AFRL-RI-RS-TR-2010-078
Final Technical Report
March 2010

ADVANCED CYBER ATTACK MODELING, ANALYSIS, AND VISUALIZATION

George Mason University

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* MARCH 2010	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* September 2006 – September 2009

4. TITLE AND SUBTITLE ADVANCED CYBER ATTACK MODELING, ANALYSIS, AND VISUALIZATION	**5a. CONTRACT NUMBER** FA8750-06-C-0246
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 33140F
6. AUTHOR(S) Sushil Jajodia and Steven Noel	**5d. PROJECT NUMBER** 7820
	5e. TASK NUMBER MW
	5f. WORK UNIT NUMBER 01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) George Mason University 4400 University Drive Fairfax, VA 22030-4422	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RIGA 525 Brooks Road Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2010-078

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project delivers an approach for visualization, correlation, and prediction of potentially large and complex network attack graphs. These attack graphs facilitate defense against multi-step cyber network attacks, based on system vulnerabilities, network connectivity, and potential attacker exploits. A new paradigm is introduced for attack graph analysis that augments the traditional graph-centric view, based on graph adjacency matrices.

15. SUBJECT TERMS
Cyber attack graphing, Information assurance, IA, Information security, User interaction, Cyber defense, Vulnerability prioritization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Thomas J. Parisi
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	113	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

TABLE OF CONTENTS

LIST OF FIGURES

1. SUMMARY

This project delivers an approach for visualization, correlation, and prediction of potentially large and complex attack graphs. These attack graphs show multi-step cyber attacks against networks, based on system vulnerabilities, network connectivity, and potential attacker exploits. We introduce a new paradigm for attack graph analysis that augments the traditional graph-centric view, based on graph adjacency matrices.

In our approach, the analysis includes all known network attack paths, while still keeping complexity manageable. It supports pre-attack network hardening, correlation of detected attack events, and attack origin/impact prediction for post-attack responses. The goal of this system is to transform large quantities of network security data into actionable intelligence.

The utility of organizing combinations of network attacks as graphs is well established. Traditionally, such attack graphs have been formed manually by security red teams (penetration testers). We have demonstrated the capability for computational generation of attack graphs, rather than relying on manual creation. This approach is based on models of network security conditions and potential attacker exploits.

Because of vulnerability interdependencies across networks, a topological attack graph approach is needed, especially for proactive defense against insidious multi-step attacks. The traditional approach that treats network data and events in isolation, without the context provided by attack graphs, is clearly insufficient.

Our innovative approach to proactive cyber security via attack graphs is called *Topological Vulnerability Analysis* (TVA). TVA combines vulnerabilities in ways that real attackers might do, discovering all attack paths through a network, given the completeness of scan data used for our analysis. Mapping all paths through the network provides defense in depth, with multiple options for mitigating potential attacks, rather than relying on mere perimeter defenses.

From its attack graphs, TVA computes recommendations for optimal network hardening. It also provides sophisticated visualization capabilities for interactive attack graph exploration and what-if analysis. TVA attack graphs support a number of metrics that quantify overall network security, e.g., for trending or comparative analyses.

Further, by mapping TVA attack paths to the network topology, we can deploy intrusion detection sensors to cover all paths using the minimum number of sensors. TVA 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 facilitates integration between TVA and intrusion detection systems.

By mapping intrusion alarms to the TVA attack graph, we can correlate alarms into multi-step attacks and prioritize alarms based on distance from critical network assets. Further, through knowledge of network vulnerability paths, we can formulate best options responding to attacks. Overall, TVA offers powerful capabilities for proactive network defense, transforming raw security data into actionable intelligence.

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 intrusion detection 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. We then place sensors to cover all attack graph paths, using the fewest number of sensors necessary.

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 generally gives near optimal results very quickly. 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.

We model composition of vulnerabilities through attack graphs, which show all paths of vulnerability allowing incremental network penetration. We propagate attack likelihoods through the attack graph, yielding a novel metric that measures the overall security of a networked system.

From this, we score risk mitigation options in terms of maximizing security and minimizing cost. For practical implementation, we can rely on our TVA attack graph tool. TVA populates attack graph models from live network scans and databases of reported vulnerabilities. As additional input to our model, we use comprehensive sources of security risk scores for individual vulnerabilities. Our flexible new attack graph metric model can be used to quantify overall security of networked systems, and to study cost/benefit tradeoffs for analyzing return on security investment.

2. INTRODUCTION

Cyber security is inherently difficult. Protocols are often insecure, software is frequently vulnerable, and educating end-users is time-consuming. Security is labor-intensive, requires specialized knowledge, and is error prone because of the complexity and frequent changes in network configurations and security-related data. Network administrators and security analysts can easily become overwhelmed and reduced to simply reacting to security events. A much more proactive stance is needed.

Furthermore, the correct priorities need to be set for concentrating efforts to secure the network. Administrators and analysts often have a vertical view of the particular component they are managing; horizontal views across/through the infrastructure are missing. This in turn shifts emphasis to vulnerabilities at the interfaces. Security concerns in a network are also highly interdependent, i.e., susceptibility to attack can depend on multiple vulnerabilities across the network. Attackers can combine such vulnerabilities to incrementally penetrate a network and compromise critical systems.

However, traditional security tools are generally point solutions that provide only a small part of the picture. They give few clues as to how attackers might exploit combinations of vulnerabilities to advance an attack on a network. It remains a painful exercise to combine results from multiple tools and data sources to understand one's true vulnerability against sophisticated multi-step attacks. It can be difficult even for experienced analysts to recognize such risks, and it is especially challenging for large dynamically evolving networks.

Our network models are created automatically from network scans and firewall rule logs. Potential attacker exploits are modeled from existing databases of reported cyber vulnerabilities. In our attack graph representation, dependencies among attacker exploits are mapped. We avoid explicit enumeration of attack states; our attack graphs thus scale quadratically rather than exponentially.

With our approach, we can efficiently compute attack graphs for realistic networks. But the attack graphs that result can often pose challenges for human comprehension. This is compounded by the fact that attack graphs are usually communicated by literal drawings of graph vertices and edges.

The general problem of graph drawing is ill-posed in the sense that many possibilities exist for what constitutes a good graph drawing. Also, finding optimal placement of graph vertices according to many of the desired criteria is NP-complete. For the relatively dense attack graphs often found in practice (e.g., within a trusted internal network), graph drawing techniques are often ineffective, producing overly cluttered drawings for graphs of larger than moderate size.

In this project, we develop techniques to help make complex attack graphs more understandable, and apply these techniques to the correlation, prediction, and hypothesis of attacks. Our approach reveals graph regularities, making important features such as bottlenecks and densely-connected subgraphs apparent. We extend an existing graph-clustering technique to show multi-step reachability across the network, the impact of network configuration changes, and the analysis of intrusion alarms within the context of network vulnerabilities.

Rather than relying solely on literal drawings of attack graphs, we augment that with visualization of the corresponding attack graph adjacency matrix [1]. The adjacency matrix represents each graph edge with a single matrix element, as opposed to a drawn line. Graph vertices, rather than being drawn explicitly, are implicitly represented as matrix rows and columns. The adjacency matrix avoids the edge clutter of drawn graphs, not only for very large graphs, but also for smaller ones.

The adjacency matrix is a concise graph representation, but alone it can be insufficient. That is, without the proper ordering of matrix rows and columns, the underlying attack graph structure is not necessarily apparent. We therefore apply an information-theoretic clustering technique that reorders the adjacency matrix so that blocks of similarly-connected attack graph elements emerge. The clustering technique is fully automatic, parameter-free, and scales linearly with graph size.

Elements of the attack graph adjacency matrix represent all one-step attacks. We extend this by computing higher powers of the adjacency matrix, to represent multiple-step attacks. That is, the adjacency matrix of power k shows all attacker reachability within k steps of the attack. Further, we combine multiple adjacency matrix powers into a single matrix that shows the minimum number of attack steps between each pair of attack graph elements.

Alternatively, we summarize reachability for all numbers of steps, i.e., the transitive closure of the attack graph. For these multi-step adjacency matrices, we retain the reordering induced by clustering, so that patterns in the attack graph structure are still apparent.

The general approach of clustering attack graph adjacency matrices (and raising them to higher powers) provides a framework for correlating, predicting, and hypothesizing about network attacks. The approach applies to general attack graphs, regardless of what the particular graph vertices and edges represent.

For example, such attack graphs could have been formed from models of network vulnerability, or from causal relationships among intrusion detection events. Attack graph vertices could also represent aggregated sub-graphs, such as aggregation by machines and exploits between them. Overall, the techniques we develop have quadratic complexity in the size of the attack graph, for scalability to larger networks.

We apply our general approach to a vulnerability-based attack graph, in which the graph vertices (network security conditions and attacker exploits) have been aggregated to machines and exploits between them. This makes the patterns of attack clear, especially in comparison to the corresponding literally drawn graph. We show how this representation can provide a concise summary of changes in the attack graph resulting from changes in the network configuration, e.g., for what-if analysis of planned network changes or impact of real network changes.

We also place intrusion alarms in the context of a vulnerability-based multi-step attack graph reachability matrix. In this way, false alarms become apparent when they occur for pairs of machines not reachable by the attacker, based on the network configuration. Also, one can infer missed detections from alarms between machines that require multiple attack steps before compromise can occur.

We develop a graphical technique for predicting attack steps (forward and backward) on the adjacency matrix. Here, we project to the main diagonal of the matrix to match rows and columns between each attack step. This technique allows one to step forward from an attack, so that the impact of an attack can be determined and candidate attack responses can be identified. Using this technique with the multi-step reachability matrix allows candidate attack responses to be prioritized according to the number of steps required to reach victim machines. Alternatively, one can step backward from an attack to predict its origin.

Advances in automatic generation of cyber attack graphs [2][3][4][5][6][7][8][9][10] [11][12][13][14][15][16][17][18] have made it possible to efficiently compute attack graphs for realistic networks. These approaches avoid the state explosion problem by representing dependencies among state transitions (i.e., attacker exploits), rather than explicitly enumerating states. The resulting exploit dependency graphs have quadratic rather than exponential complexity, and still contain the same information (implicitly) as explicitly enumerated state graphs.

Still, when attack graphs are generated for realistic networks, using comprehensive sets of modeled attacker exploits, the resulting attack graphs can be very large. Previous approaches generally use graph drawing algorithms [19], in which vertices and edges between them are drawn according to particular aesthetic criteria.

While graphs containing many vertices have been successfully drawn, these have generally been relatively sparsely connected. In fact, much of the research in graph visualization has focused on trees, e.g., as summarized in [20]. One such approach to tree visualization is treemaps [21], a technique for showing hierarchical data in a space-constrained layout. In [22], treemaps are applied to visualization of attack reachability. But network attack graphs can exhibit dense connectivity, so that tree visualizations are not a good match.

An approach has been proposed for managing attack graph complexity through hierarchical aggregation [13], based on the formalism of clustered graphs [23]. The idea is to collapse subsets of the attack graph into single aggregate vertices, and allow interactive de-aggregation. In this approach, aggregated edges of the attack graph are hidden until they are de-aggregated or otherwise highlighted. In our approach, all graph edges are visible in a single view.

Also, a critical abstraction for the hierarchical aggregation approach is the protection domain, i.e., a fully-connected subgraph (clique) of the attack graph. To avoid the expensive clique detection operation, this approach requires prior knowledge of which sets of machines form protection domains, and in practice this knowledge may not be available. In our approach, protection domains are formed automatically, without prior knowledge.

