Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances

Steven Noel, Eric Robertson, Sushil Jajodia
Center for Secure Information Systems, George Mason University
{snoel, erobert2, jajodia}@gmu.edu

Abstract

We map intrusion events to known exploits in the network attack graph, and correlate the events through the corresponding attack graph distances. From this, we construct attack scenarios, and provide scores for the degree of causal correlation between their constituent events, as well as an overall relevancy score for each scenario. While intrusion event correlation and attack scenario construction have been previously studied, this is the first treatment based on association with network attack graphs. We handle missed detections through the analysis of network vulnerability dependencies, unlike previous approaches that infer hypothetical attacks. In particular, we quantify lack of knowledge through attack graph distance. We show that low-pass signal filtering of event correlation sequences improves results in the face of erroneous detections. We also show how a correlation threshold can be applied for creating strongly correlated attack scenarios. Our model is highly efficient, with attack graphs and their exploit distances being computed offline. Online event processing requires only a database lookup and a small number of arithmetic operations, making the approach feasible for real-time applications.

1. Introduction

Since intrusion detection systems generally focus on low-level events and report them independently, network administrators are often overwhelmed by large volumes of alerts. This has motivated recent work in alarm aggregation, to reduce administrator workload and provide higher-level situational awareness. Ideally, alarm aggregates should help one distinguish coordinated, multi-step attacks from isolated events. It is also critical to know if one's network is actually vulnerable to detected attacks, and not just from the standpoint of individual machines but also in the context of the overall network and its most critical resources.

Various approaches have been proposed to correlate intrusion alarms and build attack scenarios from them. For building attack scenarios, a particularly effective form of correlation is causal correlation, which is based on analyzing dependencies among intrusion events.

One approach to causal event correlation is to apply logical rules that chain together events based on their relevant attributes. But there are several problems with rule-based approaches to event correlation. It can be difficult for complex rule systems to keep pace with online streams of events, and maintaining the rule sets needed for constructing attack scenarios from disparate events can be difficult. Also, missing events can prevent rules from assembling a proper attack scenario, and attempts at inferring hypothetical missing attacks can lead to irrelevant results.

Another approach to causal correlation is to represent relationships among events with graphs instead of logical rules. However, because this is still based on intrusion detection information only, it can potentially give irrelevant results when hypothesizing missing events. Also, because the attack scenario graphs are constructed as events occur, it may be difficult for to keep pace with online event streams.

In existing approaches, the implicit assumption is that intrusion events are caused by the execution of attacker exploits. These approaches then model intrusion events in terms of rules (preconditions/postconditions) for the implicit exploits. But the fundamental problem with these approaches is they do not include network vulnerabilities in their model, which would provide the proper context for their implied exploits. This is the source of potentially irrelevant scenarios or ambiguity for hypothesized missing events.

In this paper, we extend previous approaches to building attack scenarios by explicitly including network vulnerability/exploit relationships (i.e., the attack graph) in the model. In other words, the network attack graph is precisely the model component that adds the necessary context to the exploits implied by intrusion events. A crucial design criterion is to maintain low overhead for online event processing. Our online processing depends solely on a manageable set of pre-computed attack graph distances. To process an online intrusion event, only a distance lookup and a small number of arithmetic operations are required.

We first build a joint model of attacker exploits and network vulnerabilities. The network vulnerability model is created either manually or automatically from the output of the Nessus vulnerability scanner. From the joint exploit/vulnerability model, we then compute distances (number of steps in the shortest path) between each pair of exploits in the attack graph (for all possible network attacks). These distances provide a concise measure of exploit relatedness, which we use for subsequent online causal correlation of intrusion detection events.

As detection events occur, we map them to attack graph exploits, and look up the distances between pairs of corresponding exploits. This allows us to correlate events through attack graph information, without the online overhead of rule execution or graph building. We iteratively build event paths, with a numeric correlation score for each event. Missing events are handled in a natural way, i.e., we quantify gaps in attack scenarios through attack graph distances. Events that cannot be mapped to the attack graph initially can be considered in post-analysis and possibly merged with existing attack scenarios.

Sequences of correlation scores over event paths indicate likely attack scenarios. We apply a low-pass signal filter (the exponentially weighted moving average filter) to correlation sequences, which improves quality in the face of detection errors. We apply a threshold to filtered correlations to separate event paths into attack scenarios, i.e., only paths with sufficient correlation (sufficiently small attack graph gaps) are placed in the same attack scenario. We also compute an overall relevancy score for each resulting attack scenario, which measures the extent that it populates a path in the attack graph.

In the next section, we review related work in this area. Section 3 then describes our underlying model, and Section 4 gives details of our implementation of this model. In Section 5, we provide experimental evidence in support of our approach, and in Section 6 we summarize this work and draw conclusions.

2. Related Work

Our approach extends recent work in causal correlation of intrusion events. But rather than correlating based on dependencies among events only, we take the novel direction of including the interdependent network vulnerabilities (i.e., network attack graph) in the correlation model.

