Building Scenario Graph Using Clustering

Safaa O. Al-Mamory
School of Computer Science,
Harbin Institute of technology,
Harbin, China
Safaa_vb@yahoo.com

Hong Li Zhang
School of Computer Science,
Harbin Institute of technology,
Harbin, China
zhl@pact518.hit.edu.cn

Abstract

The increasing use of Network Intrusion Detection Systems (NIDSs) and a relatively high false alert rate can lead to a huge volume of alerts. This makes it very difficult for security analysts to detect long run attacks. In this paper, we have proposed a system that represents a set of alerts as subattacks. Then correlates these subattacks and generates abstracted scenario graphs (SGs) which reflect attack scenarios. We have conducted the experiments using Snort as NIDS with different datasets that contains multistep attacks. The resulted compressed SGs imply that our method can correlate related alerts, uncover the attack strategies, and can detect new variations of attacks.

1. Introduction

When the NIDS detects a set of attacks, it will generate many alerts that refer to security breaches. Unfortunately, the NIDS cannot deduce anything from these separated attacks. So, alert correlation is an important solution to link separated attacks, to give alerts another meaning, and to infer attack scenarios.

Alert correlation and analysis are a critical task in security management. Recently, several techniques and approaches have been proposed to correlate and analyze security alerts, most of them focus on the aggregation and analysis of raw security alerts, and build attack scenarios.

An interesting method is the work of Ning *et al.* [1]. They 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 NIDS, correlated off-line, and tried to draw a big picture (through *SG*s) of what happens in the monitored network. However, there are some shortcomings associated with this method:

- The graph explosion problem that occurs in the generated *SG*s makes the resulted graphs complex and hard to understand by the security analyst.
- Huge number of rules used to draw these graphs which represent prerequisites and consequences of alerts.
- The affects of the missed attacks by NIDS resulted in graphs that yield separated *SG*s.

To address the disadvantages of this method, we have proposed a system that can address these problems. The proposed system contains three components: alert prioritization, alert clustering, and finally correlation and *SG* generation. Also Breadth-First search algorithm was used to find the related attacks. The resulted *SG*s show that the proposed system can correlate related alerts, uncover the attack strategies, and can effectively simplify the analysis of large amounts of alerts.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 states system architecture in details. Section 4 presents our experiments. Section 5 discusses the results and Section 6 concludes this paper.

2. Related Work

Many researchers propose systems that aim to build attack scenarios depending on various techniques. Dain *et al.* [2] use data mining approach to combine the alerts into scenarios in real time. The probabilistic alert correlation [3] based on the similarities between alerts to correlate them. Measures are defined to evaluate the degree of similarity between two alarms.

Qin *et al.* [4] present an alert correlation system combining a Bayesian correlation system with a statistical correlation system using Granger Causality

Test (GCT), a time series-based causal analysis algorithm. Based on the results of this analysis the GCT module constructs a correlation graph. As the structure of the network is predetermined, the Bayes-based correlation module can discover alerts that have direct causal relationships according to domain knowledge.

The work of Ning *et al*. [1] generates *SG*s depending on pre-/post-conditions of an individual alerts. They propose an alert 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, *Prerequisites* (the conditions for an intrusion to be successful) and *Consequences* (the possible outcome of an intrusion).

3. System Architecture

The proposed system is composed of three parts: Prioritization, Clustering, and Correlation and *SG* Generation. As shown in Fig. 1, it takes the raw alerts from NIDS as input then enhances alerts quality using alert prioritization. After that alert clustering is performed using classification and merging, whereas the similar classes produced from the classification will be merged. Finally, the last component generates *SG*s.

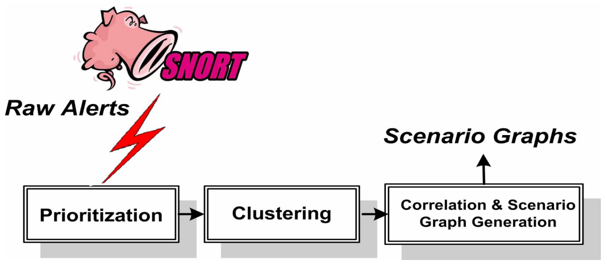

Figure 1. Proposed system components

Alert clustering aim is to handle the raw alerts produced by NIDS due to a certain attack, to produce a higher-level alert message, called *meta-alert* (*MA*), summarizing the detected attacks characteristics.

