J Comput Virol (2009) 5:271–282
DOI 10.1007/s11416-008-0103-3

ORIGINAL PAPER

IDS alerts correlation using grammar-based approach

Safaa O. Al-Mamory · Hongli Zhang

Received: 5 April 2008 / Revised: 15 July 2008 / Accepted: 28 July 2008 / Published online: 15 August 2008

Abstract Intrusion Detection System (IDS) is a security technology that attempts to identify intrusions. Defending against multi-step intrusions which prepare for each other is a challenging task. In this paper, we propose a novel approach to alert post-processing and correlation, the Alerts Parser. Different from most other alert correlation methods, our approach treats the alerts as tokens and uses modified version of the LR parser to generate parse trees representing the scenarii in the alerts. An Attribute Context-Free Grammar (ACF-grammar) is used for representing the multi-step attacks. Attack scenarii information and prerequisites/consequences knowledge are included together in the ACF-grammar enhancing the correlation results. The modified LR parser depends on these ACF-grammars to generate parse trees. The experiments were performed on two different sets of network traffic traces, using different open-source and commercial IDS sensors. The discovered scenarii are represented by Correlation Graphs (CGs). The experimental results show that Alerts Parser can work in parallel, effectively correlate related alerts with low false correlation rate, uncover the attack strategies, and generate concise CGs.

1 Introduction

The study of Intrusion Detection System (IDS) has become an important aspect of network security. As soon as the IDS detects a set of attacks it will generate many alerts referring to as security breaches. Unfortunately, it provides unmanageable amount of alerts. It has been estimated that up to 99% of these alerts are false positives alerts [1]. The main reasons for triggering false alerts are: runtime limitations of IDSs, specificity of detection signatures, dependency on environment [2], the difficulty to define precisely the boundary between abnormal and normal activities, and IDS isolation from the rest of the system and the network [3].

Many methods to deal with alerts are available, but main objectives of these investigations are: to reduce the amount of false alerts [2,4], to study the cause of these false positives [5], to recognize high-level attack scenarii [6–11], and finally to provide a coherent response to attacks, understanding the relationship among different alerts. In this paper, we focus on the methods that recognize high-level attack scenarii. Our method aims to construct the multi-step attacks from the raw alerts by parsing the alerts using the modified Left-to-Right (LR) Parser algorithm. The architecture of our system can be seen in Fig. 1. The distributed sensors over the protected network generates alerts and then forward them to the central alerts database. After correcting the time order of the alerts in the database, the alerts are fed into our Alerts Parser correlator to recognize the multi-step attacks. Alerts Parser generates Correlation Graphs (CGs) describing the detected scenarii and then forwards them to the security analyst to make the appropriate reaction decision.

We can compare the role of IDS to the lexical analyzers role, where the lexical analyzer issues tokens, but it does not know the relations connecting them. This is the situation in IDS behavior where it triggers many separated attacks while neglecting the relationships among them. In the same way, the problem of scenario discovery seems to be like the parser problem which tries to find the correct sequence of tokens and to which grammar each token belongs. The comparison between Alerts Parser phases and compiler phases is depicted

S. O. Al-Mamory (✉) · H. Zhang
School of Computer Science and Technology,
Harbin Institute of Technology, 150001 Harbin, China
e-mail: safaa_vb@yahoo.com

H. Zhang
e-mail: zhl@pact518.hit.edu.cn

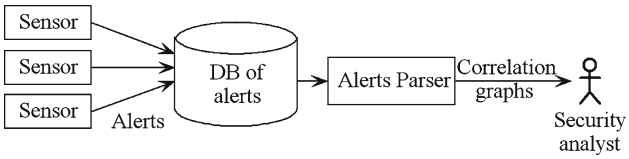

Fig. 1 Raw alerts generated by distributed sensors are stored in a central database. The alerts (after reordering) will be forwarded to Alerts Parser which produces CGs. The reaction by the security analyst would depend on the resulted CGs

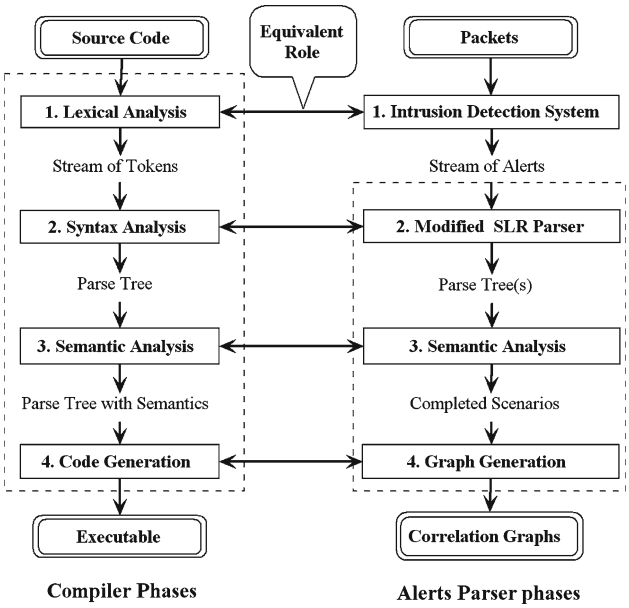

Fig. 2 Alerts Parser has three phases. Each phase has a similar phase in the compiler phases

in Fig. 2. In this paper, a novel alerts correlation method is proposed depending on LR parser. The standard LR parser algorithm is modified to meet scenario recognition requirements. The Alerts Parser compose of three phases as can be seen in Fig. 2. In the first phase, the raw alerts are fed to the modified LR parser to construct attack scenarii. The second phase tries to connect the resulted (may be separated) scenarii. Finally, the third phase converts the resulted scenarii to CGs. Instead of repairing a broken scenario afterwards [6], our method can tolerate missed attacks at the same correlation time.

The more information fed into the correlation system, the better the accuracy is obtained. Alerts can be correlated based on similar attributes [12], attack signatures [13], prerequisites/consequences knowledge [6,9], and vulnerabilities information [11]. We encode the attack signature information and prerequisites/consequences knowledge into Attribute Context-Free Grammar (ACF-grammar). The prerequisites/consequences knowledge [9] is used here as attributes (in the semantic rules) to enhance alerts correlation and to mitigate the influence of false correlations. In addition, the similarity between alerts attributes are used. The ACF-grammars are converted to LR tables. Alerts Parser would use the LR tables to construct scenarii.

