Multistep Attacks Extraction Using Compiler Techniques[1]

Safaa O. Al-Mamory, ZHANG Hongli
School of Computer Science,
Harbin Institute of technology,
Harbin, China
safaa_vb@yahoo.com, zhl@pact518.hit.edu.cn

Abstract

The Intrusion detection system (IDS) is a security technology that attempts to identify network intrusions. Defending against multistep intrusions which prepare for each other is a challenging task. In this paper, alerts classified into predefined classes. Then, the Context-Free Grammar (CFG) was used to describe the multistep attacks using alerts classes. Based on the CFGs, the modified LR parser was recruited to generate the parse trees of the scenarios presented in the alerts. The experiments were performed on two different sets of network traffic traces, using different open-source and commercial IDSs. The detected scenarios are represented by Correlation Graphs (CGs). The experimental results show that the CFG can describe multistep attacks explicitly and the modified LR parser, based on the CFG, can construct scenarios successfully.

1. Introduction

The study of IDS has become an important aspect of network security. When the IDS detects a set of attacks, it will generate many alerts referring to security breaches. Unfortunately, the IDS can not deduce anything from these separated attacks. As a result, alert correlation is an important solution to link separated attacks, to give alerts another meaning, and to infer attack scenarios.

Alert correlation function is to find out the logical relationships among the alerts. Attackers are likely to launch a series of attacks against their targets. Intelligent hackers are more likely to disguise their real purpose by launching many other minor attacks. Alert correlation is used to correlate alerts based on logical relationships among the alerts. This function will provide the security analysts with a great insight into where the initial attacks came from and where they actually end up.

An interesting method is the work of Ning *et al.* [1]. Ning *et al.* were a proposed alert correlation model based on the observation that most intrusions consist of many stages, with the early stages *preparing for* the later ones. They were collected alerts from IDS, correlated off-line, and tried to draw a big picture (through *CG*s) of what happens in the network. However, this method has overlooked, or unable to resolve, the following shortcomings. First, the graph explosion problem that occurs in the generated *CG*s makes the resulted graphs complex and difficult to understand. Second, huge number of rules used to draw these graphs representing alert's prerequisites and consequences. And finally, the affects of the missed attacks by IDS yield separated *CG*s.

The primary focus of this paper is on addressing these disadvantages. We used the compiler techniques [2] to recognize the scenarios in the raw alerts. Put another way, after categorizing alerts into classes (i.e. classification); the scenarios are represented by *context-free grammars*. These grammars are used to build LR tables. The LR tables and the raw alerts are fed to LR parser trying to construct attack scenarios; the standard LR parser algorithm is modified to meet our requirements. The resulted scenarios visually represented by correlation graphs. Instead of repairing a broken scenario afterwards, our method can tolerate missed attacks at the same time of correlation.

The remainder of this paper is organized as follows. Section 2 presents related works. Section 3 states basic concepts and assumptions. Section 4 proposes our correlation engine. Section 5 gives the empirical results. The discussion presents in section 6 and section 7 concludes the paper.

2. Related Works

Many alert aggregation and correlation approaches have recently been proposed with the goal of reducing the false alerts rates of the IDSs and building attack scenarios to recognize attack plans. Dain *et al*. [3] used

data mining approach to combine the alerts into scenarios in real time. Qin *et al*. [4] presented an alert correlation system combining a Bayesian correlation with a statistical correlation. The probabilistic alert correlation [5] based on the similarities between alerts to correlate them. Several researchers have studied the attack graph generation automatically [6]. The attack graphs represent the possible paths the attackers may take in intrusions.

With respect to the related works, in this paper, a novel correlation method is proposed which uses parsing techniques to extract attack scenarios from alerts. To our best knowledge, the using of LR parser techniques has not been addressed in the alert correlation problem.

3. Preliminaries

This section reviews relevant concepts and states our assumptions. Section 3.1 introduces classification and section 3.2 presents our model to represent scenarios. Section 3.3 discusses addresses similarity.

3.1. Alerts Classification

The alert classification scheme is designed to categorize alerts into groups that most effectively indicate their stages in multistage attacks. Each class has its name indicating the general category; the set of classes will be denoted as *CLS*. The *CLS* set contains the following classes: enumeration (*EN*), host probe (*HP*), service probe (*SP*), service compromise (*SC*), user access (*UA*), root or administrator access (*RAA*), DoS (*DS*), system compromise (*SYC*), sensitive data gathering (*SDG*), active communication remote (*ACR*), Trojan activity (*TA*), and DDoS (*DDS*). Many types of Snort IDS [7] alerts and RealSecure IDS [8] alerts have manually classified and their descriptions were taken from [9], [10] respectively.

