Practical IDS alert correlation in the face of dynamic threats

Sathya Chandran Sundaramurthy, Loai Zomlot, Xinming Ou
Kansas State University, Manhattan, Kansas, USA
{sathya, lzomlot, xou}@ksu.edu

Abstract

A significant challenge in applying IDS alert correlation in today's dynamic threat environment is the labor and expertise needed in constructing the correlation model, or the knowledge base, for the correlation process. New IDS signatures capturing emerging threats are generated on a daily basis, and the attack scenarios each captured activity may be involved in are also multitude. Thus it becomes hard to build and maintain IDS alert correlation models based on a set of known scenarios. Learning IDS correlation models face the same challenge caused by the dynamism of cyber threats, compounded by the inherent difficulty in applying learning algorithms in an adversarial environment. We propose a new method for conducting alert correlation based on a simple and direct semantic model for IDS alerts. The correlation model is separate from the semantic model and can be constructed on various granularities. The semantic model only maps an alert to its potential meanings, without any reference to what types of attack scenarios the activity may be involved in. We show that such a correlation model can effectively capture attack scenarios from data sets that are not used at all in the model construction process, illustrating the power of such correlation methods in detecting novel, new attack scenarios. We rigorously evaluate our prototype on a number of publicly available data sets and a production system, and the result shows that our correlation engine can correctly capture almost all the attack scenarios in the data sets.

I. Introduction

IDS alert correlation has been studied for more than a decade and a number of correlation methodologies have been proposed. Although research has made significant progress in creating various correlation models, what one finds in practical use still remains rudimentary. Our conversation with system administrators and security analysts indicate that there is a significant gap between the desired capability of IDS alert/event correlation technologies and what the current commercial tools can provide. The question naturally arises that why more than ten years' research into IDS alert correlation has not found wide-spread use in practice.

Several attempts in applying IDS alert correlation have suffered from many of the following limitations. Most of the approaches are based on constructing a knowledge base of known attack scenarios and thus lack the ability to detect emerging and new ones. Another problem is the difficulty in updating the knowledge base. If there is a same attack with slight modification in the sequence of events, the knowledge base may not be able to detect it. A more important concern with past works on alert correlation is the lack of rigorous evaluation on a number of data sets. For an IDS to be a practical tool it has to perform consistently over a variety of data sets.

In this paper we propose a simple and practical approach to IDS alert correlation that addresses the above challenges. Our contributions are:

- Our approach is *generic* in that it is not customized to detect pre-modeled scenarios but has the ability to detect previously unseen attack scenarios. It is also *flexible* in the sense that it can handle data from a variety of sensors like NIDS, HIDS and other relevant information for intrusion analysis. Adding a new sensor is easily done by adding a new semantic mapping for the information the sensor reports. The correlation graphs can be generated with any desired level of granularity, depending on how detailed the system administrator wants to know about specific attack scenarios.
- The correlation tool we develop can report attacks, if any, in *real time*. Our correlation tool handles network traffic *continuously*, generating attack graphs in *.svg* file format deployed as a web page on a web-server. The correlation is staged in two phases wherein we do the time-consuming activities like alert grouping and summarization in the first phase and the reasoning engine that takes those processed alerts, generating attack scenarios constitues the second phase.
- We have done rigorous evaluation of the efficiency of our tool on a number of data sets that range over a wide period of time. We call it rigorous because we used the same semantic model consistently over all the data sets and the result shows that the same model works perfectly on all of them.
- The attack scenarios produced from our correlation tool can be used as input for various prioritization methods.

II. Related Work

Alert correlation in intrusion detection systems has been a topic researched for almost ten years [1, 3, 5, 6, 8, 13]. Ning, et al. [7] proposed an approach using pre and post-conditions. The concept of hyper-alerts are introduced, which consists of the attack activity, the pre and post-conditions corresponding to that attack. The mappings of raw alerts to one of these hyper-alerts are pre-generated and stored in a knowledge base. Two hyper-alerts are correlated if the post-condition of one hyper-alert contributes to the pre-condition of another one. Cuppens, et al. [3] propose a correlation model called CRIM which provides clustering, merging and correlating of alerts from multiple IDSes. The alerts are clustered based on their similarity and merged, where each group of merged alerts represents a single attack. The correlation module takes these merged alerts and constructs a number of possible attack scenarios. CRIM specifies a number of attack modules using the LAMBDA language where each module has a pre-condition that must be true for the attack to take place, the post-condition that may result if the attack

