An Efficient Method to Intrusion Detection

Yacine Bouzida and Sylvain Gombault
Email :{Yacine.Bouzida | Sylvain.Gombault}@enst-bretagne.fr
Département RSM
GET/ ENST Bretagne
2 rue de la Châtaigneraie
CS 17607
35576 CESSON SEVIGNE CEDEX

ABSTRACT. This paper presents a new method in intrusion detection based on analyzing the audit trails of users' activities in a local area network. This approach consists of detecting the presence of known attacks in servers' audit sessions. Each attack scenario is described by a column vector containing the different occurrences of the system events that represent the attack. The detection procedure consists of examining the manifestation of the attack scenarios in the system event trace. This method could be applied to attacks on servers. The most advantages of the presented method are (1) it is easy to implement in any network having the audit mechanism, (2) it is very fast and may be used in real time and (3) it is robust.

KEYWORDS Intrusion Detection, Audit Trail, Anomaly Intrusion Detection, Misuse Intrusion Detection.

1 Introduction

Computers and network computers of today's organizations and academia are increasingly becoming the targets of computer crimes which can result in loss of productivity, competitive advantage and theft of corporate business information or of research data.

The security infrastructure provides several security services such as privacy, integrity, availability and authentication. There are many kinds of computer security applications such as: access control, system restoration, intrusion detection… The first ensures that the user is the one he claims to be and that he is a registered person. It mainly deals with cryptography and access control. The second consists of restoring the system to a working state after damages (physical or logical) have occurred. Both fields have been largely studied and have machine implementation on almost all operating systems.

Research on intrusion detection started in early 1980's [1-2]. There have been many intrusion detection systems developed since [3-4]. Any security policy violation is a potential intrusion objective [5]. Detecting intrusions can be divided into two categories : *anomaly intrusion detection* and *misuse intrusion detection.*

The former refers to intrusions that can be detected via anomalous behavior and use of computer resources. For example, if user X only uses the computer from her/his office between 9 AM and 5 PM, an activity on her/his account late in the night is therefore suspicious and might be an intrusion. Anomaly detection attempts to quantify the usual and acceptable behavior and flags other irregular behavior as potentially intrusive.

In contrast, misuse intrusion detection refers to intrusions that follow well defined attack patterns that exploit weaknesses in system and application software. Such patterns can be precisely written in advance. Therefore, from this prior knowledge about bad or unacceptable behavior, this technique seeks to detect it directly, as opposed to anomaly intrusion detection, which seeks to detect the complement of normal behavior.

Our work lies in misuse detection approach. In particular, it consists of counting the number of events' occurrences in the audit logs. Each attack scenario is described by the number of occurrences of

auditable events that constitute this attack. The proposed algorithm works as follows; Audit the client's activity during her/his connection, if the number of events audited during a client session is greater than that constituting an attack scenario, then this attack is considered as present. Hence, the execution time of the matching algorithm is not considerable when it is compared with other misuse detection systems (see for instance, section 2).

This paper is organized as follows. Section two presents the major approaches of intrusion detection. In the third section, we present our formalization of the problem. Section four presents some results of our model. Section five introduces a comparative study with related work in misuse detection, and finally, section 6 concludes the paper and presents some perspectives.

2 Main approaches to misuse detection

As presented in the previous section, misuse detection consists of collecting and storing known attacks in an attack database. This attack base will be used by the IDS to find patterns that correspond to the description of an attack stored in the attack base.

Many mechanisms have been proposed during the past decade to represent an attack scenario; rules in expert systems such as P-BEST [6], transitions in transition based IDS such as IDIOT [7] which used CPN (Colored Petri Nets) and state transition diagrams in USTAT [8], and simple signature that are used recently by the current commercial and open source IDS such as SNORT [9]. Other Mechanisms are also proposed and are using regular expressions in [10], linear time temporal logic in LogWeaver [11] and the declarative signature specification language (see for instance Sutekh [12]).

The above approaches take into account temporal constraints between events that should be respected when writing signatures. On one hand, these constraints refine the description of attacks, on the other hand, they increase the complexity of the detection algorithm.