Our approach applies information-theoretic clustering to the attack graph adjacency matrix [24]. This clustering rearranges rows and columns of the adjacency matrix to form homogeneous groups. In this way, patterns of common connectivity within the attack graph are clear, and groups (attack graph subsets) can be considered as single units. This clustering technique is fully automatic, is free of parameters, and scales linearly with graph size.

There have been approaches that view network traffic in the form of a matrix [25][26], where rows and columns might be subnets, IP addresses, ports, etc. But these approaches do not employ clustering to find homogeneous groups within the visualized matrices as we do. Also, they generally consider attack events independently of one another, as opposed to looking at sequences of events. In particular, they include none of the multi-step analyses in our approach, e.g., raising matrices to higher powers for multi-step reachability, tracing multiple attack steps by projecting to the main matrix diagonal, or predicting attack origin and impact.

The multi-step reachability matrix in our approach corresponds to the attack graph exploit distances in [12], although those distances are computed through graph traversal as opposed to our matrix multiplication. However, in the previous approach, exploit distances are not clustered or visualized; rather, they are used to correlate intrusion detection alarms. While the previous approach considers multiple steps to handle missing alerts and build attack scenarios, it does not predict attack origin and intent as in our approach.

Beyond generation of attack graphs for vulnerability analysis, there lacks a strategy for placement of intrusion detection sensors within the network infrastructure to cover known vulnerability paths. When intrusion detection sensor placement is addressed in the literature, it is usually in the context of general architectures for distributed intrusion detection, such as [27]. One paper has applied network attack modeling (based on logic programming) for placing intrusion detection sensors, for the limited case of Internet Protocol (IP) spoofing attacks. [28].

In [29], a model checker is used to find a minimal coverage of attack paths, using intrusion detection systems 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 intrusion 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 intrusion detection system 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 [29] does not identify how a minimum set of attack paths actually map to intrusion detection sensor deployment in the network infrastructure. For example, an attack from host A to host B may pass through multiple network devices, and the deployment of sensors in appropriate locations is left unanswered.

We assign intrusion detection 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 intrusion alerts according to attack distance from critical assets.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

In this section, we describe the methods, assumptions, and procedures carried out under this project. The remainder of this section is organized as follows:

- **Section 3.1:** Describes the generation of attack graphs via TVA, and how that supports optimal network security.
- **Section 3.2:** Describes our novel approach for applying adjacency matrices and other specialized matrix analyses to cyber attack graphs.
- **Section 3.3:** Describes optimal placement of intrusion detection system sensors and prioritization of intrusion alerts using attack graphs.
- **Section 3.4:** Describes our new approach for network security metrics based on attack graphs.

3.1 Topological Vulnerability Analysis

In this section, we describe the TVA approach, and show how it can be applied to provide optimal network security. The remainder of this section is organized as follows:

- **Section 3.1.1:** Describes how we build cyber attack graphs showing all possible paths of attack through a network.
- **Section 3.1.2:** Describes various applications and post-analyses of attack graphs in support of optimal network defense.

3.1.1 Building Cyber Attack Graphs

We analyze vulnerability interdependencies and build a complete map showing all possible paths of multi-step penetration into a network, organized as an attack graph. This approach is called Topological Vulnerability Analysis (TVA) [2][5][11].

TVA models the network configuration, including software, their vulnerabilities, and connectivity to vulnerable services. It then matches the network configuration against a database of modeled attacker exploits for simulating multi-step attack penetration. During simulation, the attack graph can be constrained according to user-defined attack scenarios.

TVA attack graphs map all the potential paths of vulnerability, showing how attackers can penetrate through a network. TVA identifies critical vulnerabilities and provides strategies for protection of critical network assets. This enables us to take a much more proactive stance, hardening the network before attacks occur, handling intrusion detection more effectively, and responding appropriately to attacks.

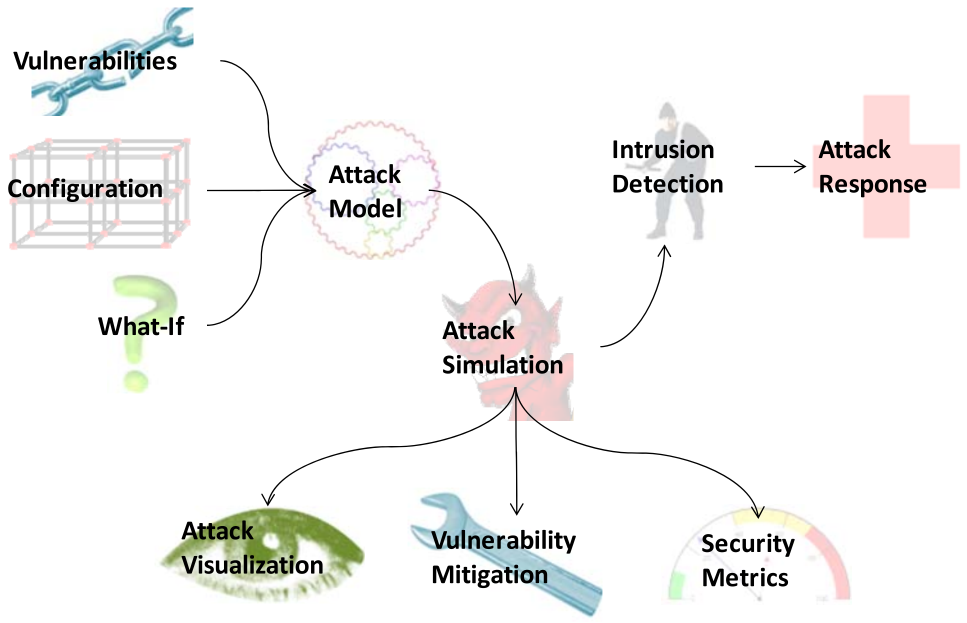


Figure 1. Overview of TVA

Figure 1 shows the overall flow of TVA. It begins by building an input attack model, based on the network configuration and potential attacker exploits. Network configuration data may include vulnerability scan reports, hosts inventory results, and firewall rules. To model incremental network penetration, we represent the fact that a given vulnerability can potentially be exploited.

From this input attack model, TVA matches modeled exploits against vulnerabilities, to predict multi-step attacks through the network. From the resulting attack graph, it then generates recommendations for optimal priority of hardening vulnerabilities. The attack graph can also be explored through interactive visualization, for more in-depth risk analysis, including "what-if" scenarios. The TVA attack graph can also support computation of various metrics for measuring overall network security.

The attack graph guides optimal strategies for preventing attacks, e.g., patching critical vulnerabilities and hardening of systems and services. But because of realistic operational constraints, such as availability of patches or the need to offer mission-critical services, there usually remain some residual attack paths through a network.

At this point, the residual attack graph provides the necessary context for dealing with intrusion attempts. This includes guidance for the deployment and configuration of intrusion detection systems, correlation of intrusion alarms, and prediction of next possible attack steps for appropriate attack response.

For example, the attack graph can guide the placement of intrusion detection sensors to cover all attack paths, while minimizing sensors redundancy. As in all cases for TVA analysis, the attack graph must be kept current with respect to changes in network vulnerabilities.

The attack graph then can filter false intrusion alarms, based on known paths of residual vulnerability. The graph also provides the context for correlating isolated alarms as part of a larger multi-step attack penetration. It also shows the next possible vulnerabilities that could be exploited by an attacker, and whether they lie on attack paths to critical network resources. This in turn supports optimal planning and response against attacks, while minimizing effects of false alarms and purposeful misdirection by an attacker.

As a simple illustration of our TVA attack graph approach, consider the small network in Figure 2. In this network, assume that the mail server and file server are for internal use only. However, outside access to the web server is needed. Thus the firewall allows incoming web connections to the web server, and blocks all other traffic from the outside. In this attack scenario, we wish to know if an attacker on the outside can compromise the mail server, through one or more attack steps.

To model this scenario, 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. In general, we rely on existing security tools to scan the network and build the input model.

For example, we could run a vulnerability scanning tool (e.g., Nessus [30]) against the hosts in the internal network to map their vulnerabilities and feed this into the TVA model. We then rely on our database of modeled exploits, pre-built to cover exploitable vulnerabilities detected by Nessus. We assume worst case, i.e., that a vulnerability is exploitable (leads to an exploit) as long as it is reported as giving sufficient control over the victim machine. This is independent of any particular code or procedure that may actually carry out such exploitation.

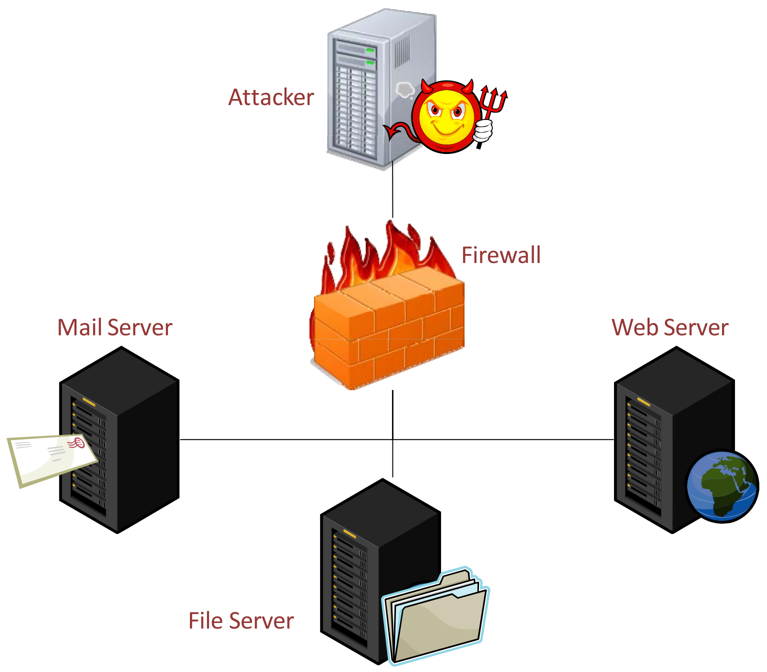


Figure 2. Small network to illustrate TVA

To incorporate the connectivity-limiting effects of the firewall, we scan through the firewall. We also scan behind the firewall, to capture vulnerabilities that are available once the attacker has reached the internal network. Alternatively, we could process the firewall rules directly for building the network model.

Figure 3 shows the resulting attack graph for this scenario. There is indeed a path from the outside to the inside mail server, via a critical vulnerability exposed through the firewall. Figure 3(a) is a high-level view of the attack graph. This shows one vulnerability being exploited (implicitly, through the firewall) from the outside to the inside. In other words, the attack graph indicates that there is one vulnerability exposed from the outside, with the potential to be exploited, allowing the attacker to progress inside. This exploit, along with all others in our model, gives the attacker the ability to execute arbitrary code at an elevated privilege.

Figure 3(b) is a more detailed view, showing that the attacker can exploit a vulnerability on the web server from the outside. Then from the web server, the attacker can attack the mail server. The box labeled "inside" represents the inside network, and implicitly all machines on the inside can exploit one another's vulnerabilities.

In the figure, the label "1" in the attack graph edge indicates that there is one exploit (implicitly, one exploitable vulnerability) from the attacker to the web server. Inside the network, there are "3 exploits," i.e., three exploitable vulnerabilities on the web server.

