Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs

Steven Noel and Sushil Jajodia
Center for Secure Information Systems
George Mason University
4400 University Drive
Fairfax, VA, 22030 USA

Abstract – Network security tools generally lack sufficient context for maintaining a well informed and proactive defense posture. Vulnerabilities are usually assessed in isolation, without considering how they contribute to overall attack risk. Similarly, intrusion alarms are logged as isolated events, with limited correlation capabilities. Security professionals are often overwhelmed by constant threats, complexity of security data, and network growth. Our approach to network defense applies attack graphs for advanced vulnerability analysis and intrusion detection. Attack graphs map paths of vulnerability, showing how attackers can incrementally penetrate a network. We can then identify critical vulnerabilities and provide strategies for protection of critical network assets. Because of operational constraints, vulnerability paths may often remain. The residual attack graph then guides optimal intrusion detection and attack response. This includes optimal placement of intrusion detection sensors, correlating intrusion alarms, accounting for missed detections, prioritizing alarms, and predicting next possible attack steps.

I. Introduction

It is inherently difficult to secure computer networks against attack. Without adequate tool support, network security is labor-intensive and error prone, because of the complexity, volume, and frequent changes in security data and network configurations. Software vulnerabilities are commonplace, patches are often unavailable, and many protocols are insecure.

Moreover, security concerns are interdependent across the network. Attackers can attack vulnerable machines and use them as stepping stones to further penetrate a network and compromise critical systems. But today's security tools are generally point solutions, giving few clues for strategic network defenses. It often remains a difficult exercise to combine results from multiple tools and data sources to mount defenses. It can be challenging for even experienced analysts to recognize multi-step attack risks, and to understand which vulnerabilities really are acceptable risks. This kind of analysis is especially challenging for large dynamically evolving networks.

By knowing the paths of vulnerability through our networks, we can reduce the impact of attacks. But traditional network vulnerability assessment tools simply scan individual machines on a network and report possible security problems. They give little guidance as to how attackers might exploit combinations of vulnerabilities among multiple hosts to advance an attack on a network. Without this knowledge, we cannot optimize our defenses, because vulnerabilities in isolation lack context.

To address these weaknesses, we capture the network configuration, (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. The resulting set of all possible attack paths, organized as an attack graph, constitutes a predictive roadmap of potential attacks.

Our attack graphs provide context for individual vulnerabilities, and support strategic vulnerability risk analysis. Critical combinations of vulnerabilities are identified for optimal network hardening, minimizing any potentially disruptive changes to the network.

Still, because of operational constraints (availability of patches, need to offer critical services, etc.), some residual network vulnerability usually remains. In such cases, attack graphs support proactive measures that reduce the impact of attacks, and guide our responses. First, we can use the attack graphs to determine optimal locations for intrusion sensors, to cover known paths of vulnerability, while minimizing the number of deployed sensors.

Then, through the predictive power of our attack graphs, we can tune intrusion detection to focus on known vulnerability paths. We can also correlate isolated intrusion alarms into multi-steep coordinated attacks. Further, we can prioritize alerts based on incremental penetration and closeness to critical network assets. We can also predict next possible attack steps for optimal response.

The next section gives an overview of our approach for predictive attack graph analysis. Section 3 illustrates the approach through a simple example. Section 4 describes our working system for predictive

attack graph analysis. Section 5 examines how this system can be extended for optimal placement of intrusion detection sensors, and Section 6 shows how our attack graphs provide context for attack correlation and response. Section 7 reviews related work, and Section 8 summarizes our approach.

II. Overview of Approach

Our approach is to capture the network configuration, from which we predict attack paths through the network. We then use the predicted paths (attack graph) for network hardening, intrusion sensor placement, alarm prioritization, and attack response.

As shown in Fig. 1, we scan the network to discover hosts, operating systems, application programs, vulnerable network services, etc. We also capture network connectivity, including the connectivity-limiting effects of devices such as firewalls and router access control lists.

We apply the network data to an attack prediction engine, using a database of modeled attacker exploits. For assumed threat source(s) and/or critical network asset(s) to protect, we predict multi-step attacks through the network. This attack graph, computed in worst-case quadratic time [1][2], represents all known attack paths through the network.

The resulting attack graph then supports proactive network defenses. In particular, from the graph we formulate optimal network hardening strategies. Then, for any residual vulnerability paths, we place intrusion sensors in the network to cover all paths, using the minimum number of sensors [3].

We also fine-tune the intrusion detection system by focusing on known vulnerability paths. We can then correlate resulting intrusion alarms with known vulnerability paths, thus detecting coordinated multi-step attacks [4]. We can also raise priority for such coordinated attacks, especially when they lie on paths to critical network assets.