A meta-alert is characterized by: alert class, which is the generalized alert type or subattack name, the source IP address, the target IP address, and time information. A reference to the log file of the NIDS is reported so that further investigation on the results can be carried out.

3.1. Alert Prioritization

Alert prioritization is performed to assess the relative importance of alerts generated by the sensors. This method has to take into account the security policy and the security requirements of the site where the correlation system is deployed [5]. Therefore, prioritizing of alerts aids in substantial reduction of alert volume.

3.2. Alert Clustering

Classification followed by aggregation is the clustering we refer to in this paper. The classification was done by using alerts abstraction. The alert classification scheme is designed to categorize alerts into groups that most effectively indicate their stage in a multistage attack. An alert can be part of multiple classes. Each class has its name that indicates the general category (e.g., Host Probe, User Access, Service Compromise, etc.). Alerts descriptions were taken from the Snort signature database [6]. Appendix A summarizes the classification scheme that we have performed. Aggregation will merge the similar classes of alerts resulted from classification within a specified time window.

3.3. Building Scenario Graphs

This component contains alert correlation and *SG* generation. In this paper, we have proposed a technique that builds simple *SG* using alert clustering and correlation. The correlation depends on the *Relation Matrix* (*RM*) that contains the similarities between every two *MA*s, and few predefined rules. Three measurements (having numerical values) have been used which are listed below.

- Msr_1: How much Similar_SourceIP(MA_1,MA_2)? This feature computes the common similar bits of two IP addresses from the left, and the result divided by 32.
- Msr_2: How much Similar_TargetIP(MA_1,MA_2)? This feature is computed such as the previous one.
- Msr_3: TargetIP(MA_1)=SourceIP(MA_2)? This feature is necessary because sometimes the attacker use one victim as a step stone to compromise another victim.

It is very important to find the strength between any two *MA*s to correlate them together or not. Computation of that strength depends on the measurements. The correlation strength will be computed for all the *MA*s and these *MA*s assumed to be in time order. We suggest representing the correlation strength of any related *MA*s in a triangle matrix (i.e. *RM*). In this matrix $V_{(1,3)}$, for example, means the relation between MA_1 and MA_3 and MA_1 comes before MA_3.

Equation (1) was used to compute the correlation strength value between any two *MA*s in *RM*. The

Is_Successor(j,i) variable in (1) is a Boolean variable (0 or 1) that determine if MA_j can occur after MA_i. If so, the similarities between its features will be computed, otherwise the value is zero. The Msr_k variable in (1) is the k^{th} measurement's value.

$$V_{(i,j)} = Is\_Successor(j,i) * \sum_{k=1}^{3} Msr_k(MA_i, MA_j) \qquad (1)$$

The *Is_Successor* variable can be used to pass the missed attacks. Assume we have a scenario ($A \to B \to C$) which means three attacks should occur in order. The *Is_Successor* can be set to return true if attack *A* followed by either *B* or *C* to pass the missed attack.

Scenario graphs can be represented by nodes (i.e. the subattacks) and arcs (i.e. the relation between two subattacks). The direction of the arcs specifies the temporal relation. The *MA* here represents subattack.

Definition1. Given *MA* contains one or more alerts. Let S_{MA} be the set of *MA*s and let *t(MA)* is the earlier time in which *MA* has occurred. Thus it can be said that *Class(MA)* is the class of an *MA* that represents a subattack within a scenario. In a multistep attack, the early step of attack prepares for later ones. So we can build a relation $Prepare\text{-}for(MA_1,MA_2)$ if $Class(MA_1)$ prepare for $Class(MA_2)$ in the scenario and $t(MA_1) \leq t(MA_2)$. For any given two *MAs* α and $\beta \in S_{MA}$, α is called a parent of β and β is called a child of α if the relation $Prepare\text{-}for(\alpha, \beta)$ is satisfied. It should be noted that any child can have more than one parent. ■

Applying the Breadth-First search algorithm depends on the parent-child relationship that has assumed in definition 1. The pseudo code of the proposed algorithm that builds *SG* is shown in Fig. 2.

```
Input: Stream of Meta-Alerts in time order
Output: Scenario graphs

Initialize Queue Q and Graph G;
For each unvisited MA ∈ RM {
   Put new unvisited MA into Q and G;
   While (! IsEmpty(Q)) {
      Q.dequeue(ActiveMA);
      Q.enqueue(all unvisited children of ActiveMA);
       Set ActiveMA as visited;
      G←G ∪ ActiveMA (Connect ActiveMA to appropriate MAs
       in G using Attach_Threshold); }
   Output G as detected scenario graph;
   Set Q and G to empty; }
```

Figure 2. Pseudo-code of SG generation algorithm

Any new *MA* is not always linked to the latest *MA* in the generated scenario. Instead, it is connected to the *MA*s with which it has a high correlation value in *RM*. So, the representation is useful for inference with multiple goals of attackers. The intention of using graph representation for the attack scenario is to give the security analyst an intrinsic view of the network status.