We report experimental results on the DARPA 2000 DDoS evaluation datasets [14] and DEFCON 8 Capture The Flag (CTF) datasets [15]. Our experiments shows that Alerts Parser can work parallel, effectively correlate related alerts with low false correlation rate, uncover the attack strategies, and generate concise CGs.

The remainder of this paper is organized as follows. In Sect. 2, we review related works. In Sect. 3, we state how to represent the attack scenarii by ACF-grammar. In Sect. 4, we propose Alerts Parser correlator to discover multi-step attacks from raw alerts. We report experimental results in Sect. 5, discuss the contributions and the results of this paper in Sect. 6, and present conclusions and future work in Sect. 7.

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 scenarii to recognize attack plans. Dain et al. [7] used data mining approach to combine the alerts into scenarii in real time. Qin et al. [8] presented an alert correlation system combining a Bayesian correlation with a statistical correlation using a time series-based causal analysis algorithm. The probabilistic alert correlation system [12] based on the similarities between alerts to correlate the alerts. Measures have been defined to evaluate the degree of similarity between two alerts. The work of Julisch [5] outlines an effective approach to reduce false positives in sensor reports by clustering alerts, then using the clusters to discover and understand the root causes of false alerts.

The Morin et al. model (M2D2) [16] has provided concepts and relations relevant to the information system security. It relies on a formal description of sensor capabilities in terms of scope and positioning, to determine if an alert is false positive. This model can be used to facilitate event aggregation and correlation.

Several researchers have studied automated generation of attack graphs [17–19]. They used vulnerability information instead of IDS alerts to construct high-level attack graphs. The attack graph represents the possible intrusion paths.

The work of Ning et al. [9] generates CGs depending on prerequisites/consequences knowledge (pre/con for short) of individual alerts. They proposed a correlation model based on the inherent observation that most intrusions consist of many stages, with the early stages preparing for the later ones. The correlation model is built upon two aspects of intrusions that are, pre/con knowledge. With knowledge of pre/con, their model can correlate related alerts by finding causal

Table 1 Classes of alerts represent stages in the multistage attacks

Classes	Abbreviations
Enumeration	EN
Host Probe	HP
Service Probe	SP
Service Compromise	SC
User Access	UA
Root or Administrator Access	RAA
SYstem Compromise	SYC
Sensitive Data Gathering	SDG
Active Communication Remote	ACR
Trojan Activity	TA
Availability Violation	AV
Integrity Violation	IV
Confidentiality Violation	CV

relationships between them. They used hyper alert correlation graphs to represent the alerts, where the nodes represent hyper alerts and the edges represent *prepare for* relation.

With the same lane to the related works, a novel correlation method is proposed, which depends on encoding scenarii information in ACF-grammars, then uses parsing techniques to extract attack scenarii from alerts. To the best of our knowledge, the use of LR parser techniques has not been addressed in the correlation problem.

3 Attack scenarii modelling

In this section, we will state how the ACF-grammar can represent the multi-step attacks, we define the terminals and nonterminals in this ACF-grammar, and the attributes assigned with these terminals and nonterminals.

The alerts can be classified into groups that most effectively indicate their stages in multistage attacks. An alert can be part of multiple classes. Each class has its name indicating the general category; the set of classes will be denoted by CLS. Table 1 presents the classes which belong to CLS. Many types of Snort IDS [20] alerts and RealSecure IDS [21] alerts have been manually classified and their description had been taken from [22,23], respectively. These classes represent the nonterminals in the ACF-grammar whereas the alerts represent the terminals of the ACF-grammar.

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

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

Definition 2 *An Attribute context-free grammar* ACF-grammar *consists of three elements, a CF-grammar, a finite set of attributes* Att*, and a finite set of semantic rules R. The set of attributes* Att *includes the prerequisites/consequences knowledge and the attributes of the alerts. Thus* ACF-grammar = <CF-grammar, Att, R >.

A finite set of attributes *Att(X)* is associated with each symbol $X \in V$. The set *Att(X)* is partitioned into two disjoints subsets, the *inherited attributes* and the *synthesized attributes*. The synthesized attributes move the data flow *upwards* and the inherited attributes move the data flow *downwards* in the derivation tree during the attribute evaluation process. In our model, we only used the synthesized attributes.

The production $p \in P$, $p : Y_j \rightarrow X_1 \ldots Xm$ ($m \geq 1$), has an attribute occurrence $X_i.a$, if $a \in Att(X_i)$, $1 \geq i \geq m$. A finite set of semantic rules is associated with each production p. We have classified these semantic rules, in our implementation, into two types: copy rules and check rules. The copy rule, as its name, copies the attribute value from X_i to Y_j, where $X_i, Y_j \in V$. The check rule checks for some conditions to be satisfied. The prerequisites/consequences knowledge [9] is used here as attributes to enhance alerts correlation and to mitigate the influence of false correlations.

Example 1 Consider a scenario in which the attacker probes the hosts to see which one is active, probes the services, compromises a service, installs a Trojan and launches it, and finally uses this Trojan to launch a DDoS attack. This scenario can be noted in DARPA 2000 scenario based datasets [14]. The ACF-grammar representing this scenario is depicted in Fig. 3. The alert messages in this figure were generated using RealSecure IDS [21].

As shown in Fig. 3, The AV, in the last production, is presented without a loop and the ε rule, assuming that it is neither can be missed nor be duplicated (RealSecure IDS triggers one alert for this attack). Moreover, the frequently use of the ε makes this scenario capable of accepting only the last attack. For this reason, a combination of the first production's nonterminals can be used to solve this problem. All the semantic rules in this figure are copy rules except for the first production's rules which are check rules. This modeling has the capability of detecting new combinations of attacks because the focusing is on alerts classes instead of alerts themselves.