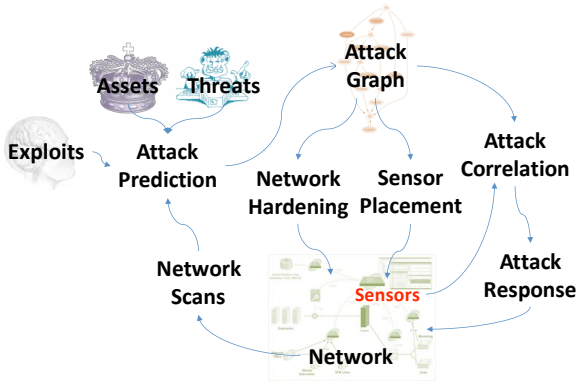

Fig. 1. System overview.

The attack graph also provides the context needed for responding appropriately to attacks. When we know the possible vulnerability paths, we can restrict our responses to exactly what prevents further penetration.

III. Illustrative Example

Fig. 2 shows a small network for demonstrating our generation of network attack graphs. In this network, the firewall protects 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 as well, although the firewall protects it from being attacked from the outside directly. The question is whether there are attack paths that allow the outside Attacker to compromise the Mail Server.

For this example, we need to capture elements of the network configuration relevant to attack penetration. This includes the existence of vulnerable software (services) on hosts, as well as connectivity allowed to vulnerable services. We also need a set of potential attacker exploits that may work against the vulnerable services.

For example, we could run a vulnerability scanning tool (e.g., Nessus [5]) against the hosts in the internal network to map their vulnerabilities, and feed this into the model. We could then rely on a database of modeled exploits based on the full set of vulnerabilities detected by Nessus. To incorporate the connectivity-limiting effects of the firewall, we could scan through the firewall (as well as behind it), to implicitly capture firewall effects, or process the firewall rules directly.

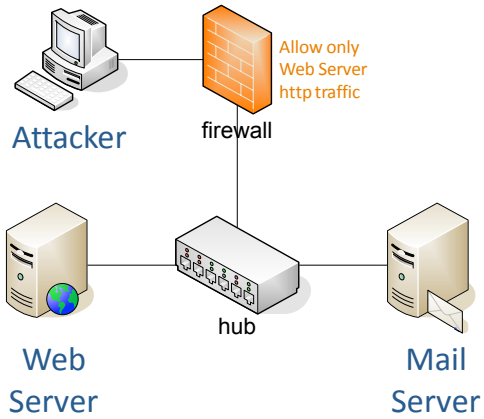

Fig. 2. Small example network.

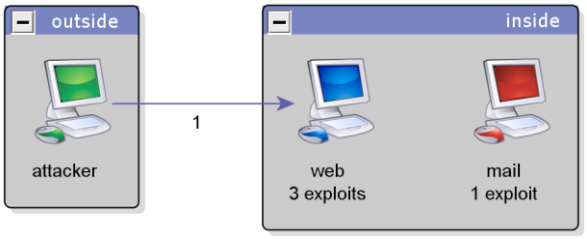

Fig. 3. Attack graph for example network.

Fig. 3 shows the resulting attack graph for this small example. There is indeed a multi-step path from the outside to the Mail Server.

In the first step, the attacker exploits the IIS vulnerability that is exposed through the firewall. Here, the number "1" indicates there is only one critical exploit from the Attacker to the Web Server, from among the 3 total vulnerabilities on the Web Server. Then, once inside, the attacker has unlimited connectivity to the Mail Server (via the hub). In this case, there is a vulnerable service on the Mail Server that allows the attacker to compromise this critical host.

This attack graph shows how hosts on a network can be exploited through multiple steps, even when the attacker cannot access them directly. It is not directly possible to compromise the Mail Server from the outside because of the policy enforced by the firewall. But our attack graph shows that the Mail Server can be reached indirectly, through a sequence of two exploits. From the outside, a traditional scanning tool like Nessus reveals no Mail Server vulnerability.

Further, the attack graph shows that addressing a single critical vulnerability (from among four) would prevent the attack. Also, tools such as Nessus generate many alerts that are merely informational (i.e., irrelevant to network penetration), while we carefully excludes these from our database of modeled exploits.

While such a result may be easy enough for an experienced analyst on a small network, for maintaining a proactive security posture on realistic networks an automated tool is crucial.

IV. Predictive Attack Graphs

For computing attack graphs for larger networks, we need scalable mathematical representations and algorithms. Modeling the attacker's control over the network as monotonic (increasing over time), we need only represent the dependencies among exploits, rather than explicitly enumerating every sequence of exploits [6]. The resulting exploit-dependency attack graphs grow only quadratically (as opposed to exponentially) with the number of exploits, so that it becomes feasible to apply them for realistic networks.

