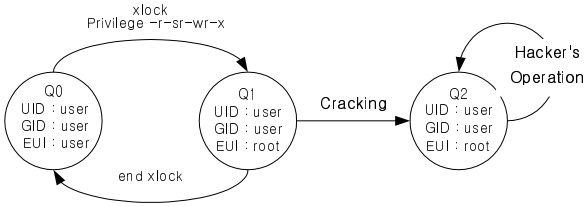

Fig. 3. An example of privilege change by buffer overflow attack.

state (Q2) caused by buffer overflow. The learning model for user privilege flows collects the sequence of normal system calls that happen before the occurrence of normal privilege change. Then, it models normal behaviors and detects intrusion by comparing each user's state of privilege flow.

The filtering subsystem imports the reformatted audit data from the preprocessor and determines which system calls are eligible and which audit data fields are sufficiently available to conduct our experiments. From this preliminary study, we analyze the data to detect privilege change from a regular user to root. By examining the result, we can assure that many events are not in use, but only 25% of the events have been used. In training an HMM, running time is roughly proportional to the number of system calls. For this reason, among the 267 different events audited by BSM, only 80 were used. It could be a good reduction technique that extracts the events around privilege change. Although abundant training data using system calls require significantly longer training time, these reduction techniques can reduce computational cost. Proposed privilege change model evaluated with taking fixed data sequence based on the situation where transitions between UID and EUID occur, which are derived from BSM data. However, during the detailed analysis of theses attacks, acquiring root privilege not only can be from user's change but also from group's change. For the sake of this case, this system detects privilege flows of both user and group.

Table 1. Events related with privilege flows.

Event ID	System Call	Event ID	System Call	Event ID	System Call
2	fork	27	setpgrp	200	setuid
11	chown	38	fchroot	214	setegid
16	stat	39	fchown	215	seteuid
21	symlink	40	setreuid	237	lchown
23	execve	41	setrgid	6158	rsh
24	chroot	69	fchroot	6159	su

3.2 Combining Multiple Models

Conventional HMM-based intrusion detection technique have used only system call ID as the measure of user's behavior. System call ID is useful measure to

monitor user's behavior. However, it is inadequate for monitoring the whole behavior. For example, we cannot know whether system call is failed or executed successfully. It is also unknown if system call is related to a critical file or an ordinary file. Therefore more information is needed to model user's behavior accurately. Solaris BSM provides additional information on user's behavior besides the system call ID: such as process ID, system call return value, related file, etc. We also model these additional information using HMMs and combine them to improve the performance.

Because additional system-call-related information is too large and various to apply to HMM directly, some data reduction technique is needed. We use self-organizing map (SOM) to reduce these additional information. System call-related information and process-related information and file access-related information are extracted from each BSM event and reduced using SOM as shown in Table 2.

Table 2. Measures extracted from the BSM to detect intrusion.

Group	Measures
System call	ID, return value, return status
Process	ID, IPC ID, IPC permission
	exit value, exit status
File access	access mode, path, file system
	file name, argument length

SOM is an unsupervised learning neural network, using Euclidean distance to compute distance between input vector and reference vector [11]. Algorithm of SOM is as follows. Here, $i(x)$ is the best matching neuron to input vector, λ_i means neighborhood function and η is learning rate.

1. Initialize the reference vectors $(w_j(n))$ $(n = 0)$
2. Compute distance and select the minimum value
   $i(x) = argmin\|x(n) - w_j(n)\|$
3. Update the reference vectors
   $w_j(n+1) = w_j(n) + \eta(n)\lambda_{i(x)}(n, j)(x(n) - w_j(n))$
4. repeat from 2 to 3 until the stop condition satisfies.

Several measures can be extracted from one audit record of BSM which includes an event and the information is normalized for the input of SOM. As an output of SOM, we can get one representative instead of many measures, that is, one record is converted into one representative.