succeeds, the attack activity itself, and other information. To correlate the modules and construct the attack scenarios the authors propose two methods, direct and indirect correlation. In the explicit correlation method two modules are correlated if successful execution of one module contributes to the initiation of another. In the indirect correlation approach ontological rules are used to correlate modules that aren't directly related but through a series of events, with one module being the initial event and the other the last event. Cheung, et al. [2] proposed a model called Correlated Attack Modeling Language (CAML) that aims at developing attack patterns. The main idea is to construct a set of modules that describe specific attacks with pre-conditions to be satisfied for the attack to occur, the attack activity itself and the post-condition that may result if the attack succeeds. The post condition of one attack module may satisfy the precondition of some other attack modules in which case both these attack modules will be linked.

Most of the above previous works adopt a pre- and post-condition correlation module. A potential drawback of ascribing a pre- and postcondition to an IDS alert is that the model itself may already have assumed some specific attack patterns. The attack modules knowledge base must be frequently updated to include newer attack patterns, which has become impractical in today's rapidly changing threat environment.

Ren, et al. [11] propose the design of an online correlation system for real-time intrusion analysis. There are two components: an off-line Bayesian-based knowledge base construction tool and an online correlation and attack graph constructor. The offline component maintains tables that specify the frequency of occurrence of possible hyper-alert types and also the correlation between different hyper-alert pairs. This correlation is dynamically updated depending upon the network traffic observed over a past time window. The online component, on receiving alerts and grouping them into different hyper-alerts, uses the knowledge base from the offline component to construct the attack graph. This approach is practical since it moves to offline the work that involves maximum processing, and it is dynamic, ie., able to adjust the causality between hyper-alerts depending on the network traffic. However, the automatic knowledge base construction can only learn and detect the type of attacks that have occurred in the past and on the network where it is trained. We propose a different approach where a generic knowledge base captures an attacker's intentions and constraints, instead of specifics of attacks.

III. Correlation Model

The biggest problem with IDS is the large volume of false alarms. IDS alert correlation can potentially help in finding attack traces in all these alerts. Our correlation model is built within the context of the SnIPS tool suite [4]. A key feature of SnIPS is using qualitative uncertainty tags and a *proof strengthening* techniques to handle the uncertainty challenge in intrusion analysis [9]. The new correlation model described in this paper would allow for more sophisticated mathematical theories to be applied to handle uncertainty, as compared to the empirically developed proof strengthening approach.

Figure 1 shows the overall architecture of the SnIPS system. Our correlation model is based on a direct semantic model for

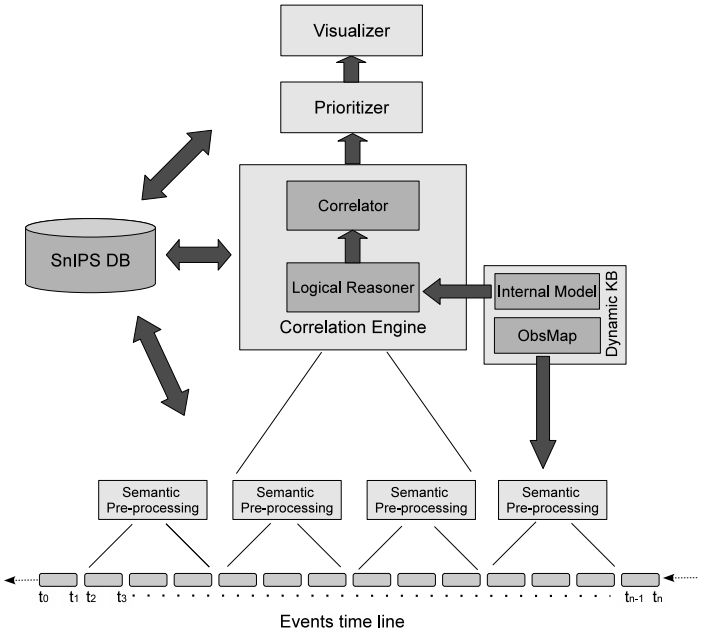


Figure 1. SnIPS System

IDS alerts, which maps an alert to its potential meanings. The semantic pre-processing layer applies this semantic model to translate raw alerts into high-level "summarized alerts" which are grouping of alerts with similar properties (source/destination address and alert type). The correlation engine then applies a two-stage process to build up an alert correlation graph. The correlation system is capable of handling dynamic knowledge base such as black-listed IP addresses which change frequently. It provides online real-time response, running continuously on streams of input events.

A. Semantic pre-processing

The pre-processing step is performed to translate and reduce the amount of information entering the reasoning engine. This process consists of the following parts.