In [1], the approach to causal correlation is to define logical rules that relate generic (network independent) events through preconditions/postconditions. As events occur, the generic rules are instantiated with attributes such as time, source/destination machine, and vulnerability type, and evaluated via Prolog to chain events together. This approach does include additional implication rules for handling missed attacks. However, because it lacks knowledge of the network vulnerabilities, it is unable to narrow down hypothesized attacks to ones that are truly relevant. Also, while this approach generates rules offline (from a set of generic exploit specifications), in online mode it still needs to evaluate the rules. The approach in [1] does include merging of identical events, which is complementary to our approach. The event merging is accomplished through clustering correlation, a form of correlation that has been described by other authors, e.g., [2][3][4].

The approach in [5] is to represent relationships among events as a graph rather than through rules. Such graphs are less complex than rule systems, and indeed we apply a similar graph representation in our approach. But the approach in [5] does not correlate events with vulnerability information, as we do. It can therefore give irrelevant results when hypothesizing missing events, because events are not grounded in real network vulnerabilities. Also, the attack scenario graphs are constructed as events occur, making it more difficult to keep pace with online event streams. In contrast, we capture relationships among attack graph elements in concise distance measurements, so that no graph manipulation is done online.

Work has been done in integrating intrusion detection with vulnerabilities information, notably [6]. However, this work considers vulnerabilities in isolation, without considering the overall impact of combined vulnerabilities on a network. Also, it does not address the critical problem of building attack scenarios from individual events. There are actually 2 vendors (Tenable Network Security and Internet Security Systems) that integrate their respective intrusion detection and vulnerability scanning tools, but again this considers vulnerabilities only in isolation.

On a related research front, work has been done in automatic construction of attack graphs from network vulnerability models. Our attack graph construction is based on such prior work [7][8][9]. Other approaches to attack graph construction have been proposed, including logic-based [10][11] and graph-based [12][13][14] approaches. These have been generally effective for assessing overall network security posture or hardening networks, although not all the proposed approaches are scalable. Our attack graph representation is based on exploit dependencies rather than security state enumeration, so that we avoid combinatorial explosion. The basic representation was first described in [14], and later modified in [8][15].

3. Underlying Model

Construction of network attack graphs is based on the application of attacker exploit rules. These rules map the conditions for exploit success (preconditions) to conditions induced by the exploit (postconditions). For example, an exploit may require user privilege on the attacker machine and yield root privilege on the victim machine. An attack graph is constructed by finding the interdependencies of exploits with respect to machines on a network.

While we employ a scalable (low-order polynomial) attack graph representation, the cost of attack graph computation still prohibits online calculation per intrusion event. The attack graph needs to be fully realized before events occur. Once an alarm is raised, its event is mapped to an exploit in the attack graph. Multiple precondition/postcondition dependencies between exploits are represented with a single graph edge, meaning that the "to" exploit depends on at least one postcondition of the "from" exploit.

A typical scenario for network vulnerability analysis includes an initial attacking machine (either outside or inside the administered network) and a set of attack goal conditions (e.g., root) on one or more machines. Given that an exploit's preconditions are met, the state of the victim machine changes per the exploit's postconditions. Upon success of an exploit, the conditions of the victim machine may meet other exploits launched from that machine. Successful exploits launched from the victim machine are linked to the exploits that provide its preconditions. By executing and linking exploits in this fashion, an attack graph is formed.

For constructing attack scenarios, we do not base the attack graph on a fixed attacker/goal scenario as is typically done in network vulnerability analysis. Neither the goal nor the attacker is known when the attack graph is computed, before intrusion events are actually considered. The assumption is that attacks can come from any machine inside or outside an administered network. The attacker may have infiltrated the network through stealth attacks, or the attack may have come from an insider who abuses his granted privileges. Similarly, the attack goal is open, since it could be any adverse condition (such as denial of service, root privilege, or unauthorized data access) on any machine. In short, our model considers the full scope of possible attack paths.

Two events that fall on a connected path in an attack graph are considered correlated (at least to some extent). Clearly, events should be fully correlated if they map to adjacent exploits in the attack graph, since this is the strongest relationship possible. Conversely, events mapped to non-adjacent exploits are only partially correlated, as shown in Figure 1. In this case, we determine the degree of event correlation through graph distance between corresponding exploits.

The graph distance between a pair of exploits is the minimum length of the paths connecting them. If no such path exists, then the distance is infinite. Graph distance measures the most direct path an attacker can take between two exploits. While longer paths might be possible between exploits, the shortest path is the best assumption for event correlation, and is the most efficient to compute. The use of minimum path length does not hinder the ability to analyze longer paths, since these paths are constructed by assembling shorter paths. Using minimum path length also resolves cycles in the attack graph, which would otherwise indicate redundant attack steps. Our graph distances are unweighted, i.e., no weights are applied to graph edges between exploits.

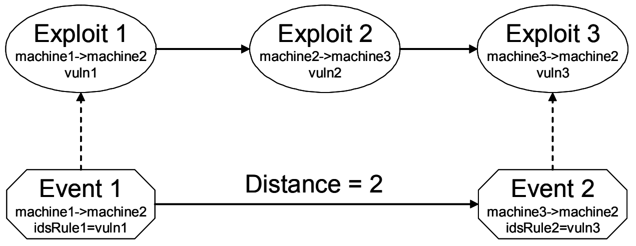

Figure 1: Partially correlated events.