Fig. 4 shows the flow of reducing audit data. In this paper, we reduce the measure size based on the locality of user action: We observe the range of the measures and save actual ones used in the system as table where we find the value of measure of current action. As a result of mapping of measures, we can obtain reduced data.

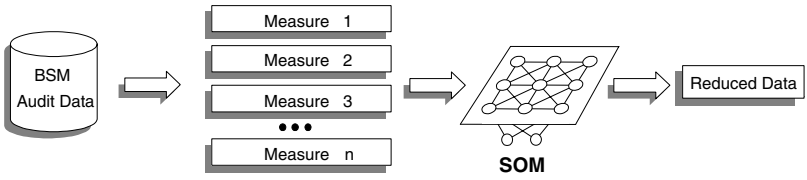

Fig. 4. Overall flow of reducing BSM audit data.

When one event is evaluated through each model, a vector of evaluation values are generated. A method to combine multiple models are required to finally decide whether current sequence is an anomaly. Combining multiple detectors is good for improving the performace of detection systems. [12] and [13] combines detectors using ensemble learning. They have applied machine learning tehniques such as artificial neural networks and support vector machine to intrusion detection and showed that combining different detectors is superior to individual approches. In this paper, we have combined HMM-based detectors that determines if current sequence is abnormal according to the measure it is responsible for:system call-related, process-related and file access-related measure. Each detector participates in the final anomaly decision.

Each detector is given a weight W_m according to their confidence. Voting method is determined. Typical voting methods include unanimity, majority and OR voting. In OR voting, anomaly is determined if at least one member votes positively. Voting is to determine whether or not the total result R is greater than or equal to the T dependent on the voting method.

$$R = \sum W_m * V_m \quad \begin{pmatrix} W_m & : & \text{model weight} \\ V_m & : & \text{model voting value} \end{pmatrix}$$

$$\text{anomalous if} \begin{cases} R = 1 & \text{(unanimity)}, \\ R \geq 0.5 & \text{(majority)}, \\ R > 0 & \text{(OR voting)} \end{cases}$$

Generally, OR voting enhances a detection rate but it increases an error rate. Unanimity improves an error rate but a detection rate also decreases.

4 Experimental Results

At first, we have checked whether the method that models privilege flow can produce a better performance. To show the usefulness of the proposed method, we compare the performance of modeling method with partial sequence data only around the privilege transition and that with all sequence data. We have conducted experiments to decide the optimal HMM parameters for effective intrusion detection. A desirable intrusion detection system would show high intrusion detection rate with low false-positive error rate. The detection rate is the

percentage of attacks detected, and the false-positive error rate is the percentage of normal data that the system alarms as attack. If the false-positive error rate is high, the IDS may not be practical because users may be warned from the IDS frequently even with the normal behaviors.

To model the HMM with normal behaviors, we have collected audit data from 10 users who have conducted several transactions such as programming, navigating Web pages, and transferring FTP data for one month. Because it is crucial to define normal traffic for IDS based on anomaly detection,we have attempted to balance the data with every-day usage of computer systems. The number of users and period of collection time should be extended at the future work.

In this paper, the BSM auditing tool of Sun Solaris operating environment is used and a total of 767,237 system calls have been collected. The performance of IDS is measured against 19 times u2r (user-to-root) intrusions with the observation of the variation of the Receiver Operating Characteristic (ROC) curve. Most attacks are attempted to acquire root privilege by exploiting a subtle temporary file creation and race condition bug.

Standard HMM has a fixed number of states, so that one must decide the size of the model before training. Preliminary study indicates that the number of optimal states roughly equals to that of distinct system calls used by the program [14]. A range of 5-15 states are examined in HMM.

To measure the runtime-performance, the program is executed 10 times in UltraSparc 10 and the results of the shell are recorded using time command. Table 3 summarizes the results of time for modeling the HMMs with different settings. As mentioned previously, training an HMM requires very expensive computation, but the model with privilege transition flows has obtained approximately 250 times faster than that with all data. We can expect to reduce the time consumption substantially with privilege transition flows.