1) Translation: SnIPS takes Snort alerts and other relevant events as input. Then it maps the observations (events) to their semantics. This process uses a set of mapping rules called *obsMap*.

Definition 1. Observations $\xrightarrow{mode}$ Internal Conditions

Definition 1 shows the formal mapping rule between observations (e.g. alerts) and internal conditions (semantics). The *mode* is used as a tag indicating the strength of the belief (e.g. *unlikely,possible, likely, or certain*). The assignment of the mode is done by interpreting the natural language description of the snort rule (IDS signature).

Example 1. *Observation mapping:*
obs(portScan(H1,H2)) $\xrightarrow{p}$ int(*probeOtherMachine*(H1,H2))

Example 1 shows an obsMap rule that maps a *portscan* alert to probing activity with the p (possible) mode.

2) Summarization: The summarization step is performed to reduce the amount of information entering the reasoning engine. We apply a data abstraction technique by grouping a set of similar "internal conditions" into a single "summarized" internal

$$\left.\begin{array}{l} int(probe(ext_1, H), c, T_1) \\ int(probe(ext_2, H), c, T_2) \\ \ldots \\ int(probe(ext_n, H), c, T_n) \end{array}\right\} int(probe(external, H), c, range(T_1, T_n))$$

Figure 2. Summarization

condition. The summarization is done on both the time stamps and IP addresses. For timestamps, if a set of internal conditions differ only by timestamp we merge them into a single summarized internal condition with a time range between the earliest and latest timestamp in the set. We also abstract over external IP address. We begin by selecting a set of internal conditions that differ only in the external source or destination IP addresses, and give a special variable, "external" as an abstraction of the external IP addresses. We do not summarize on internal IP addresses as this knowledge may be useful in the reasoning process. We maintain the mapping between the summarized internal condition and the raw internal conditions/observations in the SnIPS database, which helps us identify the low-level facts belonging to the summarized predicates. The summarized tuples are then given to the reasoning engine. The output of this module will be stored in the SnIPS database. Figure 2 illustrates the summarization process.

3) Black List IP Processor: SnIPS can digest any emerging information about the threat landscape and use them in the reasoning process. All that is needed is to provide an obsMap rule for the information. One example of such dynamic information is black listed IPs. A machine can be put into a blacklist if it has been found to be involved in malicious activities (bot activities, ssh brute-force log in attempts, etc.). Such a list can be used in two ways. The first method is to map a blacklisted IP to the predicate *compromised*, with the mode assigned by the IP's age in the list. That is, the IP address will be mapped to a higher mode if it is more recently added to the list. Over time the confidence will decrease, due to the fact that the machine could have been cleaned up. The second method is to create a snort rule that will be triggered whenever there is a communication between any local host and the black-listed IP. This alert will be mapped using obsMap rules stored at the SnIPS dynamic knowledge base (Figure 1).

Example 2. *H2 is a black-listed IP:*
obs(anyCommunication(H1,H2)) $\xrightarrow{c}$ int(*compromised*(H1))

The advantage of the second method is that it can capture all communication with a black-listed IP even if it would not trigger an alert otherwise.

B. Correlation Engine

The goal of this step is to build the scenario picture of attacks and consists of two stages: the *Reasoner* and the *Correlator*.

1) Reasoner: The goal of this reasoning process is to find all the possible semantic links among the summarized facts. It uses an *Internal Model* (see Figure 1). Definition 2 gives the formal format for reasoning rules in the internal model (called *internal rules* thereafter). The rule derives one internal condition from another, with two qualifiers: *direction of inference*, and *mode*. The direction tag has two values either *backward* or *forward*. The mode tag has been discussed before.

Definition 2. *Condition 1 → Condition 2*

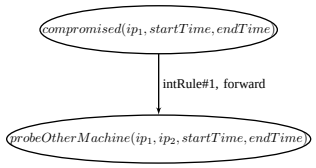


Figure 3. A proofStep example

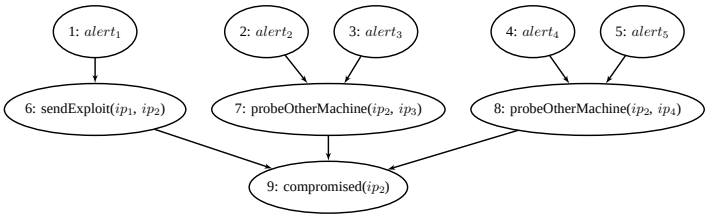


Figure 4. Correlation graph example

Example 3 illustrates one internal rule. If we know that machine H_1 is compromised, then it may perform malicious probing to another machine H_2. Conversely, if we know that a machine H_1 is performing malicious probing against another machine, we can also know that machine H_1 is compromised. So each internal rule can be used either in the forward direction, or the backward direction, in the reasoning process.