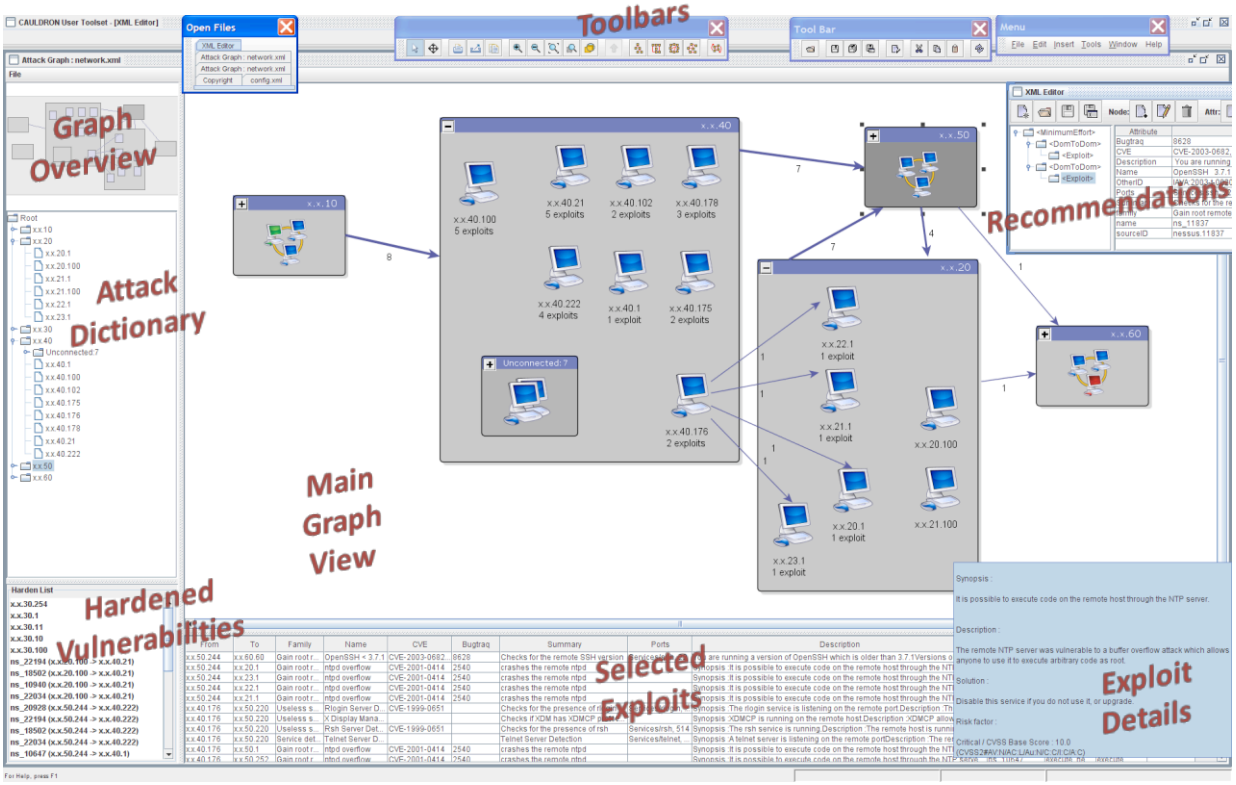

Fig. 4. Predictive network attack graph.

Based on a given attack scenario, the attack graph can be constrained by specific starting and ending points (as in the previous section). The scenario could also be less constrained, such as finding all possible attack starts leading to one or more goals, or finding all possible paths from particular starting points.

We aggregate attack graphs, to help make them understandable at a glance [7]. An important aggregation abstraction is the *protection domain*, which represents a set of machines that have full access to one another's vulnerabilities. In a raw (non-aggregated) form, the graph would be fully connected within a protection domain. Instead, we list the machines in a protection domain, along with exploits against each of their vulnerabilities. Then implicitly, once an attacker takes control of a machine within a protection domain, he can exploit all vulnerabilities on machines within it. We thus need not explicitly list every n^2 (fully-connected) exploit dependency within the protection domain, making complexity linear within the domain.

In our implementation of predictive attack graphs (Fig. 4), a high-level overview displays attack relationships among protection domains, which can be opened individually or in groups for deeper views of attack properties and relationships. In this process, no graph information is lost; one has merely to expand a folder to acquire information at a lower level.

A complete listing of exploits and associated details for any selected component is available at all times. This supports in-depth analysis of exploit details, while overall topology and network relationships are kept simple and understandable within the main graph view.

Our attack graph implementation also emulates the hardening exploitable vulnerabilities, to study the effects of remediation and what-if scenarios. Exploring the attack graph, the analyst is often faced with multiple options for remediation. This involves choosing a machine or set of machines to protect (harden), or identifying specific exploits to protect against.

We display the attack graph effects that occur when a specific machine or protection domain is hardened or when a specific exploit is neutralized. Hardened elements are maintained in a log, e.g., for reporting. We can also generate recommendations automatically, i.e., first layer (from start), last layer (from goal), and the optimal (minimum) set that that separates start from goal [8].

To aid navigation, our implementation maintains a global overview of the entire attack graph at all times, which can be used to pan the main graph view. The tool also has a graphical (tree) attack dictionary of all graph elements. The various graph views are linked, so that selecting an element in one view cause it to be selected in all views. A variety of toolbars are available for commonly used tools. This includes a suite of interactive layout tools, with manual repositioning as well as full-scale layout algorithms, continuously available to restructure the display.