The *Attach_Threshold* is used to control the membership of one *MA* to the scenarios. When this threshold is set to be small, the resulted graphs will be noisy, whereas when its value is set to be high many real attacks do not join to its *SG*s.

The generated *SG*s are concise and abstract. The references to the log file that exist in *MA*s can be used to drill-down and show more details about the low-level alerts.

4. Experiments

In this section, we report the experiments we performed to validate the suggested method. The experiments were conducted with the 2000 DARPA datasets [7] and Defcon 8 datasets [8]. Snort (Version 2.6.1) [9] was used because it is a freely available NIDS. The experiments were aimed to evaluate the effectiveness of our method in constructing attack scenarios.

4.1 DARPA Dataset Experiment

The 2000 DARPA scenario specific datasets include LLDOS 1.0 and LLDOS 2.0.2 [7]. LLDOS 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 its vulnerabilities, installs the Msream DDos software on the three compromised hosts, and actually launches a DDos attack against an off-site server. We have performed four sets of experiments, each with either the DMZ or the inside network traffic of one dataset.

The *SG*s discovered from the inside zone of LLDOS 1.0 were shown in Fig. 3. Each node represents a *MA*. The text inside the node is the class of the *MA* followed by *MA* ID. There are 15 *MA*s in this graph and there are no false alerts with it. Fig. 3 contains three subgraphs from one attacker to three victims. Three disjoint *SG*s were generated because Snort's fails to report some parts of the scenario, i.e. communication of the DDoS Trojans on the compromised hosts and DDoS attack.

The extracted *SG* from LLDOS 2.0.2 is shown in Fig. 4. It is clear from the mentioned *SG* that the attacker compromises one victim (i.e. 172.16.115.20) and installs mstream master in it. After that, he (from that victim) probes the network, compromises another victim (i.e. 172.16.112.50) and installs Mstream agent in it. Three false alerts correlated with resulted *SG* and to distinguish it, we have painted it in another color.

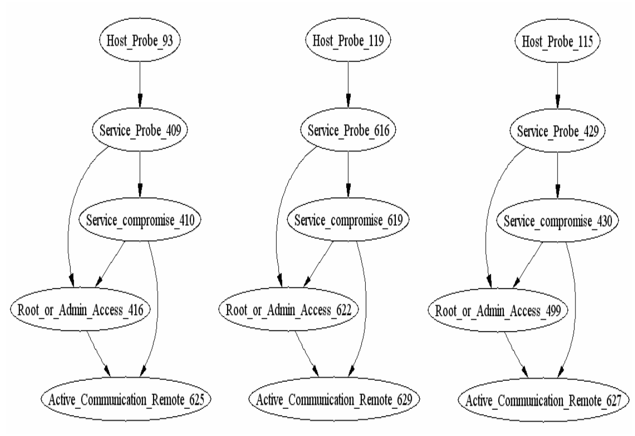

Figure 3. The *SG*s discovered in the inside zone of LLDOS 1.0

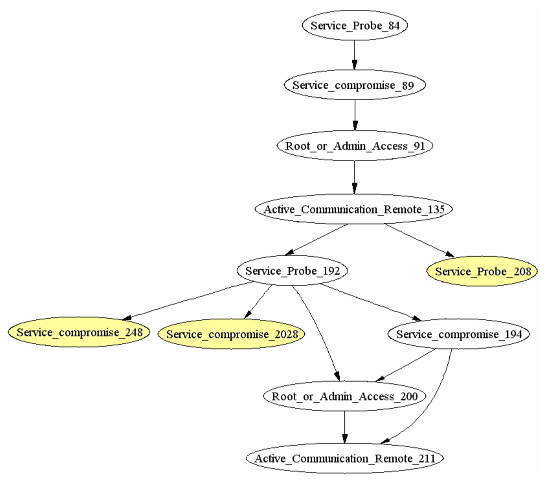

Figure 4. The SG of the LLDOS 2.0.2 inside zone

To test the effectiveness of the proposed system, we have used the measures of completeness and soundness defined in [1]. The soundness measurement (*Rs*) evaluates the rate of true alerts that appear in *SG*. The completeness measure (*Rc*) looks for missing true alerts from *SG*. Equations (2) and (3) show these measures. The results of two measures are shown in Table I.