The exploit distances are pre-computed once for an attack graph, and then applied continuously for a real-time stream of intrusion events. The exploit distances supply the necessary information to form event paths. An event is added to the end of a path if it maps to an exploit that has a finite distance from the exploit mapped to the last event in the path. Event time is naturally accounted for, because events are added at the ends of paths, which were constructed from prior events. If a new event is unreachable from all existing event paths (i.e., if the corresponding attack graph distances are infinite), then the event forms the beginning of a new path.

In Figure 2, suppose an initial event path exists as Event 1, corresponding to Exploit 1. A new Event 2 arrives, corresponding to Exploit 3. Since Exploit 3 is reachable from Event 1 with a graph distance of 2, Event 2 is added to the event path. A new event may trigger the creation of additional independent event paths. Continuing with our example, suppose a new Event 3 arrives, which corresponds to Exploit 4. Exploit 4 is reachable from both Exploit 1 and Exploit 3. Therefore, Event 3 can be correlated to Event 1 independently of Event 2. Since Event 2 might have nothing to do with Event 1, a new path is created as a record of another potential attack scenario. Thus we have the 2 paths Event 1 $\rightarrow$ Event 2 $\rightarrow$ Event 3 and Event 1 $\rightarrow$ Event 3. In the figure, these 2 paths are drawn with solid lines and dashed lines, respectively, in the event graph.

In our model, cycles in the event graph are unrolled. For example, in Figure 2, Exploit 4 can reach back to Exploit 1 through a distance of 3. Event 4 occurs after Event 3, and is identical to Event 1, i.e., it also maps to Exploit 1. For example, Exploit 4 might yield new privileges based on trust gained from the intervening 3 exploits. Thus two new paths are formed:

1. Event 1 → Event 2 → Event 3 → Event 4 (solid lines)
2. Event 1 → Event 3 → Event 4 (dashed lines)

These are shown with solid and dashed lines, respectively, in Figure 2.

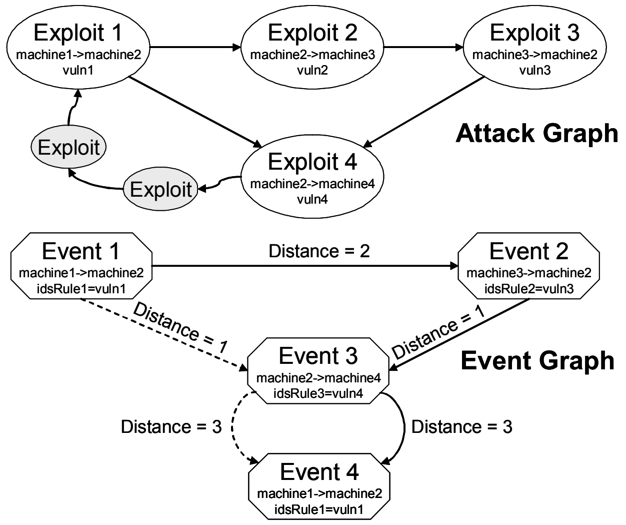

Figure 2: Creating event paths.

For the example in Figure 2, the events correspond to exploits that lie within relatively close distances to each other. But this may often not be the case. Indeed, it is reasonable to assign events whose exploits are widely separated in the attack graph to separate attack scenarios. Since event distances greater than unity represent missed detection events (according to the attack graph), it is possible that such distances sometimes occur within a set of coordinated attacks, since real attacks are sometimes missed. But when event distances become larger, larger numbers of attacks would need to be missed if they were really coming from a coordinated attack.

Thus, we apply a correlation threshold that segments event paths into highly correlated attack scenarios. In other words, a consecutive sequence of events that lies above the threshold defines an attack scenario. When individual event paths are formed from the incoming stream of events, new event paths are created when a new event is not reachable (infinite distance) from the currently existing set of event paths. In this way, event paths have an obvious beginning based on (non-) reachability. The correlation threshold provides a way to end an event path when the distance to the next event is too large, but is still finite.

The distances between events in an event path are crucial information. But because of possible false detections (positive and negative), the individual distance values are somewhat suspect. We could gain more confidence in our estimate by averaging the individual distance values. While this would capture the global trend of the event path, local trends would be lost. Also, it is convenient to invert the event distances (use their reciprocals), so that they lie in the range [0,1], with larger values representing stronger correlation. Thus the inverse distances represent similarities rather than dissimilarities.

But rather than computing the global average of inverse event distances, we compute a moving average, which has the ability to capture local trends while still providing error resiliency. An unweighted moving average defines a data window, and treats each data point in the window equally when calculating the average. However, it is reasonable to assume the most current events tend to better reflect the current security state. We therefore apply the exponentially weighted moving average, which places more emphasis on more recent events by discounting older events in an exponential manner. It is known to be identical to the discrete first-order low-pass signal filter.