Example 3. *Internal rule:*
int(*compromised*(H1)) $\xrightarrow{f}$ *int(probeOtherMachine(H1,H2))*

The output of this stage is a collection of individual "proof steps" (*proofStep*). Figure 3 gives an example of a *proofStep*. Each node is associated with a fact like *compromised*(H_1), and a time range *(startTime,endTime)*, indicating when the fact becomes true. The direction of inference (forward or backward) is also indicated in the proof step. The time range of the conclusion can be calculated based on the time range of the antecedent and the direction of the inference. All the proofSteps will be stored in the SnIPS database.

2) Correlator: The correlator module collects all the small pieces of evidence in the form of proofSteps into a possible scenario. The input of this engine is a list of *proofSteps* from the *reasoner*, and the output is a set of correlation graphs. Each graph is an illustration of attack scenarios gathered from all the pieces of evidence.

Figure 4 is an example correlation graph, which can be viewed from top to down. "*compromised*", "*probeOtherMachine*", and "*sendExploit*" are predicates used to describe various attack hypotheses. $alert_1$ is mapped to the fact that host ip_1 sent an exploit to ip_2; both $alert_2$ and $alert_3$ are mapped to the fact that ip_2 did malicious probing to ip_3, and so on. The rationale behind this correlation graph is that after ip_1 sent an exploit to ip_2, ip_2 may be compromised (node 9). Once the attacker has compromised ip_2, he can send malicious probing to any other machine. Thus these alerts are all potentially correlated in the same underlying attack sequence. Section IV explains the algorithm that computes these correlation graphs.

C. Prioritizer

The prioritizer can further refine the result of correlation by assigning each node in the correlation graph a belief value

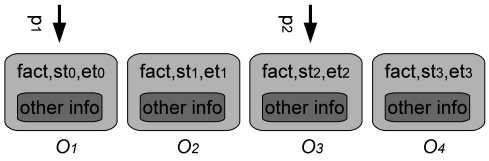


Figure 5. Correlation of different time ranged overlapping facts objects

based on an extended version of *Dempster-Shafer* evidential reasoning theory [12]. The belief values can be used to rank the correlation graph segments by the belief values of the nodes within the segments, the higher the belief value, the more likely the correlation represents a true attack. This will help the system admin to spot the most important correlation scenario. The calculation uses the mode values assigned to the observations and translates them into numeric basic probability assignment on the interpretations of the observations. Then an extended Dempster Belief Combination method is applied to calculate the belief value of each node in the graph. Detailed explanation of this module falls beyond the scope of this paper.

D. Visualizer

The final step is to introduce the output to the system admin in an easy to understand and manipulate manner. We use the *GraphViz* tool to display the correlation graphs in the *.svg* format, so that the user can use an interactive web interface to further analyze portions of a correlation graph. For example, the user can examine the raw alerts behind a summarized alert, the IDS signatures that trigger them, the payload, and other relevant information.

E. Dynamic Knowledge Base

The dynamic knowledge base is used in a number of steps described above. It includes both the obsMap relations used in the semantic pre-processing stage and the internal model used in the logical reasoning phase. The dynamism of this model comes from the fact that it can be automatically or manually updated based on the emerging threats. For example, the black-listed IP addresses change every hour. The simple obsMap relations allow for quick update of the obsMap relations for this piece of information.

F. Implementation

We use the Prolog system XSB [10] to perform the semantic mapping and logical reasoning of the input alerts. The Correlator and the Prioritizer are implemented in Java. The Visualizer is implemented using a collection of web programming languages such as PHP.