$$Rc = \frac{\#\ correctly\ correlated\ alerts}{\#\ related\ alerts} \tag{2}$$

$$Rs = \frac{\#\ correctly\ correlated\ alerts}{\#\ correlated\ alerts} \tag{3}$$

The missed alerts by NIDS degrade the effectiveness which was the situation in our experiment where the missed alerts by Snort affect the completeness measure results as shown in Table I. Also the experiment was produced accepted values for the soundness measure except LLDOS 2.0.2 inside zone because there are ten incorrectly correlated alerts. This occurred due to *Attach_Threshold* value has reduced to catch all the real alerts.

Table I
Correlation Performance of the proposed system

	LLDOS 1.0		LLDOS 2.0.2	
	DMZ	Inside	DMZ	Inside
# Correlated Alerts	*122*	*60*	*6*	*24*
# Correctly Correlated Alerts	*122*	*60*	*6*	*14*
# Incorrectly Correlated Alerts	*0*	*0*	*0*	*10*
# Related Alerts	*136*	*72*	*6*	*16*
# Missed Alerts By Snort	*14*	*12*	*0*	*2*
Completeness Measure Rc	*89.7%*	*83.3%*	*100%*	*87.5%*
Soundness Measure Rs	*100%*	*100%*	*100%*	*58.3%*

4.2 Def Con 8 Dataset Experiment

As another case study, we applied our method on the Def Con 8 Capture The Flag (CTF) datasets [8]. Unfortunately, due to the nature of the Def Con 8 CTF datasets, we did not have any information about its scenarios. Thus, we only analyze the resulted *SG*s and discuss some of its scenarios.

The resulted alerts from snort (after prioritization filtering) were 1,847,745 raw alerts. Scanning related alerts divided into two groups: host probe and service probe. Host probe alerts account for 1,255,881 (67.9 %) while service probe alerts account for 425,398 (23%). Other alerts include service compromise, DDos, Dos, etc, account for 166,466 (9.1%). The remaining *MA*s after clustering are 170,404.

There are many scenarios in this dataset, so we will discuss two of them in some details. One of the generated *SG*s that contains two scenarios can be seen in Fig. 5. The attacker in this *SG* is 10.20.1.237 and the victim is 10.20.1.12. In the first scenario, the attacker scans the host to see if it is a live with an *ICMP Ping* alert and this appeared as host probe. After that he/she scans the victim's ports by *SCAN nmap XMAS* alert to gather the active services and this appeared as service probe. Then he/she exploits the ftp service by buffer overflow attack (i.e. *FTP command overflow attempt* and other alerts) and installs a Trojan, this abstracted in this figure as service compromise and active communication remote respectively. Finally the attacker connects to a victim server through the Trojan with *BACKDOOR CDK* alert that is the Trojan activity mentioned. The second scenario aimed to get user privileges that are satisfied by some web attacks. After the attacker compromises web application, he/she gathered information about the user account then logged as a normal user. There are also some other attack scenarios that our method is able to find; many of them are *port scanning* followed by *Buffer Overflow* attacks.

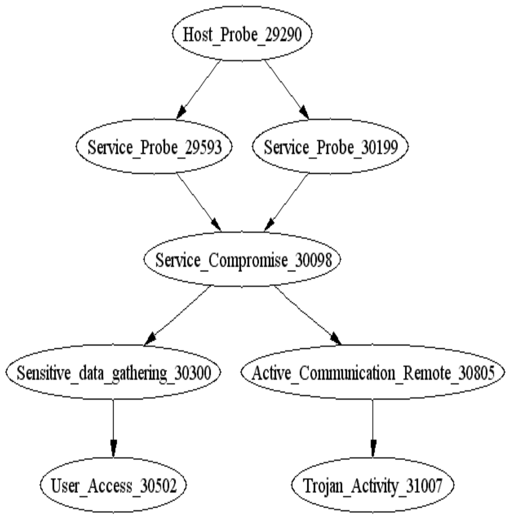

Figure 5. The scenario graph generated from Def Con 8 dataset

5. Discussion

From the literature, Ning *et al.* [1] have proposed a correlation method to extract attack strategies from alerts, which is similar to this work. The experimental results show that both approaches produce similar attack strategies. However, the difference locates in three folds. First, they have used pre-/post-conditions to correlate alerts whereas we used the scenario based approach. Second, we use few rules to produce these results whereas they have defined a large number of rules to correlate the alerts. And finally, we have represented the alerts by classes which reduce the required rules. By using alert's classes to represent scenario rules, there is ability to detect new variations of attacks. In other words, our system is more adaptive to the emerging of new attack patterns because we focus on alerts classes instead of alerts themselves.