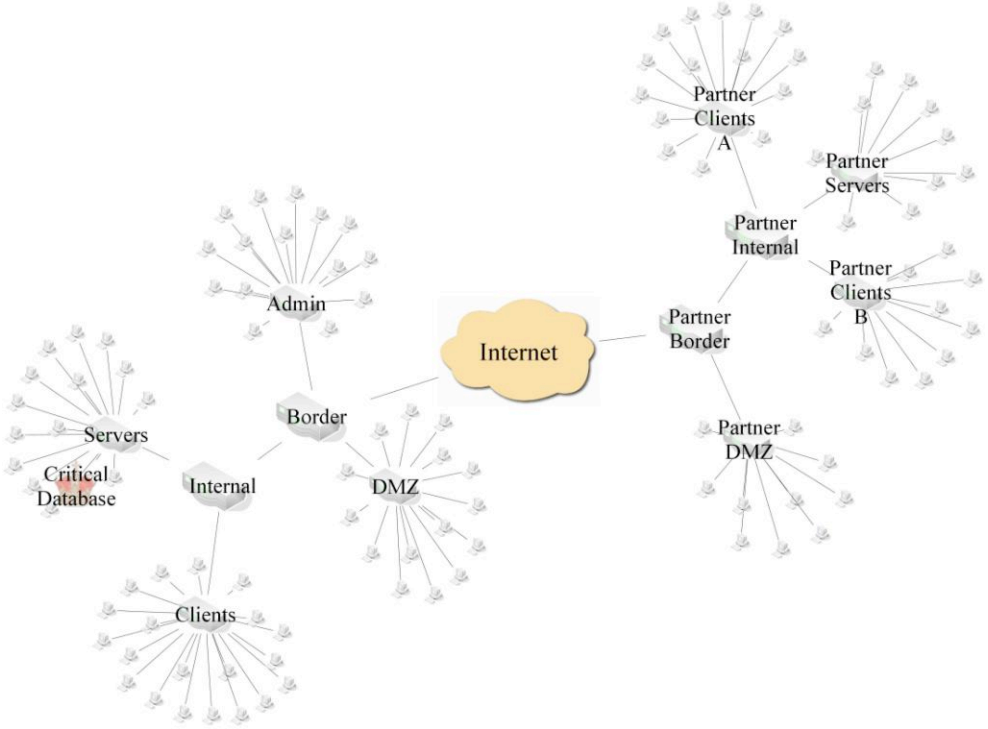

Fig. 5. Network for placing intrusion detection sensors.

V. Optimal Sensor Placement

At this point in the network defense process, we have 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. At this point, we can rely on intrusion detection, guided by our predictive attack graph.

Now, given the residual paths of vulnerability, where should we place intrusion sensors to monitor all these paths? In fact, how can we cover all critical paths with the fewest number of sensors, to minimize our deployment costs?

Consider the network in Fig. 5. 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 hosts, and firewalls (not shown) limit connectivity to help protect the network.

The network in Fig. 5 is in two parts – our enterprise network to defend, and a partner network that has been given some access to our network (via the Internet). In this scenario, we wish to protect a critical database server (crown in Fig. 5).

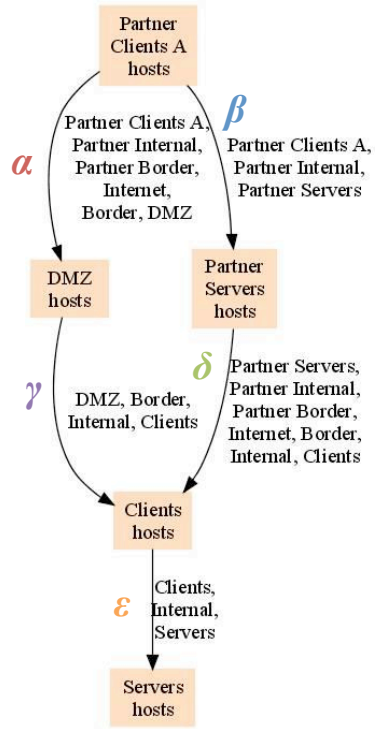

Fig. 6. Predictive attack graph to be covered.

Fig. 6 shows a high-level view of network vulnerability paths, based on our predictive attack graph. This graph shows all possible paths leading to our Server subnet, at the subnet-to-subnet (protection domain) level. Here, an edge means there is at least one exploit between given protection domains.

Each edge (set of attacks between a pair of domains) in Fig. 6 is labeled with the set of network devices that carry traffic between the pair of domains. For example, traffic between hosts from Clients subnet to Servers subnet flows through the Clients device, the Internal device, and the Servers device. Each such device is a potential location for placing an intrusion detection sensor for monitoring potentially malicious traffic.

Given this knowledge of vulnerable paths through the network, we wish to place sensors to cover all paths. To minimize costs, we seek to cover all paths (the full attack graph) using the least number of sensors.

Our optimal sensor placement is an instance of the classical set cover problem [9]. 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 sensors deployed on particular network devices. Each sensor monitors a given set of edges, i.e., can see the traffic between the given attacker/victim machines.