Let d_k be the attack graph distance between a pair of intrusion events. Then the inverse event distance is $x_k = 1/d_k$. We then apply the exponentially weighted moving average filter to a sequence of these x_k:

$$\bar{x}_k = \alpha \bar{x}_{k-1} + (1-\alpha) x_k . \qquad (1)$$

The sequence of values of $\bar{x}_k$ is the filtered version of the original sequence of inverse event distances x_k, for some filter constant $0 \le \alpha \le 1$. The filtered inverse event distances $\bar{x}_k$ are the basic measure of event correlation in our model. For convenience, we define a correlation of unity for the first event in a path (i.e., it is fully correlated with itself), even though there is no previous event to compare it to.

The inverse intrusion event distances are filtered very efficiently through the recursive formulation in Equation (1). Computation requires no storage of past values of *x*, and only one addition and 2 multiplications per data point are required.

In the exponentially weighted moving average filter, the filter constant $0 \le \alpha \le 1$ dictates the degree of filtering. As $\alpha \to 1$, the degree of filtering is so great that individual event (inverse) distances do not even contribute to the calculation of the average. On the other extreme, as $\alpha \to 0$, virtually no filtering is performed, so that $\bar{x}_k \to x_k$. Values in the range of $0.3 \le \alpha \le 0.4$ generally work well in practice.

The filtered inverse distances in Equation (1) provide a good local measure of event correlation. In particular, they perform well for the application of the score threshold for segmenting event paths into attack scenarios. But once an attack scenario is formed, the individual filtered inverse distances do not provide an overall measure of correlation for it. We introduce another score that provides a measure of relevancy for the entire scenario, based on attack path occupancy by events.

For attack scenario s_k, $|s_k|$ is the number of events in the scenario. Next, let l_k be the cumulative distance between pairs of events in the scenario. Then the attack scenario relevancy score r_k is

$$r_k = |s_k| / l_k \ . \qquad (2)$$

Because the cumulative distance l_k is the length of the attack path that the scenario maps to, this relevance score r_k is the proportion of the attack path actually occupied by an attack scenario's intrusion events.

Our model is robust with respect to inconsistencies between events and vulnerabilities. Events that cannot be mapped to an exploit in the attack graph simply remain as isolated events. This might occur because there is no known mapping from a particular event to a vulnerability, or because a certain vulnerability was not known when constructing the attack graph. The converse is that there are certain vulnerabilities in the attack graph that have no corresponding intrusion detection signature. In this case, distances between events (in event paths) can be normalized by the expected distance between corresponding exploits in the attack graph.

4. Implementation Details

Figure 3 shows the system architecture for our implantation of the model described in the previous section. The *Attack Graph Analyzer* requires a joint model of the network and attacker exploits. *Exploit Modeling* is done through manual analysis of reported vulnerabilities and known exploits. We have researched almost 2000 Nessus vulnerabilities, from which we have modeled about 650 exploits (a significant portion of Nessus vulnerabilities are irrelevant for this kind of modeling). Because we usually model exploits at a relatively high level of abstraction (e.g., in terms of access type, privilege level, and network connection), this manual process generally proceeds quickly.

Accurate modeling depends on sufficient information about vulnerabilities and exploits. Our exploit modeling is supported by an extensive database, which includes 37,000 vulnerabilities and 7,400 exploits, taken from 24 information sources including X-Force, Bugtraq, CVE, CERT, Nessus, and Snort. *Network Modeling* can be done manually, or generated automatically from Nessus vulnerability scanner output. In the case of network models created manually, we support model specification in terms of vulnerable software components (OS, patch level, web servers, configuration files, etc.), with rules to map these to Nessus vulnerabilities.

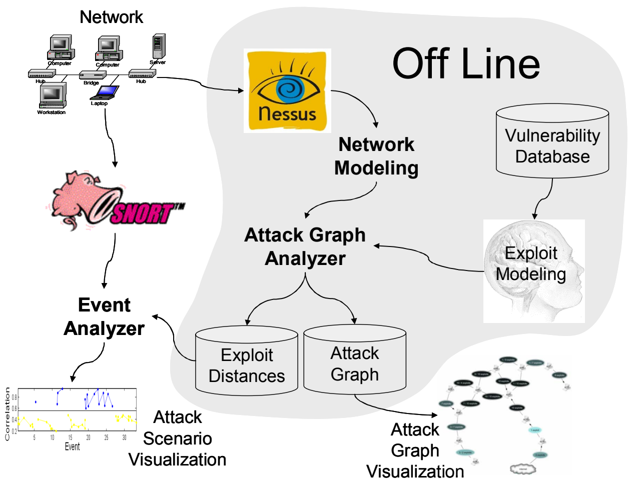

Figure 3: System architecture.

From the combined network and exploit models, we analyze attack paths and load the resulting exploit distances into an Oracle database. For efficiency, infinite distances (caused by some exploits not being reachable to others) are not recorded in the database. Rather, they are represented by their absence. In practice, a value can be chosen as an effective infinity, giving the distance computation algorithm a reasonable stopping point in declaring an exploit unreachable. Once exploit distances are calculated, they become a static image of the attack graph to be correlated with intrusion events. We can also store the attack graph itself for future offline attack graph visualization and post-analysis. All of this processing is done offline, as shown by the shaded region in Figure 3.