Thus, of the three exploitable vulnerabilities on the web server, only one of those vulnerabilities is exploitable from the outside. TVA identifies this critical vulnerability. In other words, if the single vulnerable service from attacker to web server is mitigated, the attacker has no other path to the mail server. Of course, other vulnerabilities could be mitigated as well, but the vulnerability from attacker to web server is clearly high priority.

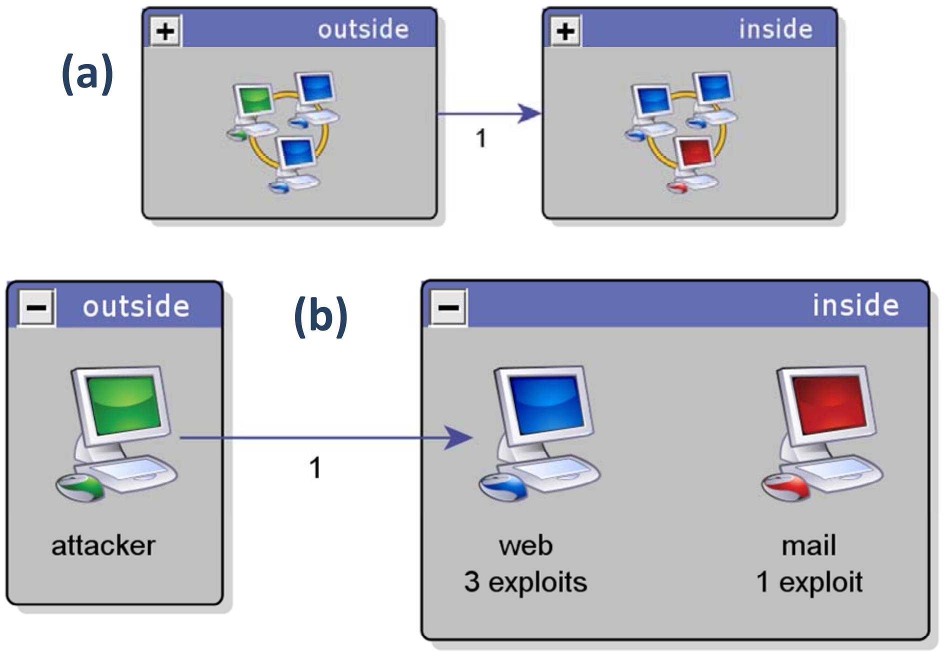


Figure 3. Attack graph for small network

This simple example 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 internal mail server from the outside because of the policy enforced by the firewall. But TVA shows that the attack goal can be reached indirectly, in this case through a sequence of two exploits. Further, it shows that addressing a single critical vulnerability from among four in the internal network would prevent this attack scenario.

By constraining the attack graph to particular start and goal points, we focus the analysis on protecting a critical asset against an assumed threat source. For example, the file server does not appear in the attack graph. This is because it does not play a part in this scenario. In other words, there are no attack paths from attacker to mail server that involve the file server.

Also, Nessus and other vulnerability scanners generate many alerts that are merely informational, and not relevant to network penetration. Our TVA approach excludes such extraneous alerts from its database of modeled exploits.

In general, many different combinations of critical vulnerabilities may prevent an attack scenario. For enterprise networks, analyzing all attack paths and drawing appropriate conclusions requires the kind of automated tool support we provide.

TVA is fundamentally a modeling and simulation approach. It relies on existing tools for gathering network configuration and vulnerability information. It also needs to be pre-populated with a database of modeled exploits that could potentially be applied to a network. So in this sense, the attack graph results are only as complete as the input model.

The benefits of a modeling/simulation approach include the ability to easily change the model for what-if analyses. But the modeling taxonomy needs to be carefully defined to reflect the realities of the network attack environment, while keeping model complexity manageable. That is, there is a tradeoff between model fidelity and model complexity that we must balance.

Also, different analysis tasks may call for variations in model details. For example, the level of detail needed for information operations support may differ from that needed for patch management. Our TVA approach can accept general models in terms of exploit preconditions/postconditions. The only requirement is to create a database of the modeled exploits, and to create network models that match exploit conditions.

3.1.2 Network Security via TVA

Security is not a one-time single-point fix, but rather a continuous process, as exemplified in the protect-detect-react life cycle. To protect from attacks, we take steps to prevent them from succeeding. Still we must understand that not all attacks can be averted in advance, and there must usually remain some residual vulnerability even after reasonable protective measures have been applied.

Indeed, the more important question is not the vulnerability itself but the magnitude of damage in case of an incident. We rely on the detect phase to identify actual attack instances. But the detection process needs to be tied to residual vulnerabilities, especially ones that lie on paths to critical network resources.

Once attacks are detected, comprehensive capabilities are needed to react to them based on vulnerability paths. We can thus reduce the impact of attacks through advance planning, by knowing the paths of vulnerability through our networks, based on pre-emptive analysis of network vulnerability scan results. To create such a proactive stance, we need to transform raw data about network vulnerabilities into attack roadmaps that help us prioritize and manage risks, maintain situational awareness, and plan for optimal countermeasures.

TVA attack graphs support proactive network defenses across the entire protect-detect-react life cycle. This includes identifying critical vulnerabilities, computing key security metrics, guiding the configuration of intrusion detection systems, correlating and prioritizing intrusion alarms, reducing false alarms, and planning optimal attack responses.

Attack graphs provide a powerful framework for proactive network defenses. A variety of analytical techniques are available for attack graphs, providing context for informed risk assessment. Attack graphs pinpoint critical vulnerabilities and form the basis for optimal network hardening.

Through sophisticated visualization techniques, purely graph-based as well as geo-spatial, we can interactively explore attack graphs. Our visualizations are designed to effectively manage graph complexity without getting overwhelmed with details. Our attack graphs also support a number of key metrics that concisely quantify the overall state of network security.

TVA automates the defense of networks against multi-step attacks. TVA attack graphs reveal the true scope of threats by mapping sequences of attacker exploits that can penetrate a network. We can then use these attack graphs to recommend ways to address the threat. This kind of automated support is critical; finding such solutions manually is tedious and error prone, especially for larger networks.

One kind of recommendation is to harden the network at the attack source, i.e., the first layer of defense. This option prevents all further attack penetration beyond the source. This is shown in Figure 4.

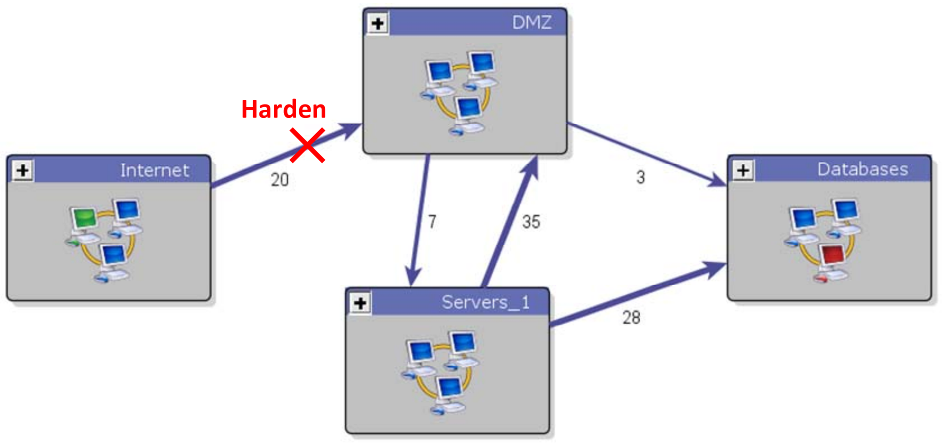


Figure 4. First-layer network hardening

Here, we use the same attack scenario, i.e., starting and ending points, as in Figure 29. However, the network configuration model is changed slightly, with a resulting change in the attack graph. In particular, the numbers of exploits between protection domains has changed.

For first-layer defense for this network configuration, the recommendation is to block the 20 exploits from Internet to Demilitarized Zone (DMZ). The idea here is not to simply rely on preventing these 20 exploits for complete protection of the network. Rather, we point out these critical first steps that give an attacker the attacker a foothold in the network. Understanding all known attack paths, not just the first layer, provides defense in depth. But we would certainly like to highlight the critical first layer.

Figure 5 shows a different kind of recommendation for network hardening, i.e., hardening the network at the attack goal at the last layer of defense. This option protects the attack goal (critical network resource) from all sources of attack, regardless of their origin. Here, as always, the assumption is that compromise of the victim (DMZ) does not imply granting of legitimate access to a subsequent victim (database server). If that is the case, such access is included as a potential attacker exploit.

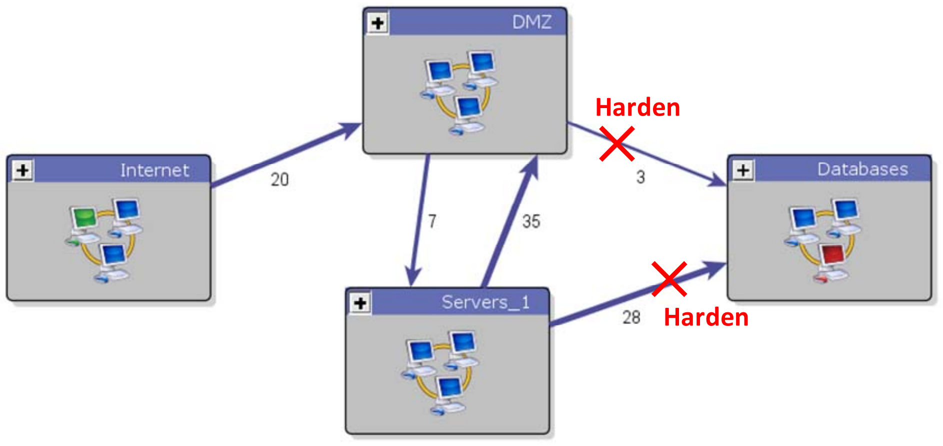


Figure 5. Last-layer network hardening

The attack graph for Figure 5 is the same as for Figure 4 (first-layer defense). For last-layer defense in Figure 5, the recommendation is to block the 3 exploits from DMZ to Databases plus the 28 exploits from Servers_1 to Databases, for a total of 31 exploits. As for the first-layer defense, we are not to simply rely on preventing these last-layer exploits for complete defense in depth. Rather, the idea is to highlight these direct attacks against critical assets, which are reachable from anywhere the attacker may be.

Another kind of recommendation is to find the minimum number of blocked exploits that break the paths from attack start to attack goal. In other words, we seek to break the graph into two components that separate start from goal, minimizing the total number of blocked exploits [10].

This is shown in Figure 6. For the minimum-cost defense, the recommendation is to block the 3 exploits from DMZ to Databases plus the 7 exploits from DMZ to Servers_1, for a total of 10 exploits. This is a savings of 10 blocked exploits in comparison to first-layer hardening, and a savings of 21 blocked exploits in comparison to last-layer hardening. As for first-layer and last-layer defenses, the idea is to highlight critical vulnerabilities that break the attacker's reach to the critical asset. After these are addressed, the residual attack graph can be analyzed for further defense in depth.