Our method provides a high-level representation of alerts revealing the causal relationships between them. As we have seen in Section 4, *SG*s generated clearly show the strategies behind these attacks. The *SG* compression is one advantage of our method.

The contribution of this paper is therefore three folds: (1) Generate compressed and easy to understand *SG*s which reflects attack scenarios. (2) Represent the scenarios by alert classes instead of alerts themselves which reduce the required rules. (3) The ability to pass the missed attacks by NIDS that are located in the middle of the scenarios.

5. Conclusion

This paper presented a systematic method for constructing attack scenarios (or *SG*s) through alert correlation, using predefined attack scenarios. The proposed system filters out the unnecessary alerts, clusters the alerts as subattacks, and then generates *SG*s using a small set of rules and Breadth-First search algorithm. The generated *SG*s correctly reflect the multistage attacks in the datasets.

6. References

[1] Ning P., Cui Y., Reeves D. S. and Xu D., "Techniques and tools for analyzing intrusion alerts," ACM Transactions on Information and System Security, Vol. 7, Issue 2,pp. 1-44, 2004.

[2] Dain O.M. and Cunningham R. K, "Fusing a heterogeneous alert stream into scenarios," Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications, pp. 1-13, 2001.

[3] Valdes A. and Skinner K., "Probabilistic alert correlation," Proceedings: Recent Advances in Intrusion Detection, LNCS 2212, 54-68, 2001.

[4] Qin X. and Lee W., "Statistical causality of INFOSEC alert data," Proceedings: Recent Advances in Intrusion Detection, LNCS 2820; Springer-Verlag, pp. 73-93, 2003.

[5] Valeur F., Vigna G., Kruegel C. and Kemmerer R. A., "Comprehensive approach to intrusion detection alert correlation," IEEE Transactions on Dependable and Secure Computing 1 (3), pp. 146-169, 2004.

[6] Snort signature database, http://www.snort.org/pub-bin/sigs.cgi.

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

[8] Def Con captures the flag (ctf) contest, http://cctf.shmoo.com/data/cctf-defcon8/, 2000.

[9] SNORT, http://www.snort.org/, 2005.

Appendix A Summery of alert's classes generated by Snort

Classes	**Description**
Enumeration	Attacker tries to identify the user name and password to exploit poorly protected resources. e.g., *FTP Bad Login* alert.
Host Probe	This type determines active hosts and sometimes reveals specific software details like versions prior to launching an attack. Usually, this is the first step of many attacks. e.g., *ICMP Ping* alert.
Service Probe	Specifying which service running on which port is falling under this class. This may be a precursor to an attack to exploit the vulnerability of that service. e.g., *RPC sadmind UDP PING* alert.
Service Compromise	This is service attack step that exploit vulnerability (usually a buffer overflow vulnerability) in a specific service or software to gain more privileges. e.g., *WEB-CGI campus access* alert.
User Access	This class means that the attacker obtain the user (non administrator) privileges. The alerts in this class may happen after the Enumeration's alerts class described above. e.g., *TELNET Access* alert.
Root or Admin. Access	Intrusive step into a target machine with the privileges of administrator. Attacker may have access to a command shell with the same privileges. May overlap with buffer overflow attacks classified here as Service Compromise, but includes attacks that may exploit configuration flaws. e.g., *RPC sadmind query with root credentials attempt UDP* alert.
Dos	Alerts that indicate the possibility of Denial of Service attacks. e.g., *DOS IGMP dos attack* alert.
System Compromise	This class means that the system is infected by the attack. e.g., *BAD-TRAFFIC tcp port 0 traffic* alert.
Sensitive Data Gathering	This class describes the alerts that attempt to gain information on users and groups that exist on the hosts, cookies information, browser version etc. e.g., *INFO web bug 1x1 gif attempt* alert.
Active Communication Remote	Attacker's goal is to install a Trojan to facilitate further attacks. e.g., *BACKDOOR netbus active* alert.
Trojan Activity	The installed Trojan tries to do some actions. e.g., *BACKDOOR CDK* alert.