ON HANDLING REPLAY ATTACKS IN INTRUSION DETECTION SYSTEMS

A. M. Sokolov, D. A. Rachkovskij

Abstract: We propose a method for detecting and analyzing the so-called replay attacks in intrusion detection systems, when an intruder contributes a small amount of hostile actions to a recorded session of a legitimate user or process, and replays this session back to the system. The proposed approach can be applied if an automata-based model is used to describe behavior of active entities in a computer system.

Keywords: intrusion detection, replay attack, probabilistic finite automata, dynamic programming, IDS, PFA, DP

1. Introduction

Intrusion detection systems (IDS) are aimed at detecting and preventing intrusive activities that were not detected by common security mechanisms of a computer system. These are ill-intended activity of legitimate users, outsider's attacks that have passed through a firewall, the use of stolen passwords, and any other activity that was not prevented by authentication, authorization, or other security subsystems.

Two categories of IDS are usually distinguished – *misuse detection systems* and *anomaly detection systems*. The former make use of traces or templates of known attacks, while the latter build profiles of non-anomalous behavior of computer system's active subjects. Both types of IDS have their advantages and drawbacks. Misuse detection systems are perfect in detecting attacks that match one of the predefined templates. However, in case of an unknown attack or a slight variation of a known one, they usually fail. Anomaly detection systems, on the contrary, learn to recognize non-intrusive behavior and try to detect deviations from it, i.e., anomalies. They are more suitable to alarm at a previously unseen attack, because, as is hypothesized, hacker's activity is different from that of a legitimate user. However, there always remains a possibility that an attack is detected while there is really no intrusion (*false positive*), or a real intrusion activity is unnoticed (*false negative*).

In this paper, we consider the problem of handling a kind of attacks the majority of IDS are vulnerable to. These are the so-called replay attacks – when an intruder acts as if she is a legitimate user, though still performing some intrusive activity. This can be done by contributing a negligible number of hostile actions to a recorded log of a legitimate user or process.

The rest of the paper proceeds as follows. In section 2, we briefly survey existent approaches to the modeling of behavior of active entities in computer systems. Section 3 explains what is a replay attack, and the main idea of our method. Three types of possible intruder's actions and their handling are described in section 4. In section 5, we make conclusions.

2. Methods of anomaly detection

Usually, four major methods of anomaly detection techniques are distinguished: instance-based learning methods, frequency methods, neural-network approaches, and finite automata methods.

Instance-based approaches are the simplest because they just memorize all seen subsequences of elements from the training data. Then, an audit sequence to be checked is considered anomalous if it contains subsequences not present in the previously memorized set. Examples of this approach are given in [1,2,3]. The authors claim that because of the restrictions operating systems impose on the variety of possible signals in the audit logs, the data sets of behavior examples will not be too large. Otherwise, it is still possible to implement some of the size-reducing techniques [3].

Frequency-based approaches are a development of instance-based methods. They check whether the statistics gathered about a particular event falls into a predefined interval or is equal to a predefined value. The interval or the value may be estimated from the training data without anomalous traces or set by an expert. This approach was used in one of the early papers that considered the need for IDS [4].

Lately, *neural-network* approaches have emerged in the field of IDS. Usually, perceptron-like networks trained by backpropagation are used [5,6,7]. The input to the network is composed of selected system parameters, and the output may be either an *anomalous/normal* behavior indicator or one of the system parameters. If,

during the process of testing, the output of the network deviates from the current value of the parameter it represents, an anomaly is signaled.

Numerous methods infer a structure and parameters of sequence-processing finite automata. These automata can be nondeterministic, with their structure obtained from analyzing computer programs' sources [8] or from sequences of system calls [9]. Non-deterministic automata can be used to model subjects with a limited behavior (e.g., programs making system calls). Probabilistic automata usually infer their structure from data originating from humans [10,11,12], that are of probabilistic nature (e.g., shell commands). One of the simplest kinds of probabilistic automata that can be used to model sequences of audit events are Markov chains [10,11]. A more powerful type of probabilistic automata are Hidden Markov Models (HMM), which were also tested in the field of anomaly detection [12].