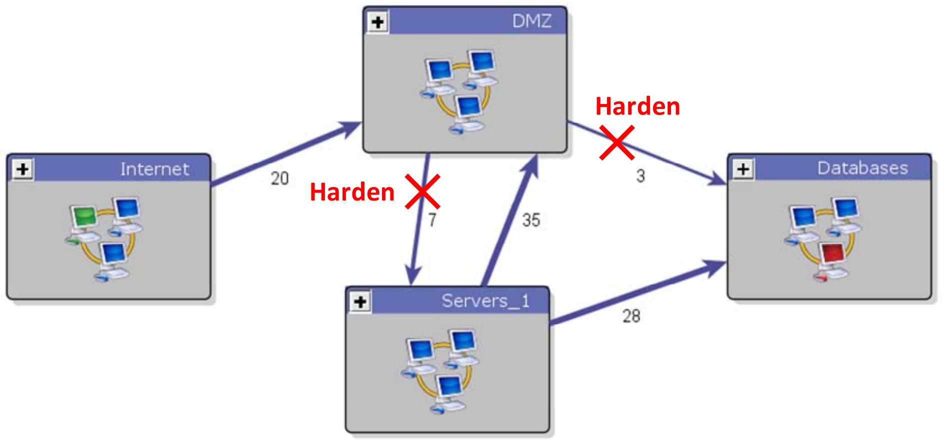


Figure 6. Minimum-cost network hardening

One of the challenges in TVA is managing attack graph complexity. In early formalisms, attack graph complexity is exponential [31][32][33][34] because paths are explicitly enumerated, leading to combinatorial explosion. Under reasonable assumptions attack graph analysis can be formulated as monotonic logic, making it unnecessary to explicitly enumerate states, leading to polynomial rather than exponential complexity [1][17][35]. Our protection domain abstraction reduces complexity further, to linear within each domain [13], and complexity can be further reduced based on host configuration regularities [36].

Thus, while it is computationally feasible to generate attack graphs for reasonably large networks, complex graphs can overwhelm an analyst. Rather than presenting attack graph data in their raw form, we present views that aid in rapid understanding of overall attack patterns. Employing a clustered graph framework, a clustered portion of the attack graph provides a summarized view, while showing interactions with other clusters. Arbitrarily large and complex attack graphs can be handled in this way, through multiple levels of clustering.

Attack graphs show how network vulnerabilities can be combined to stage an attack, providing a framework for much more precise and meaningful security metrics. Attack graph metrics can help quantify risk associated with potential security breaches, guide decisions about responding to attacks, and accurately measure overall network security. Informed risk assessment requires such a quantitative approach.

Desirable properties of metrics include being consistently measurable, inexpensive to collect, unambiguous, and having specific context. Metrics based on attack graphs have all these properties. National Institute of Standards and Technology (NIST) outlines processes for implementing security metrics [37]. The Common Vulnerability Scoring System (CVSS) [38] provides a way of scoring vulnerabilities based on standard measures. But in these cases, vulnerabilities are treated in isolation without considering their interdependencies on a target network.

In contrast, attack graph metrics are holistic measures, taking into account patterns of vulnerability paths across the network. These can also be tailored for specific attack scenarios, including assumed threat origins and/or critical resources to protect. They provide consistent measures over time, so that an organization can continually monitor security posture through the course of network operation. They can also be used for evaluating the relative security of planned network changes, so that risks can be assessed and alternatives compared well in advance of actual deployment.

One basic metric might be the overall size (vertices and edges) of the attack graph. For example, for a given attack scenario, the attack paths may constitute only a small subset of the total network vulnerabilities. This could be for a given attack starting point with the attack goal unconstrained, thus measuring the total "forward reach" of the attacker. Or it could be for a given attack goal with attack start unconstrained, measuring the "backward susceptibility" of a critical asset. Alternatively, it could be computed for constrained start and constrained goal, measuring joint attack reachability/susceptibility.

While attack graph size provides a basic indicator, it does not fully quantify levels of effort for defending against attacks. For example, the number of exploits in the first-layer hardening recommendation quantifies the effort for blocking initial network penetration. Similarly, the number of exploits in the last-layer recommendation quantifies the effort for blocking final-step critical asset compromise. The minimum-effort recommendation quantifies the overall least effort required for blocking the attacker from a critical asset.

Another idea is to normalize metrics by the size of the network, yielding a measure that could be compared across networks of different sizes. We could also extend our attack graph models to deal with uncertainties. For example, given that exploits each have individual measures of likelihood, difficulty, etc., we could propagate these through the attack graph, according to the logical implications of exploit interdependencies.

This approach could derive an overall measure for the network, e.g., likelihood of catastrophic compromise. Such a measure might then be included in more general assessments of overall business risk. We could then rank risk-mitigation options in terms of maximizing security and minimizing business cost.

This is illustrated in Figure 7. The network is scanned, providing input to the computation of a TVA attack graph. The individual risk scores for each vulnerability in the attack graph are assess, e.g., via CVSS. The structure of the attack graph implies a particular logical form for the overall attack goal. We then propagate the risk scores through the graph according to this logical structure. The result is a measure of overall risk (e.g., likelihood of compromise) that we can rank for different network configurations (attack graphs).

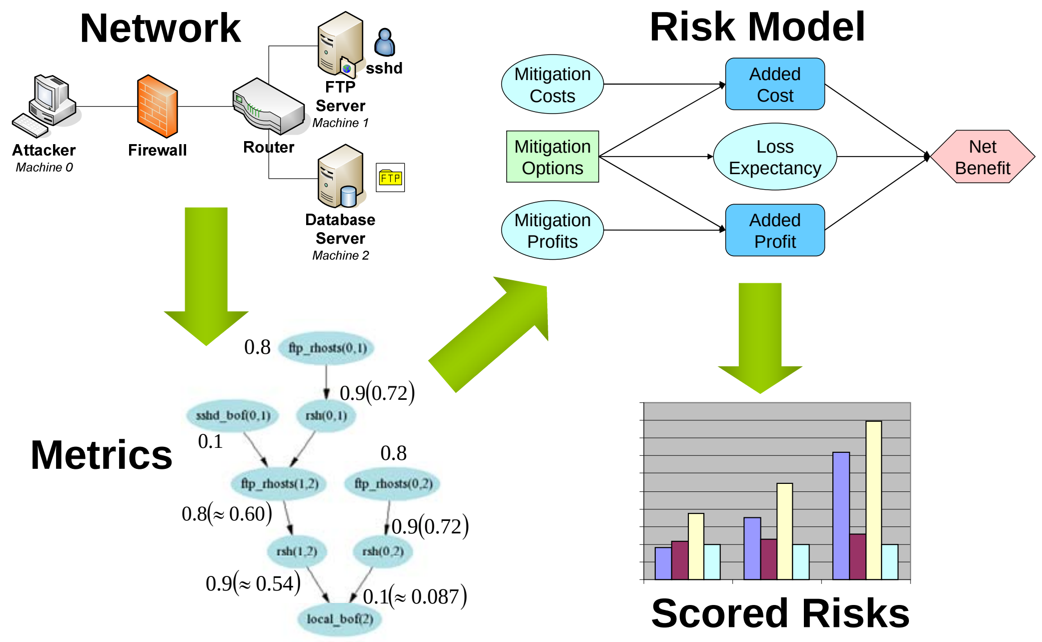


Figure 7. Propagating risk scores through TVA attack graph

The kind of precise measurement provided by attack graphs can also help clarify security requirements and guard against potentially misleading “rule of thumb” assumptions. For example, suppose there is a network with many vulnerable services, but those services are not exposed through firewalls. Then another network has fewer vulnerable services, but they are all exposed through firewalls. Comparing attack graphs, from outside the firewalls, the first network is more secure.

Or, making network host configurations more diverse, presumably to make the attacker’s job more difficult, may not necessarily improve security. For example, this may provide more paths leading to critical assets. By taking into account the diversity of configurations in our model, our attack graph metrics give precise measures for analyzing these kinds of situations.

Attack graph analysis identifies critical vulnerability paths and provides strategies for optimal protection of critical network assets. This enables us to make optimal decisions about hardening the network in advance of attack. But we must also recognize that because of operational constraints such as availability of patches and the need for offering mission-critical services, residual vulnerability paths usually remain.

The knowledge provided by TVA enables us to plan in advance and maintain a proactive security posture even in the face of attacks. For example, TVA attack graphs provide the necessary context for deployment and fine tuning of intrusion detection systems, for correlation and prioritization of intrusion alarms, and for attack response.

Knowing the paths of vulnerability through our network helps us prepare our defenses and plan our responses. This is illustrated in Figure 8.

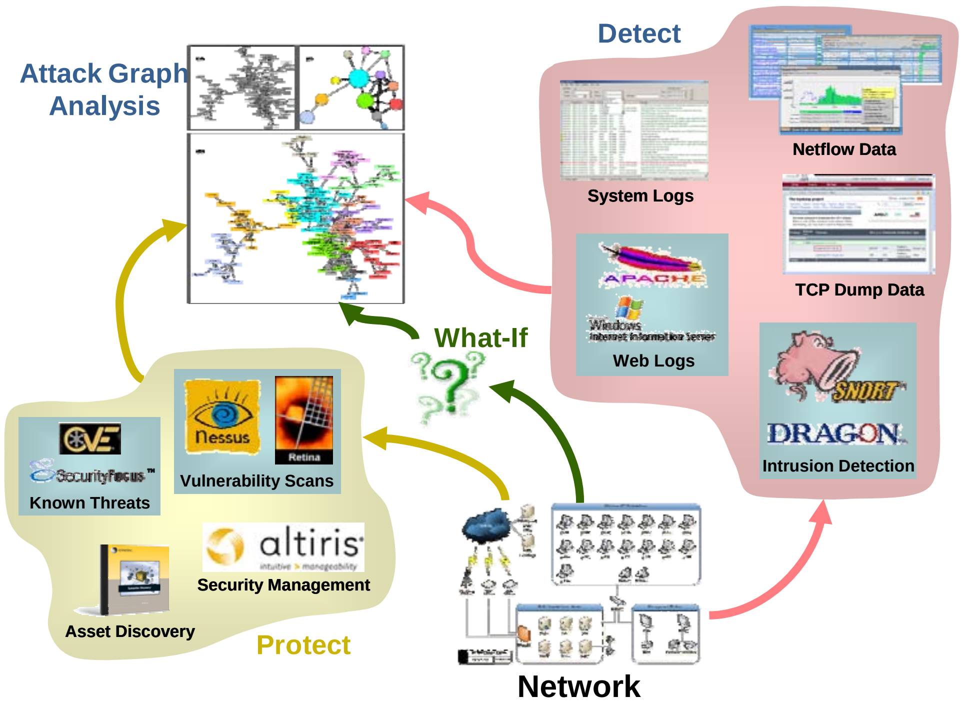


Figure 8. TVA attack graphs for protection, detection, and correlation

3.2 Attack Graph Matrices

In this section, we give an overview of our general approach for applying adjacency matrices and other specialized matrix analysis to cyber attack graphs. The remainder of this section is organized as follows:

- **Section 3.2.1:** Describes how adjacency matrices can be created for various types of cyber attack graphs.
- **Section 3.2.2:** Describes a matrix clustering algorithm that finds homogenous groups of edges (rows/columns) in the attack graph adjacency matrix.

- **Section 3.2.3:** Describes how the attack graph adjacency matrix can be transformed to represent multi-step attacks.
- **Section 3.2.4:** Describes how detected intrusions can be placed in the context of attack graph reachability matrices for predicting attack origin and impact.

3.2.1 Attack Graph Adjacency Matrices

Our approach begins with the creation of a network attack graph, through some means, based on some representation of network attacks. Our approach is very general, in that there are really no particular restrictions on the exact form of the attack graph for our approach to apply.

