Evolutionary Neural Networks for Anomaly Detection Based on the Behavior of a Program

Sang-Jun Han and Sung-Bae Cho, *Member, IEEE*

***Abstract*—The process of learning the behavior of a given program by using machine-learning techniques (based on system-call audit data) is effective to detect intrusions. Rule learning, neural networks, statistics, and hidden Markov models (HMMs) are some of the kinds of representative methods for intrusion detection. Among them, neural networks are known for good performance in learning system-call sequences. In order to apply this knowledge to real-world problems successfully, it is important to determine the structures and weights of these call sequences. However, finding the appropriate structures requires very long time periods because there are no suitable analytical solutions. In this paper, a novel intrusion-detection technique based on evolutionary neural networks (ENNs) is proposed. One advantage of using ENNs is that it takes less time to obtain superior neural networks than when using conventional approaches. This is because they discover the structures and weights of the neural networks simultaneously. Experimental results with the 1999 Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation (IDEVAL) data confirm that ENNs are promising tools for intrusion detection.**

***Index Terms*—Anomaly detection, computer security, evolutionary algorithms, intrusion detection system (IDS), neural networks.**

I. Introduction

DUE TO advances in information-communication technology, intrusion-detection systems (IDSs) have become essential tools for the security of computer systems. Generally, IDSs can be divided into two categories: misuse-detection systems and anomaly-detection systems. Misuse-detection systems use knowledge about known attacks and attempt to match current behavior against those attack patterns. Most commercial IDSs use this approach because known attacks can be detected economically and reliably with low false-positive errors. A shortcoming of this approach is that it cannot detect unknown attacks. Anomaly detection, which uses knowledge about normal behaviors and attempts to detect intrusions by noting significant deviations, has been studied actively. However, this approach suffers from high false-positive error rates because unseen normal behaviors are considered as attacks [1].

In host-based anomaly detection, the idea of learning the behavior of a program has been studied and used actively by many researchers. Normal behavior is considered from the point of view of each individual program. Profiles for each program's behavior are built, and behaviors that deviate from the profile significantly are recognized as attacks. Machine-learning methods, such as the rule-learning technique, the statistical technique, and the hidden Markov model (HMM) have been used to profile the behavior of these programs. This is because it can be viewed as a binary classification problem, which is one of the traditional problems in pattern classification [2]–[5]. Neural networks have been actively applied to IDSs. Especially, in the 1999 Defense Advanced Research Projects Agency (DARPA) intrusion detection evaluation (IDEVAL), the detection technique based on neural networks showed superior performance to the other techniques in detecting host-based attacks [6]. However, profiling normal behaviors requires much time due to the huge amount of audit data and computationally intensive learning algorithms. Moreover, to apply neural networks to real-world problems successfully, it is very important to determine the topology of the networks and the number of hidden nodes in the given problem, because performance hinges upon the structure of the neural networks. Unfortunately, although techniques have been devised to design the domain-specific network structures automatically, there is no absolute solution, [7] and typically, the network structures are designed by repeating trial-and-error cycles based on previous experiences of working on similar problems. Therefore, it takes a very long time to build normal-behavior models. This is the major drawback of the neural-network-based intrusion-detection technique.

In this paper, we employ an evolutionary neural network (ENN) to overcome these shortcomings. ENN does not require trial-and-error cycles for designing network structures and the near-optimal structure can be obtained automatically. This means that we can get better classifiers in shorter time periods. We examine the proposed method through experiments with real audit data and compare the results with those of other methods.

The rest of this paper is organized as follows. In Section II, we give a brief overview of how IDSs learn the behavior of programs. A detailed description of ENNs and the proposed methods are presented in Section III. Experimental results are shown in Section IV. The conclusion in Section V discusses future works.

II. Related Works

A. Learning the Behavior of a Program

Learning the behavior of a program is a well-known and widely used intrusion-detection paradigm. Several kinds of

Manuscript received August 3, 2004; revised April 4, 2005. This work was supported by the Ubiquitous Autonomic Computing and Network Project, the Ministry of Information and Communication (MIC), 21st Century Frontier R&D Program in Korea. This paper was recommended by Associate Editor Mark J. Embrechts.

The authors are with the Department of Computer Science, Yonsei University, Seoul 120-749, Korea (e-mail: sjhan@sclab.yonsei.ac.kr; sbcho@sclab.yonsei.ac.kr).

Digital Object Identifier 10.1109/TSMCB.2006.860136

TABLE I
REPRESENTATIVE RESEARCH ON LEARNING-PROGRAM BEHAVIOR

Researcher	Year	Techniques	Data
S. Forrest *et al.*	1996-1999	Equality matching, stide, t-stide, RIPPER, HMM	UNM
J. Stolfo *et al.*	1997-2001	RIPPER, sparse Markov tree	1998 IDEVAL, UNM
N. Ye *et al.*	2001	Decision tree, Hostelling's T^2 test, chi-square multivariate test, Markov chain	1998 IDEVAL
A.K. Ghosh *et al.*	1999-2000	Neural network, Elman network	1998, 1999 IDEVAL
V.R. Vemuri *et al.*	2002-2003	kNN, RSVM	1998 IDEVAL

intrusion can occur by inducing program misuse that exploits the bugs of the specific program. Victim programs behave differently from normal execution programs. Therefore, learning normal behaviors and recognizing significant deviations can be efficient for anomaly detection. Though there are many ways to observe the behavior of a program, capturing the system-call sequences is a typical and efficient method. In this way, a normal profile can be built by learning the patterns of the short system-call sequences. Programs that show sequences that deviate from normal sequence profiles are considered to be victims. Namely, intrusion detection by learning the behavior of a program can be transformed to the problem of learning and classifying temporal sequence data. Machine-learning techniques that are known for good solutions for this kind of problem (such as rule-learning techniques, neural networks, and the HMM) have been applied.

Program-based anomaly detection with system-call sequences can be formulated as follows. Let $P = (s_1, s_2, \ldots, s_N)$ denote the set of all system-call events generated by one execution of a program, where s_t denotes a system call event occurring at time t, and $S_t = (s_{t+1}, s_{t+2}, \ldots, s_{t+L})$, $t \leq N - L$ denotes the set of sequences made by windowing. P using a window length L at time t and R_t denotes the result of the sequence-evaluation function eval.

$$R_t = \text{eval}(S_t). \tag{1}$$

If R_t is greater than the predefined threshold, the execution of the program is recognized as an intrusion.

$$\text{alarm}(R_t) = \begin{cases} \text{normal}, & \text{if } R_t \geq \text{threshold} \\ \text{attack}, & \text{if } R_t < \text{threshold}. \end{cases} \tag{2}$$

Representative research that explores these methods is given in Table I.

Forrest and co-workers pioneered the intrusion-detection technique based on learning the behavior of a program. They proposed the techniques that use equality matching with system-call sequences as a possible prototype of the computer immune system [8]–[10]. They built the database by capturing system calls of a monitored program under normal conditions, and recognized the process that produced significantly high numbers of mismatched sequences. In [2], they compared several learning techniques such as frequency-based methods, rule learning, and HMMs. Most of their tests were conducted with University of New Mexico (UNM) datasets that are available in [11].

