Optimal IDS Sensor Placement And Alert Prioritization Using Attack Graphs

Steven Noel and Sushil Jajodia
Center for Secure Information Systems
George Mason University, Fairfax, Virginia

Abstract

We optimally place intrusion detection system (IDS) sensors and prioritize IDS alerts using attack graph analysis. We begin by predicting all possible ways of penetrating a network to reach critical assets. The set of all such paths through the network constitutes an attack graph, which we aggregate according to underlying network regularities, reducing the complexity of analysis. We then place IDS sensors to cover the attack graph, using the fewest number of sensors. This minimizes the cost of sensors, including effort of deploying, configuring, and maintaining them, while maintaining complete coverage of potential attack paths. The sensor-placement problem we pose is an instance of the NP-hard minimal set cover problem. We solve this problem through an efficient greedy algorithm, which works well in practice. Once sensors are deployed and alerts are raised, our predictive attack graph allows us to prioritize alerts based on attack graph distance to critical assets.

1. Introduction

A variety of challenges make it inherently difficult to secure computer networks against attack. Vulnerabilities in software design, implementation, and configuration are commonplace, and even the Internet itself lacks security as an original design goal. Once a machine is connected to a network, its security concerns become highly dependent on vulnerabilities across the network. Attackers can use vulnerable machines as stepping stones to penetrate through a network and compromise critical systems.

In traditional network defense, IDS sensors are placed at network perimeters, and configured to detect every attempt at intrusion. But if an attacker manages to avoid detection at the perimeter, and gain a toehold into the network, attack traffic on the internal network is unseen at the perimeter. Also, in today's highly distributed grid computing, network boundaries are no longer clear.

Organizations have a desire to detect malicious traffic throughout their network, but may have limited resources for IDS sensor deployment. Moreover, IDS usually report all potentially malicious traffic, without regard to the actual network configuration, vulnerabilities, and mission impact. Given large volumes of network traffic, IDS with even small error rates can overwhelm operators with false alarms. Even when true intrusions are detected, the actual mission threat is often unclear, and operators are unsure as to what actions they should take.

By knowing the paths of vulnerability through our networks, we can reduce the impact of attacks. Traditional tools for network vulnerability assessment simply scan individual machines on a network and report their known vulnerabilities. They give no clues as to how attackers might exploit combinations of vulnerabilities among multiple hosts to advance an attack on a network. It remains a labor-intensive and error-prone exercise for "connecting the dots" to predict vulnerability paths, and the number of possible vulnerability combinations to consider can be overwhelming.

To address these weaknesses, we focus on protecting the network assets that are mission-critical. We model the network configuration, including topology, connectivity limiting devices such as firewalls, vulnerable services, etc. We then match the network configuration to known attacker exploits, simulating attack penetration through the network and predicting attack paths leading to compromise of mission-critical assets. This approach to network attack survivability is called Topological Vulnerability Analysis (TVA) [1][2].

The resulting set of all possible attack paths (organized as an *attack graph*) is a predictive attack roadmap. The TVA attack graph assesses the true vulnerability of critical network resources, and automates the traditionally labor-intensive analysis process. TVA also encourages easy “what-if” analyses of candidate network configuration changes, and provides optimal network-hardening recommendations that require minimal changes to the network.

Even after protective measures have been applied across the network, some residual vulnerability usually remains. In such cases, TVA attack graphs can reduce the impact of attacks. The attack graph guides the placement IDS sensors across the network to cover known paths of vulnerability. In this way, all potentially malicious activity on critical paths is monitored. Conversely, no sensors are needed for monitoring traffic that does not lie on critical paths, helping to reduce costs and operator overload. In particular, our approach places sensors to cover all attack paths to critical assets, using the fewest number of deployed sensors.

Further, through the predictive power of TVA attack graphs, we prioritize IDS alerts based on the level of threat they represent to critical assets. For example, we can give lower priority to alerts that lie outside critical attack paths. Particularly severe threats are those seen as coordinated steps as an attacker incrementally advances through the network, especially if only a short distance from mission-critical assets. The attack graph also provides the context needed for responding to an attack. When an operator has strong evidence (e.g., multiple coordinated steps) of an intrusion, and knows the next network vulnerabilities the attacker could exploit next, he has confidence in taking the appropriate (and highly focused) actions for preventing further penetration.

In the next section, we review related work in attack graph analysis, IDS alert correlation, and sensor placement. Section 3 gives an overview of our system for discovering network attack paths, covering the paths with IDS sensors, and correlating and prioritizing the resulting alerts. Section 4 provides a technical description of our method for generating attack graphs and aggregating them over multiple levels. In Section 5, we describe our method for optimal placement of IDS sensors, which covers all network attack paths using the minimum number of sensors. Section 5 also describes how we leverage the known network attack paths to correlate and prioritize the resulting alerts. Finally, in Section 6 we summarize these results.

2. Related Work

Early work in automated construction of attack graphs applied symbolic model checking tools [3][4]. But model checkers have significant scalability problems, a consequence of the exponential complexity of the general state space they consider. Early graph-based approaches for analyzing attack combinations suffered similar problems with state-space explosion [5][6].

Subsequently, scalable attack graph models have emerged [1][7]. Rather than explicitly enumerating paths through state space, these models represent dependencies between state transitions (attacker exploits), eliminating path redundancies and succinctly capturing all state information. These models scale quadratically with the number of network hosts (ignoring man-in-the-middle attacks, which raise complexity to n^3). By representing fully-connected portions of the attack graph (e.g., within a subnet) more efficiently, scalability is improved to linear within each defined portion [8]. Another approach is to reduce graph complexity based on regularities in host configurations [9]. Efficient rule-based approaches for generating attack graphs have also been demonstrated [10][11][12][13].

There have also been advances that help alleviate the information overload of attack graph analysis. The approach in [8] includes highly interactive visualizations of clustered attack graphs, with high-level overviews and detail drilldown, over multiple levels of detail. A different paradigm is applied in [14], in which complex attack graphs are shown as adjacency matrices, simplified through matrix clustering techniques. In [15], a number of complementary attack graph views, including hierarchically clustered graphs, clustered matrices, and user-constrained graphs are coordinated for interactive visualization.

Once attack graphs are generated, they can be further analyzed. In [16], attack graphs are used to find optimal network configurations that prevent a given attack scenario while requiring the fewest changes to the network. Attack graphs have also been used for computing metrics of overall network security [17][18]. In [19], attack graph states are ranked for prioritizing security measures, and a rank-based security metric is defined. Overall, advances in scalability, visualization, and post analysis help make attack graph applications feasible for realistic sized networks. A more detailed review of various developments in attack graph research (as of 2005) is given in [20].

Beyond analysis of vulnerability paths, attack graphs have been applied to IDS alert correlation [21][22][23]. In fact, through our approach, attack graphs are most powerful when *predicted* paths (based on vulnerabilities) are matched with *actual* detected attacks. This helps eliminate false positives, enables prediction of missing alerts, makes alert correlation faster, and provides the context for attack response. In [24], IDS alerts are correlated into multi-step attack scenarios, using attack graph distances based on known network vulnerabilities.