For example, the graph could be based on hypothetical attacker exploits generated from knowledge of vulnerabilities, network connectivity, etc., as in [2][3][4][5] [15][16][17]. Or, the attack graph could be constructed from causal relationships among intrusion detection system alarms, as in [14][18]. We can also handle intrusion alarms placed within the context of vulnerability-based attack graphs, e.g., as in [12].

Attack graphs can be created with specified starting and goal points (to constrain the graph to regions of interest), or with starting and goal points unspecified (e.g., for intrusion alarm correlation). There are dual attack graph representations [13] in which either network security conditions or attacker exploits could be the graph vertices, with the other being the graph edges. Also, subgraphs of the attack graph can be aggregated to single vertices. Our approach handles all of these situations.

Consider a simple example in where there is a set of network machines having no connectivity limitations among them, so that the attack graph is fully connected. For such a set of 200 machines, with just one vulnerable network service on each machine (vertex), there are $200^2 = 40{,}000$ exploits (edges) that must be displayed.

If such a graph were drawn with lines for edges, it would not be apparent from the resulting mass of lines that this indeed represents a fully connected attack graph. We therefore employ an adjacency matrix visualization, in which each attack graph edge is represented by a matrix element rather than by a drawn line. In our example of 200 fully connected machines each having one vulnerable service, the attack graph adjacency matrix would simply be a 200-square matrix of all ones.

Formally, for n vertices in the attack graph, the adjacency matrix A is an $n \times n$ matrix where element $a_{i,j}$ of A indicates the presence of an edge from vertex i to vertex j. In attack graphs, it is possible that there are multiple edges between a pair of vertices (mathematically, a multigraph), such as multiple conditions between a pair of exploits or multiple exploits between a pair of machines.

In such cases, we can either record the actual number of edges, or simply record the presence (0, 1) of at least one edge. The adjacency matrix records only the presence of an edge, and not its semantics, which can be considered in follow-on analysis.

As a data structure, an alternative to adjacency matrices are adjacency lists. For each vertex in the graph, the adjacency list keeps all other vertices to which it has an edge. Thus, adjacency lists use no space to record edges that are not present. There are tradeoffs (in both space and time) between adjacency matrices and lists, depending on graph sparseness and the particular operations required. Our implementation uses Matlab [39] sparse matrices (adjacency lists) for internal computations, reserving the adjacency matrix representation for visual displays.

3.2.2 Adjacency Matrix Clustering

The rows and columns of an adjacency matrix could be placed in any order, without affecting the structure of the attack graph the matrix represents. But orderings that capture regularities in graph structure are clearly desirable. In particular, we seek orderings that tend to cluster graph vertices (adjacency matrix rows and columns) by common edges (non-zero matrix elements).

This allows us to treat such clusters of common edges as a single unit as we analyze the attack graph (adjacency matrix). In some cases, there might be network attributes that allow us to order adjacency matrix rows and columns into clusters of common attack graph edges. For example, we might sort machine vertices according to IP address, so that machines in the same subnet appear in consecutive rows and columns of the adjacency matrix. Unrestricted connectivity within each subnet might then cause fully-connected (all ones) blocks of elements on the main diagonal.

In general, we cannot rely on a priori ordering of rows and columns to place the adjacency matrix into meaningful clusters. We therefore apply a particular matrix clustering algorithm [23] that is designed to form homogeneous rectangular blocks of matrix elements (row and column intersections). Here, homogeneity means that within a block, there is a similar pattern of attack graph edges (adjacency matrix elements). This clustering algorithm requires no user intervention, has no parameters that need tuning, and scales linearly with problem size.

This algorithm finds the number of row and column clusters, along with the assignment of rows and columns to those clusters, such that the clusters form regions of high and low densities. Numbers of clusters and cluster assignments provide an information-theoretic measure of cluster optimality.

The matrix clustering algorithm is based on ideas from data compression, including the Minimum Description Length principle [40], in which regularity in the data can be used to compress it (describe it in fewer symbols). Intuitively, one can say that the more we compress the data, the better we understand it, in the sense that we have better captured its regularities.

3.2.3 Matrix Operations for Multi-Step Attacks

The adjacency matrix shows the presence of each edge in a network attack graph. Taken directly, the adjacency matrix shows every possible single-step attack. In other words, the adjacency matrix shows attacker reachability within one attack step. As we describe later, we can navigate the adjacency matrix by iteratively matching rows and columns to follow multiple attack steps. We can also raise the adjacency matrix to higher powers, which shows multi-step attacker reachability at a glance.

For a square ($n \times n$) adjacency matrix A and a positive integer p, then A^p is A raised to the power p: In other words,

$$A^p = \underbrace{AA \cdots A}_{p \text{ times}} \quad . \tag{1}$$

Here, matrix multiplication is in the usual sense. For example, an element of A^2 is

$$\left(A^2\right)_{ij} = \sum_k a_{ik} \cdot a_{kj} \; . \tag{2}$$

In Equation (2), the matching of rows and columns in matrix multiplication (index k) corresponds to matching steps of an attack graph. The summation over k counts the numbers of matching steps. Thus, each element of A^2 gives the number of 2-step attacks between the corresponding pair (row and column) of attack graph vertices. Similarly, A^3 gives all 3-step attacks, A^4 gives all 4-step attacks, etc.

For raising a (square) matrix to an arbitrary power, we can improve upon naïve iterative multiplication. This involves a spectral decomposition [41] of A. An $n \times n$ matrix always has n eigenvalues. These form an $n \times n$ diagonal matrix D and a corresponding matrix of nonzero columns V that satisfies the eigenvalue equation $AV = VD$. If the n eigenvalues are distinct, then V is invertible, so that we can decompose the original matrix A as

$$A = VDV^{-1}. \tag{3}$$

Here D is a diagonal matrix formed from the eigenvalues of A, and the columns of V are the corresponding eigenvectors of V.

It is then straightforward to prove that $A^p = VD^pV^{-1}$, via $V^{-1}V = I$. This product VD^pV^{-1} is easy to compute since D^p is just the diagonal matrix with entries equal to the pth power of those of D, i.e.,

$$D^p = \begin{bmatrix} d_1^p & 0 & \cdots & 0 \\ 0 & d_2^p & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & d_n^p \end{bmatrix}. \tag{4}$$

In our matrix multiplication, if we calculate the Boolean product rather than the numeric product, the resulting A^p simply tells us whether there is at least one p-step attack from one vertex to another, rather than the actual number of such paths. Thus, the Boolean sum

$$A \vee A^2 \vee A^3 \vee \ldots \vee A^{n-1} \tag{5}$$

tells us, for each pair of vertices (matrix elements), whether the attacker can reach one attack graph vertex to another over all possible numbers of steps.

The Boolean sum in Equation (5) is known as the *transitive closure* of A. The classical Floyd-Warshall algorithm computes transitive closure in $O(n^3)$, although there are improved algorithms, e.g., [42], that come closer to $O(n^2)$.

Frequently in practice, elements of A^p monotonically increase as p increases. In such cases, we can distinguish the minimum number of steps required to reach each pair of attack graph vertices by computing the multi-step reachability matrix

$$A + A^2 + A^3 + \ldots + A^{n-1}. \tag{6}$$

Here the matrix multiplication is Boolean and the summation is simply arithmetic. Since elements of A^p increase monotonically from zero to one (under Boolean matrix multiplication), the elements of the reachability matrix in Equation (6) give the minimum number of steps required to reach one attack graph vertex to another.

A fundamental property of attack graphs is how well connected the various graph vertices (exploits, machines, etc.) are. For example, attack graphs that have few or weak (large multi-step only) connections are easier to defend against, and those with more and stronger connections are more difficult to defend against.

Knowing the numbers and depths of attacks (e.g., through higher powers of the adjacency matrix) helps us understand large-scale tendencies across the network. Individual vertices' roles within the attack graph are also described by their numbers and depths of attacks to other vertices. For example, vertices (e.g., machines) with many attack paths through them might bear closer scrutiny. Or, we can identify critical "bottleneck" vertices in the attack graph.

3.2.4 Attack Prediction

In our approach, we place detected intrusions within the context of predictive attack graphs based on known vulnerability paths. We first compute a vulnerability-based attack graph from knowledge of the network configuration, attacker exploits, etc. We then form the adjacency matrix A for the attack graph, perform clustering on A. We then compute either the transitive closure of A, or the multi-step reachability matrix in Equation (6).

Then, when an intrusion alarm is generated, if we can associate it with an edge (e.g., exploit) in the attack graph, we can thus associate it with the corresponding element of any of the following:

- The adjacency matrix *A* (for single-step reachability).
- The multi-step reachability matrix in Equation (6) (for multi-step reachability).
- The transitive closure of *A* (for all-step reachability).

From this, we can immediately categorize alerts based on the numbers of associated attack steps. For example, if an alarm occurs within a zero-valued region of the transitive closure, we might conclude it is a false alarm, i.e., we know it is not possible according to the attack graph.

Or, if an alarm occurs within a single-step region of the reachability matrix, we know that it is indeed one of the single-step attacks in the attack graph. Somewhere in between, if an alarm occurs in a *p*-step region, we know the attack graph predicts that it takes a minimum of *p* steps to achieve such an attack.

By associating intrusion alarms with a reachability graph, we can also predict the origin and impact of attacks. That is, once we place intrusion alarm on one of the vulnerability-based reachability graphs, we can navigate the graph to do attack prediction.

The idea is to project to the main diagonal of the graph, in which row and column indices are equal. Vertical projection (along a column) leads to attack step(s) in the forward direction. That is, when one projects along a column to the main diagonal, the resulting row gives the possible steps forward in the attack.

We can predict attack origin and impact either (1) one step away, (2) multiple steps away with the number of steps distinguished, or (3) over all steps combined. Here are those 3 possibilities:

- When using the adjacency matrix *A*, non-zero elements along the projected row show all possible single steps forward. Projection also can be done iteratively, to follow step-by-step (one at a time) in the attack.
- When using the multi-step reachability matrix in Equation (6), the projected row shows the minimum number of subsequent steps needed to reach another vertex. We can also iteratively project, either choosing single-step elements only, or “skipping” steps by choosing multi-step elements.
- When using the transitive closure, the projected row shows whether a particular vertex can be subsequently reached in any number of steps. Here, iterative projection is not necessary, since transitive closure shows reachability over all steps.

We see that projection along a column of a reachability matrix predicts the impact (forward steps) of an attack. Correspondingly, we can project along a row (as opposed to a column) of such a matrix to predict attack origin (backward steps).

In this case, when one projects along a row to the main diagonal, the resulting column gives the possible steps backward in the attack. As before, we can predict attack origin using either the adjacency matrix, the multi-step reachability matrix, or the transitive closure matrix. Just as for forward projection, this gives either single-step reachability, multi-step reachability, or all-step reachability, but this time in a backward direction for predicting attack origin.

3.3 Optimal Intrusion Sensor Placement

This section describes the optimal placement of intrusion detection system sensors and the prioritization of intrusion alerts using attack graphs. In this approach, 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 intrusion detection 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 remainder of this section is organized as follows:

- **Section 3.3.1:** Provides motivation and states the optimal sensor-placement problem.
- **Section 3.3.2:** Gives an overview of our novel approach to optimal sensor placement via attack graphs.
- **Section 3.3.3:** Describes how our attack graphs provide the constraints sufficient sensor coverage.