Set cover is known to be computationally hard, one of Karp's original 21 *NP*-complete problems [10]. But there is a well known polynomial-time greedy algorithm for set cover that gives good results in practice [9]. 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 network device can see traffic for a subset of the entire attack graph, i.e., each device covers certain attack graph edges. For example, in Fig. 6, the Partner Border device covers two attack graph edges:

- Partner Clients A hosts to DMZ hosts (edge α)
- Partner Servers hosts to Clients hosts (edge δ)

The problem is then to choose a minimum set of network devices that cover all attack graph edges. Indexing devices from Fig. 6, we have the following:

- Partner Internal covers $\{\alpha, \beta, \delta\}$
- Partner Border covers $\{\alpha, \delta\}$
- Border covers $\{\alpha, \delta, \gamma\}$
- Internal covers $\{\gamma, \delta, \varepsilon\}$

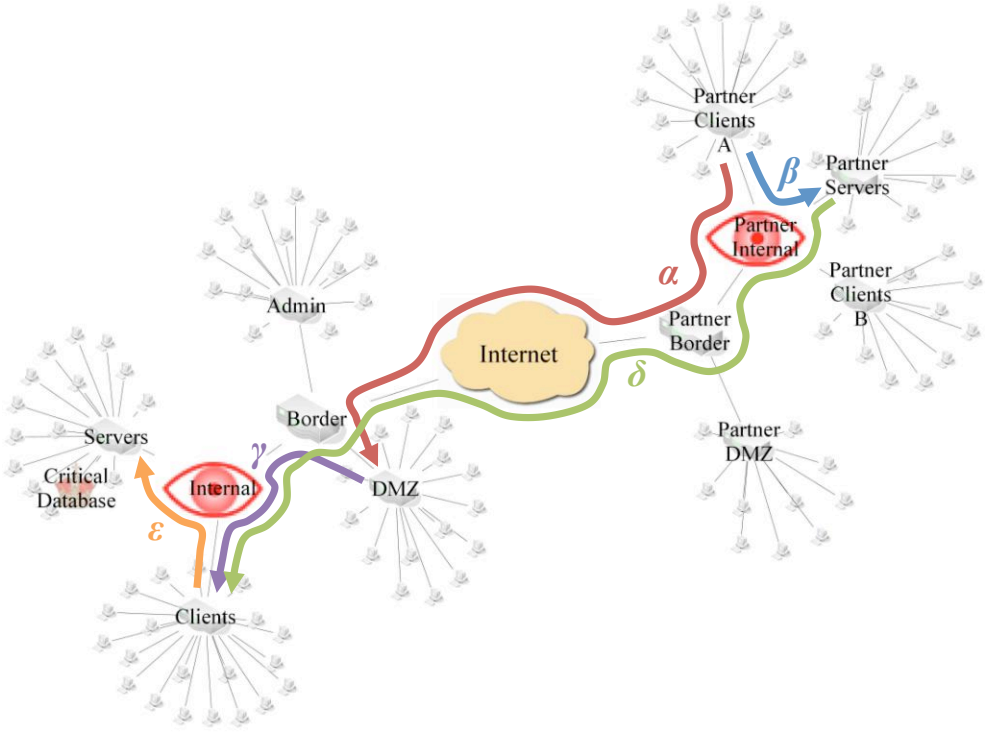

Fig. 7. Optimal placement of intrusion detection sensors.

A refinement of the greedy algorithm is to favor large sets that contain infrequent elements. In this example, the Partner Internal device covers a large set (3 attack graph edges), with the infrequently covered edge β (Partner Clients A hosts to Partner Servers hosts) so we choose it first. In the next iteration, we choose device Internal, which has the largest number of uncovered elements, i.e., γ and ε.

At this point, we have covered all 5 edges in the attack graph (Fig. 6). Our sensor-placement solution is thus complete, as shown in Figure 7. Here, red eyes show the optimal sensor locations at devices Partner Internal and Internal. From the figure, superimposed with attack graph edges, we see that sensors at these two devices cover all attack graph edges.

In this instance, we have in fact found the optimal solution. In general, the greedy algorithm approximates the optimal solution within a factor of $\ln(n)$, for n attack edges 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 [11]. However, for restricted cases in which each element (per-domain edge) occurs in at most f sets (network devices), 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. Set cover is a well-studied problem in computer science, placing our approach to sensor placement via predictive attack graphs on firm theoretical ground.

Traditionally, intrusion detection sensors are placed at network perimeters, with the idea of detecting attacks from the outside. But such deployment is limited, because traffic in the vulnerable internal network is not monitored. If an attacker avoids detection at the perimeter, subsequent attack traffic in the internal network is missed.

On the other hand, deploying sensors everywhere may be cost prohibitive, and can overwhelm analysts with floods of alerts. Our predictive attack graphs strike a balance, in which we cover known residual vulnerability paths, using the fewest sensors necessary.

VI. Attack Correlation and Response

Our attack graphs identify critical vulnerability paths and provide strategies for network hardening in advance of attack. But because of operational constraints such as availability of patches and the need

for offering mission-critical services, residual vulnerability paths usually remain.