Most recent mechanisms are based on finding events that are left by known attacks in audit files. These events describe attack scenarios and are stocked in the attack base to be used later during the detection step. However, a bad description of attacks with these events leads to high rates of false positives if this description is not precise enough and high false negatives if there are lot of attacks that are not taken into account and not stored in the attack base.

3 Presentation of our model

In our system, each attack scenario is described by the number of occurrences of system events that constitute this attack. For example, a penetration scenario, that can be used to illegally acquire root privileges for 4.2 BSD UNIX, is described as follows:

```
%cp  /bin/sh     /usr/spool/mail/root
%chmod  4755   /usr/spool/mail/root
%touch    x
%mail root <x
% /usr/spool/mail/root
root #
```

These commands are then translated into audit events which describe the attack.

The different parameters of our formalization are as follows:

- Let *Ne* be the number of auditable system events and *Ns* the number of potential known attack scenarios.

- Let *AES* be an *Ne x Ns* Attack Events Scenario sparse matrix which gives the set of events generated by each attack scenario. AES_{ij} is the number of auditable events of type *i* generated by the attack scenario *j* ($AES_{ij} \geq 0$). (see fig.1 for an example of such a matrix). AES^i is the i^{th} column vector of *AES* that represents the i^{th} attack scenario.

- Let *Ob* be an *Ne-dimensional* vector where Ob_i counts the number of events of type *i* that are present in the audit trail (*Ob* is called "*observed audit vector*").

Let *I* be an *Ns-dimensional* positive integer vector, where I_i is the number of occurrences of the i^{th} attack scenario (*I* describes all the attacks that are present in the audited file).
To capture the manifestation of one or more attacks contained in the audit trail (*i.e. Ob*), we have to find the *I* vector which maximizes the sum $\sum_{i=1}^{Ns} I_i$ (it is the pessimistic approach: find *I* that maximizes the risk); subject to the constraints $I_i x AES^i \leq Ob;\ (1 \leq i \leq Ns)$ (see eq.1).

$$\begin{cases} Max(\sum_{i=1}^{Ns} I_i) \\ (I_i \times AES^i \leq Ob)_{i=1,..,Ns} \end{cases} \qquad (1)$$

Where AES^i is the i^{th} column of *AES*, which represents the i^{th} attack scenario of the sparse matrix *AES*.
I is an *Ns-dimensional* positive integer vector.
It is clear that the system *(1)* is a *polynomial* problem (not *NP-Complete*) and its resolution is very simple:

$$I_i = \min_{j=1,..,Ne} \left\lfloor \frac{Ob_j}{AES_{ji}} \right\rfloor,\ i=1,..,Ns \ \ where\ \ AES_{ji} \neq 0 \qquad (2)$$

Here is a simple example of our model:

Example :

$$let\ \ AES = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 3 & 6 & 2 & 0 \\ 1 & 2 & 4 & 0 \\ 2 & 1 & 0 & 8 \\ 2 & 0 & 0 & 0 \end{pmatrix} \ and\ \ Ob = \begin{pmatrix} 10 \\ 8 \\ 5 \\ 9 \\ 4 \end{pmatrix}$$

The solution to this problem is

$$I_{max} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The different steps to find are described in eq.2. For our example :

$I_1 = \min(Ob_1 \div AES_{11}, Ob_2 \div AES_{21}, Ob_3 \div AES_{31}, Ob_4 \div AES_{41}, Ob_5 \div AES_{51})$
$= \min(10 \div 5, 8 \div 3, 5 \div 1, 9 \div 2, 4 \div 2) = 2$

$I_2 = \min(Ob_2 \div AES_{22}, Ob_3 \div AES_{32}, Ob_4 \div AES_{42})\ \ \{AES_{12} = AES_{52} = 0\}$
$= \min(8 \div 6, 5 \div 2, 9 \div 1) = 1$

and so on, we finally find $I_{\max} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