3.2. Scenarios Representation

Many attacks languages proposed to represent the attacks and to facilitate correlation process. In this paper, we have modeled the attacks using *CFG*.

Definition 1. A *context-free grammar CFG* is a quadruple such that $CFG = <N, T, P, S>$, where N is a finite set of nonterminals which belong to *CLS* and they are usually start with capital letter, T is a set of terminals representing the alerts and the terminals usually start with small letter, P is a finite set of productions, and S is the start symbol of *CFG* which represents the scenario name; here unusually $S \notin N$.

Definition 1 formally describes the grammar from our point of view. An element in $V = N \cup T$ is called *grammar symbo*l. The productions in P are pairs of the form $X \rightarrow \alpha$, where $X \in CLS$ and $\alpha \in V^*$. Put another way, the left hand side symbol (LHSS) X is a nonterminal, and the right hand side symbol (RHSS) α is a string of grammar symbols. An empty right hand side symbol (empty string) is denoted as an ε.

Example 1. Consider a scenario in which the attacker probe the hosts to see which host is active, probe the services, compromise a service, install a Trojan and launch it, and finally use this Trojan to launch a DDos attack. This scenario can be noted in DARPA dataset [11]. It can simply be represented, assuming RealSecure [8] alerts, in our model as follows:

Scenario	→	HP SP SC ACR TA DDS
HP	→	icmp_echo_request HP | ε
SP	→	sadmind_ping SP | ε
SC	→	admind SC
		| sadmind_amslverify_overflow SC | ε
ACR	→	rsh ACR | ε
TA	→	mstream_zombie TA | ε
DDS	→	stream_dos

Some points can be noted from this grammar. First, the order of the RHSS in the first rule specifies the time order of the attacks. Second, we put the ε (i.e. the empty string) in each rule to pass the missed attacks by IDS. In addition, the passing of missed attacks leads to generate a single *CG* instead of generating separated *CGs*. Third, the attacker may try the same attack (or other attacks from the same class) until he succeeds. As a consequence, we put the LHSS after each terminal to allow loops. Fourth, the DDS (the last rule) presented without a loop and the ε rule assuming it is neither can be missed nor duplicated (RealSecure triggers one alert for this attack). Finally, the frequently use of the ε makes this scenario capable of accepting only the last attack. For this reason, a combination of the first rule could solve this problem.

3.3. Addresses Similarity

An alert A can be characterized by a set of features. Assume that $A.f_i$ represents the i*th* feature of an alert A. These features are: alert type (or type for short), source IP, Destination IP, and time stamp. The table of alerts can be denoted as T. We consider only IPv4 addresses.

Definition 2. The addresses similarity between any two alerts can be computed as follows $IP\_Sim(A_i,A_j)=Sim\_Src(A_i,A_j)+Sim\_Dst(A_i,A_j)+Sim(A_i.DstIP,A_j.SrcIP)$, where: (1) $Sim\_Src(A_i,A_j)$ and $Sim\_Dst(A_i,A_j)$ are the source IP similarity and the target IP similarity respectively, these measures

compute the common similar bits of two IP addresses from the left then the result is divided by 32; (2) *Sim(A_i.DstIP,A_j.SrcIP)* checks if A_i.*DstIP* equal to A_j.*SrcIP*, this feature is necessary because sometimes the attacker use one victim as a step stone to compromise other victims; (3) A_i.time ≤ A_j.time.

4. Scenario Construction

The proposed system composes of two phases. The first phase uses the modified LR parser to build scenarios, whereas the second phase tries to connect the related scenarios which may be separated.

Like the lexical analyzer, the IDS triggers many separated attacks neglecting the relationships among them. Moreover, the problem of scenario discovery seems to be like the problem of the parser which tries to find the correct sequence of tokens and to which grammar each statement belongs.