Productions	Semantic Rules
Scenario→ HP SP SC ACR TA AV	Subattack$_i$.*pre* = = Subattack$_{i+1}$.*con* Spoofed(AV.*sourceIP*) Spoofed(AV.*targetIP*)
HP→ icmp_ping HP | ε	HP.*pre* = icmp_ping.*pre* HP.*con* = icmp_ping.*con*
SP→ sadmind_ping SP | ε	SP.*pre* = sadmind_ping.*pre* SP.*con* = sadmind_ping.*con*
SC→ sadmind_amslverify_overflow SC | ε	SC.*pre* = sadmind_amslverify_overflow.*pre* SC.*con* = sadmind_amslverify_overflow.*con*
ACR→ rsh ACR | ε	ACR.*pre* = rsh.*pre* ACR.*con* = rsh.*con*
TA→ mstream_zombie TA | ε	TA.*pre* = mstream_zombie.*pre* TA.*con* = mstream_zombie.*con*
AV→ stream_dos	AV.*pre* = stream_dos.*pre* AV.*con* = stream_dos.*con*

Fig. 3 One multi-step attack from DARPA 2000 datasets [14] is represented here using ACF-grammar

The attack scenario is a set of malefactor's intentions partially ordered in time. Each malefactor's intention, either can be detected by the IDS as an alert message or it can be missed. We can use (Intention → a_1 Intention $|a_2$ Intention $|\ldots|a_n$ Intention $|\varepsilon$) as a template to represent each intention where a_i represents the alert message. Some points can be noted from this template. First, the attacker may try to do the same intention using different ways until he succeeds; in this case the IDS may generate different alerts messages. Moreover, the security analyst may use different IDSs (which can trigger different alert messages referring to the same intention) to monitor the network. Consequently, we put the LHSS after each alert message to allow the loops. Second, we put the ε in this template to pass the missed attacks by the IDS.

The attack scenarii are represented by CF-grammars making the rules writing and updating an easy task. Solving several simpler problems is often easier than solving one complicated problem. We can construct individual production for each intention as we noted above. Next, these individual productions can be easily merged into a CF-grammar for the original scenario and then adding the main production (ScenarioName → I_1 I_2 ,. . ., I_k), where the variables I_i are the start variables for the individual productions (or individual intentions). The order of intentions in the right hand side of this rule refers to their time order. Some helpful techniques for building the CFG are presented in [24].

All multi-step attacks can be represented by regular grammars. The state transition is an efficient method to represent penetrations where a penetration is viewed as a sequence of actions performed by an attacker that leads from some initial state on a system to a target compromised state [25]. The state transition (or finite automata) can be easily converted to a grammar [24,26]; in this case, all resulted grammars are regular grammars. For this reason we have used the regular grammar in our modeling to represent attack scenarii.

The resulted regular grammars are then converted to LR tables which are the basic information used by the modified LR parser to construct the attack scenarii presented in the alerts. There are many types of LR parser; i.e. LR, Simple LR (SLR for short), and Look Ahead LR (LALR) [26]. While parser generators use the most complex method (LALR), we used SLR parser because it is easy to implement. Moreover, the main difference between these parsers is the speed of detecting the errors and we have deleted the error action in the modified LR parser. The resulted SLR tables are sparse; as a result, we represented them by linked lists in order to save memory.

We selected CF-grammars to model the attacks because they offer significant advantages. First, a CF-grammar gives a precise, yet easy to understand, specification of the scenarii. Second, an efficient parser can be automatically constructed for the CF-grammars [26]. And finally, scenarii evolve over a period of time, and they can be added more easily when there is an existing implementation based on grammatical description.

The using of ambiguous CF-grammars causes conflicts in the LR tables, which should be avoided when building the parsers. The conflict occurs when some finite automata states suggest two actions at the same time. There are three types of conflicts which are shift/shift, shift/reduce, and reduce/reduce conflicts. However, the shift/shift conflict occurence is rare, while shift/reduce and reduce/reduce conflicts can be avoided by using look-ahead family of parsers like LR(1) [27].

There is no general procedure to eliminate ambiguity from CF-grammar. However, in some cases, it is still possible to rewrite the CF-grammar to produce unambiguous CF-grammar. Rewriting an unambiguous CF-grammar to eliminate conflicts is somewhat of an art. Some techniques may help to remove ambiguity from CF-grammars, like changing from left-recursive to right recursive, left factoring, etc. [28]. It should be noted that we used only regular grammars. For any ambiguous regular grammar, there exists an unambiguous regular grammar [29].

4 Alerts Parser correlator

In this section, we present Alerts Parser correlator which processes the alerts produced by various IDS sensors, to produce CGs. The Alerts Parser core depends on one of the shift-reduce parser techniques, and Sect. 4.1 describes the shift-reduce parser techniques in details. Due to the reason that Alerts Parser receives alerts from different IDS sensors (i.e. Snort and RealSecure), the temporal characteristic of alerts are typically imprecise. Consequently, Sect. 4.2 discusses the alerts time constraint. To study the relation among the attacks, parsed by Alerts Parser Sect. 4.3 defines addresses similarity. Finally, the modified LR parser algorithm used by Alerts Parser is presented in Sect. 4.4.

4.1 Shift-reduce parser

Parsing is the process of determining if terminals string can be generated by a CF-grammar. The parser is capable of constructing the parse trees. Most parsing methods fall into one of two classes, called the *top-down* and *bottom-up* methods. Bottom-up parsing methods can handle a larger class of grammars than top-down methods [26]. For this reason we selected the bottom-up parsing methods, which is the shift-reduce method.

Shift-reduce parser attempts to construct a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top). While the parser primary operations are shift and reduce, there are actually four possible actions the shift-reduce parser can make: shift, reduce, accept, and error.

Initially, the stack (which uses \$ as bottom marker) is empty and the string to be parsed is on the input buffer. The parser operates by shifting zero or more input symbols onto the stack until a handle β is on top of the stack. The parser then reduces β to the LHSS of the appropriate production. The parser repeats this cycle until it detects an error or the stack contains the start symbol and the input string is empty. After entering this configuration, the parser halts and announces successful completion of parsing.

The LR parser is a shift-reduce parser which has the importance and popularity for many reasons [26]. First, this technique can be used to construct parsers that recognize context-free languages. Second, LR parsers are predictive descent parsers, i.e., they avoid the use of backtracking which is quite expensive and slow. Third, grammars amendable to LR parsing are more powerful than grammars that can be parsed by other predictive parsers. Finally, no other left-to-right input scanner can detect a syntax error in the input string as fast as the LR parsers.

Synthesized attributes can be evaluated by the LR parser as the input is being parsed. The parser can keep the values of the synthesized attributes (which are associated with the CF-grammar symbols) on its stack. Whenever a reduction is made, the values of the new synthesized attributes are computed from the attributes appearing on the stack for the RHSSs of the reducing production [26].