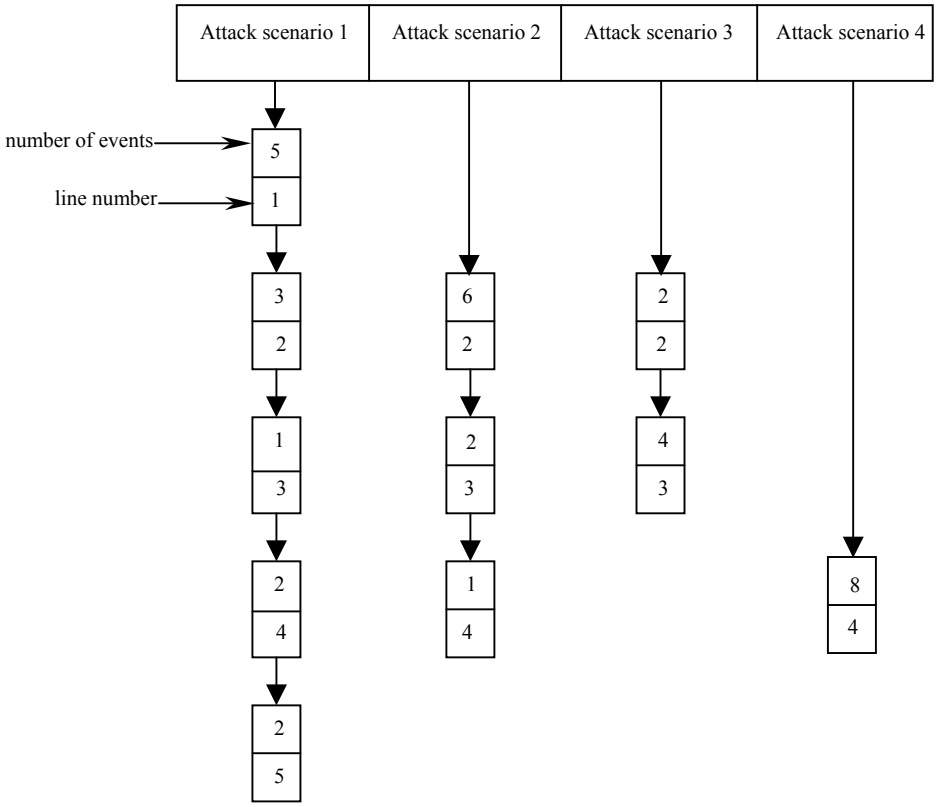

Figure. 1.: Attack Events Scenario sparse matrix *AES* Corresponding to the example above.

This approach is similar to those based on thresholds but in our case we can specify the number of many events occurrences. The signatures are not precise but the detection algorithm used is very efficient, not NP-Complete (polynomial) and may be used in real time.

4 Results with our model

To perform realistic experiments, we have used the same kinds of users and the same attack-events matrix (see table.1) as those experimented in [13-14].

An example of an attack type [13-14], which allows the attacker to print any file, is given by the following commands :

```
>touch f
>lpr –s f
>rm f
>ln –s/etc/security/passwd f
```

These commands are then translated into audit events which appear in the *AES* matrix.

	Attack Scenario																							
	a_2	l_1	l_2	l_3	l_4	a_3	a_4	a_5	l_5	l_6	l_7	l_8	a_6	a_7	a_8	a_9	l_9	l_{10}	l_{11}	a_{10}	l_{12}	l_{13}	l_{14}	l_{15}
user_login fail	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
user log (23h to 6h)	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
short_Session	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
use_SU OK	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
user_SU fail	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
who,w,finger,…	.	3	.	.	.	.	.	.	8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
more,pg,cat,…		.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	5	.	.
ls OK	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ls fail	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
df,hostname,uname	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.
arp,netstat,ping	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.
ypcat	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.
lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	1	.	.	.	.	.	.	.	.
rm, mv	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
whoami, id	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4
rexec,rlogin,rsh	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
proc_Execute	.	3	.	.	.	35	5	.	8	3	2	3	.	.	10	3	.	300	.	2	.	5	.	4
proc_setpetri	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.
file_open fail	.	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
file_open fail cp	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	.	.	.	.	.	.	.
file_open .netrc	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
file_read lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	..	.	.	.	.	.
file_read passwd...	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	.	.
file_write passwd,… fail	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
file_write cp Ok	.	.	.	.	.	.	.	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.
file_Unlink rm	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	50	.	.	.	.	.	.	.
file_mode	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.

Table. 1. : The attack-Events matrix used to validate the experiments (from [13]).

For example, attack a_2 corresponds to a guessing attack password "*there are more than three failed login attempts during a session",* and attack a_9 permits any user print any file on which he has no privileges [13]. All other attacks are defined in [13].