When Snort intrusion detection events are logged in the database, this triggers Oracle stored procedures in the *Event Analyzer* to process them. For each Snort event, we map the Snort identifier to the corresponding Nessus vulnerability identifier. In the case that a Snort identifier maps to multiple Nessus identifiers, we report all the identifiers, and conservatively select the shortest distance from among the candidate exploits for computing the correlation score. The lookup of pre-computed attack graph distances is based on source and destination IP addresses and Nessus vulnerability identifier. Note that only the distances between exploits are looked up, and no processing of the actual attack graph occurs online.

Event paths are formed in the manner described in the previous section, i.e., by adding new events to the ends of paths if the new event is reachable from the last event in the path, etc. For each path of intrusion events, the *Event Analyzer* inverts the distances between events (converts them from dissimilarities to similarities), then applies the exponentially weighted moving average filter in Equation (1) to the inverse distances. The correlation threshold is then applied, as described in the previous section, which segments event paths into highly correlated attack scenarios. In practice, proper values of correlation threshold should be based on expected rates of missed detections.

5. Experiments

In this section, we demonstrate our approach through various experiments. The first experiment focuses on the application of correlation threshold for separating event paths into highly-correlated attack scenarios and the interaction between threshold value and low-pass filter constant. To instill a deeper understanding of this, we examine a small number of attacks in greater detail, as opposed to showing statistical results for large number of attacks. In the second experiment, we show more clearly how low-pass filtering makes it easier to distinguish regions of similar attack behavior in the presence of intrusion detection errors. The third experiment is a larger-scale scenario to demonstrate overall performance.

5.1 Scenario Building via Correlation Threshold

Figure 4 is a concise summary of the attack graph for this experiment. The network model in this experiment is generated from Nessus scans of real machines. In the figure, an oval between a pair of machines represents the set of exploits between that machine pair. In most cases, there are 2 numbers for exploit sets, reflecting the fact that some exploits are in one direction (from one machine to another), and other exploits are in the opposite direction. Unidirectional sets of exploits are drawn with directional arrowheads; for sets of exploits in both directions, arrowheads are omitted. This is a variation of the aggregated attack graph representation described in [15].

In this experiment, only remote-to-root exploits are included, to make results easier to interpret. That is, each exploit has preconditions of (1) execute access on the attacking machine and (2) a connection from the attacking machine to a vulnerable service on the victim machine, and postconditions of (1) execute access and (2) superuser privilege on the victim machine. Since connections to vulnerable services exist in the initial network conditions, and each exploit directly yields superuser access on the victim machine, the shortest exploit distance between machines is always one. In interpreting these distances from the figure, the actual numbers of exploits between pairs of machines are therefore irrelevant.

The important information from Figure 4 is the attack graph distances between the 8 intrusion events, which we can determine directly from the figure. The arrow beside "Event *x*" indicates the direction (source and destination machine) of the event. So the distance from Event 1 (an exploit from machine m23 to m80) to Event 2 (an exploit from machine m80 to m52) is one, the distance from Event 2 to Event 3 is 2, etc.

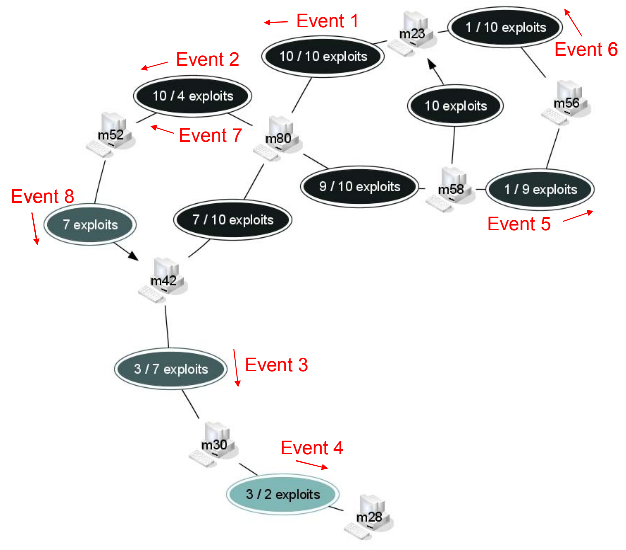

Figure 4: Aggregated attack graph.

Counting the distance from Event 4 to Event 5 is a bit more subtle. Here one must realize that "3/2 exploits" means there 3 exploits from m30 to m28, one of which is associated with Event 4. Then from the Event-4 exploit, in counting the shortest path to Event 5, there is one exploit from m28 to m30, one from m30 to m42, etc., for a total distance of 5. Figure 5 shows the full attack graph for this experiment, although it is cumbersome to use this complex graph for visually counting event distances.

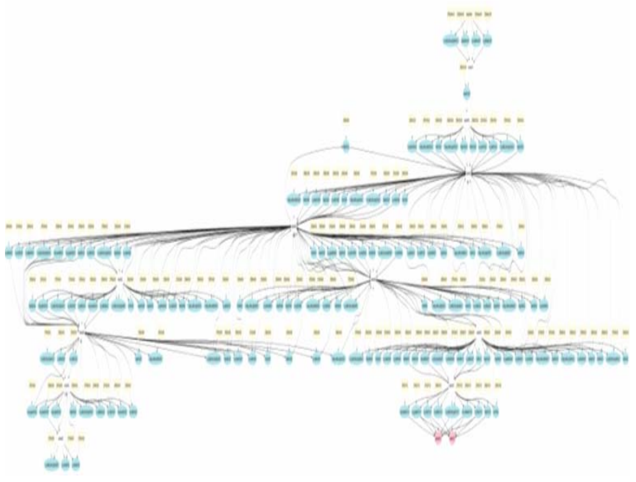

Figure 5: Non-aggregated attack graph.