Despite these advances, a key step has been overlooked. In particular, previous work does not address the placement of IDS sensors within the network infrastructure to cover known vulnerability paths. When IDS sensor placement is addressed in the literature, it is usually in the context of general architectures for distributed intrusion detection, such as [25]. One paper has applied network attack modeling (based on logic programming) for placing IDS sensors, for the limited case of Internet Protocol (IP) spoofing attacks. [26].

In [27], a model checker is used to find a minimal coverage of attack paths, using IDS or other protection measures. This approach provides a weak kind of optimality, giving the minimum set of measures that block the attacker from the end goal (the unsafe state of the model checker), assuming that *each* such measure is successful. However, this is not a safe assumption, given the high likelihood of missed IDS detections. In other words, with such minimum coverage, if only one attack is missed, the remaining uncovered paths may readily allow network penetration to critical network assets. While such minimal coverage may be appropriate for assured hardening measures such as software patches and firewall rules, it is clearly insufficient for IDS deployment. In contrast, we cover all possible attack paths (not just a minimum subset), using a minimum number of sensors while maintaining polynomial complexity.

Further, the approach in [27] does not identify how a minimum set of attack paths actually map to IDS sensor deployment in the network infrastructure. For example, an attack from host A to host B may pass through multiple network devices. But given all possible paths and network devices, on which physical device(s) should we deploy sensors? This is precisely the problem that we address.

In particular, we assign IDS sensors to network devices so that they cover all known paths of vulnerability through the network. If desired, we can focus these paths based on known threat sources and critical network assets. Our sensor placement is optimal, in that only a minimum number of sensors are needed. Once sensors are placed, we use our predictive attack graph to prioritize the resulting IDS alerts according to distance from critical assets. Our approach was first proposed at the Cyberspace Research Workshop [28], organized by the newly formed Air Force Cyber Command.

3. System Overview

Because attackers can exploit vulnerabilities as stepping stones to new vantage points, considering network components and vulnerabilities in isolation is clearly insufficient. We have implemented a comprehensive TVA tool that discovers multi-step attacks, modeling network penetration as real attackers might do. This is a custom tool, written in the Java programming language, with full-featured user interface and attack graph visualization capabilities. This TVA tool computes an attack graph showing all possible paths through a network. The approach that we propose here is to place IDS sensors to cover these predicted TVA paths, and to use this predictive context for prioritizing IDS alerts.

In the TVA approach, a network is scanned to catalog hosts, their operating systems, application programs, and vulnerable network services. We also capture network connectivity, including the effects of connectivity-limiting devices such as firewalls and router access control lists (ACLs). With the resulting network configuration, a database of modeled attacker exploits, and a specification of threat origin and critical network assets, we compute the attack graph comprising all known attacks through the network.

In particular, from network scans our TVA tool builds a model of the network configuration. This configuration is then subjected to simulated attacks from our TVA exploit database. Exploits are modeled in terms of preconditions and postconditions. When all preconditions for an exploit are met (e.g., from the initial network state), the exploit is successful, and its postconditions are induced. These postconditions in turn provide potential preconditions for other exploits. The resulting set of exploits, joined by their precondition/postcondition dependencies, forms the attack graph, predicting all possible attacks through the network. We integrate with popular network scanning tools (e.g., Nessus [29], Retina [30], and FoundScan [31]) to automate the network model building process. We also continually monitor sources of reported vulnerabilities, keeping the TVA exploit database current with respect to emerging threats.

TVA attack graphs can follow pre-defined attack scenarios, e.g., based on assumed threat sources (attack starting points) or critical assets to be protected (attack ending points). We can then constrain the attack graph with respect to these starting and/or ending points. This allows an organization to focus on realistic threat sources, while insuring the safety of critical assets. Algorithmically, these constraints are applied in two passes. The forward pass traverses the graph in a forward direction from the starting point(s), and the backward pass traverses the graph in a backward direction from the ending point(s). These passes can be applied independently, to constrain the graph in one direction or the other, or combined as a joint constraint.

In their low-level form, TVA attack graphs for realistic sized networks can be large and complex. Our approach aggregates attack graphs at various levels of detail, e.g., host, subnet, etc. We then apply analysis to the appropriate level of graph abstraction, to help keep complexity manageable. In fact, we aggregate elements of the network model in advance, so that attack graph computations are more efficient. Our aggregated structures retain all underlying low-level information, so that no information is lost compared to the full low-level attack graph. This information is also available for interactive drilldown in attack graph visualization.

Once we create the TVA attack graph predicting all possible paths through the network, we can formulate optimal network defenses. Of course, the first step is to reduce risk by hardening the network in advance of attack [16]. Still, given requirements for mission-critical services, availability of patches, etc., some residual vulnerability paths often remain on a network. The next step is to deploy IDS sensors to monitor traffic and detect potentially malicious activity along these paths.

In fact, our proposed placement of IDS sensors is optimal, in the sense that a network's attack graph is entirely covered, using the fewest required number of sensors. We argue that this sensor placement problem is an instance of the classical set cover problem [32], which is NP-hard. To solve this problem, we apply a polynomial-time greedy heuristic that is known to give good solutions in practice.

Once sensors are deployed and the IDS starts generating alerts, our attack graphs provide the necessary context for correlating and prioritizing those alerts. For example, if two alerts lie in a sequence along the attack graph, there is strong potential that these are multiple steps along a single attack, and should be taken very seriously. Further, we can predict the minimum number of future steps before the attacker reaches a given critical network asset, and can prioritize the alert accordingly. Based on our knowledge of possible attack paths, we can formulate optimal responses for stopping any further progress by the attacker.

4. Attack Graph Generation and Aggregation

Figure 1 shows a small network, which we have implemented in a laboratory testbed for demonstrating the automated generation of attack graphs via our TVA tool. In this network, the purpose of the firewall is to protect the internal network from outside attack. It is configured to allow only hypertext transfer protocol (HTTP) traffic to the internal web server, and all other traffic initiated from the outside is blocked. The web server is running a vulnerable version of Microsoft Internet Information Server (IIS), which is reachable from the outside through the firewall. The mail server has vulnerable software deployed as well, although the firewall protects it from direct attack from the outside. The question we pose is whether there are paths that allow the outside attacker to compromise the mail server.

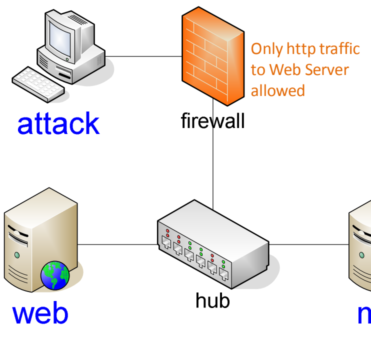

Figure 1. Small testbed network for demonstrating attack graph analysis.

To capture the network configuration for Figure 1, we use the output of the open-source Nessus vulnerability scanner. First, we scan from the outside *through* the firewall, targeting the internal network. We then scan the internal network *behind* the firewall, to see the attacker's options once he gains entry to the internal network. When then merge the resulting scan results into an overall TVA network model. This model then serves as the initial conditions for a TVA attack simulation, in which we apply a database of simulated exploits derived from Nessus vulnerabilities.