Cohen [12] and Lee *et al.* [3] applied a rule-learning method to learning the behavior of a program, where they proposed a modeling method that used the dynamic length of a window depending on the context of the system-call sequence [13]. They showed that the context-dependent window-length method outperforms the conventional fixed-length method with UNM and 1999 DARPA IDEVAL data. The DARPA IDEVAL dataset collected by Lincoln Labs at Massachusetts Institute of Technology (MIT) is the largest corpus of data for evaluating security systems [14], [15].

Ye *et al.* introduced three properties of audit data: the frequency property, the duration property, and the ordering property. They applied various probabilistic techniques to intrusion detection [4]. In their experiments with 1998 IDEVAL data, the Markov chain based on an ordering property showed superior performance to the other techniques. This verifies that the ordering of audit events provides useful information to detect intrusions.

Ghosh and co-workers [16]–[18] and Elman [19] exploited some variations of neural networks: the feed-forward back propagation and the Elman recurrent network for classifying system-call sequences. Their experimental results with 1998 and 1999 DARPA IDEVAL data illustrate that the Elman networks can improve the performance.

Liao and Vemuri used the k-nearest neighbor (kNN) classifier and the robust-support vector machine (RSVM) for profiling computer programs. The kNN classifier was employed in [20] with an interesting analogy between classifying text documents and detecting intrusion using the sequences of system calls. In [21], their previous method is extended and compared to the performance of the RSVM, the conventional SVM, and the kNN classifier with 1998 IDEVAL data.

B. Evolutionary Approaches for Intrusion Detection

Evolutionary approaches have been used by several researchers to optimize intrusion detectors in an automatic manner. Wang *et al.* used the evolutionary algorithm for discovering neural networks for intrusion detection [22]. The connections of the network and its weights were encoded with binary bits and evolved simultaneously. Their detection system was evaluated with World Wide Web (WWW) log data and showed an accuracy rate of 95%. However, in their experiment, they used their own dataset rather than a public-benchmark dataset. The binary encoding of the weight also limited the search space of the evolutionary algorithm highly.

Hofmann *et al.* proposed the evolutionary learning of radial-basis-function networks (RBFN) for intrusion detection [23]. They targeted a network-based IDS. Their evolutionary algorithm performed two tasks simultaneously: selecting the optimal feature set and learning the RBFN. The binary-bits system was used to encode the 137 possible features of the network packet headers and three components of the RBFN, including the type of basis function, the number of hidden neurons, and the number of training epochs. In the experiments with the network audit dataset, the RBFN optimized with the evolutionary algorithm outperformed the normal MLP and the normal RBFN.

Gonzalez *et al.* proposed an intrusion-detection technique based on evolutionary-generated fuzzy rules [24]. The condition part of the fuzzy detection rules was encoded with binary bits and fitness was evaluated using two factors: the accuracy and the coverage of the rule. The performance was compared to the methods of different genetic algorithms and without the fuzziness of rules using two network audit datasets: their own wireless dataset and the Knowledge Discovery and Data Mining (KDD) Cup 99 dataset (a small version of the 1998 DARPA dataset).

These two streams of intrusion-detection research have been developed separately in the field of computer security. However, here we propose a new hybrid intrusion-detection method by putting the paradigm of learning the behavior of a program and evolutionary learning together.

III. Intrusion Detection With ENNs

In this paper, we focus on improving the neural-network-based intrusion-detection technique. The main benefit of using a neural network is the ability to generalize from limited, noisy, and incomplete data. This generalization capability provides the potential to recognize unseen patterns, i.e., not exactly matched patterns that are different from the predefined structures of the previous input patterns. The neural network has been recognized as a promising technique for intrusion detection because the intrusion detector should ideally recognize not only previously observed attacks but also future unseen attacks. Thus, this technique can help improve the robustness of IDSs. However, in the conventional approaches, there have been some problems due to the static and regular structure of the neural networks.

First, we have to find the neural-network topology that is optimal to intrusion detection. Because the performance of the neural network is highly dependent on the network topology, it is very important to find the topology that is optimal for the problem domain. The works of Ghosh and co-workers are representative of neural-network-based intrusion-detection techniques and they showed the best performance in the 1999 DARPA IDEVAL. They applied two kinds of network topologies: the standard multilayer perceptron (MLP) and the Elman recurrent network. They applied the standard MLP for both

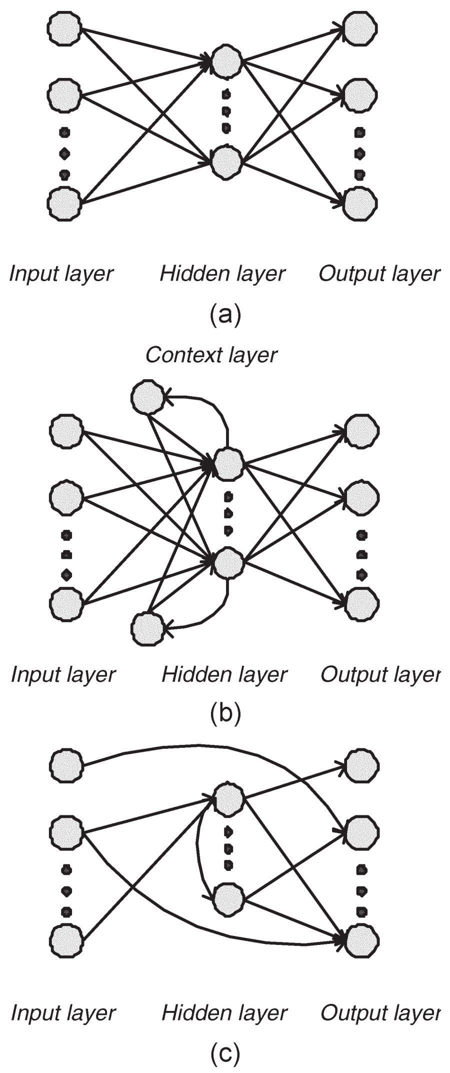


Fig. 1. Comparison of the neural-network structures. (a) Standard multilayer perceptron. (b) Elman neural network. (c) Evolutionary neural network.

program-based anomaly and misuse detections with system-call sequences [18]. To improve the detection performance of neural networks, they employed the neural networks of different topologies with the Elman recurrent network [17]. This is more suitable for processing temporal sequence data (like system-call sequences) than the standard MLP because it maintains state information between the input samples. Their experimental results verify that the Elman network is superior to the standard MLP in program-based intrusion detection. However, the topology of the Elman network is still static and regular. There is room to improve the network topology to approach the optimal structure.