However, we have implemented a sparse matrix to represent this attack events matrix and tested our model on some simulated users' behavior after introducing some known attacks in the audited user's behavior. The tool used in [13-14] is called *GASSATA* "*Genetic Algorithms for a Simplified Security Audit Trail Analysis*" and its formalization is as follows:

$$\begin{cases} \begin{pmatrix} 1 & i & Na \\ & R_i & \end{pmatrix} \begin{pmatrix} \\ H_i \\ \\ \end{pmatrix} = maximum \\ \begin{matrix} 1 \\ j \\ Ne \end{matrix} \begin{pmatrix} 1 & i & Na \\ & AE_{ij} & \\ & & \end{pmatrix} \begin{pmatrix} 1 \\ H_i \\ Na \end{pmatrix} \begin{matrix} 1 & 1 \\ i & \le & j \\ Na & Ne \end{matrix} \begin{pmatrix} \\ O_i \\ \\ \end{pmatrix} \end{cases} \quad (3)$$

Where *Na* is the number of the known attack types, *Ne* is the number of auditable events, *H* is an *Na-dimensional* hypothesis binary vector, where $H_i=1$ if the attack *i* is present according to the hypothesis and $H_i=0$ otherwise (*H* describes a particular attack subset), *R* is an *Na-dimensional* weight vector, where R_i $(R_i>0)$ is the weight associated with attack *i* (R_i is proportional to the risk inherent in the attack scenario *i*). However, the same weight for all attacks (i.e. $R_i=1$, *for i=1,..,Na*) is used, *AE* is an *Ne* x *Na* Attacks-Events matrix (eq.3) which gives the set of events generated by each attack. AE_{ij} is the number of audit events of type *i* generated by the attack scenario *j* $(AE_{ij} \geq 0)$ and *O* has the same role as *Ob* in our model which represents the number of occurrences of each event in the audited session. (For more details see [13-14]).

However, the problem presented in eq.3 is *NP-Complete* and [13] used an heuristic method based on genetic algorithms [15-16] to solve this problem.

Some shortcomings of this method*, w*hich do not appear in our model, are:

- by using a binary coding for the individuals, it cannot detect the multiple realization of a particular attack,

- if the same event or group of events occur(s) in several attack scenarios, a **malicious** intruder realizing these attacks simultaneously without duplicating this event or group of events, it fails to find the actual attacks,

- by using Genetic Algorithms, if there is more than one optimum solution, it provides randomly one of them, and

- if an intruder knows the period of a session, he can perform an attack during two or more different sessions, it will also fail to detect this attack. ([13] proposed to execute *GASSATA* whenever possible (every 18 seconds for example) by considering the whole audit trail from the beginning of the user's session).

To illustrate the first three drawbacks, let us apply the algorithm to the above example, the optimum *H* vector will be one of the following:

$$H\max=\begin{pmatrix}1\\0\\1\\0\end{pmatrix}, H\max=\begin{pmatrix}0\\1\\0\\1\end{pmatrix}, \text{ or } H\max=\begin{pmatrix}0\\0\\1\\1\end{pmatrix}.$$

In addition to these three drawbacks, if the number of attack scenarios is great enough, the execution time using Genetic Algorithms will be considerable and the solution given to the fourth disadvantage will not be realistic. The following figure shows the execution time in seconds versus number of attacks in the Attack-Events matrix: **a)** when using an heuristic method (**from [13]**). **b)** when using our model.

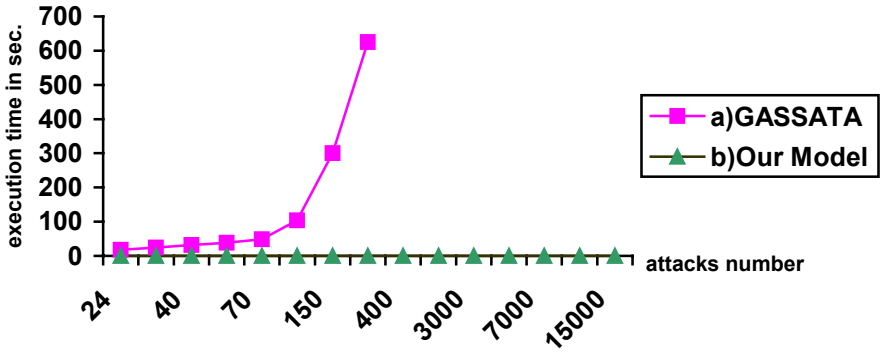

Figure. 2.: Execution time with GASSATA and our new model.