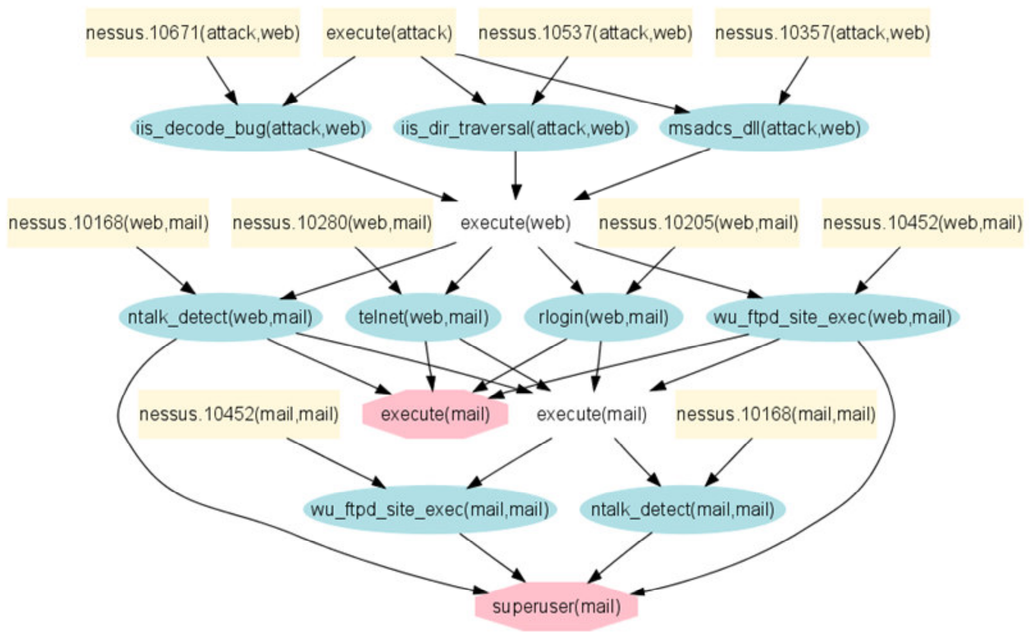

Figure 2. Attack graph for testbed network in Figure 1.

The TVA attack simulation begins on the outside machine, and ends with the compromise of the mail server. Figure 2 shows the resulting attack graph. Yellow boxes are initial network conditions, and blue ovals are attacker exploits. Here, a condition of the form *nessus.xxxxx(from, to)* represents the fact that the *from* machine can connect to a service on the *to* machine, and that this service has a particular *xxxxx* vulnerability detected by Nessus. So for example, initial condition *nessus.10671(attack, web)* means that the web server has Nessus vulnerability number 10671 (IIS Remote Command Execution) [33] (also identified as CVE-2001-0333 and CVE-2001-0507 under MITRE's Common Vulnerabilities and Exposures [34]), and that the attack machine can connect to that vulnerable service on the web server.

In Figure 2, network conditions of the form *execute(machine)* represent the attacker's ability to execute arbitrary code on a particular machine. The attacker can initially execute code on his own machine, as indicated by the *execute(attack)* in a yellow box. A condition such as *execute(web)* is induced as a postcondition of one or more exploits (in this case, by 3 different exploits), so it does not appear in a yellow box (i.e., not an initial condition). Conditions defined as overall attack goals (in this case, executing code with superuser privilege level on the mail server) are shown in red octagons.

So in Figure 2, the *iis_decode_bug(attack, web)* exploit requires 2 preconditions to be met, i.e., *execute(attack)* and *nessus.10671(attack, web)*. Since these are part of the initial network conditions, this exploit is successful, and yields the postcondition *execute(web)*. i.e., the attacker can now execute code on the web server. Two other exploits (*iis_dir_traversal* and *msadcs_dll*) are also possible from the attack machine against the web server. Once the attacker can execute code on the web server, four subsequent exploits are possible, each from the web server to the mail server. Two of those (*ntalk_detect* and *wu_ftpd_site_exec*) give the ability to execute code as superuser on the mail server, i.e., the goal has been reached. The other two (*telnet* and *rlogin*) give the ability to execute code, but without superuser privilege level. Then two subsequent exploits (*wu_ftpd_site_exec* and *ntalk_detect*) elevate attacker privilege to superuser on the mail server.

The attack graph allows us to reason about network defense strategies. For example, in Figure 2, removing Nessus vulnerabilities 10671, 10537, and 10357 on the web server would stop the attack. Or, we can conclude that fixing mail server vulnerabilities 10280 and 10205 are essentially irrelevant hardening options, i.e., 10452 and 10168 would need to be fixed anyway, and together will successfully prevent the attack (assuming it is sufficient to block the attacker from gaining superuser privilege).

Of course in practice, network attack graphs are usually much more complex than Figure 2. For example, Figure 3 is an attack graph generated by our TVA tool for an operational network of only 17 machines, across 4 subnets, with between 2 and 6 exploitable vulnerabilities per machine. Despite the complex relationships in this graph, you can still see dependency patterns among exploits. There are densely connected parts of the graph, connected by relatively sparse sets of edges. These patterns are in fact a consequence of regularities in the network configuration, of which we can take advantage for summarizing attack patterns.

These regularities are a direct reflection of how the network is organized, and are a natural choice for aggregating the attack graph into multiple levels of abstraction. This is illustrated in Figure 4. Here, Figure 4(a) is the original attack graph, showing all details. In Figure 4(b), the graph has been aggregated to the level of machines and sets of exploits between them. For example, the circled region contains 4 machines, with exploit sets between each pair of them. In other words, all the network conditions for a particular machine in Figure 4(a) are collapsed to a single "machine" vertex in Figure 4(b), and all the exploits between a particular pair of machines (in each direction if applicable) are collapsed to a single machine-to-machine exploit set in Figure 4(b). Figure 4(b) thus represents a summary of Figure 4(a), providing more of an overview of the attack.