Figure 6 shows the sequence of distances for the events in this experiment. Because every event is reachable from the previous event, only a single event path is generated.

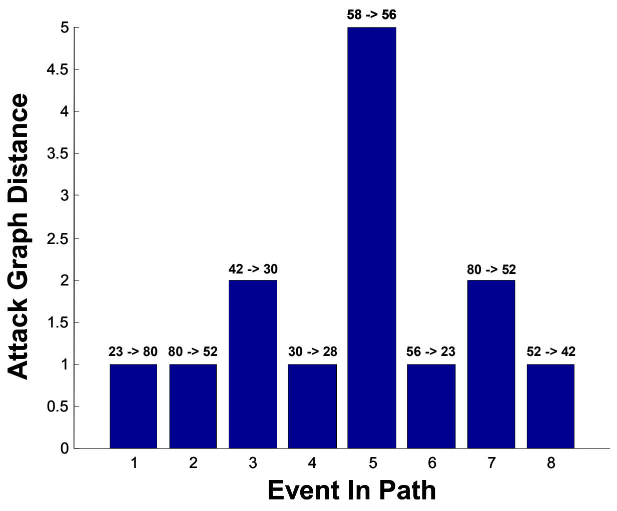

Figure 6: Attack graph distances for events.

Figure 7(a) shows the inverse of the attack graph distances from Figure 6, filtered via Equation (1), for different values of filter constant α. The vertical axis is the filtered inverse distance (i.e., the correlation score), the horizontal axis is the event number, and the axis into the page is $0.1 \le \alpha \le 0.9$. We apply a correlation threshold value of $T = 0.6$, shown as a horizontal plane.

For $\alpha = 0.1$ (front of page), very little filtering is applied, so that the filtered sequence looks very similar to the original sequence of inverse distances. In this region of α values, for the threshold T = 0.6, the event path is separated into 4 short attack scenarios:

1. Event 1 $\rightarrow$ Event 2
2. Event 4
3. Event 6
4. Event 8

The remaining events (3, 5, and 7) fall below the threshold and are considered isolated. However, the more likely scenario is that the distances=2 for Event 3 and Event 7 represent missed detections, since they are in the region of fully-correlated events. The distance=5 for Event 5 would require an unlikely high number of missed detections, so it is probably really is the start of a separate (multi-step) attack.

The problem is that, without adequate filtering, event distances are not being considered in the context of the recent history. One could lower the threshold to below T = 0.5 in this case, which would yield these most likely attack scenarios:

1. Event 1 $\rightarrow$ Event 2 $\rightarrow$ Event 3 $\rightarrow$ Event 4
2. Event 6 $\rightarrow$ Event 7 $\rightarrow$ Event 8

However, in general values below T = 0.5 are not particularly strong correlations, so this is not advisable.

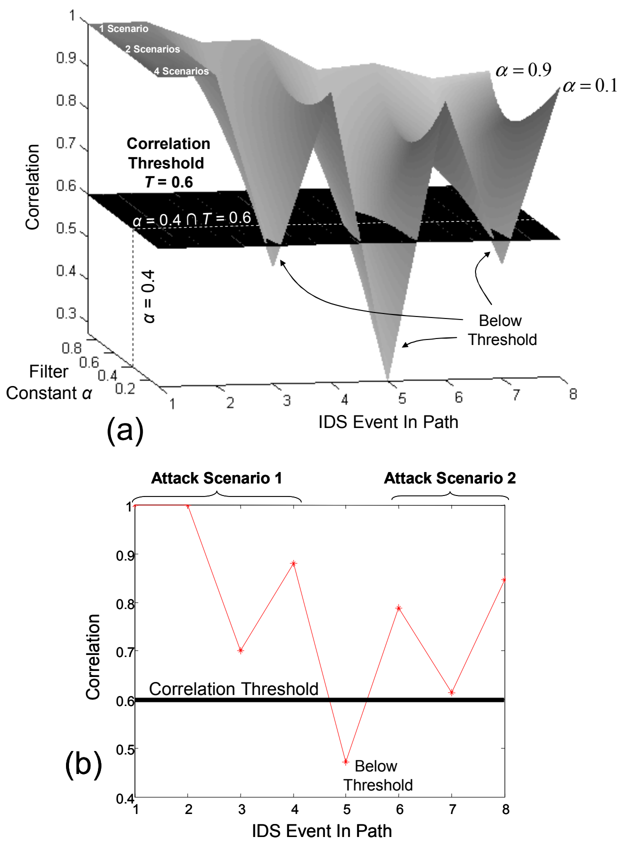

Figure 7: Distance filtering and threshold.