3.3.1 Statement of Problem

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, intrusion detection 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 intrusion detection sensor deployment. Moreover, intrusion detection systems usually report all potentially malicious traffic, without regard to the actual network configuration, vulnerabilities, and mission impact. Given large volumes of network traffic, intrusion detection systems 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.

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 of intrusion detection 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 intrusion 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.

3.3.2 Overview of Approach

Because attackers can exploit vulnerabilities as stepping stones to new vantage points, considering network components and vulnerabilities in isolation is clearly insufficient. Our approach discovers multi-step attacks, modeling network penetration as real attackers might do. We compute an attack graph showing all possible paths through a network. We use attack graphs to place intrusion detection sensors that cover these predicted paths, and use this context for prioritizing intrusion alerts. This is illustrated in Figure 9.

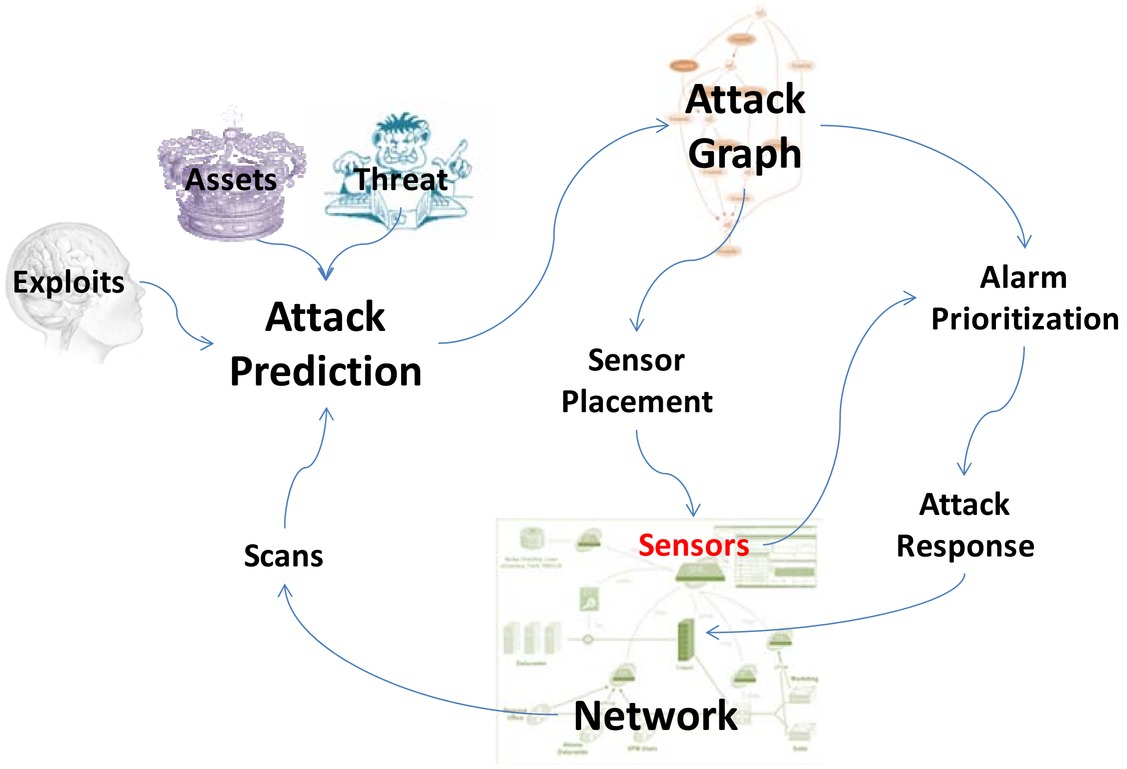


Figure 9. Intrusion detection sensor placement via attack graphs

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. 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 TVA 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, Retina, and FoundScan) 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. 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 intrusion detection sensors to monitor traffic and detect potentially malicious activity along these paths.

Our placement of intrusion detection sensors is optimal, in the sense that a network's attack graph is entirely covered, using the fewest required number of sensors. This sensor placement problem is an instance of the classical set cover problem [43], 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 intrusion detection system 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.

3.3.3 Predictive Attack Graphs

Figure 10 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.

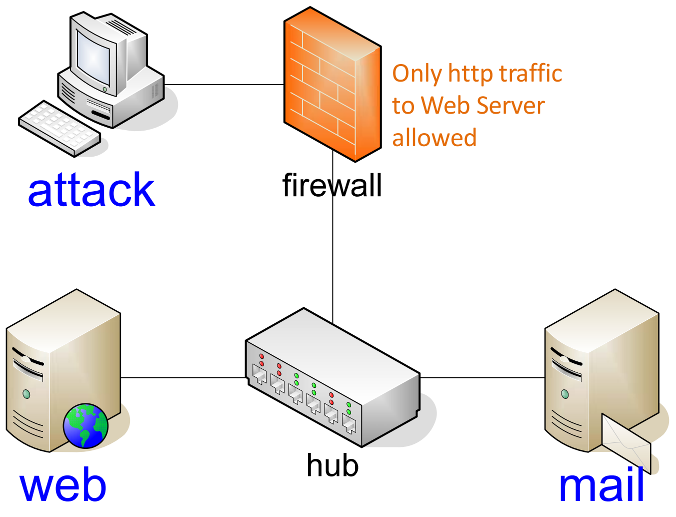


Figure 10. Small testbed network for demonstrating attack graph analysis

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.

To capture the network configuration for Figure 10, 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.

The TVA attack simulation begins on the outside machine, and ends with the compromise of the mail server. Figure 11 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.

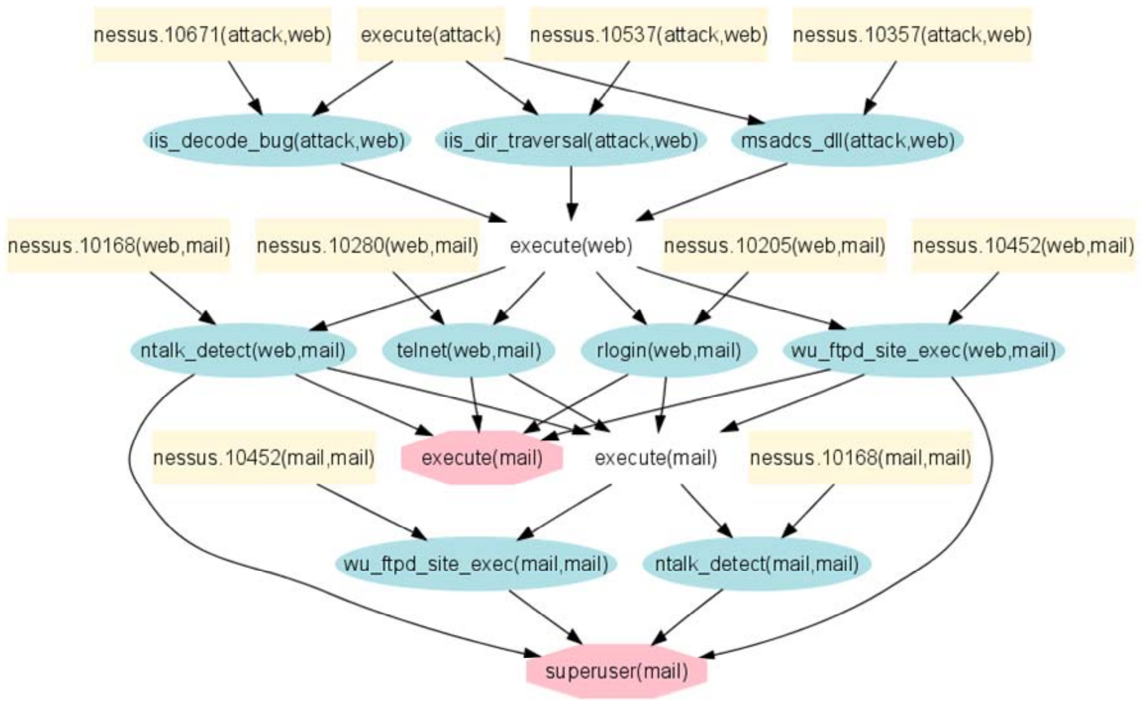


Figure 11. Attack graph for testbed network in Figure 10

So for example, initial condition *nessus.10671(attack, web)* means that the web server has Nessus vulnerability number 10671 (IIS Remote Command Execution) (also identified as CVE-2001-0333 and CVE-2001-0507 under MITRE's Common Vulnerabilities and Exposures), and that the attack machine can connect to that vulnerable service on the web server.

In Figure 11, 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 11, 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 11, removing Nessus vulnerabilities 10671, 10537, and 10357 on the web server would stop the attack. This is shown as Solution 1 in Figure 12. 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). This is shown as Solution 2 in Figure 12.

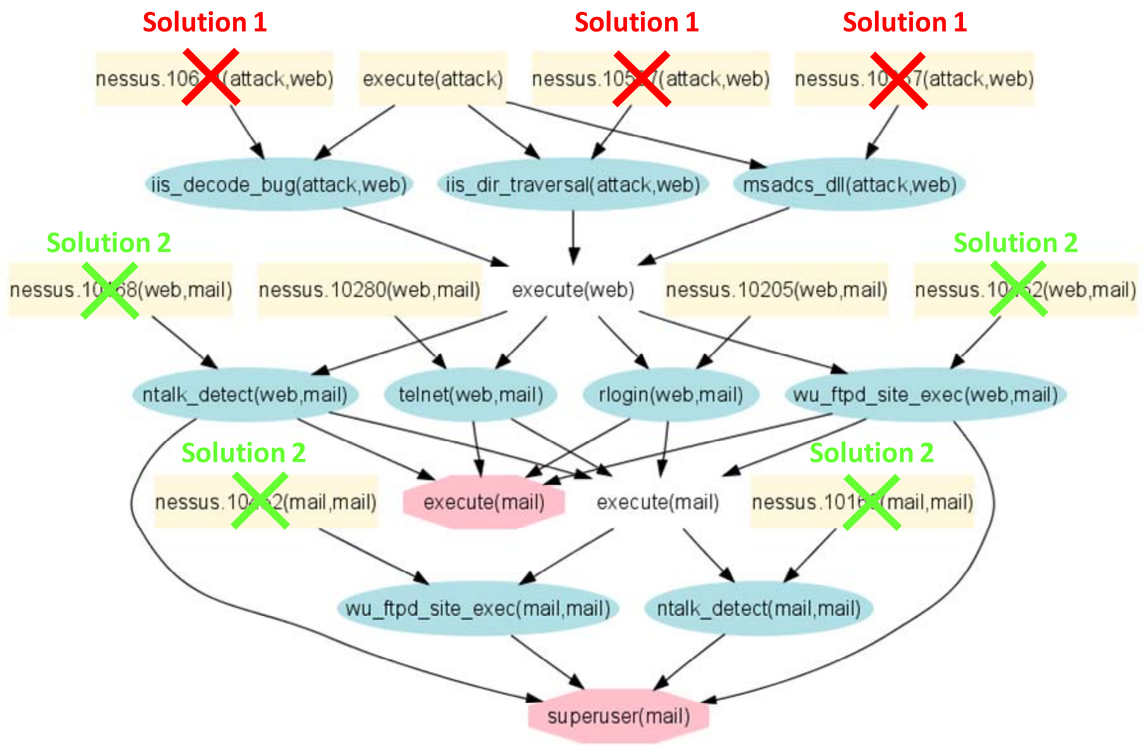


Figure 12. Recommended solutions for hardening testbed network

Of course in practice, network attack graphs are usually much more complex than Figure 2. For example, Figure 13 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.