3. Modeling replay-attacks

IDS, and particularly anomaly detection systems, are known to be vulnerable to replay attacks. If an intruder gains access to the system audit logs or is able to eavesdrop user sessions, a replay attack can be carried out. Usually, an intruder replays the recorded session back to the system, with only a few inserted, deleted, or altered elements that serve her aim. Due to the negligible quantity of hostile elements in the sequence, this replayed session will most likely be marked by an IDS as legitimate. Almost every IDS can be fooled in such a way, especially those that mainly rely on frequency-based methods.

In [12], there was an attempt to tackle replay attacks by introducing an upper limit on the possible value of the probability of a session. If the probability of a session is higher (a "too similar" session), it is considered an anomalous repetition of some previously seen session, i.e., a replay attack. Accordingly, sessions with the probability lower than the limit are not considered as containing replay attacks. This method has some serious flaws. In computer systems, namely in UNIX-based ones, there usually exists a plenty of pseudo-users running in the background – the so-called daemons. Daemons tend to generate highly regular log records, because they usually perform fixed and simple tasks. So, the probability of a daemon's session could evaluate to high values, even in the case when there is no replay attack present. Therefore, marking those sessions as anomalous would result in high false positive error rate. The second drawback is that this method does not allow us to identify added, altered, or deleted elements in a log sequence. Thus, even if we succeeded in detecting an attack, we would not be able to point the intruder's contribution out.

We will further describe our method for detecting and analyzing replay attacks that deals with the above drawbacks.

3.1 Underlying model

Unlike the situation in pattern recognition tasks, observed sequences of audit events are usually not distorted. For example, in speech, the same words manifest themselves differently due to individual characteristics of a speaker or peculiarities of the environment. But in case of computer audit events, we usually observe events their creator meant to create. Hence, it is not obligatory to introduce unobservable states having a meaning of "what really occurred", as in HMM [12], or to use advanced structure-processing methods that handle sequence components possessing a gradual degree of similarity [13, 14].

Our method is based on recovering the sequence of states that led to generation of a given log session and comparison with the most probable sequence of states passed to generate the same session with the consideration of intruder's contribution. We take Deterministic Probabilistic Finite Automaton (DPFA) [15] as our basic model. In DPFA, states emit symbols probabilistically, but the next state is uniquely determined by the current state and the symbol. With this case of automata, it is possible to recover the sequence of states that generated given output symbol sequence, provided we know the initial state. A DPFA with observable states having direct interpretation can suffice, e.g., Markov chains with either fixed or variable memory length [16] (applications of Markov models to the anomaly detection task were mentioned in section 2). Then, every state of an automaton will have a sequence of symbols (events) associated with it. Those symbols are observable in the log sequence, just like in the common Markov chains.

Let $M=\langle Q,\Sigma,p_0,p\rangle$ be a DPFA, where Q is a set of internal states, Σ is an output alphabet, p_0 is the initial probability distribution over the starting states, and $p(q^b,\sigma|q^a)$ gives the joint conditional probability of transition from state q^a to state q^b generating output symbol σ. In case of user session logs, Σ is naturally thought of as a set of shell commands or processes a user launches. Q can be associated with Σ^L (Markov chains of fixed order) or with a subset of Σ^L (Markov chains with variable memory length). Let the total number of states in Q be N.

Here we will not be interested in methods of learning automata M with examples of legitimate behavior. We just assume that it has already been built during a learning phase (see [15] for a survey of such learning techniques) and describes behavior of a legitimate user.

Thus, the probabilities $p(q^b,\sigma|q^a)$ are considered to be known. For example, for Markov chains of fixed order L, these probabilities are equal to zero for states q^b whose associated strings s^b are not suffixes of string $s^a\sigma$, where s^a is an associated symbolic sequence of state q^a. In the following, lower indices of states will denote their positions in a sequence, while upper indices will be used to distinguish between different states in Q. The lower index of y has the meaning of position in the considered sequence.

To introduce a notion of the beginning (login) and the end (logoff) of a session, we assume that the automaton always starts from the special state q^α and always ends in the special state q^ω. So, $q_0=q^\alpha$ and $q_{n+1}=q^\omega$, where n is the length of a session. The fact that the initial state q^α is fixed will allow us to unambiguously recover the sequence of states given the output sequence of symbols or events (follows from the definition of DPFA).