Second, it is hard to find the optimal number of hidden nodes. Finding the number of hidden nodes suitable for the problem domain is also crucial to the good performance of the neural networks. However, repeating this trial-and-error process mainly finds the number of hidden nodes, which is only suitable for the given problem domain. Ghosh and co-workers trained 90 neural networks in total for each program: ten, 15, 20, 25, 30, 35, 40, 50, and 60 hidden nodes and ten networks for each number of hidden nodes. Then, a neural network that

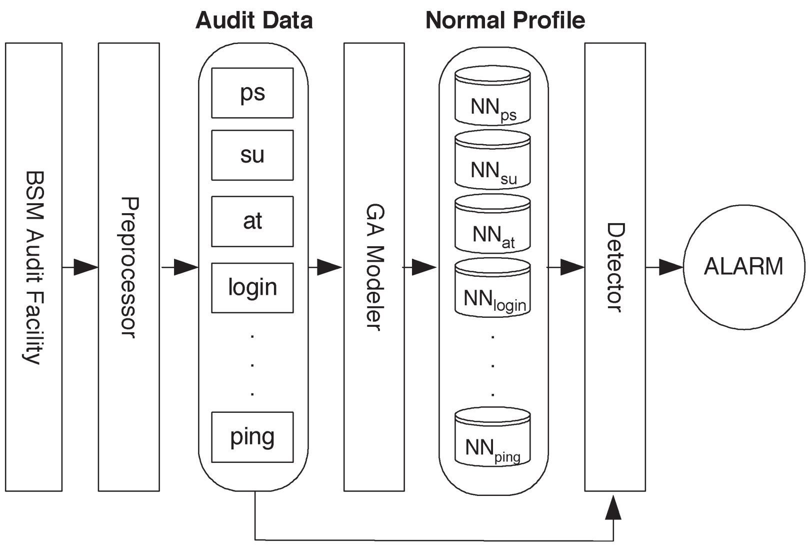


Fig. 2. Overall architecture of the proposed method.

showed the best performance against the validation data was selected [17]. In intrusion detection, generally the amount of the training data that is extremely large makes the learning algorithm more expensive. Moreover, we have to execute the learning algorithm that takes a long time because program-based intrusion detection requires many neural networks. For this reason, it is necessary to use the technique that not only defines the internal parameters of the neural network but also designs the structure of the neural network automatically.

In the field of automatic neural-network design, various constructive and pruning algorithms were proposed [7]. The constructive algorithm starts with the minimum number of hidden nodes and connections and adds necessary hidden nodes and connections during the algorithm runs. On the other hand, pruning algorithms design the network by deleting unnecessary hidden nodes and connections. However, because neither technique searches the overall structure space but only a limited area in the given environment, it is difficult to find the optimal network structure. To remedy these drawbacks, the evolutionary algorithm was introduced. Evolutionary algorithms can be applied to all general search problems and can provide a global search ability that is not sensitive to the initial condition. The neural network can be designed automatically by the evolutionary algorithm, which determines the internal weights, topologies, and the number of hidden nodes through the evolution of several generations. Moreover, we can get more sophisticated and nonregular structures that might outperform the conventional structures because there is less structural restriction in the ENN.

In this paper, the ENN-based intrusion-detection technique is proposed to overcome the drawbacks of the static and regular technique based on the conventional neural network. The ENN eliminates the trial-and-error cycles of the regular neural network because the learning of the ENN includes designing its structure. In addition, because the ENN has no structural restrictions, we can get more optimal neural networks than when using conventional neural networks with typical structures. Thus, we can obtain better neural networks with less effort in shorter time periods with ENN. Fig. 1 compares the structures of the standard MLP, the Elman network, and the ENN. While the standard MLP and the Elman network have regular topologies with static numbers of hidden nodes, the topology of the ENN is irregular, and the number of hidden nodes is determined automatically on the problem domain.

Fig. 2 illustrates the overall architecture of the ENN-based intrusion-detection method. We use system-call-level audit data provided by the Basic Security Module (BSM) of the Solaris operating system. The preprocessor monitors the execution of specified programs and generates system-call sequences by programs. The GA modeler builds normal behavior profiles using ENN. One neural network is used per program. New data is input to the corresponding neural networks. If the evaluation value exceeds the predefined threshold, the alarm is raised.

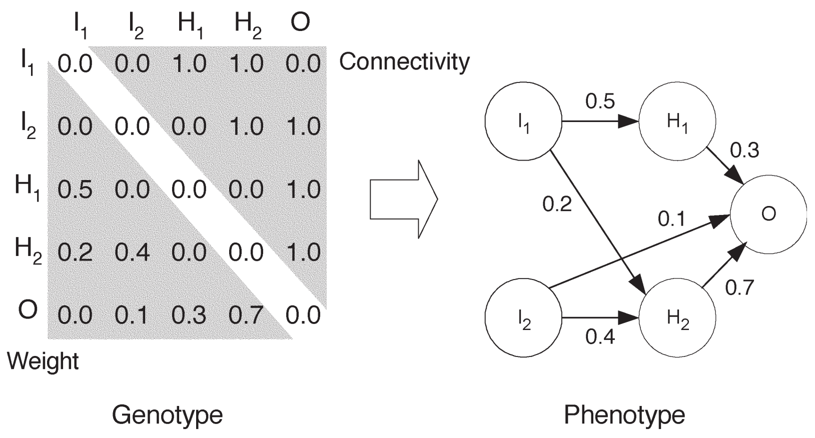
A. Modeling Normal Behavior

The overall procedure of normal-behavior modeling is shown in Fig. 3. First, the initial population of the neural networks is generated with random initial weights and full connection. Then, the individual neural network is learned partially, using an error back-propagation algorithm. After that, the fitness of each individual is assigned and the next population is generated by genetic operations. This evolution cycle is repeated until the stop criteria are satisfied or the maximum number of generations is reached.

A combination of Baldwinian and Lamarckian learning (weight training and inheritance) is applied because it enables us to speed up evolutionary learning. The search space is too large to be discovered only using a global search method (the evolutionary algorithm) within a reasonable time because the structures and weights of the neural network evolve simultaneously. Hence, the local search method is employed. The local search method (partial learning) helps discover the weights quickly and the evolutionary algorithm helps the local search algorithm avoid the local optima problem by varying the starting point of the local search. Weight learning can be

```
PROCEDURE evoNN(PopSize, MaxGeneration, DataSet)
        NEURALNETWORK Pop[PopSize]
        NUMBER Fitness[PopSize]
        NUMBER Generation := 0
        Initialize(Pop)
        WHILE Generation < MaxGeneration
                PartialLearning(Pop)
                FOR EACH Individual in Pop
                        Individual.Fitness := Test(Individual, DataSet)
                ENDFOR
                NEURALNETWORK NextPop[PopSize]
                NextPop := Selection(Pop)
                Crossover(NextPop)
                Mutation(NextPop)
                Pop := NextPop
                Generation += 1
        ENDWHILE
END PROCEDURE
```

Fig. 3. Procedure for evolving neural networks.



	I_1	I_2	H_1	H_2	O
I_1	0.0	0.0	1.0	1.0	0.0
I_2	0.0	0.0	0.0	1.0	1.0
H_1	0.5	0.0	0.0	0.0	1.0
H_2	0.2	0.4	0.0	0.0	1.0
O	0.0	0.1	0.3	0.7	0.0

Fig. 4. Example of genotype–phenotype mapping.

accelerated faster with weight inheritance, which enables us to use the result of the local search in the evolution process directly. The back-propagation algorithm that is a standard and widely used training algorithm is used to train the weights.