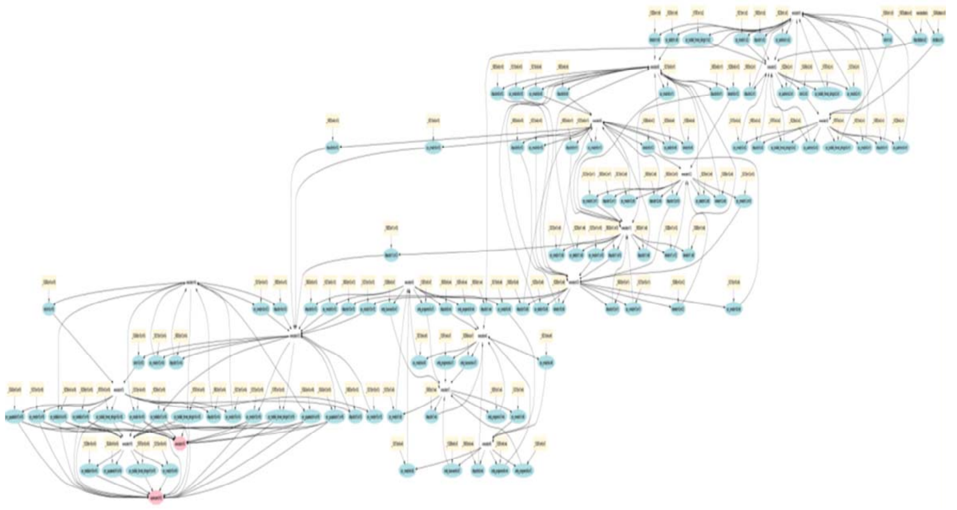


Figure 13. More complex attack graph for 17-machine operational network

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 14.

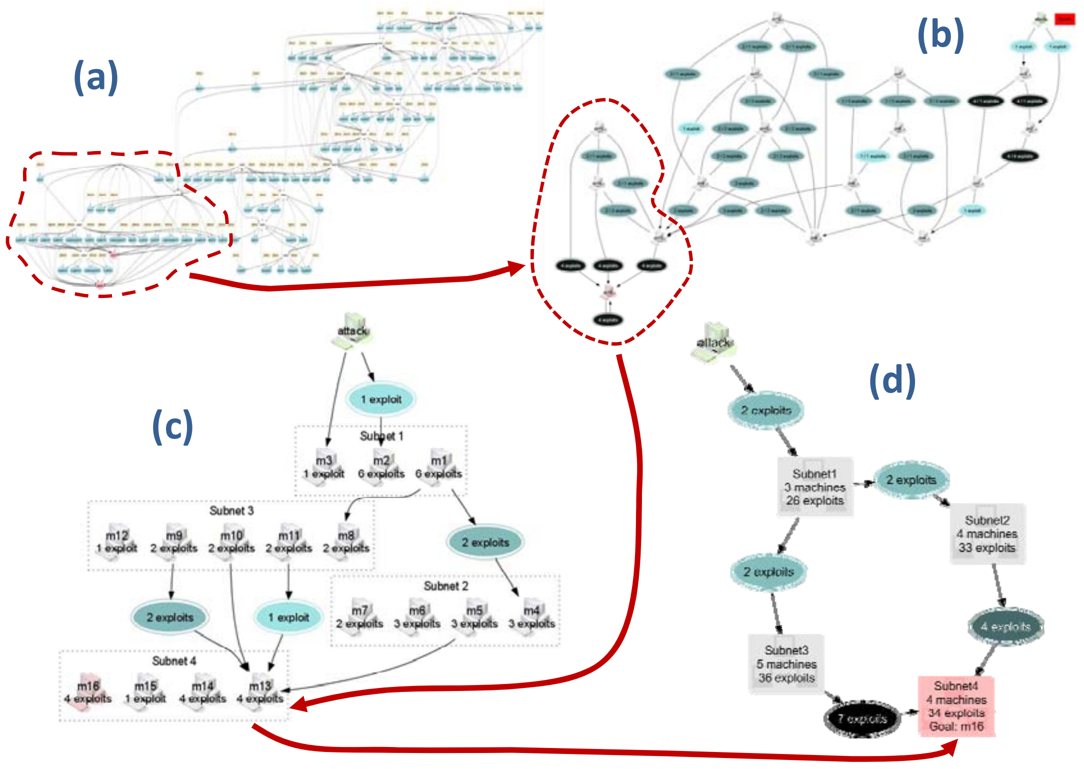


Figure 14. Aggregation of complex attack graph over multiple levels of detail

Figure 14(a) is the original attack graph, showing all details. In Figure 14(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 14(a) are collapsed to a single "machine" vertex in Figure 14(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 14(b). Figure 14(b) thus represents a summary of Figure 14(a), providing more of an overview of the attack.

In Figure 14(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 14(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 14(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 14(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.

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 15, 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.

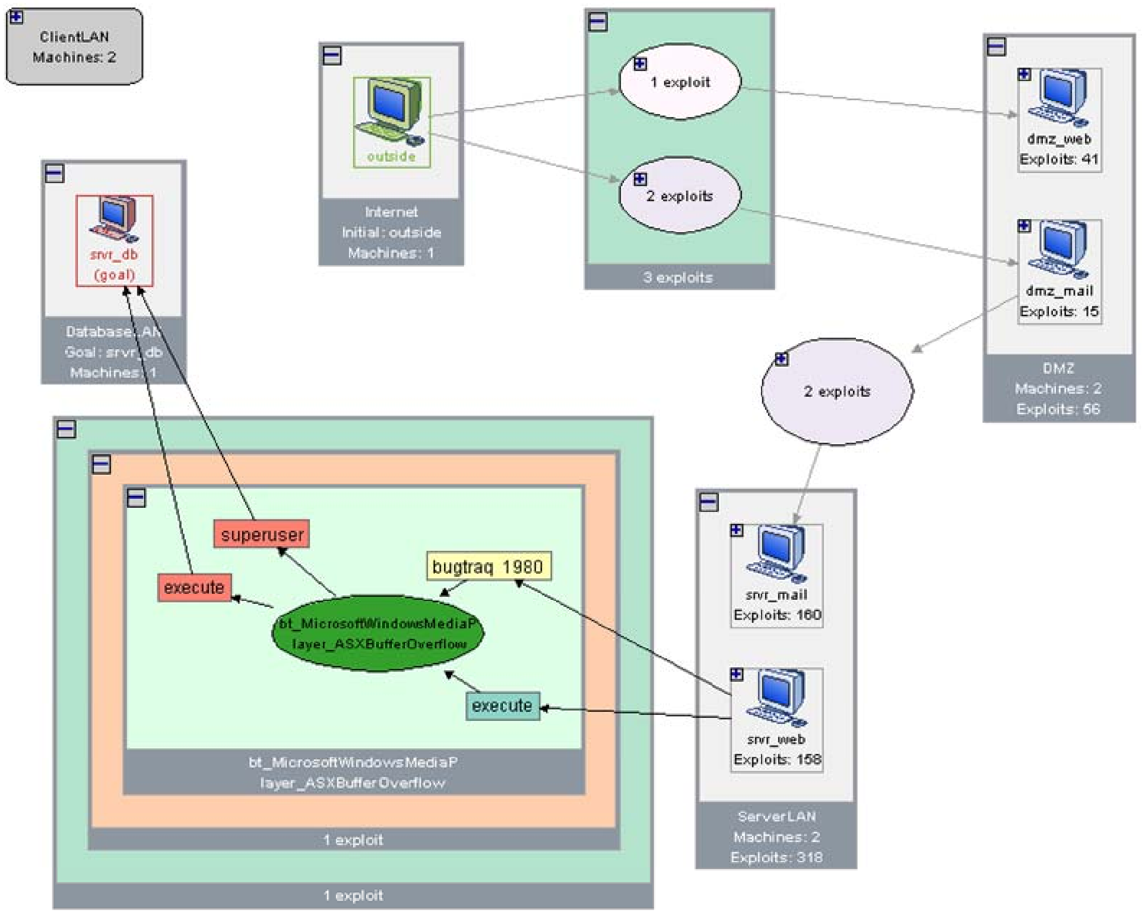


Figure 15. TVA tool attack graph visualization for 8-machine testbed network

In Figure 15, 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 Local Area Network (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.

3.4 Security Metrics from Attack Graphs

Today's information systems face sophisticated attackers who combine multiple vulnerabilities to penetrate networks with devastating impact. The overall security across a network cannot be determined by simply counting the number of vulnerabilities. To accurately assess the security of networked systems, we must understand how vulnerabilities can be combined to stage an attack.

The remainder of this section is organized as follows:

- **Section 3.4.1:** Gives an overview of our novel approach to network security metrics based on attack graphs.
- **Section 3.4.2:** Describes our particular attack graph model for computing security metrics.
- **Section 3.4.3:** Explains how we propagate individual vulnerability scores through the attack graph.

3.4.1 Overview of Approach

Practitioners have traditionally taken a binary view of information security, leading to endless reassessments, disagreement, and gridlock. In this view, the system is either secure or it is not secure. The reality is that in any complex system, there will be elements with differing degrees of integrity, value, or exposure to threats. Decision makers in diverse areas of business and engineering use metrics for determining whether a projected return on investment justifies its costs. Providing security for networks is such an investment.

We describe a powerful new metrics-based approach for managing security risks in networked environments. Through cost-benefit analysis of security options, we answer such questions as "How much security is enough?" or "How should we best invest our limited security resources?" Our approach explicitly incorporates uncertainty, supports varying levels of modeling detail, and provides measures of confidence. Based on results of simulated incremental network penetration, we measure overall probability and cost of cyber attacks. In this way we score candidate mitigation choices in terms of maximized security and minimized costs, and make recommendations for optimal choices.

We populate our models automatically from network scans and threat-management services. The data in our model also include prior expert judgments of ease, availability, and other measured dimensions of known vulnerabilities. We compute probabilistic network attack models that account for potential threats, specific network vulnerabilities and their interactions, and protection of mission-critical resources. In this way, we break free from rigid binary modeling, to handle uncertainties such as relative likelihood of an attack.

By increasing security spending, an organization can decrease the risk associated with security breaches. Such tradeoff analysis requires quantitative rather than qualitative models. Previous approaches to security metrics have focused on individual vulnerabilities. Traditional metrics such as percentage of patched systems ignore interactions among network vulnerabilities. Such metrics are limited, because vulnerabilities in isolation lack context. Attackers can combine related vulnerabilities to incrementally penetrate information systems, potentially leading to devastating consequences.

We meet this challenge by capturing vulnerability interdependencies, measuring security in the exact way that real attackers could penetrate information systems. We analyze all attack paths through a system, providing a metric of overall system risk. Through this metric, we analyze tradeoffs between security costs and security benefits. Decision makers can therefore avoid investing too much in security measures that do not pay off. Similarly, they can understand when investing too little for security leads to unacceptable risk of disaster. Our metric is consistent, unambiguous, makes underlying assumptions explicit, and provides context for understanding security risk alternatives.

3.4.2 Attack Graph Model

Attack graphs model how multiple vulnerabilities may be combined for an attack. They represent system states using a collection of security-related conditions, such as the existence of vulnerability on a particular host or the connectivity between different hosts. Vulnerability exploitation is modeled as a transition between system states.