For larger values of α (going into the page), more filtering is applied, so that distance recent history is considered more strongly. In this case, the threshold does separate the path into the 2 most likely attack scenarios. A cross section for $\alpha = 0.4$ is shown in Figure 7(b). For overly large values of α (e.g., in the region of $\alpha = 0.9$), so much filtering is applied that the entire path is considered a single attack scenario. In other words, it misses Event 5 as the start of a new attack scenario.

5.2 Signal Filtering for Detection Errors

Next, we describe an experiment that more clearly shows the need for low-pass signal filtering for handling intrusion detection errors. In particular, this experiment demonstrates how low-pass filtering makes it easier to distinguish regions of similar attack behavior through the application of a correlation score threshold.

The results of this experiment are shown in Figure 8. Here, the horizontal axis is the event in an event path. The vertical axes of the 4 plots are (respectively) raw attack graph distance between events, global average of inverse event distance, filtered inverse event distance, and unfiltered inverse event distance.

As a ground truth, the event path is divided into 7 regions. Region 1 (Events 1-7) is an uncoordinated series of events, i.e., one in which the events are unrelated and scattered across the network, so that distances between events are relatively long. Region 2 (Event 8) is a pair of events that occur immediately together in the attack graph (i.e., event distance=1, fully correlated). Region 3 (Events 9-14) is an uncoordinated series of events. Region 4 (Events 15-17) is a series of fully correlated events, and Region 5 (Events 18-24) is an uncoordinated series of events.

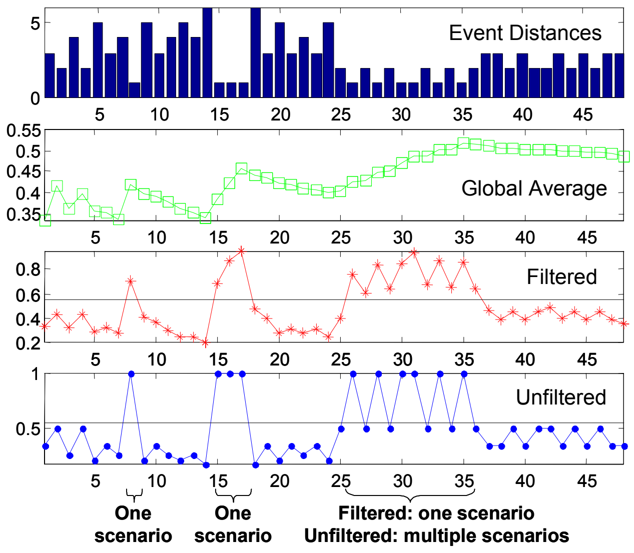

Figure 8: Filtering inverse event distances.

Regions 6 and 7 (Events 25-36 and Events 37-48, respectively) are a bit more subtle. In Region 6, the attack graph distances between events fluctuate between one and two. This represents a series of events for a single (multi-step) attack, or at least the work of a fairly consistently successful attacker. We could assume the distance=2 event pairs are from missed detections. In Region 6, the attack graph distances between events fluctuate between 2 and 3. In this case, it seems more likely to be an uncoordinated series of events that happen to occur more closely on the attack graph than say Region 1.

In Figure 8, we include global average (2nd from top in the figure) as a comparison to moving average. While each value captures the overall average inverse distance up to a given event, that does not allow us to make local decisions (e.g., through a correlation threshold) for separating the path into individual attack scenarios. Even the occurrence of fully-correlated Region 4 events cannot be distinguished through the application of a threshold.

For the unfiltered inverse distances (bottom of Figure 8), we can correctly distinguish the isolated pair of fully correlated events in Region 2, as well as the unbroken path of fully correlated events in Region 4. But there are problems for Region 6. This is the region in which fully correlated events are mixed with distance=2 events. This could be expected in a real sequence of attacks, when some of the attacks go undetected. Here, the unfiltered correlations fluctuate strongly, causing problems for setting a threshold for segmenting event paths into likely scenarios. At the threshold shown of 0.55, this region is segmented into multiple very small attack scenarios. The threshold could be lowered (to below 05), but that would cause problems for Region 7. Here, distance=2 and distance=3 event pairs are occurring. In this case, it is much less likely a coordinated attack is occurring. It would mean one or 2 attacks are repeatedly being missed, with no fully correlated events occurring. Lowering the threshold to handle Region 6 would cause Region 7 to be segmented into multiple very small scenarios.

In contrast, when the threshold is applied to the filtered version of the inverse event distances (2nd from bottom in Figure 8), this correctly forms attack scenarios corresponding to Regions 1 through 7. When filtering is applied, the distance for a new event takes into account the recent history of events, so that distances occurring after shorter distances tend to become shorter and distances occurring after longer distances tend to become longer. The degree of this effect is controlled by the filter constant α .

5.3 Performance

This experiment demonstrates overall performance for the implementation of our approach (see Section 4 for implementation details), using a large number of network attacks. In particular, we apply our implementation to a network of 9 victim machines, separated into 3 subnets, as shown in Figure 9.