4.2 Time constraint of alerts

When using different IDS sensors to monitor a real network, alerts order becomes a critical issue. A number of factors affect the timestamp of alerts such as clock discrepancy of IDSs, network delay, IDS system workload, etc. [30]. The correct order of alerts has an influence on the correctness of the Alerts Parser.

To solve this problem, we used the conservative approach proposed by Wang et al. [11]. They classified the alerts to *concurrent* and *non-concurrent* alerts depending on t_{con} threshold; the real problem is with the concurrent alerts. They addressed this problem by reordering alerts inside a time window before feeding them to the correlation engine; assuming the delays are bounded by a threshold t_{max}. They postponed the processing of an alert a_1 with a timestamp t_1 until t_{max} time has passed since the time of receiving a_1. Then, the postponed alerts are reordered so they arrive at the correlation engine in the correct order [11]. There is a trade off between the performance and the correlation effectiveness when setting the value of t_{con}.

4.3 Addresses similarity

An alert A can be characterized by a set of features. Assume that A. f_i represents the ith feature of an alert A. The used features in this paper are: alert type, source IP, target IP, and time stamp; where alert type is the subattack name, source IP is usually the address of the attacker, target IP is usually the address of the victim, and time stamp is the time information. The alerts table is denoted by T. We consider only IPv4 addresses.

Definition 3 *The addresses similarity between any two alerts can be computed as follows:* $\text{IP\_Sim}(A_i, A_j) = \text{Sim}(A_{i.SrcIP}, A_{j.SrcIP}) + Sim(A_{i.DstIP}, A_{j.DstIP}) + Sim(A_{i.DstIP}, A_{j.SrcIP})$, *where: (1)* $\text{Sim}(A_{i.SrcIP}, A_{j.SrcIP})$ *and* $\text{Sim}(A_{i.DstIP}, A_{j.DstIP})$ *are the source and target IP similarities respectively; (2)* $\text{Sim}(A_{i.DstIP}, A_{j.SrcIP})$ *also checks if* $A_{i.DstIP}$ *is similar to* $A_{j.SrcIP}$*; this feature is necessary because sometimes the attacker uses one victim as a step stone to compromise other victims; These measures compute the common similar bits of any two IP addresses from the left then the result is divided by 32; (3)* $A_i.time \leq A_j.time$.

Assume that we want to compute the similarity between two source IP addresses Sim (192.168.0.1,192.168.2.8), then the computation of similarity will be as follows:

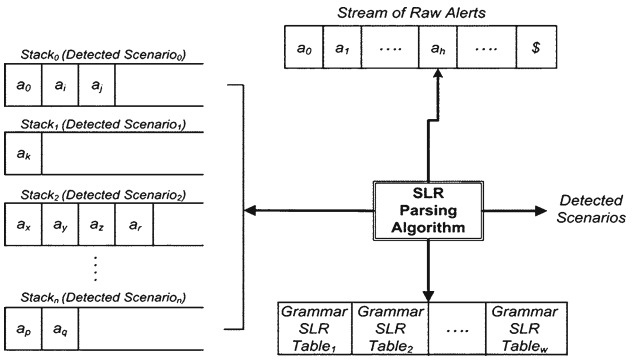

Fig. 4 The LR parsing framework is modified to be suitable for alerts correlation. The SLR tables and alerts are fed to the parser. The detected scenarii are forwarded to the second phase for more process

11000000.10101000.00000000.00000001
11000000.10101000.00000010.00001000

The number of common similar bits (the underlined bits) from the left is 22; as a result, the value of this measure will be $22/32 = 0.688$. Some attacks (like Mstream_Zombie) use the broadcast address such as 255.255.255.255 which has a special treatment. It gets a maximum similarity with any IP address because it is forwarded to all computers in the network.

4.4 The LR parser to correlating alerts

We used the Alerts Parser to discover the multi-step attacks presenting in the raw alerts. The Alerts Parser core is the LR parser. The LR parser has stringent style. Consequently, we performed some modifications to make it flexible. First, we eliminated the error action because it is undesirable in the scenario recognition process. The use of ε (i.e. the empty string) was to pass the missed attacks and to avoid the error action. Second, instead of using one SLR table, we used many SLR tables each contains one or more scenarii (one SLR table for each CF-grammar). Finally, we used many stacks (instead of single stack) which are created dynamically depending on the number of scenarii in the alerts. The modified LR parser can be shown in Fig. 4.

Each scenario can be represented by a CF-grammar or sometimes the CF-grammar can contains some scenarii. The SLR table should be built for each CF-grammar; it represents the scenario template. The scenario template is denoted here by TPLT and the set of scenario templates by S_{TPLT}.

Definition 4 *Given S_{TPLT}, the modified LR parser represents each resulting scenario by a stack. The stack STK is seven tuples = <V, State, Att_Val, Reference, SourceIP, TargetIP, Step Stone >, where State represents the current state of a finite machine, Att_Val is the value of synthesized attribute for V, Reference is an array of alerts indices, and Step Stone is the intermediate victims. The last three elements are the properties of the STK. The set of the STKs is denoted by S_{STK}.*

```
Procedure Scenario_Recovery
input: alerts in time order
output: a set of scenarios
method:
    for each alert A∈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 ∈ idx() do
            Call LR_Parsing(A, STK_j); } }
    Return set of STKs 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;
```

Fig. 5 The main procedures of the Alerts Parser

When a new alert becomes available, it is first being 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 these conditions have frequently satisfied. We called this checking as *STK_Selection*. Before shifting the new alert to any stack, the semantic rules can be used to check the consistency of the new alert with the compared stack. If the *STK_Selection* fails to find any STK, then a new STK will be created. The main Alerts Parser procedures can be shown in Fig. 5.

The role of $ symbol is to enforce the LR parser to check if some scenarii (i.e. STKs) are completed in order to send them to the second phase. This can be done by inserting dummy $ symbols (in addition to the last $ symbol) between alerts to detect the completed scenarii as soon as possible. This insertion removes the completed STKs.

One advantage of the Alerts Parser 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 detected scenarii will be forwarded to central processing unit to complete the second phase. The second phase tries to connect the resulting scenarii. Refer to Sect. 5.3 for more details.

Sometimes, the single scenario spread over more than one STK; as a consequence, the second phase connects the separated scenarii. The third phase receives the detected attack scenarii as input and uses Graphviz package [31] to visualize them.