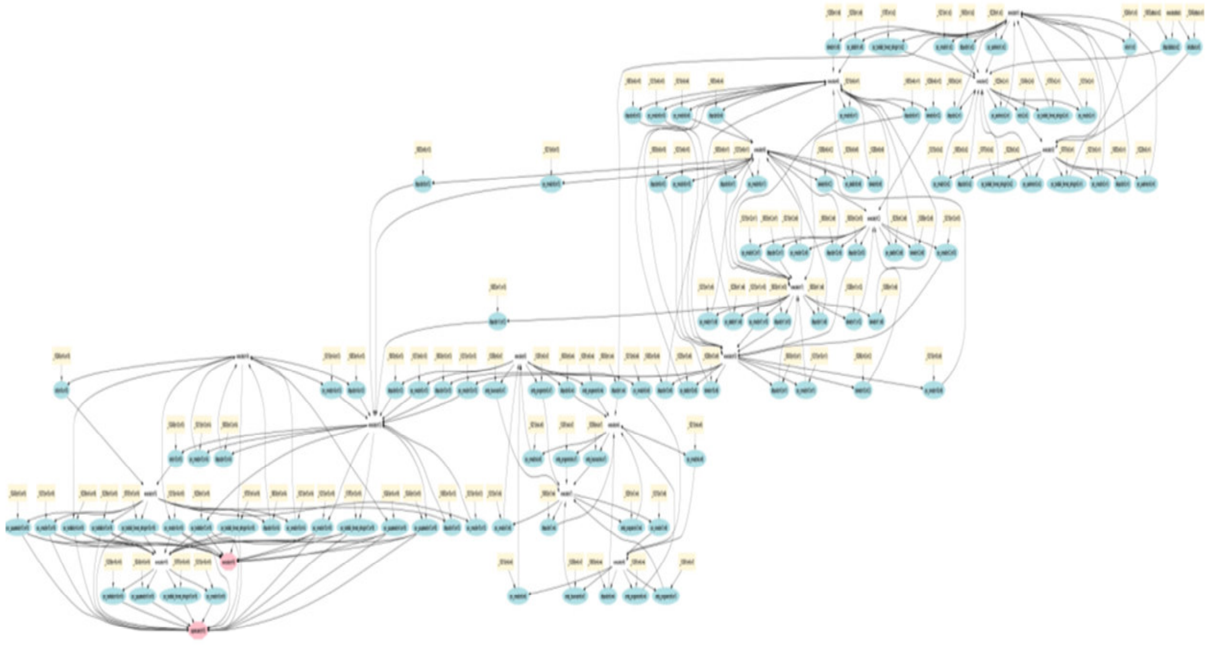
Figure 3. More complex attack graph for 17-machine operational network.

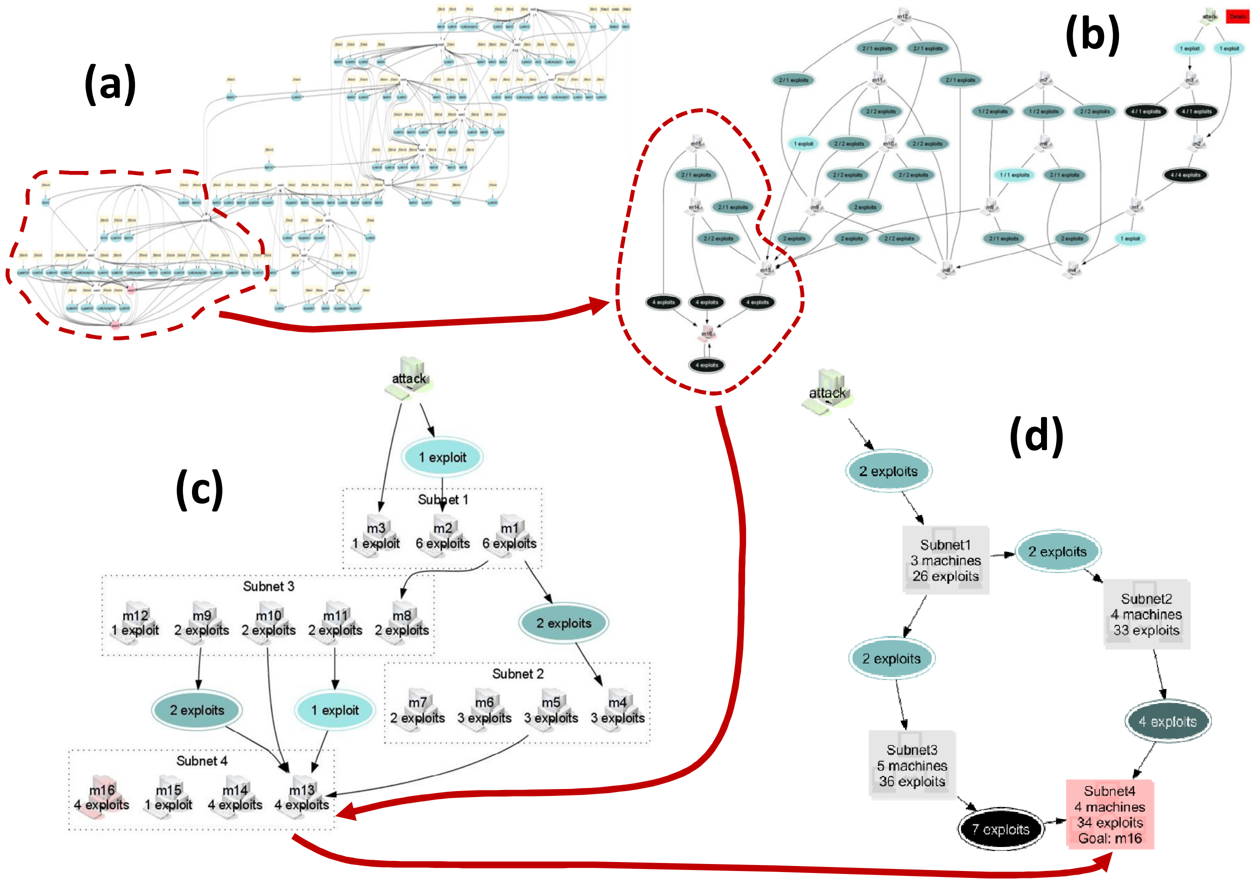

Figure 4. Aggregation of complex attack graph over multiple levels of detail.

In Figure 4(b), the circled portion of the attack graph is fully connected, i.e., this sub-graph forms a clique. This is because these machines are in the same subnet (broadcast domain), so that they have unrestricted access to one another's vulnerable services. We can incorporate this knowledge of the network structure when building the input network model. Within such a fully connected sub-graph, it is sufficient to represent only those exploits to which a machine is vulnerable, since all machines in that sub-graph can exploit those vulnerabilities. We call such a set of machines a *protection domain* [8], which forms a natural level of aggregation, as shown in Figure 4(c). Using this representation, graph size scales linearly within a protection domain (and remains quadratic across domains).

To reduce analysis complexity even further, we can aggregate machines in a protection domain to a single graph vertex, as shown in Figure 4(d). Here, we also aggregate the machine exploit sets to a single set between each pair of domains. Complexity still scales quadratically, but now as a function of the number of protection domains rather than the number of machines, thus greatly reducing complexity. With this high-level view of the attack graph, we can very efficiently reason about network defense strategies. For example, from Figure 4 (d), we immediately conclude that preventing the 2 exploits into Subnet 1 will prevent the attack. Or, if that is not possible, a second choice is to prevent the 2 exploits from Subnet 1 to Subnet 2, and the 2 exploits from Subnet 1 to Subnet 3. Other options are possible, though they involve fixing a greater number of vulnerabilities, and allow deeper penetration by the attacker.