The automaton operates as follows: at time step k an output symbol $\sigma_{k+1} \in \Sigma$ and next state q_{k+1} are chosen according to joint conditional probability distribution p. So, the probability of generating session $\boldsymbol{r}$ consisting of output symbols $r_0,r_1,\ldots,r_n$ by a walk over a sequence of states $\boldsymbol{q}=q_0,q_1,\ldots,q_n$ is $p(\boldsymbol{q},\boldsymbol{r})= \Pi_{k=0,n}\, p(q_{k+1},r_k|q_k)$.

3.2 Detecting hackers' input

Usually, it is not easy to obtain examples of intrusive behavior and labeled replay attacks in particular. So, it is rather difficult to train some classifier to distinguish between "clean" and "abused" sessions using their examples. More likely, only expert knowledge will be available about characteristics of intruder's possible actions (e.g., corruption likeliness and possible number of consequently contributed commands, see section 4).

We will think of an intruder's contribution as of corrupting noise applied to eavesdropped session $\boldsymbol{r}$ and distinguish several types of corruption the intruder is able to make. The first type is when a replay-session is formed from an original session by only changing commands, insertions and deletions of commands are not allowed. The second type further allows insertions, and the third type includes all types of corruptions – changes, insertions and deletions of commands.

Let us denote the corrupted version of session $\boldsymbol{r}=r_0,r_1,\ldots,r_n$ as $\boldsymbol{y}=y_0,y_1,\ldots,y_n$. In general, m is not equal to n, since the number of elements in the sequence may change due to corruption. Let us assume that we have an already learnt automaton M. Given $\boldsymbol{y}$, we can get $\boldsymbol{q}(\boldsymbol{y})$, the sequence of states through which M passed to generate $\boldsymbol{y}$, because M is a DPFA and we know the initial state q^α. If we are not able to accomplish this because of, e.g., the absence of some symbols in the alphabet of the legitimate user, or the absence of transitions between certain states of M, it means we have already detected an attack.

Because of the absence of *a priori* information about legitimacy of a given session, we assume replay attack is possibly present. Our idea of detecting a session containing a replay attack is to replace M with another automaton M^+ (see section 4), that will represent the user's and the intruder's contributions simultaneously. Then, we have to search for the most probable sequence of states $\boldsymbol{q}^\#$ of M^+, by a walk over which $\boldsymbol{y}$ could be obtained:

$$\boldsymbol{q}^\# = \underset{\boldsymbol{q}\in Q^{+m}}{argmax}\, p(\boldsymbol{q}|\boldsymbol{y}) = \underset{\boldsymbol{q}\in Q^{+m}}{argmax}\, \Pi_{k=0,n}\, p(q_{k+1},r_k|q_k). \quad (1)$$

Here Q^+ is the set of states of M^+. Having found $\boldsymbol{q}^\#$, we compare it with the sequence of states $\boldsymbol{q}(\boldsymbol{y})$. If $\boldsymbol{q}^\#$ and $\boldsymbol{q}(\boldsymbol{y})$ are not equal, we say that at the states where they differ an intruder had changed, inserted or deleted commands.

As an illustration, consider a series of objects (Fig. 1). An object corresponds to the possible set of states of the automaton at a particular time step and can be in any of N states. The edges between states are assigned weights equal to the probability of transition between them. So, the task of finding the most probable sequence of states becomes the task of finding a path with the largest product of weights in the obtained graph.

The algorithm of detecting a hacker's input can be summarized as follows.

First, find a sequences of states $\boldsymbol{q}(\boldsymbol{y})$ of M that lead to generation of the given log sequence $\boldsymbol{y}$. If this step fails for some reason – signal an anomaly.

Second, obtain automaton M^+ from M taking into account type of corruption and stochastic characteristics of the corruption noise. Note that M is converted into M^+ in such a way that the sequence $\boldsymbol{q}(\boldsymbol{y})$ also exists in M^+.