Alerts Parser time complexity is related to the number of alerts and the number of scenarii in the alerts. The time for processing each new alert with *STK_Selection* function is linear in (S_{STK}). It is well known that LR parser has a linear time complexity [26]. Consequently, the total time complexity of the scenario recovery procedure (Fig. 5) is $O(n*(S_{TPLT}+S_{STK}))$ where n is the number of alerts.

Two optimizations points can be noticed: first, S_{TPLT} is related to the network vulnerabilities which are limited. Some attack graph methods can generate the scenarii (which the network is vulnerable to) automatically [32]. Second, the dealing with flooding alerts makes the performance of *STK_Selection* function decreasing as more and more alerts received. Depending on site policy, they are either neglected (such as HP alerts) or aggregated.

5 Experimental results

This section describes the experiments conducted to evaluate our system. The Alerts Parser was tested on an AMD Athelon processor 2.01 GHz with 512 RAM running Windows XP. We used two different IDS sensors: RealSecure 6.0 [21] and Snort 2.6.1 [20]. Two sets of experiments were performed. The first set of experiments was conducted with the DARPA 2000 scenario specific datasets [14]. The sensor alerts reported by RealSecure and Snort sensors on these datasets were forwarded to Alerts Parser to test system effectiveness and to show its ability to differentiate true and false alerts. We used DARPA 2000 datasets for this purpose because they have well known attack scenarii, which can be referenced in the included description or previous work, such as [9].

For the second set of experiments, DEFCON 8 CTF datasets [15] and the DARPA 2000 datasets were chosen. DEFCON 8 CTF datasets contain intensive attacks launched by competing hackers. We chose this datasets to test the performance of Alerts Parser because they have a high alert rate (about 41 alerts per second) [33]. Snort was used with this set of experiments to test system performance.

5.1 Effectiveness tests

The datasets in this set of experiments are the DARPA 2000 datasets from MIT Lincoln Laboratory [14]. They consist of two multi-step attack scenarii, namely LLDDoS 1.0 and LLDDoS 2.0.2. LLDDoS 1.0 contains a series of attacks in which the attacker probes the network, probes the active hosts for Solaris Sadmind, breaks into these hosts with their vulnerabilities, installs the Mstream DDos software on the three compromised hosts, and actually launches a DDos attack against a victim server. 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. The objective of this set of experiments is to demonstrate the effectiveness of the proposed approach in scenario recovery. The RealSecure and Snort sensors were used in this set of experiments.

At the very beginning, the generated alerts from RealSecure and Snort were saved in alerts repository. Then, we used the approach mentioned in Sect. 4.2 to process the timestamp attribute. Hereafter, these alerts were fed to the Alerts Parser to construct the attack scenarii. Alerts Parser depends on SLR tables which were built as stated in Sect. 3. Our experiments with DARPA 2000 datasets shows results like the described one in [9], validating the correctness of our correlation method.

The CG which is discovered from the LLDDoS 1.0 inside zone can be seen in Fig. 6. Each node in this figure represents a subattack (i.e., CF-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 means that either the attacker did some trials to succeed or some alerts have been aggregated. There are no false alerts in this figure. In addition, the extracted CG from LLDDoS 2.0.2 is shown in Fig. 7.

To test the effectiveness of the Alerts Parser, we used the measures of completeness and soundness defined in [9]. The soundness measure (Rs) evaluates the rate of true alerts appearing in the CG. The completeness measure (Rc) looks for missing true alerts from the CG. Equation (1) shows these measures. The results of the two measures are given in Table 2. The missed alerts by the IDS sensors degrade the effectiveness which was the situation in our experiments. The missed alerts by the used IDS sensors affect the completeness measure results as shown in Table 2. In addition, the experiments produces good results for the soundness measure.

$$R_c = \frac{\#\ of\ correctly\ correlated\ alerts}{\#\ of\ related\ alerts},$$
$$R_s = \frac{\#\ of\ correctly\ correlated\ alerts}{\#\ of\ correlated\ alerts} \quad (1)$$

5.2 Alerts differentiation tests

The goal of this test was to see if the Alerts Parser can differentiate false and true alerts. The Alerts Parser has shown the ability to reduce the correlation false alert rate by not correlating false alerts. The CGs only show true alerts classes that

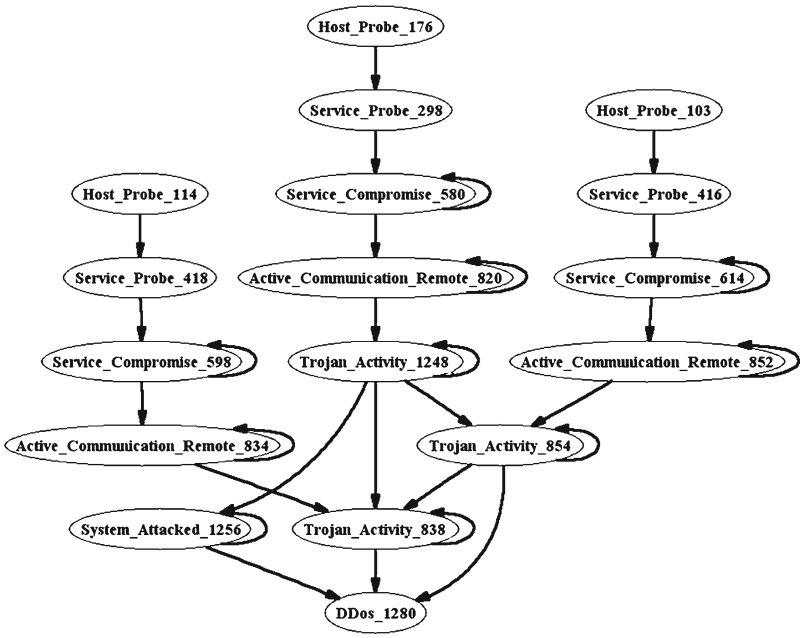

Fig. 6 The detected scenario from the inside zone of LLDDoS 1.0 dataset. This correlation graph contains three sub scenarii from one attacker to three victims. *Nodes* in this correlation *graphs* represent attacks and edges represent time order