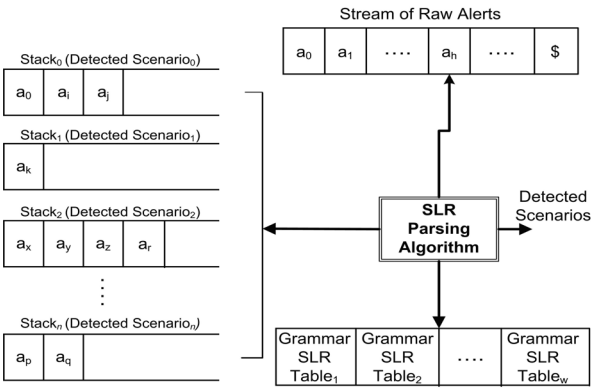

Figure 1. The modified SLR parsing algorithm

The LR parser has stringent style, so we performed some modifications to make it flexible. These modifications are as follows. First, we have eliminated the error action because it is undesirable in scenario recognition process. Second, many SLR tables are used, instead of using one, to represents the scenarios. Finally, we used many stacks which are created dynamically depending on the alerts, and this can be shown in Fig. 1.

Each scenario can be represented by a grammar or sometimes the grammar can contains some scenarios. The SLR table should be built for each grammar, it represents the scenarios templates. We denoted it as *TPLT* and the set of scenarios templates as S_{TPLT}.

Definition 3. Given S_{TPLT}, the LR parser represents each resulted scenario by a stack. The stack *STK* is a sextuple = <*V, State, References, SrsIP, DstIP, StepStone*>, where *State* represents the current state of a finite machine, *References* is an array of alerts indices, and *StepStone* is the intermediate victims. The last three elements are the properties of the *STK*. The set of the *STK*s is denoted as S_{STK}.

When a new alert becomes available, it is first checked against the existing S_{STK} using *IP_Sim* function and the *State* on the top of each *STK*. It can be added to more than one *STK* if the above conditions have frequently satisfied. We called this checking as *STK_Selection*. If the *STK_Selection* fails to find any *STK*, then a new *STK* will be created. This process can be shown in Fig. 2.

Procedure *Scenario_Recovery*
input: alerts in time order
output: a set of scenarios
method:
  for each alert $A \in T$ ***do*** {
    if alert *A* related to network vulnerabilities ***then***{
      idx()=STK_Selection*(A);*
      if empty(idx()) ***then*** {
        Create a new STK_i;
        Call LR_Parsing(*A*, STK_i); }
      else
        for each $STK_j \in$ idx() ***do***
          Call LR_Parsing(*A*, STK_j); } }
  Return set of *STK*s which contain accepted scenarios
      and related information to generate reports;

Procedure *LR_Parsing*
input: an alert *A* to be correlated, *STK*, S_{TPLT}
output: updated *STK*
method:
  STAT = current state of *STK*;
  Action = S_{TPLT}(STK_{ID}, *STAT*, *A*);
  switch (*Action*) {
    case "shift": Shift *A* to *STK*;
      Update state of *STK*; break;
    case "reduce": *R* = the reduce rule;
      Remove RHSS(s) of *R* from *STK*;
      Insert LHSS of *R* on top of *STK*;
      Update state of *STK*; break;
    case "accept": Direct *STK* to the second phase as
      accepted scenario;
      Remove *STK* from S_{STK}; }
  Return updated *STK* or accepted scenario;

Figure 2. The main procedures of the system

One advantage of the proposed system is the capability of working in parallel where the S_{TPLT} can be partitioned and distributed over many processing units. In other words, each processing unit will have a subset of S_{TPLT}.

The second phase tries to connect the related scenarios. Sometimes, the single scenario spread over more than one *STK,* so this phase connects them. In addition, this phase produces *CG*s which reflect the detected scenarios.

Time complexity of the proposed system related to the number of alerts and the number of scenarios in the alerts. The time for processing each new alert with *STK_Selection* function is linear with S_{STK}. It is well

known that LR parser has a linear time complexity [2]. So the total time complexity of scenario recovery is $O(n*(S_{TPLT}+S_{STK}))$ where n is the number of alerts.

Two optimizations points can be noticed: First, *TPLT* is related to the network vulnerabilities and their numbers are limited. Some *attack graph* methods can generate the scenarios, which the network vulnerable to, automatically [6]. Second, the dealing with flooding alerts makes the performance of *STK_Selection* function decreasing as more and more alerts are received. Depending on site policy, they are either neglected (such as *HP* alerts) or aggregated or both.