There are two approaches in employing the neural network for evaluating the abnormality of system-call sequences. The first approach is classifying the current sequence into two classes: normal behavior and attack behavior. It is sufficient to use the standard neural network in this case, where the neural network has two output nodes that correspond to the normal and attack behaviors, respectively. The second approach is predicting the next sequence at time $n + 1$ with the current input at time n and comparing the result with the real sequence at time $n + 1$. In the work of Ghosh and co-workers, they used this method because the Elman network was designed for retaining information concerning previous inputs. In this case, the number of output nodes is the same as that of the input nodes. The input at time $n + 1$ is compared with the output at time n, and the difference is used as the measure of abnormality. Because our ENN does not use temporal information, the first approach is taken. There are L input nodes because the system-call sequence S_t, which is generated at time t with window length L is used as the input. Ten input nodes are used because we set the window length as 10. The maximum number of hidden nodes is 15. The number of hidden nodes and the connectivity among them are determined dynamically by the evolutionary algorithm.

The anomaly detector uses only attack-free data in the training phase, but to train the supervised learner (such as a neural network), the dataset labeled as attack is also needed. Ghosh and co-workers discussed the neural network that classified all data as attack by default, using randomly generated data as anomalous input. Then, the real normal data were exposed in order to classify the normal data at a particular area of input

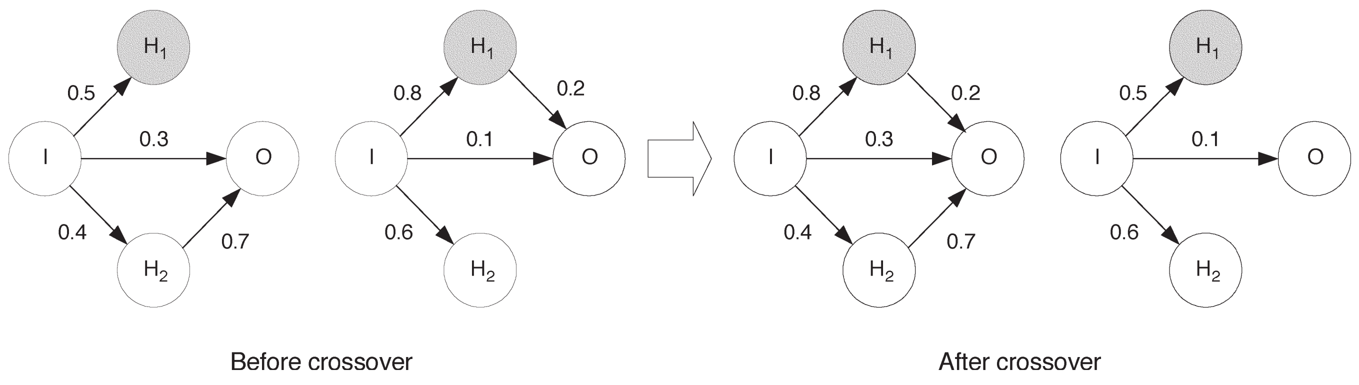


Fig. 5. Example of crossover operation.

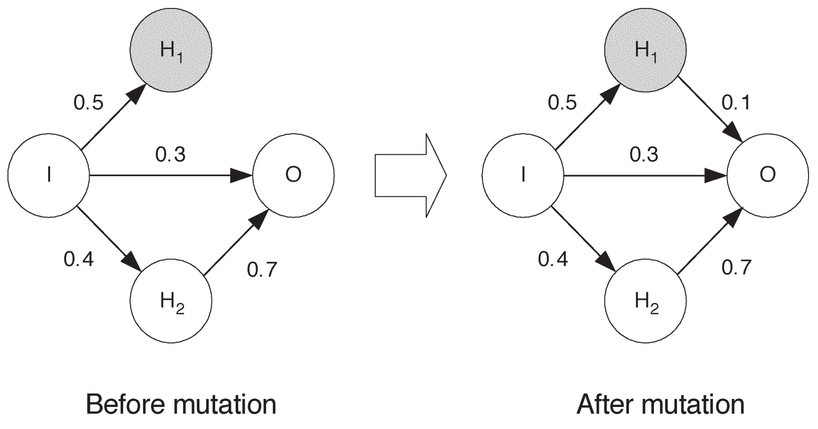


Fig. 6. Example of mutation operation.

space [17]. Their method works well, except that the neural network sometimes forgets the anomalous inputs previously exposed while learning the normal data. In this paper, we take a variant approach. The system-call sequences are generated at random and used as anomalous data. The training data is generated by mixing real normal sequences and artificial intrusive sequences. This is to prevent the forgetting of the input samples. The artificial and real data are mixed at the ratio of 2 to 1. Artificial sequences help neural networks form the decision boundary between the normal sequences and the intrusive sequences correctly. In this way, we can also obtain the neural network that classifies all system-call sequences (except the given normal sequence) as attack behavior.

There are three major problems in applying the evolutionary algorithm to real-world problems: how to define the genotype representation of the solution, what kinds of genetic operators are appropriate, and how to evaluate the fitness of each individual. In this paper, we have applied the evolutionary algorithm to neural-network learning as follows.

1) Genotype Representation: There are several genotype representations for neural networks. These include binary, tree, linked list, and matrix representation. We have used a matrix-based genotype representation because it is straightforward to implement and easy to apply genetic operators. When N is the total number of nodes in a neural network including input, hidden, and output nodes, the matrix is $N \times N$, where the entries consist of connection links and the corresponding weights. In this model, each neural network uses only forward links. In the matrix, the upper right triangle (see Fig. 4) has the connection-link information, where 1 means that there is a connection link, and 0 means that there is no connection link. The lower left triangle describes the weight values that correspond to the connection-link information. Fig. 4 shows an example of genotype–phenotype mapping for a neural network that has two input nodes, two hidden nodes, and one output node. The number of hidden nodes can vary within the maximum number of hidden nodes in the course of genetic operations.

The total number of neurons can be created by subtracting the number of hidden nodes that have no connection or are not reachable from the input neurons from the maximum number of neurons N. The number of neurons in a certain layer cannot be counted because there is no restriction in the network structure. In the traditional MLP of fixed and fully connected, there is an explicit layer structure (one input and one output layer, one or multiple hidden layers), and all neurons in each layer are fully connected with the neurons in the previous layer. However, in our ENN, the layer structure is not fixed. The input neurons can be connected directly with the output neurons as well as with the hidden neurons. So, we cannot define which hidden neurons belong to a certain layer.

This matrix-based genotype representation can include some network structures and nodes that are redundant and meaningless. We inhibit the emergence of the connections between the input neurons and the connections between the output neurons at the initialization and genetic-operation stages because the back-propagation algorithm cannot be applied to this kind of

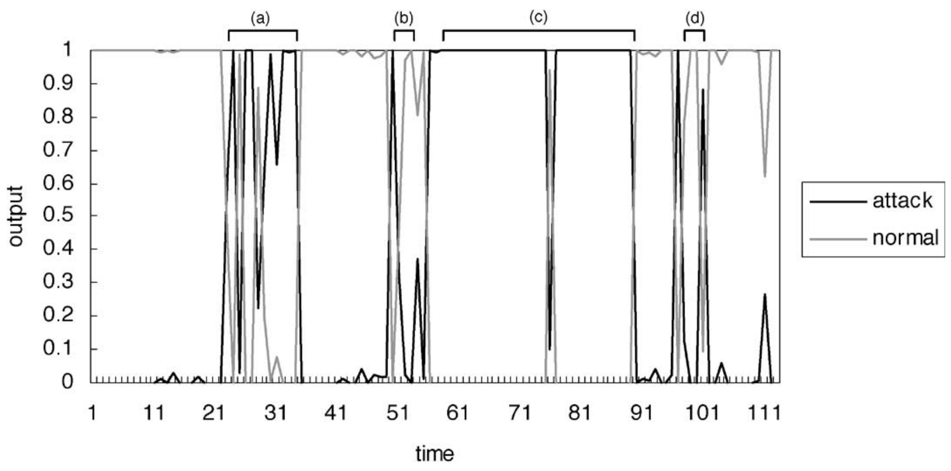


Fig. 7. Changes of output values as a U2R attack proceeds. (a), (b), and (d): Short fluctuations of the output values of the normal events. (c) Temporal locality of the anomalous events.