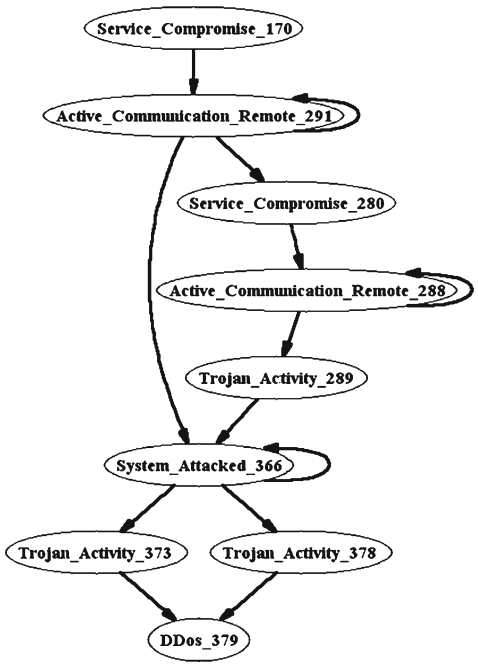

Fig. 7 The detected scenarii from the inside zone of LLDDoS 2.0.2 dataset. The attacker uses one victim as a step stone o attack another victim. Though the IDS sensors were missed some attacks, Alerts Parser correlated the related attacks

Table 2 Correlation effectiveness of the Alerts Parser

	LLDDoS 1.0		LLDDoS 2.0.2	
	DMZ	Inside	DMZ	Inside
# Correctly correlated alerts	140	87	14	29
# Related alerts	144	91	16	32
# Correlated Alerts	140	87	14	30
Rc (%)	97.22	95.60	87.50	90.63
Rs (%)	100	100	100	96.67

Table 3 Alerts differentiation test results of the Alerts Parser

	LLDDoS 1.0		LLDDoS 2.0.2	
	DMZ	Inside	DMZ	Inside
# Alerts	140	87	14	30
# True alerts	140	87	14	29
# Observed attacks	89	60	8	14
# Detected attacks	57	39	6	11
DR (%)	64.05	65	75	78.57
CFAR (%)	0	0	0	3.33

are correlated. We only computed the false alert rates of the correlated alerts.

The Detection Ratio (DR) and Correlation False Alert Ratio (CFAR) can be computed using Eq. (2) as defined in [9]. The correlation false alert rates for the Alerts Parser are shown in Table 3. False and true alerts were determined according to the description of the datasets. We can look at the correlated alerts and compare them to the description of the scenario to see if they are true or false alerts.

$$DR = \frac{\#\ of\ detected\ attacks}{\#\ of\ observed\ attacks},$$
$$CFAR = 1 - \frac{\#\ of\ true\ alerts}{\#\ of\ alerts} \quad (2)$$

It should be noted that the used IDS sensors (sometimes) generated duplicate alerts for the same attack. As a result, we counted them as the same attack. In other words, the alerts from RealSecure and Snort that refer to the same attack are computed as one attack. For this reason, we do note, in Table 3, that the number of attacks are less than true alerts. In addition, sometimes the attacker tries the same attack with different stack pointers until he succeeds; as a result, we computed these attacks as different attacks.

It can be concluded from the results shown in Table 3 that the Alerts Parser can efficiently differentiate true and false alerts. In addition, the missed attacks by the IDSs have a negative effect on the results and this can be noted from the detection ratios.

5.3 Performance tests

The objective of the third set of experiments is to evaluate the Alerts Parser performance 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.

We applied the Alerts Parser on the DEFCON 8 CTF datasets. Unfortunately, due to the nature of the DEFCON 8 CTF datasets, we did not have any information about the attack scenarii within it. Thus, we analyzed the resulted alerts depending on Snort signature database [22]. Hereafter, we encoded these alerts into regular grammars which were used for building SLR tables. About 40 regular grammars have been written for DEFCON datasets, where some regular grammars contain more than one scenario.

The DEFCON 8 CTF datasets generated a large amount of alerts. The resulting alerts from Snort IDS were 1847745 raw alerts. Scanning related alerts were divided into two groups: host probe and service probe. Host probe alerts account for 1255881 (67.9%) whereas service probe alerts account for 425398 (23%). Other alerts include service compromise, Dos, etc., account for 166466 (9.1%). In Sect. 4.4, we referred to the processing of flooding alerts which are either neglected or aggregated. 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 33,818 alerts.

As can be seen in Fig. 8, the Alerts Parser measures its own processing time per alert (averaged per 1,000 alerts). Clearly, the processing time of alerts increases when the scenarii in the alerts increase. The increasing of scenarii in the alerts leads to increase the number of stacks making the Alerts Parser slow down. The DEFCON 8 CTF 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.

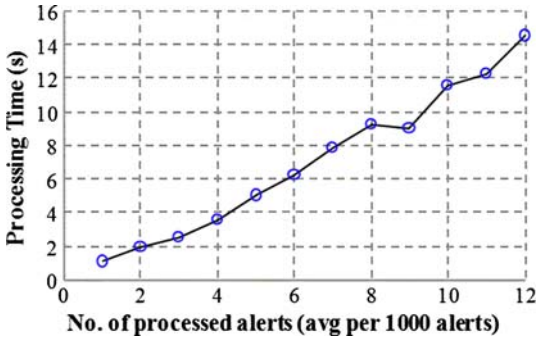

Fig. 8 The processing time of Alerts Parser on DEFCON 8 CTF datasets

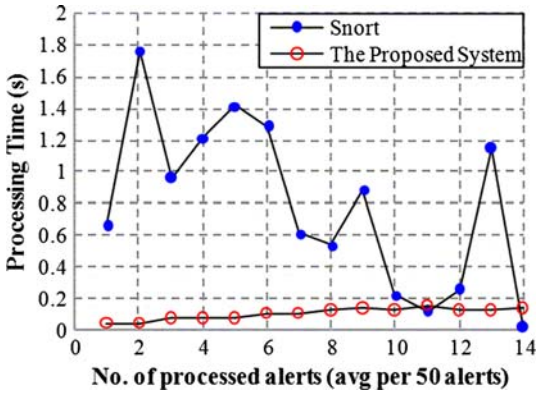

Fig. 9 Alerts Parser processes alerts of DARPA 2000 datasets faster than Snort sensor

Moreover, we have compared the processing time for the Alerts Parser to the delay between receiving two consecutive alerts from Snort. The DARPA 2000 datasets were used for this purpose. Figure 9 shows the processing time per alert (averaged per 50 alerts). Obviously, the Alerts Parser works faster than Snort in processing the entire dataset.