Our predictive attack graphs allow us to plan in advance, and maintain a proactive security posture in the face of attacks. Knowledge of network vulnerability paths helps us prepare our defenses and plan our responses, tailored to our network and its critical assets. In particular, attack graphs provide the necessary context for deploying and fine tuning of intrusion detection systems, for correlating and prioritizing intrusion alarms, and for responding to attacks.

Once sensors are deployed and are generating intrusion alarms, we can leverage predictive attack graphs for intrusion alarm correlation and prioritization. This requires mapping alarms to their corresponding elements (exploits) in the residual attack graph. This in turn requires a common alarm format, using identifiers that match those in the attack graph model (exploit database).

Specifications such as Intrusion Detection Message Exchange Format [12] (IDMEF) or ArcSight [13] event logs provide possible future integration for predictive attack graphs. For example, there is an IDMEF plugin [14] for the popular Snort intrusion detection system [15].

Data exchanges in IDMEF are in XML, enforced through a formal schema. Fig. 8 shows the XML schema for an IDMEF alert. The IDMEF data model is designed to accommodate alerts from heterogeneous tools. The critical data for our attack graphs are source and target (attacker and victim) network addresses, and an alarm identifier that can be mapped to our exploit database. In IDMEF, these are supported by the Source, Target, and Classification elements (respectively).

When intrusion alarms are generated, our predictive attack graphs provide the necessary context for correlating and prioritizing them. We can place a high priority on alarms that lie on vulnerability paths through our network. We can prioritize them even further based on graph distance to given critical assets. In other words, events that are very close to critical assets (in terms of next attack steps) should be given higher priority.

This kind of precise attack graph analysis determines not only whether a host is vulnerable to a given attack, but also whether the attacker can traverse through firewalls to reach the host's vulnerable port, and whether that attack could lead to subsequent network compromise. Our prioritization thus also serves as a powerful form of false-alarm reduction, e.g., restriction to alarms along critical paths.

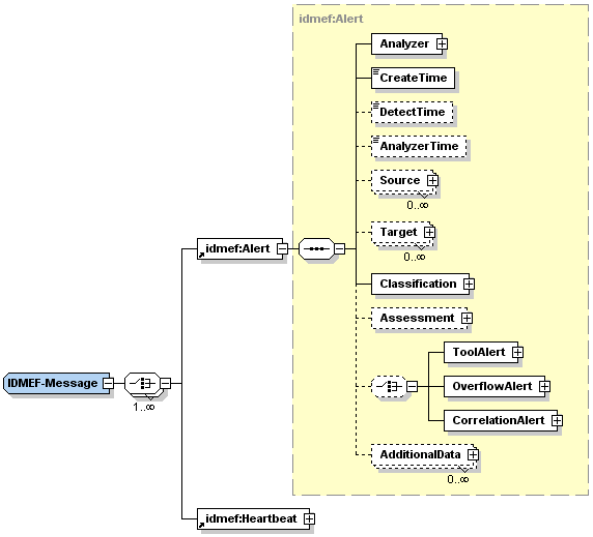

Fig. 8. Message format for intrusion alerts.

It is important to predict network vulnerability paths, as we do. Alarm correlation that does not take network vulnerabilities into account is limited [16]. Pre-computing our predictive attack graphs in advance of attack has the additional advantage of rapid correlation, i.e., faster than an intrusion detection system can generate them [4][17].

Our predictive attack graphs allow us to correlate intrusion alarms based on attack causality. A set of seemingly isolated events may in fact be shown as multiple steps of incremental network penetration. Also, the context provided by these attack graphs allows us to predict missed events (false negatives), helping to mitigate inaccuracies in our intrusion detection systems [4].

Attack graph analysis and visualization need not be limited to abstract cyber views. In many situations, it may be important to understand the physical location of attacks, for assessing threat sources and mission impact. We can thus embed the predictive attack graph into a geo-spatial visualization, as shown in Fig. 9.

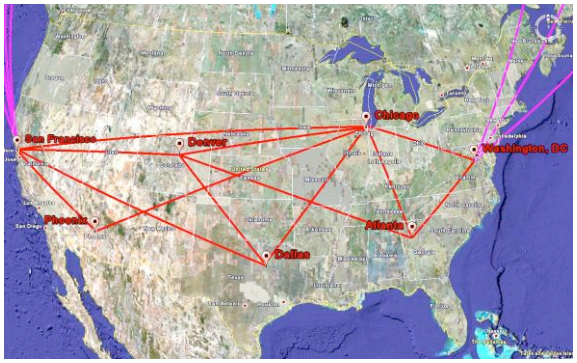

Fig. 9. Geo-spatial attack graph view.

In this view, elements of the attack graph are clustered around major geographic network, and graph edges show exploits between those centers. Interactive visualization capabilities can support drilldown for further details at a desired level of resolution.

VII. Scalability

For computing attack graphs, we need scalable mathematical representations and algorithms. We assume the attacker's control over the network increases monotonically over time [6]. This is the conservative assumption that once an attacker gains control of a network resource, there is no need to relinquish it to further advance the attack.