TABLE II
LIST OF SYSTEM CALLS FREQUENTLY USED

exit	fcntl	ioctl
fork	rename	pipe
creat	mkdir	setuid
unlink	fchdir	utime
chdir	pathconf	setgid
chown	open - read	mmap
access	open - write	audit
stat	open - write,creat	munmap
lstat	open - write,trunk	seteuid
readlink	open - write,creat,trunk	putmsg
execve	open - read,write	getmsg
vfork	open - read,write,crea	auditon
setgroups	close	memcntl
setpgrp	getaudit	sysinfo

TABLE III
LIST OF THE SETUID PRIVILEGED PROGRAMS

accton	ff.core	ps	sulogin
admintool	ffbconfig	pt_chmod	top
allocate	kcms_calibrate	ptree	ufsdump
at	kcms_configure	pwait	ufsrestore
atm	list_devices	quota	uptime
atq	login	rcp	utmp_update
chkey	mkcookie	rdist	volcheck
crontab	mkdevalloc	rlogin	w
ct	mkdevmaps	rsh	whodo
deallocate	newgrp	sacadm	xlock
eject	nispasswd	sendmail	yppasswd
exrecover	passwd	ssh	
fdformat	ping	su	

connection. The hidden neurons without any connection can emerge as a result of the operation. These kinds of neurons are ignored at the partial-learning stage (back-propagation algorithm).

2) Genetic Operations: The crossover operator produces a new descendant by exchanging partial sections between two neural networks. It selects two distinct neural networks randomly and chooses one hidden node as the pivot point. Then, they exchange the connection links and the corresponding weight information based on the selected pivot point as shown in Fig. 5.

The mutation operator changes a connection link and the corresponding weight of a randomly selected neural network. It performs one of two operations: addition of a new connection or deletion of an existing connection. The mutation operator selects two nodes of a neural network randomly. If there is no connection between them, it connects two nodes with random weights. Otherwise, it removes the connection link and weight information. Fig. 6 shows an example of the mutation operation.

3) Fitness Evaluation: Fitness is calculated as the detection rate for the training data. The most popular selection method is the roulette-wheel selection, but a problem with this method is that the population converges into one locally optimal solution rapidly when the fitness values of the population differ greatly. For this reason, we have used rank-based selection, in which the individuals' selection probabilities are assigned according to the ranks of the individuals, which are based on the fitness values.

B. Anomaly Detection

Fig. 7 illustrates the changes of the output values as a user to root (U2R) attack that gains the root privilege at time 60. We can see that the output value of the attack node increases dramatically and remains constant for some time in (c), which begins when the attack succeeds. In (a), (b), and (d), the output value of the attack node fluctuates and exceeds the output value of the normal nodes for very short time periods, but it is not sufficient to decide if the process can be classified as attack. Therefore, it is important to recognize the temporal locality of anomalous events like that presented in (c). To do this, it is necessary to consider the previous output values as well as the current output values. For this purpose, we have defined a

TABLE IV
ATTACKS INCLUDED IN THE TEST DATA

Name	Description	Count
eject	Exploiting buffer overflow in the 'eject' program	2
ffbconfig	Exploiting buffer overflow in the 'ffbconifg' program	2
fdformat	Exploiting buffer overflow in the 'fdformat' program	3
ps	Race condition attack in 'ps' program	4

new measure of abnormality that has a leaky integrator. When o_1^t denotes the output value of the attack node, o_2^t denotes the output value of the normal node, and w_1, w_2, and w_3 denote the weights of these values, the evaluation score r_t is calculated as follows:

$$r_t = w_1 \cdot r_{t-1} + w_2 \cdot o_t^1 + w_3 \cdot o_t^2. \tag{3}$$

It retains the evaluation value of past evaluations with decay, and we get higher abnormality in the current process, as the output value of the attack node is higher and the output value of the normal node is lower. In this way, we can measure the abnormality of the behavior of the program robustly to reduce fluctuation and recognize the temporal locality of abnormal behavior like that in (c).

A threshold is defined to check whether the abnormality exceeds it, determining whether the current process is attack or not. However, the decision boundaries vary from program to program because different neural networks are used to evaluate different program behavior. Thus, applying a threshold to overall neural networks is not feasible. To solve this problem, we have normalized the evaluation values statistically. First, we test the training data using the trained neural network and calculate the mean and variance of r_t. Then, under the assumption that r_t is normally distributed, we transform r_t to the corresponding value in the standard normal distribution R_t. When m is the mean of r_t and d is the standard deviation against the training data, the normalized evaluation value R_t is calculated as follows:

$$R_t = \text{eval}(S_t) = \frac{r_t - m}{d}. \tag{4}$$

If R_t exceeds the predefined threshold, the current process is considered to be an attack.

IV. EXPERIMENTS

A. Experimental Settings

In BSM audit data, there are about 280 system-call events, but rarely used system calls are not significant to represent the program's normal behavior. Therefore, we select only critical system calls to model normal behaviors by selecting the system calls in the training data of the first day, resulting in 45 system calls selected. The system calls are assigned numbers between 0 and 44 and all the remaining system calls are assigned to 45. Table II shows the 45 system calls that are selected from the training data.

To verify the proposed method, we used the 1999 DARPA intrusion-evaluation dataset [25]. It contains four kinds of attacks: denial of service, probe, remove to local (R2L), and U2R. In this paper, our experiments focus on detecting attempts of U2R attacks to gain root privilege by privileged program misuse. Thus, we monitor only the SETUID privileged programs that are the target of most U2R attacks. Table III is the list of the SETUID privileged programs that run on the victim Unix servers.

TABLE V
PARAMETERS OF THE ENN

Parameter	Value
population size	20
# of generation	100
crossover rate	0.3
mutation rate	0.08
# of input nodes	10
maximum # of hidden nodes	15
# of output nodes	2
w_1 of Eq. 3	0.5
w_2 of Eq. 3	0.5
w_3 of Eq. 3	-0.5

The 1999 IDEVAL data consists of five weeks of audit data. The 1–3-week dataset is used for training and the 4–5-week dataset is used for testing. We used the 1 and 3 week dataset, which does not contain any attacks for training neural networks. The 4–5 week dataset was used for testing. The test data contain 11 instances of four types of U2R attacks as shown in Table IV.

The parameter values used in the experiment are given in Table V. The weight to $r_{t-1} (w_1)$ and weight to the output value of the attack node (w_2) are set to 0.5. Consequently, the input that is obviously intrusive results in 1. The weight to the output value of the normal node (w_3) is set to negative (-0.5) to cut down the r_t of the partially intrusive input.