IV. Correlation Algorithm

The correlation algorithm takes the collection of proofSteps output from the *Reasoner* and builds a set of correlation scenarios in the form of graph segments. The Correlator follows the following steps:

- **Step 1:** Translate the input proofSteps into a form that can be handled by the engine. We will call this translated object O_i (Figure 5). An object contains a number of fields, including the fact associated with it, the start time, and the end time.
- **Step 2:** All translated objects O_i will be classified based on the facts associated with them. For example, all objects with the fact *compromised*(h) will be in one group and so on. Figure 5 gives an example of one group. We can assume that each fact in the figure has the form *compromised*(h). Each group of objects will be sorted ascendingly by the end time (et_i), and then by the start time (st_j).
- **Step 3:** Each group will then be correlated using time overlapping between objects. Figure 5 illustrates the correlation process. There are two sliding pointers to track the correlation process. The first pointer p_1 will start at the first object O_1. The second pointer p_2 will move to the second object. If the time range of O_2 overlaps with O_1 then the intersection of the two time ranges will be taken and stored in a variable $intTimeRange$. Pointer p_2 then moves to O_3 and takes its time range to intersect with $intTimeRange$. The process stops when $intTimeRange$ becomes empty, meaning we can no longer correlate the facts represented by the objects. A new graph node will be created for all the objects that have a non-empty time-range intersection, which will have *fact, intTimeRange* as its fields. Then p_1 will move forward until the time-range intersection for objects between p_1 and p_2 becomes non-empty.

Figure 1 and 2 show the pseudo code for this algorithm. Line 9 in Algorithm 2 includes constructing the edges of the graph, which utilizes the "other info" field in each object to connect the merged nodes with one another. Details of this part of the algorithm is omitted due to space limitation.

Algorithm 1 Correlation engine

```
1:  function CORR(ProofStepsSet)
2:      ObjectsSet ← Translate all proofSteps in ProofStepsSet
3:      for each Object(O) in ObjectSet do
4:          ObjectsGroupList ← group by fact of Object O.
5:      end for
6:      for each ObjectGroupList do
7:          sort ascendingly by the endTime of the TimeRange.
8:          Graph← CREATEGRAPH(ObjectGroupList)
9:      end for
10:     return Graph
11: end function
```

V. Experimentation

We have tested our correlation model on a number of publicly available datasets and on our departmental network as well. For the datasets the tcpdump of the network traffic is obtained and the correlation is done offline. **The construction of the reasoning model is done completely separately from the evaluation and without any knowledge about the specifics of the data sets**. This testing is to ensure that our correlation model works fine and is able to identify different types of attacks. The evaluation on the departmental network analyzes data from various sources like Snort IDS, black-list logs from computer clusters, and so on, and produces real-time attack scenario graphs in Scalable

Algorithm 2 Create graph

```
1: function CREATEGRAPH(ObjectGroupList)
2:     for each Object(O_i) in ObjectGroupList do
3:         intTimeRange ← find the intersection of timeRange with the next O_i
4:         if intTimeRange is Empty then
5:             NodesHash ← create new node with intTimeRange and O_i's fact and use O_i as the key for the hash table.
6:         end if
7:     end for
8:     for each Node in NodeHash do
9:         Graph ← build the edges from the related O_i
10:    end for
11:    return Graph
12: end function
```

Vector Graphics (svg) format which can be viewed on a web-browser. The generated svg file is hosted on a web-server so that the System Administrator can view the current security state of the system as the tool updates the file.

The Snort alert collection and correlation were carried out on a Ubuntu server running a Linux kernel version 2.6.32 with 16GB of RAM on an eight-core Intel Xeon processor of CPU speed 3.16GHz. So far we have not encountered any performance bottleneck in our algorithm.

We have tested our correlation system on the following data sets

- Lincoln Lab DARPA intrusion detection evaluation data set
- Honeynet Project
- Treasure Hunt

A. Lincoln Lab DARPA intrusion detection evaluation data set

1) Lincoln Lab Scenario (DDOS) 2.0.2 Data Set: In this attack scenario, the goal of the attacker was to break into a network on the internet through a remote buffer-overflow exploit, install software required to launch DDOS attack on the hosts inside the internal network. The attacker first breaks into the DNS server for the internal network through the remote sadmind buffer-overflow exploit and installs the *mstream* DDOS master software. He then uses the HINFO records in the captured machine to probe for machines within the internal network. Once he found the vulnerable hosts, he broke into them and installed the server software one of them. The vulnerability exploited is the well known *sadmind* vulnerability in the Solaris system. The attacker then telnets to the remotely compromised machine, starts the DDOS master and launches attack against other systems on the internal network.