Under this monotonicity, it is sufficient to represent the dependencies among exploits, rather than explicitly enumerating every sequence of exploits. The resulting exploit-dependency attack graphs grow quadratically rather than exponentially [18]. In particular, worst-case complexity for n network hosts is $O(n^2)$. By grouping hosts into protection domains, complexity is reduced to $O(n)$ within each domain [19]. In terms of the database of potential attacker exploits, complexity is $O(e)$, for e exploits.

Fig. 10 shows attack graph computation times for networks of various sizes. In each case, a subnet contains 200 hosts, and each host has 5 vulnerabilities. Each subnet has incoming vulnerable connections from two other subnets, and symmetrically, outgoing vulnerable connections to two other subnets. This is a ring topology, in which the number of network connections grows linearly with the number of subnets. Thus it is not worst-case, in which network size grows quadratically with the number of subnets.

From one subnet to another, there are 500 connections to vulnerabilities in the victim subnet. Thus there are 2×500=1,000 incoming and 2×500=1,000 outgoing vulnerable connections (a grand total of 2,000) for each subnet. Computation times (total run time in seconds) are based on increasing numbers of subnets, from 20 subnets (4,000 hosts) to 200 subnets (40,000 hosts). Run times are for a quad-core Intel Xeon CPU at 1.86 GHz, with 4 GB RAM.

In this experiment, overall network size (number of vulnerable connections) grows linearly with the number of subnets (and hosts). This shows how graph generation time depends proportionally on the size of the input network. This excludes any time for generating the input model (network and exploits) itself, although this has the same worse-case complexity, and can be created in advance. This also excludes any time for placing intrusion detection sensors, which has only linear complexity.

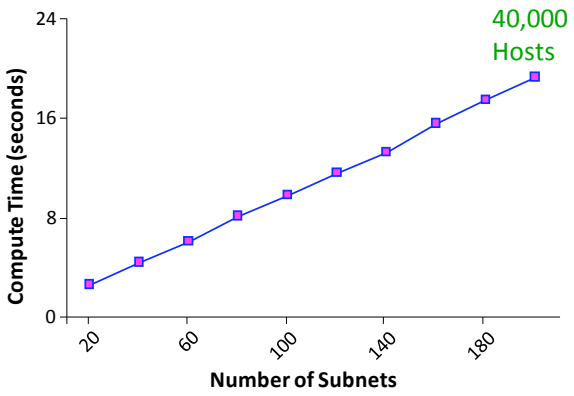

Fig. 10. Computation time for attack graph generation.

Graph visual layout performance is a separate issue, and is not included in the execution times of Fig. 10. For example, computing graph visual layout for the 100-subnets case (20,000 hosts) takes 14 minutes. But in all cases, once the initial layout is computed, performance of user interaction (repositioning, drilldown, etc.) is immediate.

Visual layout computation is needed for a cyber view of network attack graphs. Such layout induces spatial coordinates onto an abstract information graph. But when we embed the attack graph in geo-spatial visualizations such as Fig. 9, spatial coordinates are already given. Thus no graph layout computation is needed. In such cases, visualizing complex attack graphs is much faster than for purely abstract cyber views.

VIII. Related Work

In other work, we have integrated with a number of network tools for building predictive attack graphs, including the Nessus, Retina [20], and FoundScan [21] vulnerability scanners. We have also processed data from the Sidewinder firewall [22] to capture network connectivity to vulnerable host services, and have integrated with Symantec Discovery asset inventory [23] for gathering host configuration data. For maintaining our database of modeled attacker exploits, we rely on a number of sources, including NIST's National Vulnerability Database (NVD) [24], the Bugtraq security database [25], the SecurityFocus forum [26], the Open Source Vulnerability Database (OSVDB) [27], and the Common Vulnerabilities and Exposure (CVE) referencing standard [28], and Symantec DeepSight [29].

Earliest approaches to attack graph generation are generally based on explicit enumeration of attack states, with scalability problems [30][31][32]. With the

practical assumption of monotonic logic, attack graph complexity was shown to be polynomial rather than exponential [6][33]. Graph complexity has been further reduced, to worst-case quadratic in the number of hosts [18][19]. By grouping hosts into protection domains (e.g., subnets) as we do [19], complexity is reduced to $O(n)$ within each domain.

In general, attack graph research has largely focused on scalability, with relatively little work on aspects of model population. Notable exceptions include [34][35][36], although these are more theoretical frameworks than practical model population. Commercial capabilities for attack graph analysis remain limited, especially in the area of visualization for large-scale graphs [37][38]. An annotated review of attack graph research (as of 2005) is given in [39].

IX. Summary and Conclusions

Our predictive attack graphs are a powerful way of understanding the context and relative importance of vulnerabilities across systems and networks. These graphs map all potential paths of vulnerability, showing how attackers can penetrate through a network. Our attack graphs identify critical vulnerabilities and provide strategies for protection of critical network assets. This allows us to harden the network before attacks occur, to handle intrusion detection more effectively, and to responding appropriately to attacks.