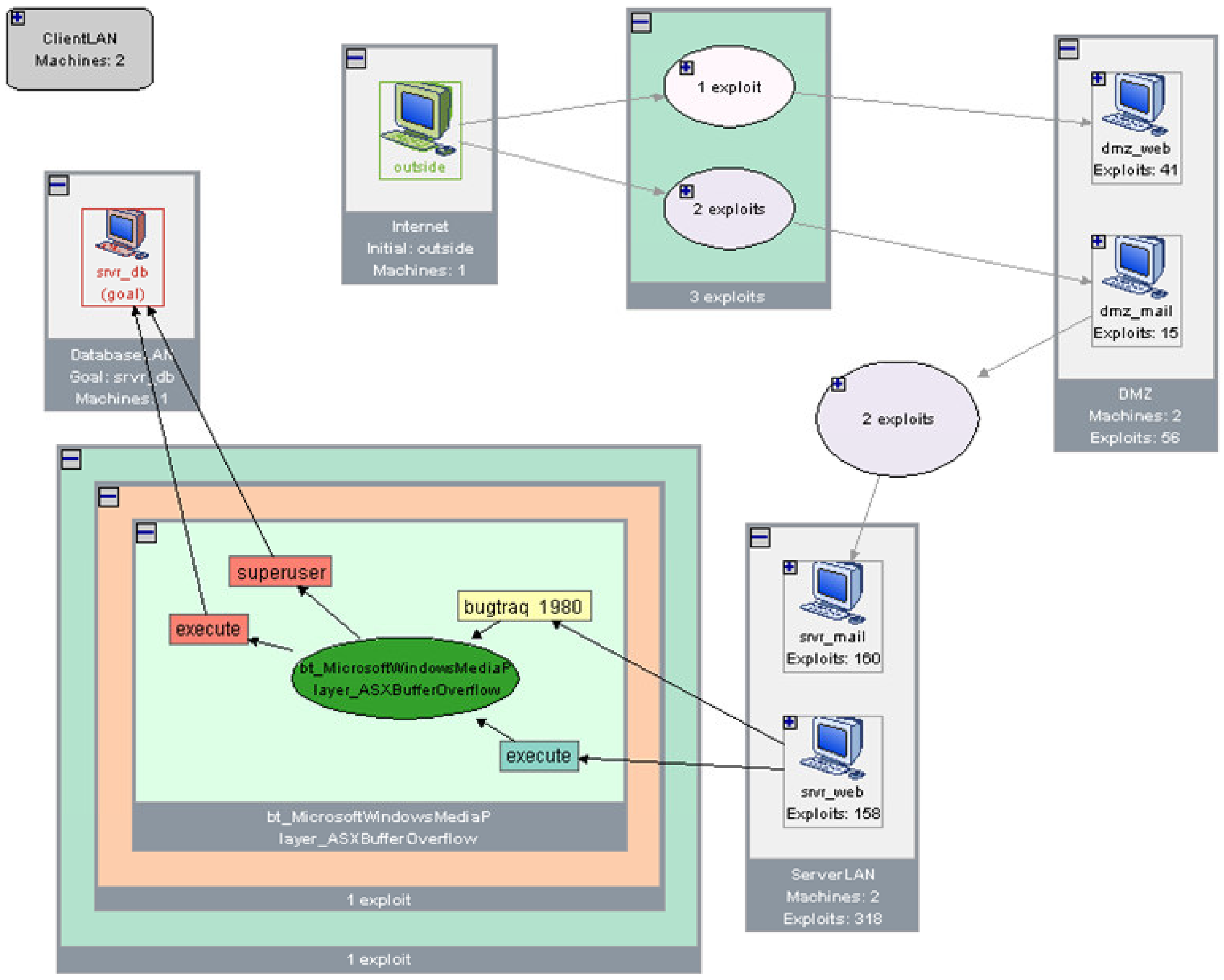

Figure 5. TVA tool attack graph visualization for 8-machine testbed network.

No information is lost in our aggregation of the network model and attack graph. Indeed, it is entirely reversible, so that all the details of the low-level attack graph are available. This is illustrated in Figure 5, which demonstrates more advanced visualization capabilities of our TVA tool, for an 8-machine testbed network. Here, a variety of levels of detail are shown in a single view of the attack graph. With the interactive visualization capabilities provided by our TVA tool, the analyst can start with a high-level overview of the attack, and drill down as needed for particular attack details.

As our TVA tool shows in Figure 5, there are exploits from the outside, to the DMZ web server (one exploit) and to the DMZ mail server (2 exploits). Once inside the DMZ, the attacker can exploit the web server in 41 different ways, and exploit the mail server in 15 different ways. Once the DMZ mail server is compromised, the attack can proceed from there to the mail server in the Server LAN (via 2 different exploits). Inside the Server LAN, the mail server can be compromised 160 different ways, and the web server can be compromised 158 ways. Once the Server LAN web server is compromised, the attacker can launch a single exploit against the database server (in the Database LAN), which is the defined goal of the attack. The visualization drilldown shows the full details (preconditions and postconditions) for this exploit.

5. *IDS Sensor Placement and Alert Prioritization*

At this point in the network defense process, our TVA tool has captured the network configuration, used it to predict all possible paths of vulnerability through the network, and applied hardening measures to help reduce known paths. But because of real-world mission and operational constraints, we are unlikely to eliminate all paths. Our next line of defense is to rely on IDS.

Now, given our knowledge of the network configuration and residual paths of vulnerability, where should we place IDS sensors so as to monitor all these paths? Moreover, what placement will cover all critical paths with the fewest number of sensors, to minimize our deployment costs? The residual attack graph defines the sources and destinations of traffic to be monitored. Then, through analysis of network topology, we identify sensor locations that cover the critical paths.

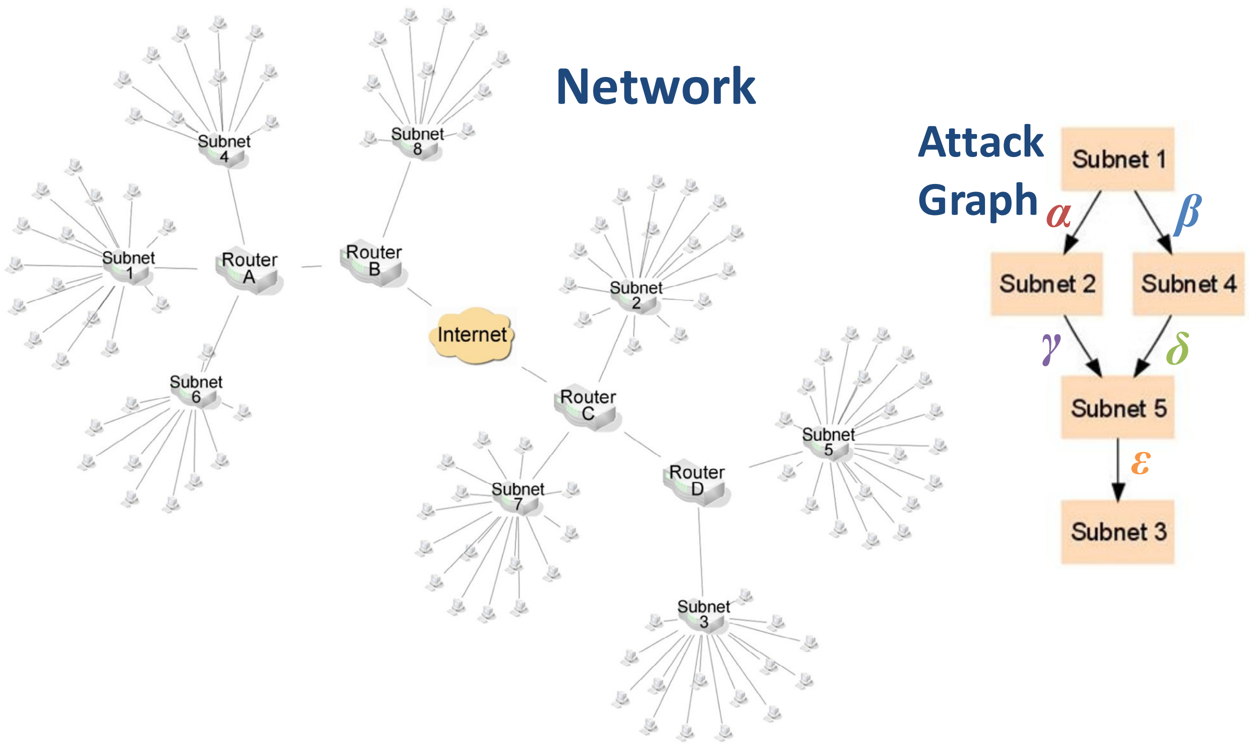

Figure 6. Testbed (hybrid real and simulated) network and its high-level attack graph.