B. Results

1) Changes of Fitness: We investigated the change of fitness to find whether the neural networks evolve or not, because it is possible for good neural networks to be obtained not by evolution but by chance. Fig. 8 shows the changes of fitness. Fitness increases as evolution proceeds and the maximum fitness converges to 0.9. This shows that we can find better neural networks by evolution and that the evolved neural networks can classify the training data at about 90% accuracy.

2) Changes of Network Structure: Fig. 9 illustrates how the number of connections and hidden nodes changes as evolution

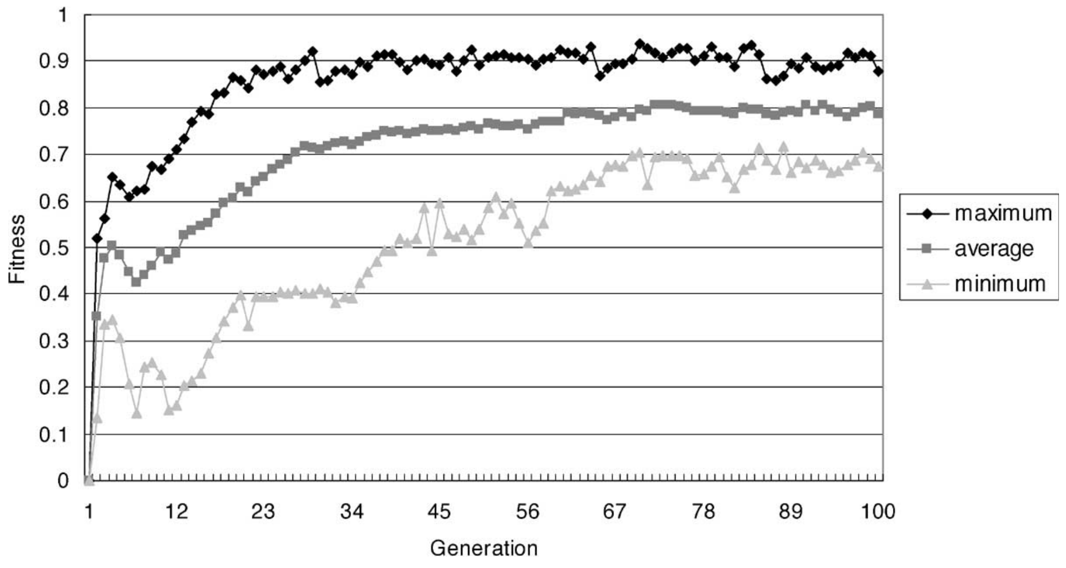


Fig. 8. Changes of fitness as evolution proceeds.

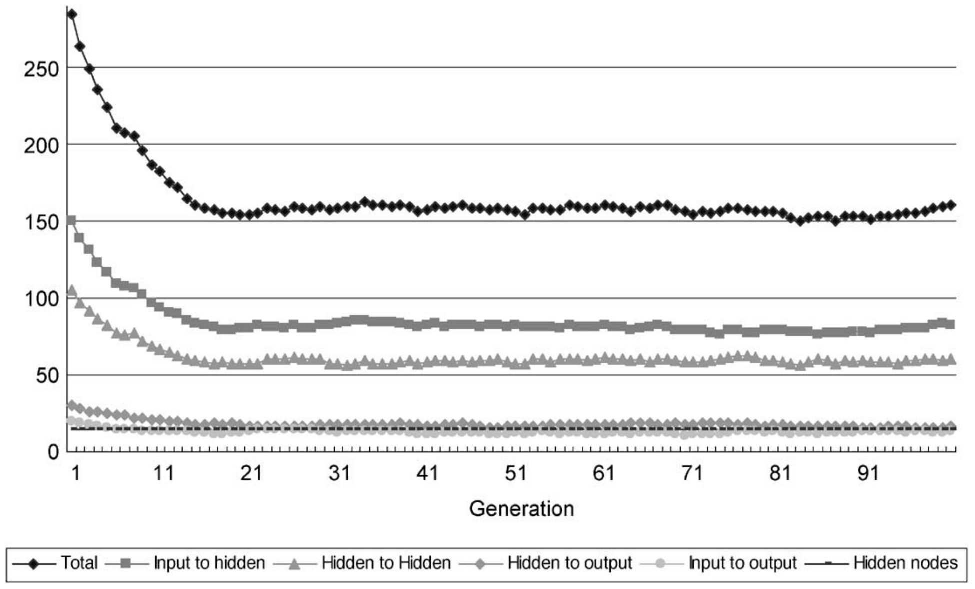


Fig. 9. Changes of network structure as evolution proceeds.

proceeds. At the initial generation, the neural networks have in total 285 connections because the ENN starts with a set of full connections. The number of connections decreases as evolution proceeds, while the fitness increases. This shows that the evolutionary algorithm optimizes the network structure by pruning the unnecessary connections. Meanwhile, the number of hidden nodes does not change until the last generation. This means that the evolution algorithm does not eliminate any hidden node to maintain the representational power.

3) Comparison of Training Time: The time required for training the typical neural network of the static structure and the ENN is compared. The training program was run on a computer with a dual Intel Pentium Zeon 2.4-GHz processor, 1-GB RAM, and Sun Solaris 9 operating system. For neural networks with static structures, the number of hidden nodes varied from ten and 60, and for each number of hidden nodes,

TABLE VI
COMPARISON OF THE TIME REQUIRED FOR TRAINING

Type	Number of hidden nodes	Time(sec)
NN of static structure	10	235.5
	15	263.4
	20	454.2
	25	482
	30	603.6
	35	700
	40	853.6
	50	1216
	60	1615
	Sum	17 hours 50 mins
ENN	15	1 hour 14 mins

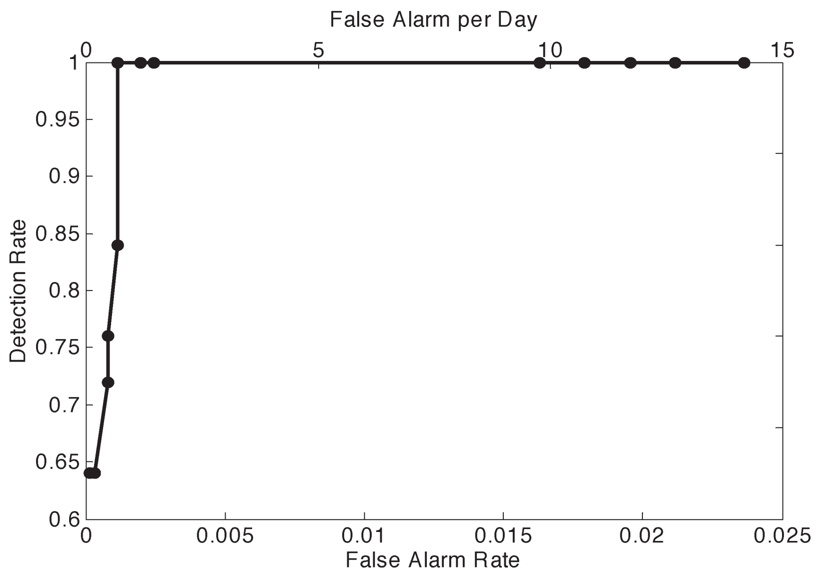


Fig. 10. Intrusion-detection performance of ENN.

ten networks were trained. A total of 90 networks were trained. Error back-propagation algorithm was iterated until it reached 5000 epochs. The ENN has at most 15 hidden nodes, and 20 neural networks were evolved to the 100th generation. Both neural networks had ten input nodes and two output nodes and were trained with the training data of the login program that consisted of 1905 sequences.