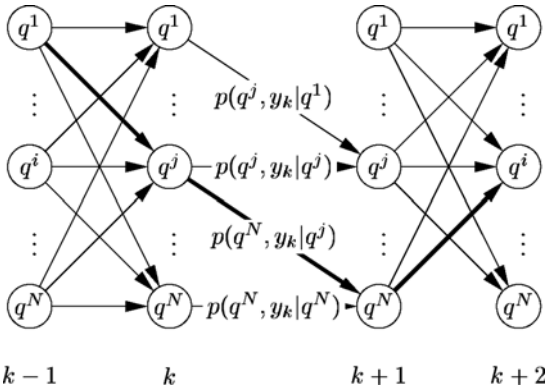

Fig. 1. A part of a graph that represents an automaton, in which the most probable sequence of states $q^{\#}$ is searched. Objects are columns of circles (states). A part of the possible most probable path is given in bold.

In the structure corresponding to M^+, find the most probable sequences of states $q^{\#}$ through which it passed to generate the considered log sequence.

Third, compare the state sequences $q(y)$ and $q^{\#}$ of M^+ to find out whether they coincide. If they don't, report their differences – possible hacker's contribution.

Evaluation of (1) can be done efficiently if we use a dynamic programming (DP) scheme (see, e.g., [17]). It may be fruitful to consider several most probable sequences of states as the sequences to compare with $q(y)$. This option is especially worth considering, if their probabilities are close to each other.

4. Handling three types of corruptions

4.1 Changes

Let us assume that we have an expert estimation of the probability ρ of changing a command. That is, for each command in a log there is the probability ρ that it will be changed by an intruder and the probability 1-ρ that it won't. For the sake of simplicity, ρ does not depend on the place in the session log where this change occurs, on the current context, and on the command being replaced. We assume y_k that will replace command r_k in a sequence is drawn by an intruder from a uniform distribution over $\Sigma\backslash\{r_k\}$.

Then, in (1) the probability $p(q_{k+1},y_k|q_k)$ becomes dependent on whether symbol y_i coincides with symbol r_k that is necessary to emit to transit to the state q_{k+1} from q_k.

$$p(q_{k+1},y_k|q_k) = p(q_{k+1},r_k|q_k)((1-\rho)\delta_{q_k y_k} + \rho(1-\delta_{q_k y_k})/(|\Sigma|-1)), \quad (2)$$

with $p(q_{k+1},r_k|q_k)$ being the original uncorrupted probability of generating r_k in state q_k and going into state q_{k+1}. We evaluate expression (1) and compare $q^{\#}$ with $q(y)$. In the places where they differ, we say that the corresponding commands were changed by an intruder.

4.2 Insertions

Besides changing commands, we will additionally allow insertions. Let us again assume that we have an expert estimation of the probability μ of inserting an intruder's command at any place in a session. We complement the set of states Q with the same quantity of special states Q*, having one-to-one correspondence with the states from Q. Thus, the whole set of the automaton's states becomes $Q \cup Q^*$.

The probability of entering any of these additional states from any $q \in Q$ with generating an output symbol $y \in \Sigma$ is set equal to $\mu/|\Sigma|$. The probability of transition to $q^i \in Q$ from some $q^{*j} \in Q^*$ with generating an output symbol $y \in \Sigma$ is set to be equal to $p(q^i,y|q^j)$ – the probability of transition from the state $q^i \in Q$ (corresponding to $q^{*i} \in Q^*$) to $q^i \in Q$ with generating the same symbol y. Transitions between additional states are prohibited. Formally,

$$p(q^{*i},y|\, q^j) = \mu/|\Sigma|, \quad P(q^i,y|\, q^{*j}) = p(q^i,y|\, q^j), \quad p(q^{*i},y|\, q^{*j}) = 0, \quad y \in \Sigma,\ q^i, q^j \in Q,\ q^{*i}, q^{*j} \in Q^* \quad (3)$$

where $p(q^i,y|q^j)$ is given by the expression (2) that takes changes into account. The rest of probabilities of transition (between states from Q^*) and the whole graph structure (except for added states) remain unchanged (Fig. 2). In Fig. 2, every object corresponds to a command in a considered log including inserted commands.