We collected the tcpdump data from the Lincoln Lab website and ran Snort on it. The alerts from Snort were logged into a MySQL database. The correlation engine read the alerts from the database, did clustering and merging of the alerts, constructed an attack scenario based on the knowledge base and generated an svg file hosted on a web-server. The graph we obtained is shown in Figure 6. In this graph and the subsequent graphs the nodes of *box* shape represent Snort alerts and *oval* shaped nodes represent hypotheses. The first graph in the figure shows the

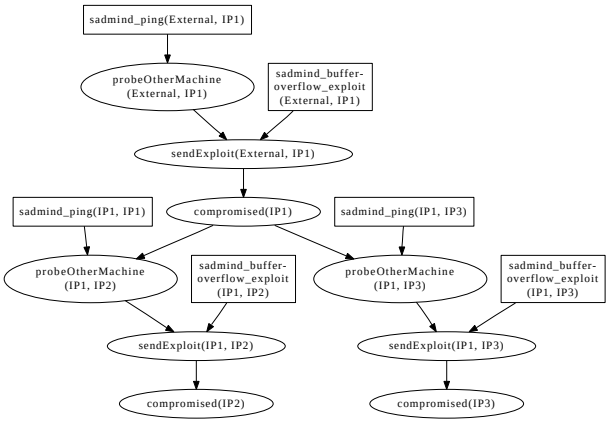


Figure 6. Lincoln Lab Scenarion (DDOS) 2.0.2 Result

graph corresponding to the probing of the attacker for services that have the sadmind vulnerability. The alerts pertaining to this activity are grouped as a single group of snort alerts and are correlated to an abstract event that says *an internal machine, in this case the DNS resolver, is getting probed by some external IP*. The System Administrator can click the snort alerts block to see the alerts, their payload and a detailed description that says how this particular type of attack can occur. All these features are implemented as hyper-links in svg. The second graph segment shows how the DNS resolver of IP address IP1 is compromised along with the Snort alerts that support this hypothesis. There are also probes emanating from this compromised machine to internal machines of IP addresses IP2 and IP3. This captured the scenario described in the truth file that after capturing the DNS resolver the attacker started probing for other machines inside the internal network for more machines that run services with sadmind vulnerability. All the Snort alerts that supported the probe are grouped as an abstract predicate *probeOtherMachine*. The third and fourth segments capture the fact that the internal hosts are compromised through sadmind buffer-overflow exploit. This abstract representation depicts the scenario very clearly so that it is easy to interpret there are malicious activities going on. Further investigation for the specifics of the activity can be done by clicking the alerts.

2) Lincoln Lab 1998 Intrusion Detection Data Set : The 1998 intrusion detection data set from Lincoln Lab has both test and training data. Only the training data's truth file was publicly available. So we used the training data, which was seven weeks long in evaluating the performance of our correlation engine. In the first week's data there was a lot of background traffic and it was meant to test whether the IDS was working fine. It just had two attacks per day. Attacks were added gradually from the second week onwards. The traffic from second to the seventh week had 100 instances of 25 attack types.

We compared our results with the truth file published on the website (http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attacks.html). Each day had specific types of attacks targetted at specific machines. The attack types included teardrop, land attack, ipsweeping, portsweeping etc., to name a few. Our correlation engine was able to correctly identify the machines that were under attack along with their attack type for each day of the data. The best part about this is that, we were able to idenitfy the different types of attacks even without encoding the specific attack patterns. Our knowledge

base, which is an abstract model, successfully captured all the specific attack types and visualiazing them as a svg file makes decision making an easy process for the user. Figure 7 shows the graph for the machine under land attack, captured by our correlation engine. Seeing this graph one can easily conclude that the machine is under land attack where one sends a packet to a machine with the same IP address and port number that causes the machine to lock itself. There are two sets of snort alerts, each identifying two different types of attacks. One group identified packets with the same source and destination IPs being sent and another group identifies smurf attack. The correlation engine identified these groups of alerts to be part of the same attack (the tcpdump has background traffic too) and built the scenario graph. There are two arrows pointing towards the node labelled "compromised(IP1)". One of them is because we have a sendExploit, for which an internal model rule says there can be a possible compromise. Applying the internal rule in the backward direction we can say that if we have a compromised host then there might have been an exploit sent to it.

This graph is just one instance of the attack type captured by our correlation model. We captured 90 of the 100 attack instances. The false negative is due to the fact that we used the latest Snort signatures that do not contain attack rules to capture attacks that were prevelant in 1998. Neverthless, we tried to obtain the older Snort rules but we couldn't obtain signatures old enough to capture those remianing 10 attacks.

B. Data Set from the Honeynet Project

This data set is from a forensics challenge organized by The Honeynet Project, an international non-profit research organization in security. The data set we obtained is from the event Scan 34 (http://old.honeynet.org/scans/scan34/). The challenge was to analyze the various log files posted on the website and to figure out what exactly happened in the honeynet. The honeynet had 3 systems named *bridge*, *bastion* and *combo*. The *bridge* machine performed routing and filtering, *bastion* was the IDS running Snort and *combo* was the victim machine that was assigned 11.11.79.67 as its IP address with many other virtual IP addresses in the vicinity of its physical address. We obtained the Snort log file and converted it into an observation file readable by our correlation engine.