Table VI illustrates the result of this experiment. The conventional approach that repeats the trial-and-error cycle requires about 17 h and 50 min. However, in the case of the ENN, it only takes 1 h and 14 min. Thus, the evolutionary approach can reduce the learning time. Another advantage is that the near-optimal network structure can be obtained.

4) Comparison of Detection Performance: To show the performance visually, the receiver operating characteristic (ROC) curve, which is a traditional way to represent the performance of the classifier, is used. However, the detection/false-alarm plot has been used widely in the field of intrusion detection since the MIT Lincoln Laboratory began to use this kind of plot. It is basically the same (with the traditional ROC curve), except that the number of true negatives per day is plotted in the horizontal axis. Therefore, we have also added the axis of false alarm per day in the ROC curve to yield fair comparison with the previous results over the same dataset. The neural network with the highest fitness is selected and used for testing. Fig. 10 depicts the ROC curve, which illustrates the intrusion-detection performance of the proposed method. It produced 0.7 false alarms (0.0011% false-alarm rate) at 100% detection rate: There were seven false alarms in the whole test dataset. Most false alarms were raised from the "ffbconfig" program, of which the training data were not sufficient. The ENN dataset with a sufficient amount of training detected intrusions almost perfectly. In the 1999 DARPA IDEVAL, the method that showed the best performance when detecting U2R attacks is the work of Ghosh and co-workers that learns system-call sequences with the Elman recurrent neural network [6]. It showed three false alarms at a 100% detection rate [16].

TABLE VII
COMPARISON OF INTRUSION-DETECTION PERFORMANCE WITH THE BEST EXISTING RESULTS ON THIS DATA

Type	False alarms	Detection rate
ENN	0.7	100%
Elman network	3	100%

Although the work of Ghosh and co-workers was produced without knowledge of the data, the performance of the ENN in our experiment was comparable to that of the work of Ghosh and co-workers. This result illustrates that the ENN can produce better neural networks than conventional neural networks of static and regular structure. Table VII summarizes the intrusion-detection performance of the ENN and the Elman network.

5) Comparison of Network Structures: The structure of the neural network that shows the best performance to learn the behavior of the ps program is shown in Fig. 11. The structure is more complex than the general MLP because the ENN does not restrict the topology of the network. Table VIII compares the ENN trained with the behavior of the ps program, the standard MLP, and the Elman network in terms of network structure. The MLP and the Elman network have the same number of nodes: ten input nodes, 15 hidden nodes, and two output nodes. In the case of the Elman network, there are 15 context nodes because each hidden node has a corresponding context node.

In terms of the total number of connections, the Elman network has the largest number of connections because the context node receives input from a single hidden node and sends its output to all nodes in the hidden layer. The total number of connections of the standard MLP and ENN does not differ much. However, the ENN has more various types of connections, including the connections from the input node to the output node and from one hidden node to another hidden node. The Elman network produced better performance by retaining the contextual information between samples with

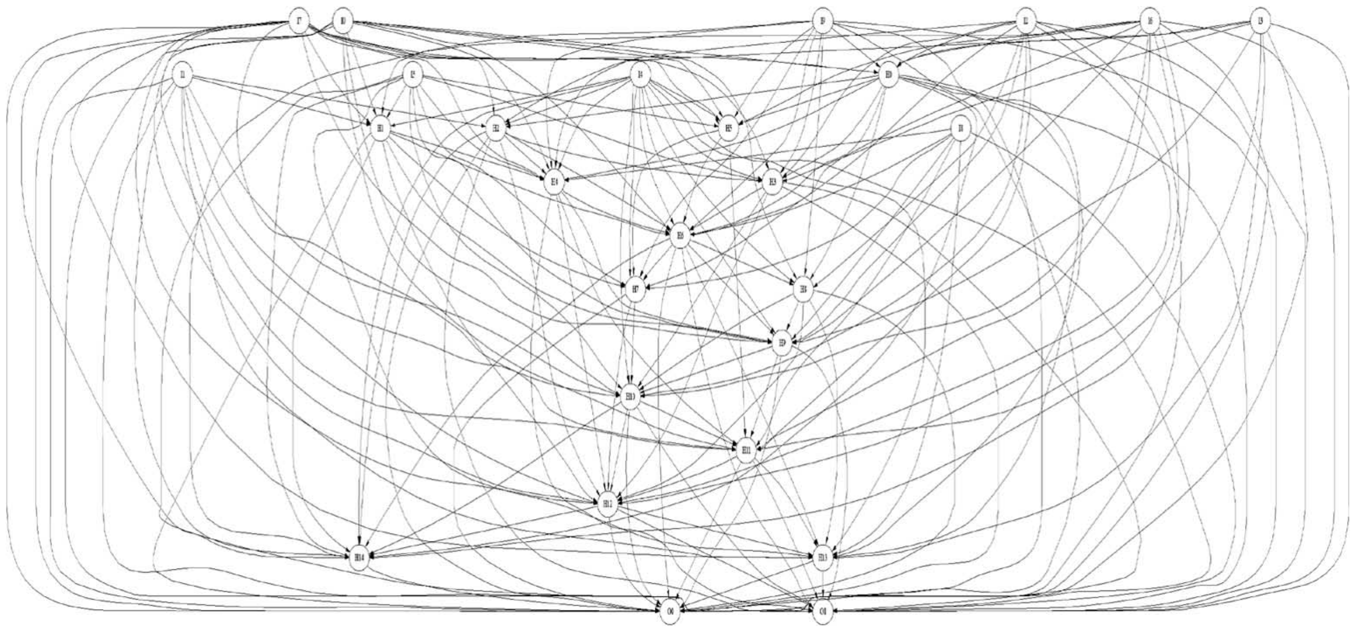


Fig. 11. Structure of neural network that shows the best performance in learning ps program.

TABLE VIII
COMPARISON OF NETWORK STRUCTURE. (A) MLP; (B) ELMAN NETWORK; (C) ENN

(a)

From\To	Input	Hidden	Output
Input	0	150	0
Hidden	0	0	30
Output	0	0	0

(b)

From\To	Input	Context	Hidden	Output
Input	0	0	150	0
Context	0	0	225	0
Hidden	0	15	0	30
Output	0	0	0	0

(c)

From\To	Input	Hidden	Output
Input	0	86	15
Hidden	0	67	19
Output	0	0	0

recurrent topologies by adding the context layer. On the other hand, the ENN attempted to increase the modeling power by forming a nonregular and complex network structure.

V. CONCLUSION

This paper proposes an ENN for improving the performance of anomaly-detection techniques based on learning the behavior of a program. The proposed method not only improved the detection performance but also reduced the time required for training. This is because it learned the structures and weights of the neural network simultaneously. In experiments with the 1999 DARPA IDEVAL data, the ENN-based detector showed good detection performance compared to the work of Ghosh and co-workers, which produced the best performance in the 1999 DARPA IDEVAL program. We verified that the time required for learning can be reduced without any loss of detection performance.