5. Empirical Results

The experiments which were performed to evaluate our system are described in this section. The proposed system was tested on an AMD Athelon processor 2.01 GHz with 512 RAM running Windows XP. Two set of experiments were conducted to test system effectiveness and performance, which are presented in sections 5.1 and 5.2 respectively.

5.1. Effectiveness

The objective of the first set of experiments is to demonstrate the effectiveness of the proposed algorithm in scenario recovery. The sensor alert report by RealSecure network sensor (Version 6.0), executed on the DARPA 2000 datasets, has been made available by researchers at North Carolina State University as a part of the TIAA project [12]. This sensor alert report was used in this set of experiments.

The 2000 DARPA scenario specific datasets include LLDDOS 1.0 and LLDDOS 2.0.2. LLDDOS 1.0 contains a series of attacks which was described in example 1. LLDDOS 2.0.2 includes a similar sequence of attacks run by a more sophisticated attacker. Four sets of experiments were performed, each with either the DMZ or the inside network traffic of one dataset.

We mapped each alert type, reported by RealSecure IDS, to a subattack name using [10]. These subattack names are listed in section 3.1. Hereafter, we encoded them in grammars (i.e. scenarios) which used to build SLR tables. Then, these SLR tables and the alerts are fed to LR parser. Our experiments with this datasets show results like the described one in [1], validating the correctness of our correlation algorithm.

The *CG* which is discovered from the inside zone of LLDDOS 1.0 can be seen in Fig. 3. Each node in this figure represents a grammar nonterminal, the text inside the nodes is the class of the alerts followed by their IDs. The edges represent the time order. The loops in the graph mean that either the attacker does some trials to succeed or some alerts have aggregated in this node.

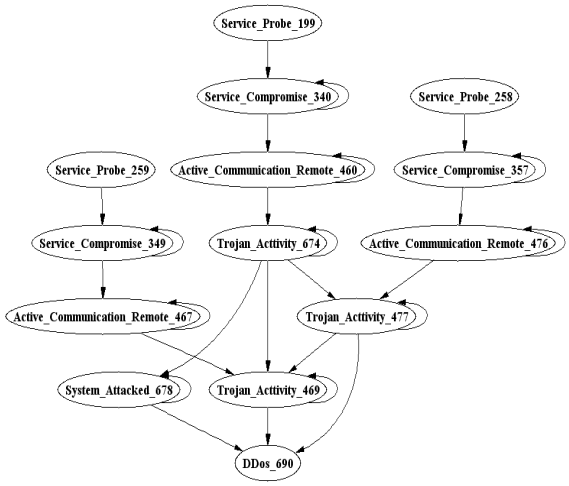

Figure 3. The detected scenarios in the inside zone of LLDDOS 1.0 dataset

The *CG* which is discovered from the inside zone of LLDDOS 1.0 can be seen in Fig. 3. Each node in this figure represents a grammar nonterminal, the text inside the nodes is the class of the alerts followed by their IDs. The edges represent the time order. The loops in the graph mean that either the attacker does some trials to succeed or some alerts have aggregated.

To test the effectiveness of our system, we used the measures of completeness and soundness defined in [1] for comparison purposes. The soundness measure (*Rs*) evaluates the rate of true alerts appearing in the *CG*. The completeness measure (*Rc*) looks for missing true alerts from *CG*. Equation (1) shows these measures. The results of the two measures are given in Table 1.

Table 1. Effectiveness of the proposed system

	LLDDOS 1.0		LLDDOS 2.0.2	
	DMZ	Inside	DMZ	Inside
# Correctly Correlated Alerts	57	44	6	16
# Related Alerts	67	51	8	20
# Correlated Alerts	57	44	6	20
# Missed Alerts By Real Secure	10	7	2	4
Completeness Measure *Rc*	85.07%	86.27%	75%	80%
Soundness Measure *Rs*	100%	100%	100%	80%

$$Rc = \frac{\#\ correctly\ correlated\ alerts}{\#\ related\ alerts},\ Rs = \frac{\#\ correctly\ correlated\ alerts}{\#\ correlated\ alerts} \quad (1)$$

The missed alerts by IDS degrade the effectiveness which was the situation in our experiment, where the missed alerts by RealSecure IDS affect the completeness measure results as shown in Table 1. In addition, the experiments were produced good results for the soundness measure.