This comparison is not performed on a real network with real users. It is only a simulation of some users' behavior after introducing some known attacks in the audited user's behavior. The experimentation showed that our model finds the solutions in real time, presents less false negatives and finds exactly all the attacks that are added in the audit trails.

5 Conclusion

We have presented a new method in intrusion detection which consists of analyzing the audit trail by detecting the existence of well known attacks with a simple comparison of the occurrences number of auditable events that constitute these attacks. Our simulation results are more interesting than those obtained when using an heuristic method. We are currently implementing this new method in a real local area network with real conditions and real users.

Unlike the different models, cited in section 2, that take into account the order and temporal constraints of the different audited events, our model does not take into account these constraints. Of course, this does not refine the description of attack scenarios but the detection algorithm is efficient and fast enough to be used in real time and might be used to detect attacks combined by more than one user. On the other hand, we are seeking for the possibility to add negative conditions that might be used to verify whether an attack is not realized when some events occur and then lessen the false positive rate.

6 References

1. J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James. P. Anderson Co., Fort Washington, Pennsylvania, April 1980.
2. D. Denning: An Intrusion Detection Model, IEEE Transactions on Software Engineering, Vol. 13 (2), 1987, pp. 222,232.
3. S. Axelsson: Research in Intrusion Detection Systems : A Survey; Technical Report N°98-17, Department of Computer Engineering, Chalmers University of Technology, Gotberg Sweden, December 15th 1998, Revised on August 19th 1999.
4. H. Débar, M. Dacier, A. Wespi: Towards a taxonomy of intrusion detection systems, Computer Networks 31(8) 805-822, N .H Elsevier, 1999.
5. F. Cuppens et al. Recognizing Malicious Intention in an Intrusion Detection Process. Second International Conference on Hybrid Intelligent Systems, Santiago, Chili, December 2002.
6. Ulf Lindqvist, Phillip A. Porras. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). In Proceedings of the *1999 IEEE Symposium on Security & Privacy*, pages 146-161, Oakland, California, May 9-12, 1999. IEEE Computer Society Press.
7. S. Kumar, E. Spafford: A pattern matching model for misuse intrusion detection, Proc. *17th* National Computer security Conf. October 1994, pp. 11-21.
8. K. Ilgun: Ustat, a real time intrusion detection system for UNIX, Proc. IEEE Symp. On Research on Security and Privacy, Oakland, CA, May 1993, pp. 16-28.
9. M. Roesch: Snort Lightwight intrusion detection for networks, Proceedings of LISA '99: 13th Systems Administration Conference. Seattle, Washington, USA, November 7–12, 1999.

10. P. Uppuluri and R. Sekar: Experiences with Specification Based Intrusion Detection System. RAID 2001 (Recent Advances in Intrusion Detection). Springer-Verlag, Lecture Notes in Computer Science 2212.

11. M. Roger and J. Goubault-Larrecq. Log auditing through model checking. In Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), Cape Breton, Nova Scotia, Canada, June 2001, pages 220-236. IEEE Comp. Soc. Press, 2001.
12. J.-P. Pouzol and M.Ducassé: From declarative signatures to misuse IDS. RAID 2001 (Recent Advances in Intrusion Detection). Springer-Verlag, Lecture Notes in Computer Science 2212.
13. L. Mé: Audit de Sécurité par Algorithmes Génétique, University of Rennes 1, PhD Thesis, Order N° 1069, July 7th 1994.
14. L. Mé: Gassata, a genetic algorithm as an alternative tool for security audit trail analysis, presented at RAID, the first international workshop on Recent Advances in Intrusion Detection, October 1998.
15. D. E. Goldberg: Genetic Algorithms in Research, Optimization and Machine Learning, Addison Wesley USA, 15767, 1991.
16. M. Gen, R. W. Cheng: Genetic Algorithms and engineering design, John Wiley and Sons, Inc., 1997.