Passing over a state from Q^* means insertion of a command that appears to be generated from this state. After having searched for the most probable sequence of states using a DP scheme, we check if states from

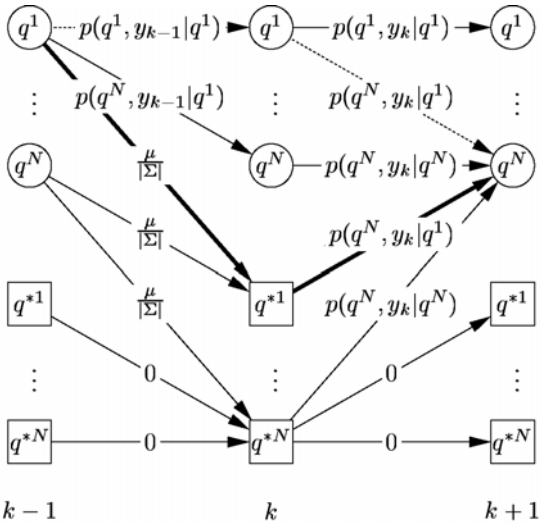

Fig. 2. A part of modified graph to handle insertions. For all i, states q^{*i} are in one-to-one correspondence to states q^i. Note that probabilities of transition from added states $q^{*i} \in Q^*$ to states from Q are equal to those between corresponding states $q^i \in Q$ and the same destination states. The differences between path $q(y)$ (dashed) and the most probable path $q^\#$ (bold) found in the graph give possible places of insertions.

Q^* appear in the obtained sequence $q^\#$. If so, symbols generated by these states could be inserted by an intruder.

Equation (3) prohibits transitions between additional states. It means that only one command at each time step can be inserted. If, however, we allow such transitions, several consequent insertions can be handled. This would require additional expert knowledge about the likeliness of neighboring inserted commands. Expressed in the form of probability, this knowledge would participate in the above expressions.

4.3 Deletions

Using deletions as additional means of performing some ill-intended activity is somewhat trickier than using just insertions and changes of commands, but nevertheless possible. Let us suppose that two adjacent deletions do not occur. We change the graph in Fig. 1 to allow connections not only to the neighboring object, but also to jump over one object. These "jumped-over" objects represent deleted components of the observed sequence and are inserted to the graph after each existing object, thus doubling the total number of objects. In Fig. 3, $\sigma(q^j)$ is the last symbol of the string that corresponds to the state q^j. That is, the automaton must emit $\sigma(q^j)$ to change from some q^i into q^j. We can still search for a path in the obtained graph with the largest product of edges' weights, so calculating expression (1). Again, by comparing $q^\#$ with $q(y)$ we can find out the deleted elements of the audit sequence. The set of states used to represent objects can be extended as in

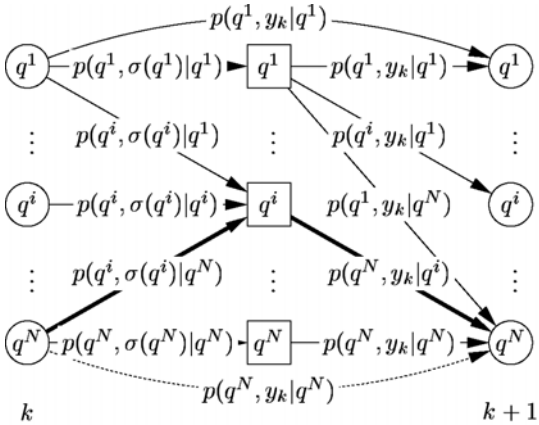

Fig. 3. A part of a graph to handle deletions. If the found most probable sequence of states $q^\#$ (bold) passes through added, "jumped-over" objects, it means that the symbols in the log sequence corresponding to those objects were possibly deleted by an intruder. Dashed line gives a part of the sequence $q(y)$ recovered from log.

case of handling insertions. Thus, we will obtain a structure handling all three types of corruption.