5.2. Performance

The objective of this set of experiments is to evaluate the performance of the correlation engine and to show its ability to work in parallel. The performance metric includes the processing time of each alert. Snort IDS was used in this set of experiments. DEFCON 8 Capture The Flag (CTF) datasets [13] were chosen which contains attacks launched by competing hackers.

We applied the proposed system on the DEFCON 8 CTF dataset. Unfortunately, due to the nature of the DEFCON 8 CTF dataset, we did not have any information about the attack scenarios within it. Thus, we analyzed the resulted alerts depending on Snort signature database [9]. Hereafter, we encoded them in grammars which used to build SLR tables. About 40 grammars have been written for this dataset, where some grammars contain more than one scenario.

The DEFCON 8 CTF dataset was generated large amount of alerts. In section 4, we referred to the processing of flooding alerts which are either neglecting or aggregation. In this set of experiments, we neglected *host probe* alerts and *Dos* flooding alerts then aggregated the remaining alerts in two minutes time window. The remaining alerts (which are used in the experiments) are 33818 alerts.

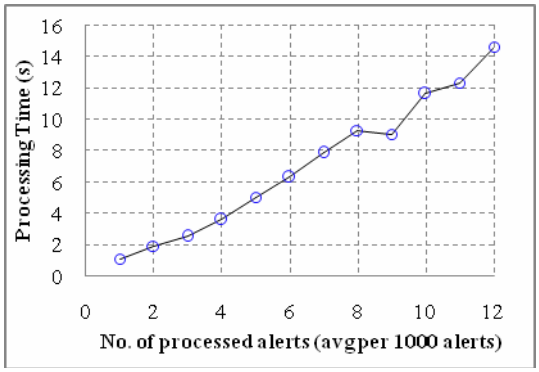

Figure 4. The processing time of our system

As can be seen in Fig. 4, the proposed system measures its own processing time per alert (averaged per 1000 alerts). Clearly, the average processing time scales with the number of received alerts and the number of scenarios in these alerts as expected. The increasing of scenarios in the alerts leads to increase the stacks number making the proposed system slows down. The DEFCON 8 CTF datasets were used here because they contain large number of scenarios and alerts. This datasets are unusual and contain huge number of attacks in a short period of time; as a consequence, our system will exhibit a better performance in real-world because attacks are usually less intensive.

Moreover, we have compared the processing time for the proposed system to the delay between receiving two consecutive alerts from Snort. The DARPA datasets were used for this purpose. Fig. 5 shows the processing time per alert (averaged per 50 alerts). Obviously, the proposed system works faster than Snort in processing the entire dataset.

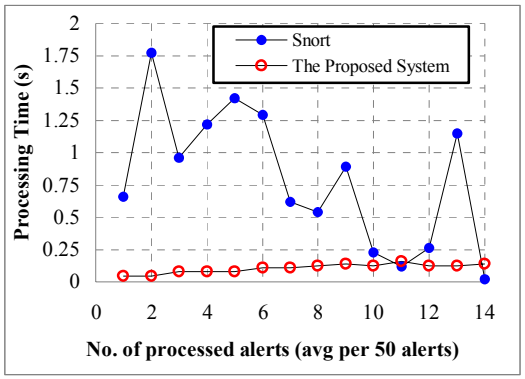

Figure 5. Processing time for DARPA dataset

We tested the capability of the proposed system to work in parallel. We used a cluster of seven nodes in our test with one node as a master; each node has the same properties noted at the beginning of section 5. We were partitioned and distributed the S_{TPLT} over the processing nodes and ran the system many times with different number of processing nodes. As shown in Fig. 6, the processing time is declined whenever the processing nodes are increased. We can conclude from these results that the proposed system has the capability to work in parallel.

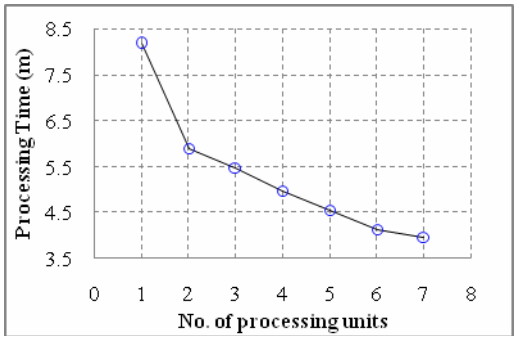

Figure 6. The relation between the number of processing units and the processing time

A natural way to correlate alerts is to search all the received alerts for those who prepare for a new alert. This nested loop procedure is assumed by many correlation methods [14]. As we have noted in section 4, the total time complexity of the proposed system

was $O(n*(S_{TPLT}+S_{STK}))$, which seems to be an attractive solution **for** the alerts correlation problem.