Table 3. A comparison of performance in running time.

Modeling	State/sequence	Number of sequence	Timestamp
All data	5/20	767218	5 hours 07 min
All data	15/30	767208	6 hours 29 min
Privilege change data	5/20	5950	26.3 sec
Privilege change data	15/30	5792	57.5 sec

In addition to the running time, we have compared the detection rates and the false-positive error rates for two cases in ROC curve as shown in Fig. 5 and 6. The model with privilege transition flows yield higher performance than that with all data. There exists a clear gap between the minimum and maximum detection rates for the model with all data.

We have compared the result at same detection rate as shown in Table 4. The results indicate that privilege flows data are useful to detect intrusions when we have modeled normal behaviors by HMM. The model with 10 states and 30

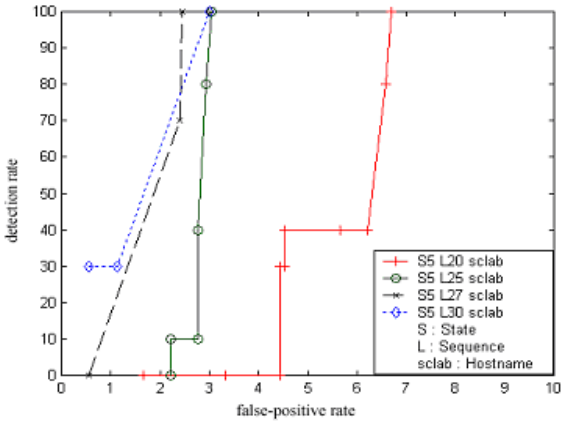

Fig. 5. ROC for privilege change data.

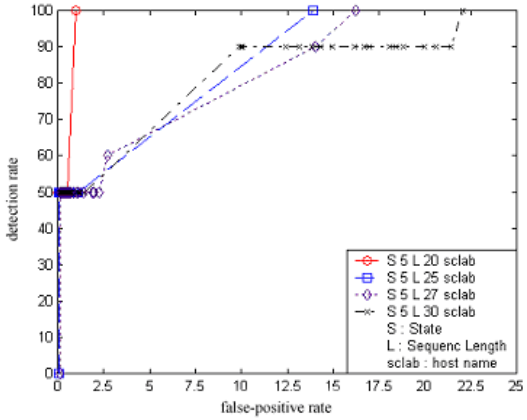

Fig. 6. ROC for all train data.

sequences in privilege transition flows data is seen to be the most effective in this experiment, while the model with 10 states and 30 sequences in all train data is not good enough. The lowest false-positive error has been obtained when the number of states is 10 and the number of sequences is 20 in all data.

Next, we have conducted the experiments to compare the detection methods based on system call ID and that on additional measures reduced by SOM and the combined method. Two models have combined by unanimity voting, majority voting and OR voting methods are tested. Each models is given the same voting weight.

For this experiment, we have used data obtained from one graduate student for one week. He has mainly used text editor, compiler and programs of their

Table 4. The anomaly detection rate for each modeling method.

Modeling	State/sequence	Threshold	Detection rate	F-P error
All data	5/20	-92.4	100%	4.234%
	10/25	-102.1	100%	3.237%
	10/20	-93.7	100%	1.6165%
	15/30	-118.3	100%	12.426%
Privilege change data	5/20	-53.8	100%	6.707%
	15/25	-65.2	100%	2.367%
	10/27	-73.1	100%	2.439%
	10/30	-81.2	100%	0.602%

Table 5. The performance of combining multiple models.

	only system call ID	SOM reduced	Voting method		
			unanimity	majority	OR voting
detection rate	100	100	100	100	100
false-positive error rate	5.33%	23.53%	1.18%	25.75%	25.75%

own writing. Approximately total 20,000 system call events have been collected and among them 10,000 are used for training and another 10,000 are for testing. 17 cases of u2r intrusion, one of the most typical intrusions, are included in the user's test data set.