Consider the testbed network in Figure 6, which we implement through a combination of real and replicated machine scans, and simulated network connectivity and firewall effects. There are 8 subnets, with 10-20 hosts in each subnet, and routers (and the internet backbone) providing connectivity among the subnets. There are vulnerabilities on many of the network hosts. Though not shown explicitly, the firewalls limit connectivity and help protect the network. Still, vulnerabilities remain on the network, and many are stepping stones, giving new vantage points for further penetration.

The right side of Figure 6 is a high-level view of attacks through this network, based on TVA tool results. We assume that Subnet 3 contains critical network assets to be protected. The attack graph shows all possible paths leading to Subnet 3, at the subnet-to-subnet (protection domain) level. Here, an edge means there is at least one exploit between given protection domains. While other paths may exist through this network, only the ones shown are relevant to the protection of Subnet 3. So it is precisely these paths that need to be monitored by the IDS.

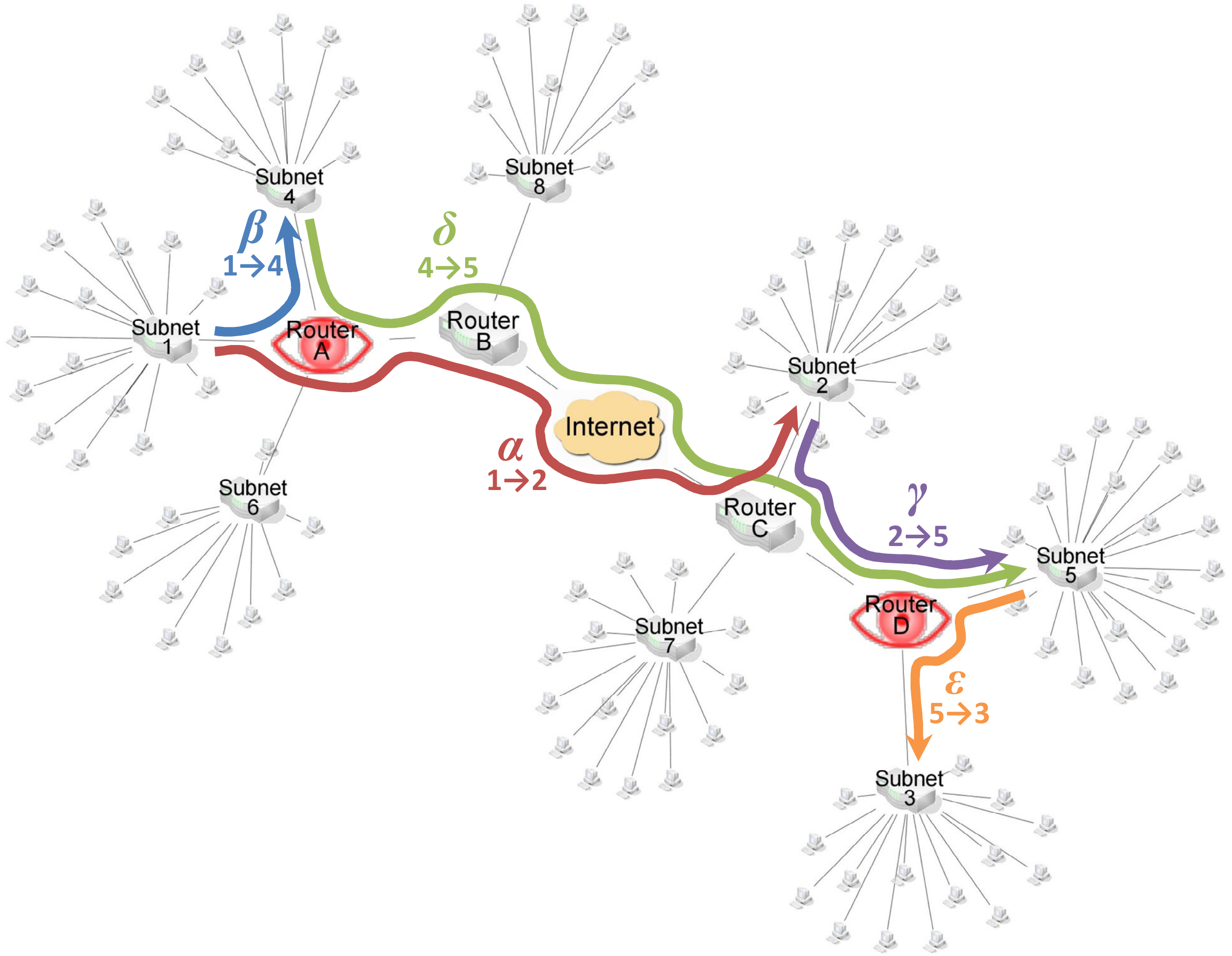

Figure 7. Optimal sensor placement for testbed network in Figure 6.

Now, given our knowledge of critical paths through the network, we can analyze the network topology for placing sensors to cover all paths. To minimize costs, we seek to cover all critical paths (the attack graph) using the least number of sensors. As we have defined it, this optimal sensor placement is an instance of the classical *set cover* problem [32]. In set cover, we are given certain sets of elements, and they may have elements in common. The problem is to choose a minimum number of those sets, so that they collectively contain all the elements. In this case, the elements are the edges (between protection domains) of the attack graph, and the sets are IDS sensors deployed on particular network devices. Each IDS monitors a given set of edges, i.e., can see the traffic between the given attacker/victim machines.

Consider Figure 7, which builds from the testbed network in Figure 6. Here, through the network topology, we trace the routes of each subnet-to-subnet edge of the attack graph. For example, the vulnerable paths from Subnet 1 to Subnet 2 are shown as a red route, from Subnet 1 to Subnet 4 as a blue route, etc. The problem is then the selection of a minimum set of routers (sensors) that covers all the vulnerable paths in the attack graph.

Set cover is known to be computationally hard, one of Karp's original 21 NP-complete problems [35]. Fortunately, there is a well known polynomial-time greedy algorithm for set cover that gives good results in practice [32]. The greedy algorithm for set covering follows this rule: at each stage, choose the set that contains the largest number of uncovered elements.