6. Discussion

From the literature, Ning *et al.* [1] have a proposed correlation method to extract attack strategies from intrusion alerts. The experimental results on the DARPA 2000 dataset and DEFCON 8 CTF dataset show that our graph representation results are similar to their results. However, there are some differences.

The results of section 5 and the results in [1] can now be compared. The measures average values of DARPA experiments are presented in Table 2. It also shows the number of nodes and edges in the resulted *CG* of LLDDos 1.0 inside zone. As can be seen, the results indicate that, overall, our system is enhanced the completeness measure and the size of the *CG*s.

Table 2. Comparison with the previous system

	Ning's system	The proposed system
Completeness Measure (avg.)	79.28%	81.59%
Soundness Measure (avg.)	95.06%	95%
Number of nodes	44	14
Number of edges	153	26

The proposed system provides a high-level representation of correlated alerts which reveals the causal relationships among them. The generated *CG*s (cf. Section 5) clearly show the strategies behind these attacks. One advantage of our method is the compressing of the resulted *CG*s. Such an abstracted *CG*s provide a concise view of the real attacks, which helps the security analysts to understand the results of alert correlation.

The contribution of this paper is three folds: First, the scenarios are represented by grammars which make the rules writing and updating an easy task. Second, the proposed system generates compressed and easy to understand *CG*s which reflects the attack scenarios. Finally, our method can tolerate missed attacks by IDS at the same time of correlation.

It should be noted that our approach has several limitations. First, our method depends on the underlying IDSs to provide alerts. Second, our approach is not fully effective for the coordinated attacks in which many attackers cooperate to do some goals. Finally, it is worth mentioning that the results produced by our correlation engine are only as good as the classification information provided by the user.

7. Conclusions

This paper presented a method for constructing attack scenarios using compiler techniques. After representing the multistep attacks by *CFG*s, the modified LR parser was used to detect the scenarios presented in the alerts. The *CFG* makes the scenario writing and updating an easy task, it also describes multistep attacks explicitly.

8. References

[1] P. Ning, Y. Cui, D. S. Reeves, and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Trans. Information and System Security*, 7(2), 2004, pp. 274–318.

[2] Aho A. V., R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques, and Tools*, Addison-Wesley Publishing Company, 1986.

[3] O. M. Dain and R. K Cunningham, "Fusing a heterogeneous alert stream into scenarios," *in Proc. 2001 ACM Workshop on Data Mining for Security Applications*, 2001, pp. 231–235.

[4] X. Qin and W. Lee, "Statistical causality analysis of INFOSEC alert data," *in Proc. 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, Sep 2003, pp. 591–627.

[5] A. Valdes and K. Skinner, "Probabilistic alert correlation," *in Proc. Recent Advances in Intrusion Detection*, LNCS 2212, 2001, pp. 54–68.

[6] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," *in Proc. 9th ACM Conference on Computer and Communications Security (CCS'02)*, 2002, pp. 217–224.

[7] M. Roesch, "Snort–lightweight intrusion detection for networks," *in Proc. 1999 USENIX LISA Conference*, 1999, pp. 229–238.

[8] Internet Security Systems, RealSecure intrusion detection system, http://www.iss.net.

[9] Sourcefire, Snort signature database, http://www.snor-t.org/pub-bin/sigs.cgi, 2007.

[10] Internet Security Systems, RealSecure signatures reference guide, http://documents.iss.net/literature/RealSecu re/RS_Signatures_6.0.pdf, 2001.

[11] MIT Lincoln Lab., 2000 DARPA intrusion detection scenario specific datasets, http://www.ll.mit.edu/IST/ ideval/data/2000/2000_data _index.html.

[12] P. Ning, "TIAA: a toolkit for intrusion alert analysis," http://discovery.csc.ncsu.edu/software/correlator.

[13] DEFCON captures the flag (CTF) contest, http://cctf.shmoo.com/data/cctf-defcon8/, 2000.

[14] L. Wang, A. Liu, S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts", Computer Communications, 29, 2006, pp. 2917–2933,.

[1] This work was supported by 973 project no. (2007CB311101).