In spite of the fact that few papers have published performance data, data are available though not in every paper. The correlation system created by Debar et al. [34] reported to process one alert per second [35]. Haines et al. [36] have a very modest alert rate of about 1-10 alerts per second. The work of Zhou et al. [37] reported to process 53 alerts per second. The Alerts Parser has an alert rate of about 70 alerts per second. This is far greater than any alert rate seen currently on large networks [33].

We tested the capability of the Alerts Parser 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 Sect. 5. We 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. 10, the processing time declines whenever the processing nodes increase. We can conclude from these results that the Alerts Parser has the capability to work in parallel increasing

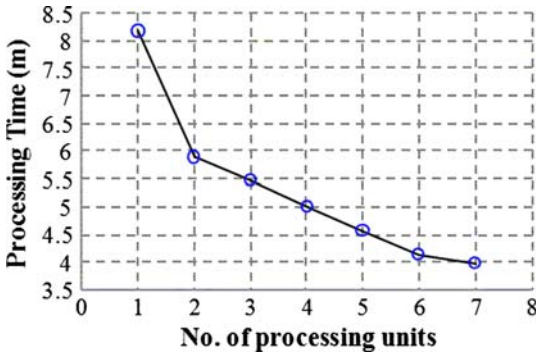

Fig. 10 The relation between the number of processing units and the processing time

the processed alerts rate to about 100 alerts per second (using three nodes).

6 Discussion

We have proposed a novel correlation method to extract attack strategies from intrusion alerts. This method employed the ACF-grammar and LR parser to process alerts. The representation of multi-step attacks by ACF-grammar makes the description easy to understand and update. The experimental results on the DARPA 2000 datasets and DEFCON 8 CTF datasets shows that our graph representation results are accurate and condensed. The CGs produced by our system as we believe convey more information to the security analyst than the individual alerts.

We have compared our method with three methods having different techniques. The first one is the method of Ning et al. [9] which uses the knowledge of prerequisites/consequences for each alert to build attack scenarii. The second method, which is the work by Qin et al. [8], uses statistical causality analysis to correlate alerts, and the third system employs the hidden colored Petri-Net to predict the next goal of intruder by Dong et al. [10]. We were unable to compute the soundness and completeness measures for some methods because they have published few results. Therefore, we used different measures which appeared in [8]. Equation (3) shows these measures. True causality rate (TCR) measures the ratio of the correctly correlated alerts. False causality rate (FCR) measures the ratio of the incorrectly correlated alerts.

$$TCR = \frac{\#\ of\ correct\ causal\ alerts}{total\ \#\ of\ causal\ relationships},$$

$$FCR = \frac{\#\ of\ incorrect\ causal\ alerts}{total\ \#\ of\ causal\ alerts} \quad (3)$$

Some measures have been used in this comparison, which are: detection ratio, true causality rate, false causality rate, and the size of the resulted graphs. The comparison was made using DARPA 2000 datasets. The results of comparison are

Table 4 General comparison among the four methods for the DARPA datasets

	Ning et al.	Qin et al.	Dong et al.	Alerts Parser
DR (%)	56.43	–	96.31	70.66
TCR (%)	79.27	97.08	96.16	92.73
FCR (%)	4.94	4.42	10	0.83
# of Nodes	44	19	–	17
# of Edges	153	21	–	29

presented in Table 4. It can be noted that our system has enhanced all measures over Ning et al. system. Qin et al. and Dong et al. methods have better results for TCR measure. Due to the reason that the value of TCR is affected by the missed attacks, our method produced lower value for it. Table 3 shows the number of the missed attacks by the IDS sensors in our experiments. However, we can note (from the FCR measure) that the soundness of our results is the best. In addition, we can note the amount of noise in their results. We think that they made a tradeoff between the TCR and FCR measures.

The proposed method provides a high-level representation of the correlated alerts, which reveals the causal relationships among the alerts. The resulting CGs (cf. Sect. 5) clearly shows the strategies behind these attacks. One advantage of our method is the compressing of the resulted CGs. Such an abstracted CGs provide a concise view of the real attacks, which help the security analysts to understand the results of alert correlation. The number of edges and nodes in Table 4 represents the size of the CG which appeared in LLDDoS 1.0 inside zone; our method was produced CG which has acceptable size.

The more information is fed to the Alerts Parser, the better the accuracy is obtained. The proposed model makes use of vulnerability information, attack knowledge, and prerequisites/consequences knowledge of the alerts. The vulnerability information can be used to write scenarii related to the protected network. The scenarii, which the network is vulnerable to, can be generated automatically using attack graph methods [32]. Attack knowledge was used to write the CF-grammars. And finally, the prerequisites/consequences knowledge of the alerts was used as filters to reduce the false correlations; furthermore, they make the correlation process more reliable.

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 [11]. As we have noted in Sect. 4.4, the time complexity of the Alerts Parser was $O(n*(S_{TPLT}+S_{STK}))$, which seems to be an attractive solution for the alerts correlation problem.

To recapitulate, the contributions of this paper are three folds: First, the scenarii are represented by CF-grammars

which make the rules writing and updating an easy task. Second, the Alerts Parser generates compressed and easy to understand CGs which reflect the attack scenarii. Finally, our method can tolerate missed attacks by IDS at the same time of correlation.

Prior to applying our approach it is important to consider its limitations. First, our method depends on the underlying IDSs to provide alerts. Though our reasoning process can pass the undetected attacks, the attacks missed by the IDSs certainly have a negative effect on the results as we have seen on the previously presented materials. Second, our approach is not fully effective for the coordinated attacks in which many attackers cooperate to do some goals. Nevertheless, it is not clear whether any method can correlate coordinated attacks without making mistakes. Finally, it is worth mentioning that the results produced by Alerts Parser are only as good as the classification information provided by the user. This process requires expert users who have certain skills and insights in the attack scenarii.

7 Conclusions and future work

This paper presented a method for constructing attack scenarii through alert correlation using compiler techniques. The ACF-grammar was used to describe the multi-step attacks using alerts classes. Based on ACF-grammars, the modified LR parser was used to detect the scenarii presented in the alerts. The ACF-grammar makes the scenario writing and updating an easy task; furthermore, it describes multi-step attacks explicitly. Experiments performed on different datasets showed that the Alerts Parser effectively correlates multi-step attacks in spite of the missed attacks by the IDS.

Several issues worth investigation in the future. First, the proposed approach has the capability to compensate the missed attacks by the IDS; as a consequence, we intend to develop this capability. Second, we want to deploy this approach in real environment. Finally, we believe that there is a room for enhancement in the performance of our approach as new research avenue.