We were able to confirm that the honeynet was compromised which conforms to the ground truth published in the website (http://old.honeynet.org/scans/scan34/sols/sotm34-anton.html). Some of the noise correlations include MySQL worm attack against the bridge which was not part of the honeypot. This was due to the fact that the Snort IDS was not configured to be context aware and so it generated alerts for packets that were indeed malicious but weren't targetted towards the honeypot. Figure 8 is a correlation graph for one of the attacked machines. First we have exploits being sent from an external machine to a machine inside the honeypot and the internal machine gets compromised. There are Snort alerts to support the fact that the machine is compromised. The internal machine then starts probing for vulnerable services on other machines, which is a typical attacker action. All the individual attack steps are indeed supported by some Snort alerts but they form an attack scenario only when they are correlated and that's what exactly SnIPS does.

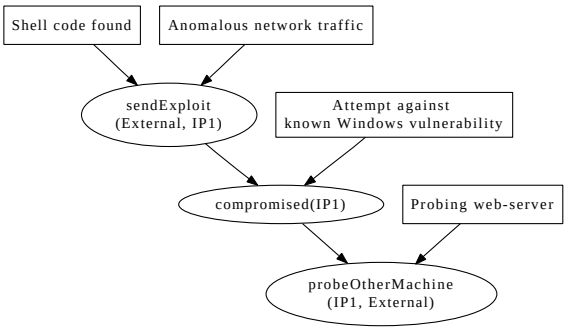


Figure 8. Honeynet Result

C. Date Set from the Treasure Hunt event - UCSB

Treasure Hunt [14] is an event organized as part of the graduate-level security course at the University of California at Santa Barbara. The class was divided into two teams: Alpha and Omega and the goal was to compete against each other in breaking into a payroll system and performing a money transaction. To avoid interference with each other there were two identical subnets provided separeately for both the teams. Each team has to perform a multi-stage attack and they had a number of tasks to do, each within a time period of 30 minutes. The first team to finish the task gets the higher points. The following is a general description of the two identical subnets. There was a webserver in a DMZ zone accessible directly outside the Local Area Network. There was a file server, a MySQL server and a transaction server. One must first compromise the web server in order to access the other servers. The task is to compromise the web server and then change the entries in a specific table in the MySQL database and then exploit the transaction service vulnerability and schedule a paycheck transfer.

We obtained the tcpdump data from the Treasure Hunt web site and ran Snort on it. Next we ran SnIPS on the alerts generated by Snort. The SnIPS system generated correlation graphs. We were not able to find the truth file for this activity but we can certainly say that every packet was an attack packet as there was no legitimte background traffic. We were able to identify the multi-stage attack in the packet capture of both Alpha and Omega teams. Figure 9 shows the correlation graph generated for the Alpha team by SnIPS. The correlation graph corresponds to the attack activities during the entire event. Figure 9(a) shows how the web server got compromised and started probing for file server, MySQL server and the transaction server. These probes are malicious and are meant for finding out vulnerable services and exploiting them. Figure 9(b)is a correlation graph for the event that the file server got compromised and Figure 9(c) for MYSQl server probing for the file server. Within Figure 9(c) we can see that the web server gets compromised and then probes for the other servers which is exactly the requirement of that assignment.

VI. Conclusion and Future work

In this paper we presented a technology to correlate intrusion detection alerts. The strength of this technology comes from is flexibility to handle the dynamism of the emerging new threats.