In this experiment, subnet x.x.100.0 services internet traffic with a web server and an FTP server. Subnet x.x.128.0 supports administrative servers and an Oracle database server. Subnet x.x.200.0 is for administrative purposes. Traffic between subnets is filtered as shown in Figure 9. Traffic within each subnet is unfiltered, so that there is full connectivity to vulnerable services among machines in a subnet.

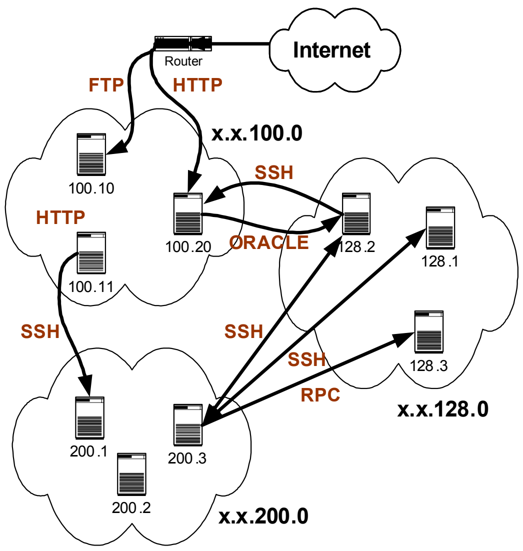

Figure 9: Network connectivity for third experiment.

The attack graph in this experiment contains 105 (machine-dependent) exploits. While there are $105^2=11025$ possible distances between 105 exploits, the exploits leading from the internet are not reachable from the remaining exploits, and such infinite distances are not recorded (using an adjacency list representation). In particular, there are 10,395 recorded exploit distances.

We then injected 10,000 intrusion events, mixed with random traffic. We included isolated events as well as multi-step attacks. Using a filter constant of $\alpha = 0.4$ and a correlation threshold of 0.55, we correctly distinguished the multi-step attacks from the isolated events.

In online mode, it takes less than 4 minutes to process 10,000 events (about 24 milliseconds per event). This is on a machine with a 2-GHz processor, 1 megabyte of main memory, and two 100-gigabyte 15,000 RPM SCSI disk drives. Neither memory nor disk traffic showed more than 30% load.

6. Summary and Conclusions

In this paper, we extend previous approaches to attack scenario building by explicitly including the network attack graph in the model. The attack graph provides the necessary context for intrusion events, and provides the graph distances upon which our correlations are based. Our online event processing depends on pre-computed attack graph distances only, and requires only a lookup and 4 arithmetic operations.

To compute attack graph distances (offline), we build a model of attacker exploits and network vulnerabilities. We can create the network vulnerability model automatically from output of the Nessus vulnerability scanner. We then compute the distance of the shortest path between each pair of exploits in the attack graph. These distances are a concise measure of exploit relatedness, which we use for subsequent online causal correlation of intrusion detection events.

From the online stream of intrusion events, we build individual event paths based on attack graph reachability. The inverse distance between each event in a path is a measure of correlation. We apply a low-pass filter to sequences of inverse distances to provide resiliency against detection errors. The application of a threshold to the filtered distances separates event paths into highly correlated attack scenarios. We also compute an overall relevancy score for each resulting attack scenario.

We demonstrate our approach through several experiments. The results show that the approach generates attack scenarios with a high degree of causal correlation. We demonstrate the effectiveness of correlation thresholding, and well as its relationship to degree of applied filtering. We demonstrate real-time performance, processing an event every 24 milliseconds.

7. Acknowledgements

The work of Sushil Jajodia was partially supported by the Air Force Research Laboratory, Rome under the contract F30602-00-2-0512, the Army Research Office under the grant DAAD19-03-1-0257, and the Army Research Laboratory under the contract W911QX-04-C-0101.

8. References

[1] F. Cuppens, A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[2] Y.-S. Wu, B. Foo, Y. Mei, S. Bagchi, "Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS," in *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.

[3] A. Valdes, K. Skinner, "Probabilistic Alert Correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Davis, California, 2001.

[4] O. Dain, R. Cunningham, "Building Scenarios from a Heterogenous Alert Stream," in *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, West Point, New York, June 2001.

[5] P. Ning, D. Xu, C. Healey, R. St. Amant, "Building Attack Scenarios through Integration of Complementary Alert Correlation Methods," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, February, 2004.

[6] B. Morin, L. Mé, H. Debar, M. Ducassé, "M2D2 : A Formal Data Model for IDS Alert Correlation," in *Proceedings of the 5th Symposium on Recent Advances in Intrusion Detection*, Zurich, Switzerland, October 2002.

[7] R. Ritchey, B. O'Berry, S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," in *Proceedings of 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002.

[8] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, "Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs," *Proceedings of 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.

[9] S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher, 2004.

[10] R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2000.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2002.

[12] L. Swiler, C. Phillips, D. Ellis, S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, June 2001.

[13] J. Dawkins, C. Campbell, J. Hale, "Modeling Network Attacks: Extending the Attack Tree Paradigm," in *Proceedings of Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Johns Hopkins University, June 2002.

[14] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS)*, Washington, DC, November 2002.

[15] S. Noel, S. Jajodia, "Managing Attack Graph Complexity through Visual Hierarchical Aggregation," in *Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, 2004.