Acknowledgments The authors thank the anonymous reviewers for their valuable comments and suggestions in improving this paper. This research was supported by Project 863 (2007AA01Z442).

References

1. Stefan, A.: The base-rate fallacy and its implications for the intrusion detection. In: Proceedings of the 6th ACM Conference on Computer and Communications Security, pp. 1–7. Kent Ridge Digital Labs, Singapore (1999)
2. Pietraszek, T., Tanner, A.: Data mining and machine learning-towards reducing false positives in intrusion detection. Inf. Secur. Tech. Rep. **10**, 169–183 (2005)
3. Yu, J., Reddy, Y.V.R., Selliah, S., Reddy, S., Bharadwaj, V., kankanahalli, S.: TRINETR: an architecture for collaborative intrusion detection and knowledge-based alert evaluation. J. Adv. Eng. Inform. **19**, 93–101 (2005)
4. Perdisci, R., Giacinto, G., Roli, F.: Alarm clustering for intrusion detection systems in computer networks. Eng. Appl. Artif. Intell. **19**, 429–438 (2006)
5. Julisch, K.: Clustering intrusion detection alarms to support root cause analysis. ACM Trans. Inf. Syst. Secur. **6**, 443–471 (2003)
6. Ning, P., Xu, D., Healey, C.G., Amant, R.S.: Building attack scenarios through integration of complementary alert correlation methods. In: Proceeding of 11th Annual Network and Distributed System Security (NDSS'04), pp. 97–111 (2004)
7. Dain, O.M., Cunningham, R.K.: Fusing a heterogeneous alert stream into scenarios. In: Proceeding of ACM Workshop on Data Mining for Security Applications, pp. 231–235 (2001)
8. Qin, X., Lee, W.: Statistical causality analysis of INFOSEC alert data. In: Proceeding of 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003), Pittsburgh, pp. 591–627 (2003)
9. Ning, P., Cui, Y., Reeves, D.S., Xu, D.: Techniques and tools for analyzing intrusion alerts. ACM Trans. Inf. Syst. Secur. **7**, 274–318 (2004)
10. Yu, D., Frincke, D.: Improving the quality of alerts and predicting intruder's next goal with hidden colored petri-net. Comput. Netw. **51**, 632–654 (2007)
11. Wang, L., Liu, A., Jajodia, S.: Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. J. Com. Commun. **29**, 2917–2933 (2006)
12. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Proceeding of International Symposium on Recent Advances in Intrusion Detection, LNCS vol. 2212, pp. 54–68. Springer, Heidelberg (2001)
13. Cuppens, F., Miège, A.: Alert correlation in a cooperative intrusion detection framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 202–215 (2002)
14. MIT Lincoln Lab., DARPA Intrusion Detection Scenario Specific Datasets. http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html
15. DEFCON Captures the Flag (CTF) Contest. http://cctf.shmoo.com/data/cctf-defcon8/
16. Morin, B., Me, L., Debar, H., Ducasse, M.: M2D2: A formal data model for IDS alert correlation. In: Proceeding of International Symposium on Recent Advances in Intrusion Detection, pp. 115–137 (2002)
17. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceeding of Ninth ACM Conference on Computer and Communications Security (CCS'02), pp. 217–224 (2002)
18. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceeding of IEEE Symposium on Security and Privacy (S&P'02), pp. 273–284 (2002)
19. Noel, S., Jajodia, S.: Correlating intrusion events and building attack scenarios through attack graph distance. In: Proceeding of 20th Annual Computer Security Applications Conference (ACSAC'04) (2004)
20. Roesch, M.: Snort-lightweight intrusion detection for networks. In: Proceeding of USENIX LISA Conference, pp. 229–238 (1999)
21. Internet Security Systems, RealSecure Intrusion Detection System. http://www.iss.net
22. Sourcefire, Snort signature database, http://www.snort.org/pub-bin/sigs.cgi (2007)

23. Internet Security Systems, RealSecure Signatures Reference Guide. http://documents.iss.net/literature/RealSecure/RS_Signatures_6.0.pdf
24. Sipser, M.: Introduction to the theory of computation, second ed. Massachusetts Institute of Technology, Cambridge (2006)
25. Ilgun, K., Kemmerer, R.A., Porras, P.A.: State transition analysis: a rule-based intrusion detection system. IEEE Trans. Soft. Eng. **21**, 181–199 (1995)
26. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers, Principles, Techniques, and Tools. Addison-Wesley, Reading (1986)
27. Power, J.: Notes on Formal Language Theory and Parsing, Technical Report. National University of Ireland, Maynooth (2002)
28. ÆMogensen. T.: Basics of Compiler Design, vol. 5, 2nd edn. http://www.diku.dk/~torbenm/Basics (2007)
29. White, J.: Algorithms and Foundations Qualifier, Technical Report, January (2007)
30. Serrano, A.: Integrating Alerts from Multiple Homogeneous Intrusion Detection Systems, Master Thesis, North Carolina State University, Raleigh (2003)
31. AT & T Research Labs, Graphviz-open source graph layout and drawing software, http://www.research.att.com/sw/tools/graphviz/
32. Hamza, L., Adi, K., El Guemhioui, K.: Automatic generation of attack scenarios for intrusion detection systems. In: Proceeding of IEEE AdvancedInternational Conference on Telecommunications and International Conference on Internet and Web Applications and Services (2006)
33. Valeur, F.: Real-Time Intrusion Detection Alert Correlation, PhD Dissertation, University of California, Santa Barbara (2006)
34. Debar, H., Wespi, A.: Aggregation and correlation of intrusion detection alerts. In: Proceeding of International Symposium on Recent Advances in Intrusion Detection, Davis, pp. 85–103 (2001)
35. Debar, H., Wespi, A.: Aggregation and Correlation of Intrusion Detection Alerts, Presentation Slides, October (2001)
36. Haines, J., Ryder, D.K., Tinnel, L., Taylor, S.: Validation of sensor alert correlators. IEEE Secur. Priv. Mag. **1**(1), 46–56 (2003)
37. Zhou, J., Heckman, M., Reynolds, B., Carlson, A., Bishop, M.: Modeling network intrusion detection alerts for correlation. ACM Trans. Inf. Syst. Secur. **10**, 1–31 (2007)