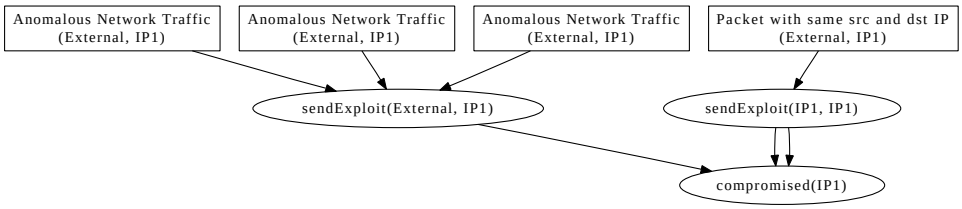
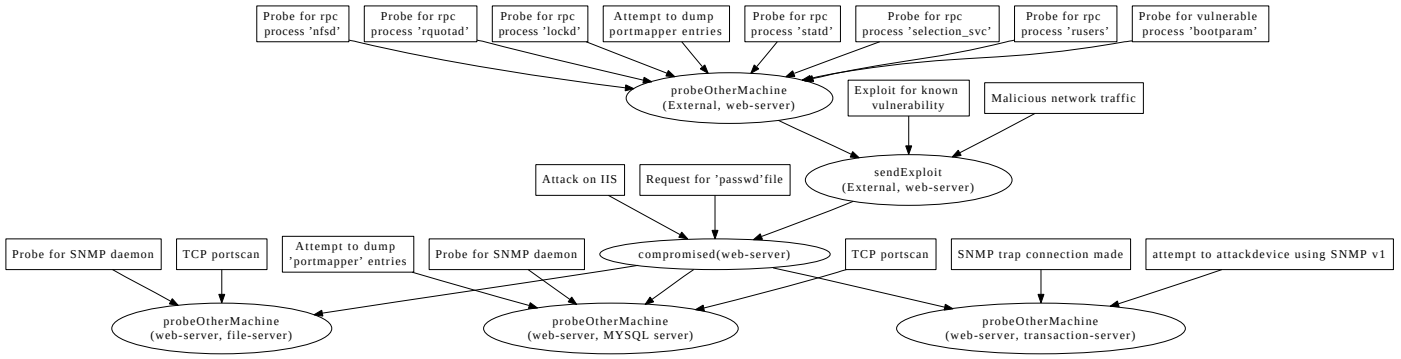


Figure 7. Lincoln Lab 1998 Result



(a) Part I

(b) Part II (c) Part III

Figure 9. Treasure Hunt Result

This tool is a real-time engine to help the system administrator to spot attack scenarios on the fly. We have performed rigorous evaluation of the correlation model, and the results indicate that such a correlation model can effectively capture a variety of attack scenarios from a number of data sets.

The correlation technology forms a solid base to build other analysis theories, making the output more precise and useful. For example the prioritizing engine currently uses and extended version of Dempster-Shafer theory. Incorporating more information sources into the system can also make the attack scenarios more accurate, and the correlation model allows for easy extension due to its straightforward semantics. We already have the ability to handle a dynamic black-list IP addresses. Our plan is to widen the window of our system by getting information from multiple sources. This can be done by adding system and server logs, and IDS systems other than Snort could all provide valuable intrusion information. Once such information can be consumed by an inference system and a more accurate correlation graph will be produced.

VII. Acknowledgment

This material is based upon work supported by U.S. National Science Foundation under grant no. 1038366 and 1018703, AFOSR under Award No. FA9550-09-1-0138, and HP Labs Innovation Research Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, AFOSR, or Hewlett-Packard Development Company, L.P.

References

[1] Steven Cheung, Ulf Lindqvist, and Martin W Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 284–292, Washington, D.C., 2003.

[2] Steven Cheung, Ulf Lindqvist, and Martin W Fongx. An online adaptive approach to alert correlationx. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[3] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 2002.

[4] Argus Lab. Snort intrusion analysis using proof strengthening (SnIPS). http://people.cis.ksu.edu/~xou/argus/software/snips/.

[5] Benjamin Morin, Hervé, and Mireille Ducassé. M2D2: A formal data model for IDS alert correlation. In *5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[6] Peng Ning, Yun Cui, Douglas Reeves, and Dingbang Xu. Tools and techniques for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2):273–318, May 2004.

[7] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer & Communications Security (CCS 2002)*, pages 245–254, 2002.

[8] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances. In *20th Annual Computer Security Applications Conference (ACSAC 2004)*, pages 350– 359, 2004.

[9] Xinming Ou, S. Raj Rajagopalan, and Sakthiyuvaraja Sakthivelmurugan. An empirical approach to modeling uncertainty in intrusion analysis. In *Annual Computer Security Applications Conference (ACSAC)*, Dec 2009.

[10] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David S. Warren, and Juliana Freire. XSB: A system for efficiently computing well-founded semantics. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

[11] H. Ren, N. Stakhanova, and A. Ghorbani. An online adaptive approach to alert correlationx. In *The Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2010.

[12] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[13] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004.

[14] G. Vigna. Teaching Network Security Through Live Exercises. In C. Irvine and H. Armstrong, editors, *Proceedings of the Third Annual World Conference on Information Security Education (WISE 3)*, pages 3–18, Monterey, CA, June 2003. Kluwer Academic Publishers.