As a future work, it is necessary to find network structures that are particularly good for intrusion detection by analyzing the evolved structures. Further improvement of detection performance can be expected. By combining multiple expert neural networks that are evolved with speciation, we can obtain more accurate models. Another possible solution is to design an evolutionary algorithm that adopts the advantage of a dynamic window length by optimizing the window length as well as the network structure. The problem of long learning time should be dealt with also. A probable solution to this is to speed up the partial learning stage by substituting the back-propagation algorithm with an up-to-date fast training algorithm such as a scaled conjugate gradient. Applying the ENN described here to other types of datasets will be a good extension of this work.

REFERENCES

[1] T. F. Lunt, "A survey of intrusion detection techniques," *Comput. Secur.*, vol. 12, no. 4, pp. 405–418, Jun. 1993.

[2] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Proc. IEEE Symp. Security and Privacy*, Oakland, CA, May 1999, pp. 133–145.

[3] W. Lee, S. Stolfo, and P. K. Chan, "Learning patterns from Unix process execution traces for intrusion detection," in *Proc. AAAI Workshop AI Methods Fraud and Risk Management*, Providence, RI, 1997, pp. 50–56.

[4] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 31, no. 4, pp. 266–274, Jul. 2001.

[5] S.-B. Cho, "Incorporating soft computing techniques into a probabilistic intrusion detection system," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 2, pp. 154–160, May 2002.

[6] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Comput. Netw.*, vol. 34, no. 4, pp. 579–595, Oct. 2000.

[7] X. Yao, “Evolving artificial neural networks,” *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[8] S. Forrest, S. A. Hofmeyr, and A. Somayaji, “A sense of self for Unix processes,” in *Proc. IEEE Symp. Research Security and Privacy*, Oakland, CA, 1996, pp. 120–128.
[9] ——, “Computer immunology,” *Commun. ACM*, vol. 40, no. 10, pp. 88–96, 1997.
[10] S. A. Hofmeyr, A. Somayaji, and S. Forrest, “Intrusion detection using sequences of system calls,” *J. Comput. Secur.*, vol. 6, no. 3, pp. 151–180, 1998.
[11] University of New Mexico, *Computer Immune Systems—Data Sets and Software.* [Online]. Available: http://www.cs.unm.edu/immsec/systemcalls.htm
[12] W. W. Cohen, “Fast effective rule induction,” in *Proc. 12th Int. Conf. Machine Learning*, Tahoe City, CA, 1995, pp. 115–123.
[13] E. Eskin, W. Lee, and S. J. Stolfo, “Modeling system calls for intrusion detection with dynamic window sizes,” in *Proc. DARPA Information Survivability Conf. and Expo. (DISCEX II)*, Anaheim, CA, 2001, pp. 165–175.
[14] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Webber, S. Webster, D. Wyschograd, R. Cunninghan, and M. Zissan, “Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation,” in *Proc. DARPA Information Survivability Conf. and Expo.*, Los Alamitos, CA, 2000, pp. 12–26.
[15] K. Kendall, “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems,” M.S. thesis, Dept. Elect. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, 1998.
[16] A. K. Ghosh, C. C. Michael, and M. A. Schatz, “A real-time intrusion detection system based on learning program behavior,” in *Proc. 3rd Int. Symp. Recent Advances Intrusion Detection*, Toulouse, France, 2000, pp. 93–109.
[17] A. K. Ghosh, A. Schwartzbard, and M. Schatz, “Learning program behavior profiles for intrusion detection,” in *Proc. 1st USENIX Workshop Intrusion Detection and Network Monitoring*, Santa Clara, CA, Apr. 1999, pp. 51–62.
[18] A. K. Ghosh and A. Schwartzbard, “A study in using neural networks for anomaly and misuse detection,” in *Proc. 8th USENIX Security Symp.*, Washington, DC, 1999, pp. 23–36.
[19] J. L. Elman, “Finding structure in time,” *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, 1990.
[20] Y. Liao and V. R. Vemuri, “Use of K-nearest neighbor classifier for intrusion detection,” *Comput. Secur.*, vol. 21, no. 5, pp. 439–448, Oct. 2002.
[21] W. Hu, Y. Liao, and V. R. Vemuri, “Robust support vector machines for anomaly detection in computer security,” in *Proc. Int. Conf. Machine Learning and Applications*, Los Angeles, CA, Jun. 2003, pp. 168–174.
[22] L. Wang, G. Yu, G. Wang, and D. Wang, “Method of evolutionary neural network-based intrusion detection,” in *Proc. Int. Conf. Info-tech and Info-net*, Beijing, China, Oct. 2001, vol. 5, pp. 13–18.
[23] A. Hofmann and B. Sick, “Evolutionary optimization of radial basis function networks for intrusion detection,” in *Proc. Int. Joint Conf. Neural Networks*, Portland, OR, Jul. 2003, vol. 1, pp. 415–420.
[24] F. Gonzalez, J. Gomez, M. Kaniganti, and D. Dasgupta, “An evolutionary approach to generate fuzzy anomaly signatures,” in *Proc. 4th Annu. IEEE Information Assurance Workshop*, West Point, NY, Jun. 2003, pp. 251–259.
[25] MIT Lincoln Laboratory, *DARPA Intrusion Detection Evaluation*. [Online]. Available: http://www.ll.mit.edu/IST/ideval/index.html

Sang-Jun Han received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 2002 and 2004, respectively.

Since 2004, he has been an Assistant Engineer at Digital Media R&D Center, Samsung Electronics Co., Ltd., Suwon, Korea. His research interests include evolutionary computation, pattern recognition, and intelligent man–machine interfaces.

Sung-Bae Cho (S’88–M’90) received the B.S. degree in computer science from Yonsei University, Seoul, Korea, in 1988 and the M.S. and Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, in 1990 and 1993, respectively.

From 1991 to 1993, he worked as a Member of the Research Staff at the Center for Artificial Intelligence Research, KAIST. From 1993 to 1995, he was an Invited Researcher of Human Information Processing Research Laboratories at the Advanced Telecommunications Research (ATR) Institute, Kyoto, Japan. In 1998, he was a Visiting Scholar at University of New South Wales, Canberra, Australia. Since 1995, he has been a Professor in the Department of Computer Science, Yonsei University. His research interests include neural networks, pattern recognition, intelligent man–machine interfaces, evolutionary computation, and artificial life.

Dr. Cho is a member of the Korea Information Science Society, International Neural Network Society (INNS), the IEEE Computer Society, and the IEEE Systems, Man, and Cybernetics Society. He was awarded outstanding paper prizes from the IEEE Korea Section in 1989 and 1992 and another one from the Korea Information Science Society in 1990. In 1993, he also received the Richard E. Merwin Prize from the IEEE Computer Society. In 1994, he was listed in *Who’s Who in Pattern Recognition* from the International Association for Pattern Recognition and received the Best Paper Awards at the International Conference on Soft Computing in 1996 and 1998. In 1998, he received the Best Paper Award at World Automation Congress. He was listed in *Marquis Who’s Who in Science and Engineering* in 2000 and in *Marquis Who’s Who in the World* in 2001.