In our case, each router can see traffic for a subset of the entire attack graph, i.e., each router covers certain attack graph edges. The problem is then to choose a minimum set of routers that cover all edges. From Figure 7, we have the following:

- Router A covers {(1,2), (1,4), (4,5)} = $\{\alpha, \beta, \delta\}$
- Router B covers {(1,2), (4,5)} = = $\{\alpha, \delta\}$
- Router C covers {(1,2), (4,5), (2,5)} = = $\{\alpha, \delta, \gamma\}$
- Router D covers {(2,5), (4,5), (5,3)} = = $\{\gamma, \delta, \varepsilon\}$

Here, the element (x, y) means an attack graph edge set from Subnet x to Subnet y.

A refinement of the greedy algorithm is to favor large sets that contain infrequent elements. In this example, Router A is a large set (3 elements) with the infrequent element β = (1,4), so we choose it first. In the next iteration, we choose Router D, which has the largest number of uncovered elements, i.e., γ = (2,5) and ε = (5,3). At this point, we have covered all 5 elements (edges in the attack graph). Our sensor-placement solution is thus complete, shown in Figure 7 as red eyes at the optimal sensor locations Router A and Router D.

In this instance, we have in fact found the actual optimal solution. In general, the greedy algorithm approximates the optimal solution within a factor of $\ln(n)$, for n elements to be covered, though in practice it usually does much better than this. In our case, n is the number of attack graph edges, aggregated by protection domains, which is usually much smaller than the number of edges between individual machines. The greedy algorithm has been shown to be essentially the best possible polynomial-time approximation algorithm for general set cover [36]. However, for restricted cases in which each element (per-domain edge) occurs in at most f sets (routers), a polynomial-time solution is possible that approximates the optimum to within a factor of f.

Using appropriate data structures, the greedy algorithm for set cover can be implemented in $O(n)$, where n is again the number of per-domain attack graph edges. More nearly optimal solutions for set cover may be possible through more sophisticated algorithms, with longer run times, such as simulated annealing [37] and evolutionary algorithms [38]. Set cover (and its dual the hitting set problem) is one the most well-studied problems in computer science. While experimental validation of complexity and solution optimality are outside the scope of this paper, our formulation of IDS sensor placement as an instance of set covering places our proposed approach on firm ground.

Once IDS sensors are in place and alerts are generated, we can use the attack graph to correlate alerts, prioritize them, predict future attack steps, and respond optimally. Figure 8 shows attack graph details for the testbed network in Figure 6, where each graph edge is a set of exploited vulnerabilities from one machine to another. Within each subnet (the shaded regions in the figure), the machines have unrestricted access to one another's vulnerabilities. Paths in the graph all lead to the assumed critical network assets (shown in the figure as crowns).

We can prioritize alerts based on attack graph distance to critical assets. That is, attacks closer to a critical asset are given higher priority, since they represent a greater risk. At any point that an attack is detected, we can use to graph to predict next possible steps, and take specific actions such as blocking specific source/destination machines and destination port.

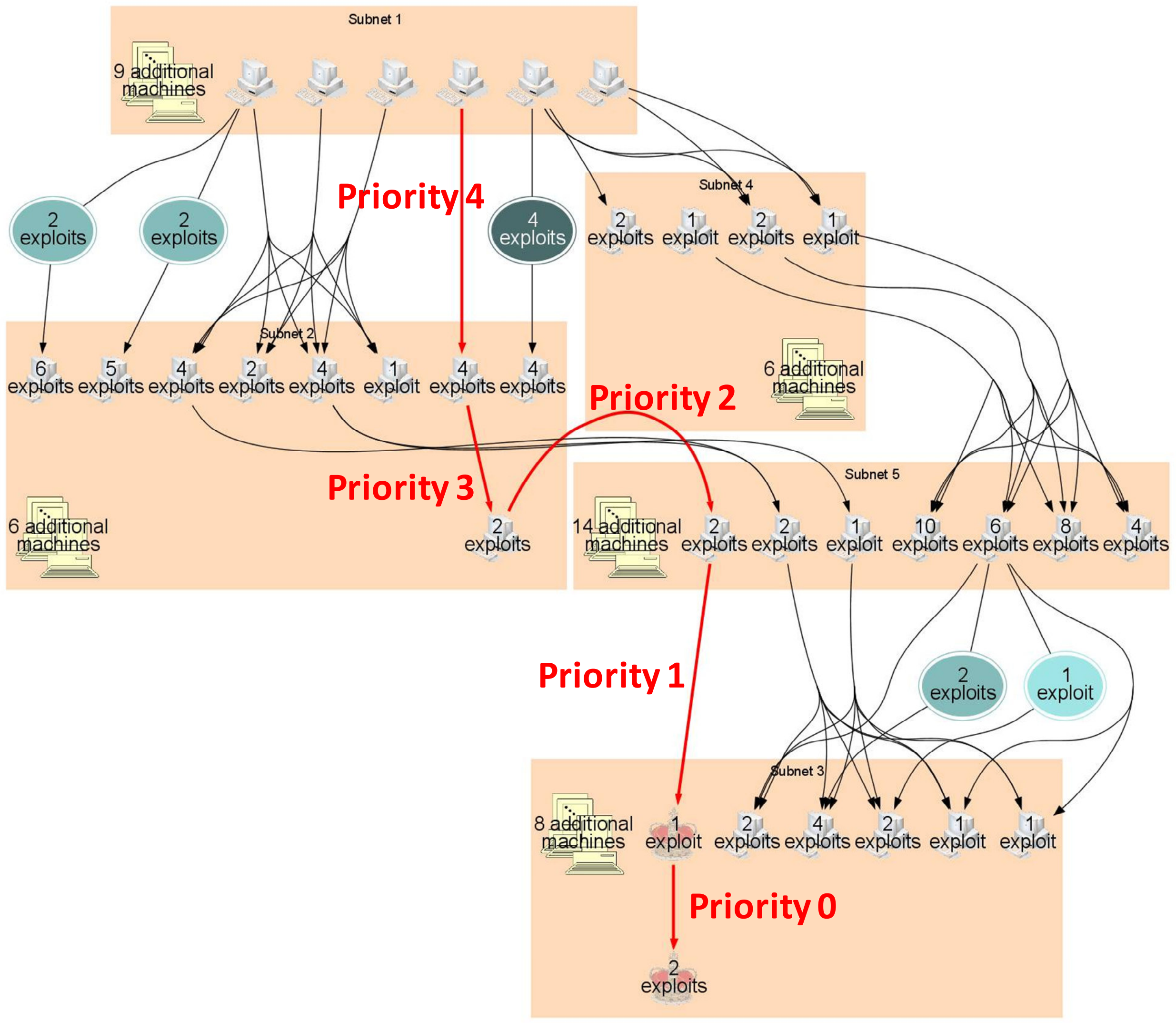

Figure 8. Priority of alerts for testbed network in Figure 6.