We have used the best result from each model because subthreshold for each model may differ from each other. Detection rate of the combined method does not change because each model's detection rate is 100%. However, False-positive error rate has enhanced compared to the others as shown in Table 5.

5 Conclusions

In this paper, we have proposed an anomaly detection-based IDS using the privilege transition flows data and combining multiple hidden Markov models. Experimental results show that the training of the privilege transition flows is substantially faster than that of conventional data without any loss of detection rate. Moreover, modeling privilege change data has less error than conventional modeling. This method can open a new way of utilizing computation-intensive anomaly detection technique in the real world, based on behavioral constraints imposed by security policies and on models of typical behavior for users. We can also reduce false positive error rate by combining multiple hidden Markov models, thereby improving the reliability of the HMM-based intrusion detection system. It is because some attacks cannot be detected by reflecting single aspect of user events.

Currently, the ROC curve that shows the change of the error rate as the threshold changes is adopted to demonstrate the detection capability of the system. For the deployment of the system in the real world, an automatic mechanism to adjust the threshold appropriately is needed. In the future, studies on

the discriminative event extraction and modeling among user behaviors must be followed.

Acknowledgments

This research was supported by University IT Research Center Project.

References

1. H. S. Vaccaro and G.E. Liepins, "Detection of anomalous computer session activity," *In Proceedings of IEEE Symposium on Research in Security and Privacy*, pp. 280-289, 1989.
2. K. E. Price, "Host-based misuse detection and conventional operating system's audit data collection," *M.S. Dissertaion*, Purdue University, Purdue, IN, December 1997.
3. T. Lane and C. E. Brodley, "An application of machine learning to anomaly detection," *In Proceedings of the National Information Systems Security Conference*, pp. 366-380, Washington, DC, October 1997.
4. T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *In Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pp. 150-158, 1997.
5. A. K. Ghosh, A. Schwartzbard and M. Schatz, "Learning program behavior profiles for intrusion detection," *In Proceedings of Workshop on Intrusion Detection and Network Monitoring*, pp. 51-62, Santa Clara, USA, April 1999.
6. C. Warrender, S. Forrest and B. Pearlmutter, "Detecting intrusion using calls: Alternative data models," *In Proceedings of IEEE Symposium on Security and Privacy*, pp. 133-145, May 1999.
7. D. Y. Yeung and Y. Ding. "Host-based intrusion detection using dynamic and static behavioral models," *The Journal of the Pattern Recognition society*, December 2001.
8. L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, 1989.
9. L. R. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, pp. 4-16, January 1986.
10. B. A. Kuperman and Eugene H. Spafford. "Generation of application level audit data via library interposition", *CERIAS TR 99-11*, COAST Laboratory, Purdue University, West Lafaytte, IN, October 1998.
11. Kohonen, T., *Self-Organizing Maps*, Springer press, 1995.
12. S. Mukkamala, A. H. Sung and A. Abraham, "Intrusion Detection Using Ensemble of Soft Computing Paradigms," *Third International Conference on Intelligent Systems Design and Applications, Intelligent Systems Design and Applications, Advances in Soft Computing,* Springer Verlag, Germany, pp. 239-248, 2003.
13. L. Didaci, G. Giacinto and F. Roli, "Ensemble Learning for Intrusion Detection in Computer Networks," *In Proceedings of the Workshop on Machine Learning, Methods and Applications, held in the context of the 8th Meeting of the Italian Association of Artificial Intelligence (AI*IA),* pp. 10-13, September, 2002.
14. J. H. Choy and S. B. Cho, "Anomaly detection of computer usage using artificial intelligence techniques, " *Lecture Notes in Computer Scinece 2112* , pp. 31-43, Springer 2001.