We model the network configuration, including connectivity to vulnerable services. We then match the network configuration against a database of modeled attacker exploits, simulating multi-step attack penetration. From the resulting attack graphs, we compute recommendations for optimal network hardening. We also provide sophisticated visualization capabilities for interactive attack graph exploration.

By mapping our attack graphs to the network topology and devices, we can deploy intrusion detection sensors to cover all vulnerable paths, using the minimum number of sensors. Our attack graphs then provide the necessary context for correlating and prioritizing intrusion alerts, based on known paths of vulnerability through the network. Standardization of alert data formats and models can facilitate integration between our system and commodity intrusion detection systems.

By mapping intrusion alarms to our predictive attack graph, we can correlate alarms into multi-step attacks, and prioritize alarms based on paths to critical network assets. Knowledge of network vulnerability paths allows us to formulate best options for responding to attacks. Overall, our predictive attack graphs offer powerful capabilities for proactive network defense.

Acknowledgments

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; by Federal Aviation Administration under the contract DTFAWA-08-F-GMU18; by Air Force Office of Scientific Research under grants FA9550-07-1-0527 and FA9550-08-1-0157; and by the National Science Foundation under grants CT-0716567, CT-0716323, and CT-0627493. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

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," *Indian Statistical Institute Monograph Series*, World Scientific Press, 2008.

[3] S. Noel, S. Jajodia, "Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs," *Journal of Network and Systems Management*, September 2008.

[4] S. Noel, E. Robertson, S. Jajodia, "Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances," *20th Annual Computer Security Applications Conference*, 2004.

[5] Nessus Vulnerability Scanner, http://www.nessus.org.

[6] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," *ACM Conference on Computer and Communications Security*, 2002.

[7] S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, "Multiple Coordinated Views for Network Attack Graphs," *ACM Workshop on Visualization for Computer Security*, 2005.

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

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

[10] R. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*, 1972.

[11] U. Feige, "A Threshold of Ln N for Approximating Set Cover," *Journal of the ACM*, 45(4), 1998.

[12] Internet Engineering Task Force, The Intrusion Detection Message Exchange Format, http://www.ietf.org/rfc/rfc4765.txt.

[13] ArcSight, Enterprise Security Management Product, http://www.arcsight.com/.

[14] SourceForge, Snort Plugin for IDMEF Format, http://sourceforge.net/projects/snort-idmef.

[15] Sourcefire, Snort – The De Facto Standard for Intrusion Detection/Prevention, http://www.snort.org/.

[16] P. Ning, Y. Cui, D. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," *ACM Conference on Computer and Communications Security*, 2002.

[17] L. Wang, A. Liu, S. Jajodia, "Using Attack Graphs for Correlating, Hypothesizing, and Predicting Network Intrusion Alerts," *Computer Communications*, 29(15), 2006.
[18] S. Noel, J. Jajodia, "Understanding Complex Network Attack Graphs through Clustered Adjacency Matrices," *21st Annual Computer Security Applications Conference*, 2005.
[19] S. Noel, S. Jajodia, "Managing Attack Graph Complexity through Visual Hierarchical Aggregation," in *Visualization and Data Mining for Computer Security*, 2004.
[20] Retina Security Scanner, http://www.eeye.com/.
[21] FoundScan, http://www.foundstone.com/.
[22] Sidewinder, http://www.securecomputing.com/.
[23] Discovery, http://www.centennial-software.com/.
[24] National Vulnerability Database, http://nvd.nist.gov/.
[25] Bugtraq, http://www.securityfocus.com/vulnerabilities.
[26] Security Focus, http://www.securityfocus.com/.
[27] Open Source Vulnerability Database, http://osvdb.org/ .
[28] Common Vulnerabilities and Exposures, http://cve.mitre.org/.
[29] Symantec DeepSight Threat Management System, https://tms.symantec.com/Default.aspx.
[30] C. Phillips, L. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," *New Security Paradigms Workshop*, 1998.
[31] R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," *IEEE Symposium on Security and Privacy*, 2000.
[32] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs," *IEEE Symposium on Security and Privacy*, 2002.
[33] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, R. Cunningham, "Validating and Restoring Defense in Depth Using Attack Graphs," *MILCOM Military Communications Conference*, 2006.
[34] F. Cuppens, R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," *Workshop on Recent Advances in Intrusion Detection*, 2000.
[35] S. Templeton, K. Levitt, "A Requires/Provides Model for Computer Attacks," *New Security Paradigms Workshop*, 2000.
[36] R. Ritchey, B. O'Berry, S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," *18th Annual Computer Security Applications Conference*, 2002.
[37] Skybox Security, http://www.skyboxsecurity.com/.
[38] RedSeal Systems, http://www.redseal.net/.
[39] R. Lippmann, K. Ingols, "An Annotated Review of Past Papers on Attack Graphs," Lincoln Laboratory Technical Report ESC-TR-2005-054, 2005.