As an example operational scenario, in Figure 8, assume that a security operator sees the Priority-4 IDS alarm. From the attack graph, he knows that this potential attack is still at least three steps away from mission-critical machines. Because of the possibility of a false alarm, the operator might delay taking action initially. Then, if he sees the Priority-3 alarm (one step closer to a critical machine), the attack graph shows that this is an immediate next possible step, providing further evidence that this is a real attack (versus a false alarm). If the operator still delays action (e.g., after detail drilldown yields no conclusive result), a subsequent Priority-2 alarm (now only one step away from the critical machine) might then cause him to respond. The attack graph shows exactly which source/destination machines and ports the operator should block to prevent attacker access to the critical machine, while avoiding disruption of other potentially mission-critical network services. This is the kind of highly focused attack response capability provided by our predictive TVA attack graphs coupled with deployed IDS sensors.

6. Summary

In our approach to network defense, we focus on critical paths through the network that lead to compromise of critical assets. This analysis supports optimal placement of IDS sensors, prioritization of alerts, and effective attack response. By analyzing the network configuration, assumed threat sources, and potential attacker exploits, we predict all possible ways of reaching critical assets (the attack graph). We then place IDS sensors to cover all attack graph paths, using the fewest number of sensors necessary. We describe a greedy algorithm for the NP-hard sensor placement (set cover) problem, which we illustrate through example. We also prioritize the resulting IDS alerts based on attack graph distance to critical assets and provide predictive context for attack response.

Acknowledgements

This material is based upon work supported by Homeland Security Advanced Research Projects Agency under the contract FA8750-05-C-0212 administered by the Air Force Research Laboratory/Rome; by Air Force Research Laboratory/Rome under the contract FA8750-06-C-0246; and by Federal Aviation Administration under the contract DTFAWA-04-P-00278/0001.

References

1. 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.), Springer, 2005.
2. S. Jajodia, S. Noel, "Topological Vulnerability Analysis: A Powerful New Approach for Network Attack Prevention, Detection, and Response," in *Indian Statistical Institute Monograph Series*, World Scientific Press, under review.
3. R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, California, 2000.
4. O. Sheyner, J. Haines, S. Jha, R. Lippman, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2002.
5. D. Zerkle, K. Levitt, "Netkuang – A Multi-Host Configuration Vulnerability Checker," in *Proceedings of the 6th USENIX Unix Security Symposium*, San Jose, California, 1996.
6. L. Swiler, C. Phillips, D. Ellis, S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, 2001.
7. P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, 2002.
8. 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.
9. W. Li, *An Approach to Graph-Based Modeling of Network Exploitations*, PhD dissertation, Department of Computer Science, Mississippi State University, 2005.
10. V. Swarup, S. Jajodia, J. Pamula, "Rule-Based Topological Vulnerability Analysis," in *Computer Network Security*, selected papers from the 3rd International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, 2005.

11. X. Ou, W. Boyer, M. McQueen, "A Scalable Approach to Attack Graph Generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, 2006.

12. M. Danforth, *Models for Threat Assessment in Networks*, PhD dissertation, University of California, Davis, 2006.

13. S. Bhattacharya, S. Ghosh, "An Artificial Intelligence Based Approach for Risk Management Using Attack Graph," in *Proceedings of the International Conference on Computational Intelligence and Security*, Harbin, China, 2007.

14. S. Noel, S. Jajodia, "Understanding Complex Network Attack Graphs through Clustered Adjacency Matrices," in *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, Arizona, 2005.

15. S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, "Multiple Coordinated Views for Network Attack Graphs," in *Proceedings of the Workshop on Visualization for Computer Security*, Minneapolis, Minnesota, 2005.

16. L. Wang, S. Noel, S. Jajodia, "Minimum-Cost Network Hardening Using Attack Graphs," *Computer Communications*, 29, 2006.

17. L. Wang, A. Singhal, S. Jajodia, "Measuring the Overall Security of Network Configurations Using Attack Graphs, in *Proceedings of 21st IFIP WG 11.3 Working Conference on Data and Applications Security*, Redondo Beach, California, 2007.

18. J. Pamula, S. Jajodia, P. Ammann, V. Swarup, "A Weakest-Adversary Security Metric for Network Configuration Security Analysis," in *Proceedings of the 2nd ACM Workshop on Quality of Protection*, Alexandria, Virginia, 2006.

19. V. Mehta, C. Bartzis, H. Zhu, E. Clarke, J. Wing, "Ranking Attack Graphs," in *Proceedings of Recent Advances in Intrusion Detection*, Hamburg, Germany, 2006.

20. R. Lippmann, K. Ingols, *An Annotated Review of Past Papers on Attack Graphs*, Technical Report ESC-TR-2005-054, MIT Lincoln Laboratory, 2005.

21. P. Ning, Y. Cui, D. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington DC, 2002.

22. L. Wang, A. Liu, S. Jajodia, "Using Attack Graphs for Correlating, Hypothesizing, and Predicting Intrusion Alerts," *Computer Communications*, 29, 2006.

23. S. Mathew, R. Giomundo, S. Upadhyaya, M. Sudit, A. Stotz, "Understanding Multistage Attacks by Attack-Track based Visualization of Heterogeneous Event Streams," in *Proceedings of the 3rd International Workshop on Visualization for Computer Security*, Alexandria, Virginia, 2006.

24. S. Noel, E. Robertson, S. Jajodia, "Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances," in *Proceedings of the 20th Annual Computer Security Applications Conference*, Tucson, Arizona, 2004.

25. C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné, A. Thomas, "A Hardware Platform for Network Intrusion Detection and Prevention," in *Proceedings of 3rd Workshop on Network Processors & Applications*, Madrid, Spain, 2004.

26. M. Rolando, M. Rossi, N. Sanarico, D. Mandrioli, "A Formal Approach to Sensor Placement and Configuration in a Network Intrusion Detection System," in *Proceedings of the ACM International Workshop on Software Engineering for Secure Systems*, Shanghai, China, 2006.

27. S. Jha, O. Sheyner, J. Wing, *Minimization and Reliability Analyses of Attack Graphs*, Technical Report CMU-CS-02-109, School of Computer Science, Carnegie Mellon University, 2002.

28. S. Noel, S. Jajodia, “Attack Graphs for Sensor Placement, Alert Prioritization, and Attack Response,” Cyberspace Research Workshop, Shreveport, Louisiana, November 2007.

29. Nessus vulnerability scanner, Tenable Network Security, http://www.nessus.org/nessus/.

30. Retina vulnerability scanner, eEye Digital Security, http://www.eeye.com/html/products/Retina/.

31. FoundScan vulnerability scanner, Foundstone (a division of McAfee), http://www.mcafee.com/us/local_content/datasheets/ds_foundtsone60.pdf.

32. T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press and McGraw-Hill, 2001.

33. Nessus vulnerability number 10671 (IIS Remote Command Execution), Tenable Network Security, http://www.nessus.org/plugins/index.php?view=single&id=10671.

34. Common Vulnerabilities and Exposures (CVE), The MITRE Corporation, http://cve.mitre.org/.

35. R. Karp, “Reducibility among Combinatorial Problems,” in *Complexity of Computer Computations*, 1972.

36. U. Feige, “A Threshold of Ln N for Approximating Set Cover,” *Journal of the ACM*, 45(4), 1998.

37. S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, 1997.

38. R. Kalapala, M. Pelikan, A. Hartmann, *Hybrid Evolutionary Algorithms on Minimum Vertex Cover for Random Graphs*, MEDAL Report No. 2007004, University of Missouri–St. Louis, 2007.