4.4 Illustrative example

Consider the illustrative example in Fig. 4. Let Q be a set consisting of symbols {*a*,*b*,*c*,*d*,*e*,*f*,*x*,*z*} and L=1. Then, Q^*={*a*,b*,c*,d*,e*,f*,x*,z**}. Suppose that session *y=aczdex* arrived to be tested. Then, $q(y)=\alpha aczdex\omega$. After creating a corresponding graph, depicted in Fig. 4, we search for the most probable path on it. Suppose the found most probable sequence is $q^{\#}=\alpha abcc^*def\omega$ (bold lines in Fig. 4). The most probable sequence $q^{\#}$ passes through state *c**, what means the corresponding symbol *z* in *y* was inserted by an intruder and should not be included in the resulting recovered sequence *r =abcdef.* Comparing with *y=aczdex*, we obtain that symbol *b* could be deleted, *z* – inserted, and symbol *f* was replaced with *x.*

5. Discussion

We proposed a method that can be applied to handling replay attacks in which a DPFA-based model is used to model legitimate active entity's behavior in a computer system. It allows one to specify the intruder's contributions to legitimate sessions. All three types of corruption (changes, insertions, deletions) can be handled simultaneously, as the changes their handling introduces to the underlying automaton are independent of each other.

It is feasible that an intruder will not insert her commands at the beginning of a session and would try to hide them somewhere in the middle of it, that some command context or type of change will be more advantageous for her. If so, the probabilities ρ and μ may be assumed dependent on the place in an audit sequence, context, commands being replaced or inserted, etc. The search procedures for the most probable sequence of states will not change. An adequate model of adversary that carries replay attacks out (e.g., which type of distortions the attacker uses, feasibility of deletions, whether to allow several consequent corruptions, mixtures of different types of corruptions, and so on) would facilitate implementation of the presented method and increase its performance.

In the proposed approach, an anomaly is signaled if at least one difference between state sequences occurs, but it may be worth considering an option of signaling it after encountering several differences (or differences of particular kind). A definite alarm policy should be determined after taking into account specific knowledge about users, their behavior, system peculiarities, assumed hacker's capabilities, etc.

The difficulty of obtaining an adequate expert estimation of the probabilities ρ and μ, and their dependencies (e.g., on the context) may limit the method's performance and accuracy. If we have data containing replay attacks, the option of estimating these probabilities from those data has to be considered.

An inaccurate estimation of probabilities and model (automaton) M used can lead to higher false positive and/or false negative rates. In case of sudden change of user's behavior (change of tasks, misprints, etc.),

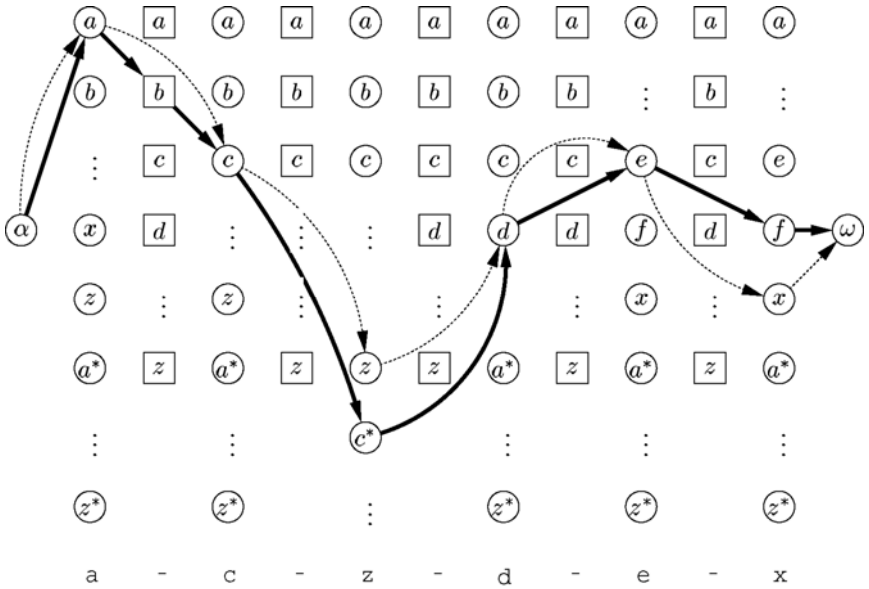

Fig. 4 An illustrative example to detecting different types of corruption. Dashed lines mark the recovered sequence of states $q(y)=\alpha aczdex\omega$ corresponding to the sequence *y=aczdex.* The found sequence most probable sequence $q^{\#}=\alpha abcc^*def\omega$ is given in bold. Circles correspond to the states from $Q\cup Q^*$. Square nodes denote states in objects that were added to handle deletions. We compare $q(y)$ and $q^{\#}$ to find differences – *b* could be deleted, *z* – inserted, and symbol *f* was replaced with *x.*

this method may also signal replay attacks. But the fact that the method reports the essence of alleged anomaly may help a security officer to make an appropriate decision.

Classification and recognition systems often provide a confidence value along with the result of classification or recognition. In the task of replay attack detection, the usefulness of an analogous empirical coefficient can be considered. As a variant, a ratio R between $P(q^{\#})$ in M^{+} and $P(q(y))$ in M, or between $P(q^{\#})$ in M^{+} and $P(q(y))$ in M^{+}, can be proposed. The obtained value of R can be brought to an expert's attention as an auxiliary indicator of an anomaly in the session, or can be used as an automatic threshold filter of anomalous sessions.

Further research and experiments with real data are required to estimate how the proposed approach and its modifications will improve handling of replay attacks.

Bibliography

[1] A. Somayaji. Automated response using system-call delays. In USENIX Security Symposium 2000, pages 185–197, 2000.

[2] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, pages 120–128. IEEE Computer Society Press, 1996.

[3] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security, 2(3): 295–331, 1999.

[4] D. E. Denning. An intrusion-detection model. In Proc. IEEE Symposium on Security and Privacy, pages 118–131, 1986.

[5] J. Ryan, M.-J. Lin, and R. Miikkulainen. Intrusion detection with neural networks. Advances in Neural Information Processing Systems, pages 254–272, 1998.

[6] D. Endler. Intrusion detection: Applying machine learning to solaris audit data. In Proc. Annual Computer Security Applications Conference (ACSAC'98), pages 268–279, Los Alamitos, CA, December 1998. IEEE Computer Society Press. Scottsdale, AZ.

[7] A. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In Proceedings 1-st USENIX Workshop on Intrusion Detection and Network Monitoring, pages 51–62, Santa Clara, California, April 1999.

[8] D. Wagner and R. Dean. Intrusion detection via static analysis. In Francis M. Titsworth, editor, Proceedings of the 2001 IEEE Symposium on Security and Privacy, pages 156–169, Los Alamitos, CA, May 14–16 2001. IEEE Computer Society.

[9] C. C. Michael and A. Ghosh. Two state-based approaches to program-based anomaly detection. ACM Transactions on Information and System Security, 5(2), 2002.

[10] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In Predicting the Future: AI Approaches to Time-Series Problems, pages 5–12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07.

[11] N. Ye. A markov chain model of temporal behavior for anomaly detection. In Proceedings of the 2000 IEEE Systems, Man and Cybernetics, Information Assurance and Security Workshop, pages 171–174, 2000.

[12] T. Lane. Hidden markov models for human/computer interface modeling. In IJCAI-99 Workshop on Learning About Users, pages 35–44, 1999.

[13] E. M. Kussul and D. A. Rachkovskij. Multilevel assembly neural architecture and processing of sequences. In A. V. Holden and V.I. Kryukov, editors, Neurocomputers and Attention: Vol. II. Connectionism and Neurocomputers, pages 577–590. Manchester University Press, 1991.

[14] D. A. Rachkovskij. Representation and processing of structures with binary sparse distributed codes. IEEE TKDE, 13(2): 261–276, 2001.

[15] K. P. Murphy. Passively learning finite automata. Technical Report 96-04-017, Santa Fe Institute, 1995.

[16] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. Machine Learning, 25(2-3): 117–149, 1996.

[17] M. I. Schlesinger and V. Hlavác. Ten lectures on statistical and structural pattern recognition. Kluwer Academic Publishers, Dordrecht/Boston/London, 2002.

Author information

Artem M. Sokolov, Dmitri A. Rachkovskij – International Research and Training Center of Information Technologies and Systems; Pr. Acad. Glushkova, 40, Kiev, 03680, Ukraine; e-mails: sokolov@ukr.net, dar@infrm.kiev.ua