As an example, consider Figure 16. The left side shows a network configuration, and the right side shows the attack graph for compromise of the database server by a malicious workstation user. In the network configuration, the firewall is intended to help protect the internal network. The internal file server offers File Transfer Protocol (FTP), Secure SHell (SSH), and Remote SHell (RSH) services. The internal database server offers ftp and rsh services. The firewall allows ftp, ssh, and rsh traffic from a user workstation to both servers, and blocks all other traffic.

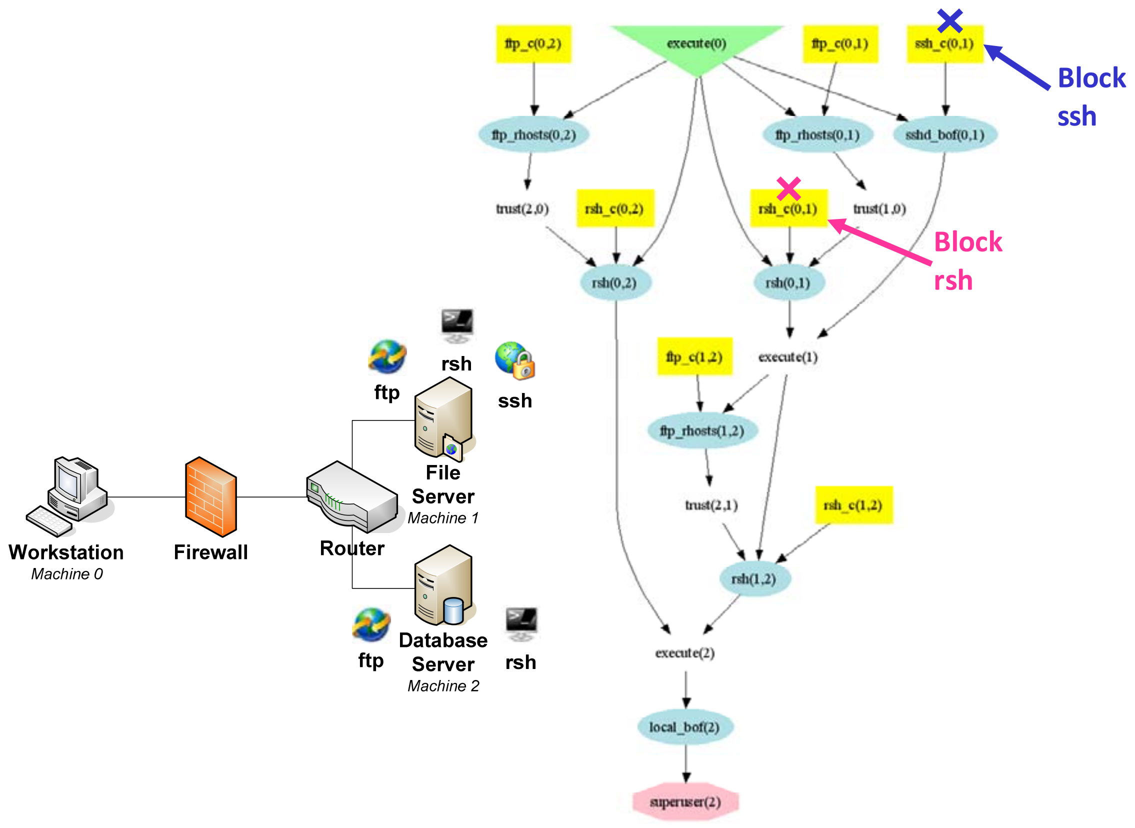


Figure 16. Example network, attack graph, and network hardening choices

In the attack graph, attacker exploits are blue ovals, with edges for their preconditions and postconditions. The numbers inside parentheses denote source and destination hosts. Yellow boxes are initial network conditions, and the green triangle is the attacker's initial capability. Conditions induced by attacker exploits are plain text. The overall attack goal is a red octagon. The figure also shows the impact of blocking ssh or rsh traffic (to the fileserver) through the firewall, i.e., preventing certain exploits in the attack graph.

The graph includes these attack paths:

1. *sshd_bof(0,1)* → *ftp_rhosts(1,2)* → *rsh(1,2)* → *local_bof(2)*
2. *ftp_rhosts(0,1)* → *rsh(0,1)* → *ftp_rhosts(1,2)* → *rsh(1,2)* → *local_bof(2)*
3. *ftp_rhosts(0,2)* → *rsh(0,2)* → *local_bof(2)*

The first attack path starts with *sshd_bof(0,1)*. This indicates a buffer overflow exploit executed from *Machine 0* (the workstation) against *Machine 1* (the file server), i.e., against its secure shell service. In a buffer overflow attack, a program is made to erroneously store data beyond a fixed-length buffer, overwriting adjacent memory so that it gets executed as code. The result of the *sshd_bof(0,1)* exploit is that the attacker can execute arbitrary code on the file server.

The *ftp_rhosts(1,2)* exploit is now possible, meaning that the attacker exploits a particular ftp vulnerability to anonymously upload a list of trusted hosts from *Machine 1* (the file server) to *Machine 2* (the database server). The attacker can leverage this new trust to remotely execute shell commands on the database server, without providing a password, i.e., *rsh(1,2)*. This exploit establishes attacker control over the database server, as a user with regular privileges.

A local buffer overflow exploit is then possible on the database server, which runs in the context of a privileged process. The result is that the attacker can execute code on the database server with full privileges.

In practice, attack graphs are usually much more complex than Figure 16. We can take advantage of certain regularities in graph structure to reduce complexity. An important attack graph abstraction is the protection domain, representing parts of the system with unrestricted access within the domain. An example is a managed subnet that has no device such as firewall that restricts connectivity within the subnet.

Using a monotonicity argument [17], we can reduce attack graph complexity as shown in Figure 17. We begin with a specified starting condition for the attack, and an attack goal condition. Exploits that are possible in one step (distance *d* = 1) from the start are then induced. From each of these first-layer exploits, a new layer (*d* = 2) is induced.

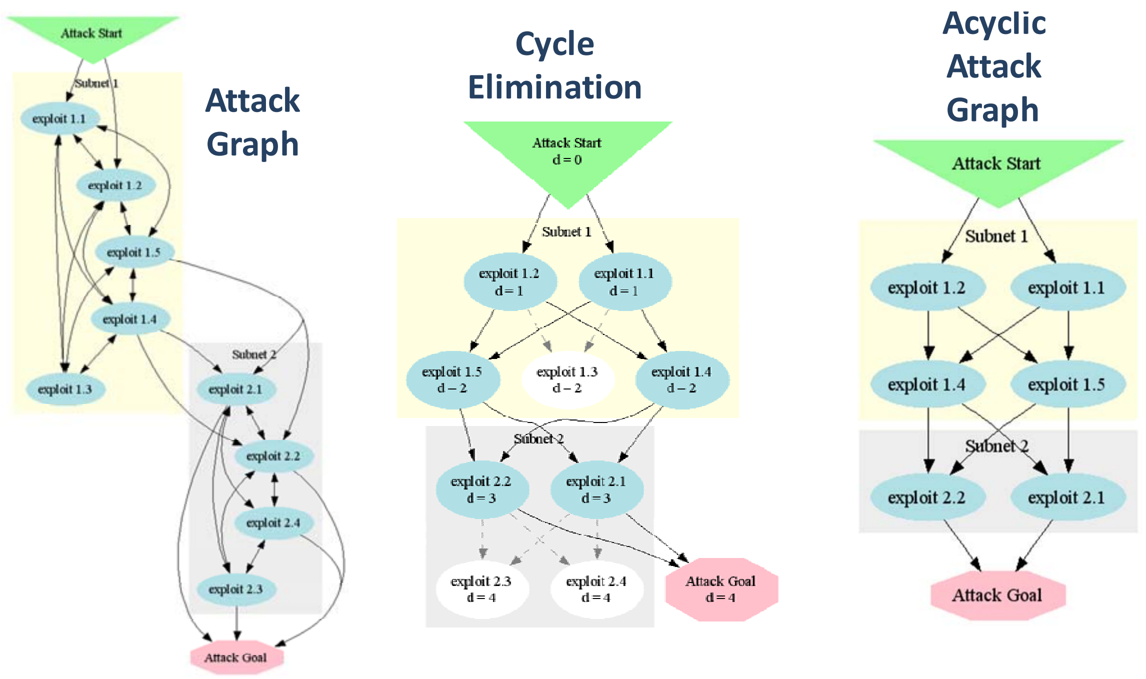


Figure 17. Removing attack graph cycles for fully-connected subnets

From a monotonicity standpoint, no backtracking is needed, so any possible edges from *d* = 2 to *d* = 1 are omitted. This is iterated through each layer of possible exploits, until the attack goal is reached. Then any exploits not backward reachable from the goal are removed. The result is an acyclic attack graph, from start to goal, in which we can propagate attack graph metrics.

In our approach, causal relationships among exploits are based on a well-accepted attack graph model. Attack graphs can be generated using our TVA attack graph tool. For practical applications in building attack graph models, the TVA tool integrates with vulnerability scanners including Nessus, Retina, and FoundScan. It also integrates with the Sidewinder firewall to capture network connectivity to vulnerable host services, and with Centennial Discovery asset inventory for gathering host configuration data.

The existence of our attack graph tool and knowledge base justifies the practicality of our attack graph metrics. We adopt a straightforward approach towards the logical relationships among exploits and their likelihoods. The information our system needs already exists in various standards and commercial products (integrated through our TVA tool), further supporting the practicality of our approach.

3.4.3 Propagating Vulnerability Scores

In practice, vulnerabilities often remain in a network, even after the vulnerabilities are discovered. Vendors may be slow to release software patches, or deployment may be delayed because of excessive cost. Attackers often leverage even correctly functioning services to gain new capabilities. An organization will often trade security risk for availability of services.

Our attack graph metric propagation quantifies this risk, based on strength that such residual paths may eventually be realized by attackers. When a network is more secure, attack strength is reduced and our attack graph metric is lower. Preventing exploits removes certain paths, in turn reducing attack strength. When the attacker cannot reach the goal, our metric is zero. When the attacker is completely assured of reaching the goal, the metric is unity.

Our approach to computing an overall attack graph security metric relies on metrics for individual system vulnerabilities as input. Given such a measure of risk of each vulnerability as input, we then combine individual measures according to the logical structure of the attack graph. In particular, conjunctive relationships (Boolean ANDs) decrease attack strength, since multiple conditions are required. Disjunctive relationships (ORs) reduce attack strength, since only one condition among multiple options is required.

When metric values are combined conjunctively, the combined metric is lower than either input. Intuitively, when both conditions must be met for an attack to succeed, more is required from the attacker, and the overall system is more secure. Alternatively, when values are combined disjunctively, the combined metric is higher than either input. In this case, when only one condition must be met (from among multiple options) for an attack to succeed, less is required from the attacker, and the overall system is less secure.

Our attack graphs encode both conjunctive and disjunctive attack relationships. For example, in Figure 16, the attacker cannot upload the list of trusted hosts if the ftp connection does not exist; neither can this happen if the attacker cannot use *Machine 1* as a normal user. Such a relationship is conjunctive. On the other hand, if a condition can be satisfied in more than one way, it does not matter which path the attacker follows to satisfy it, making the relationship disjunctive.

To compute our attack graph metric, we propagate measures of exploited vulnerabilities through the attack graph, from start to goal, according to conjunctive and disjunctive dependencies. When one exploit must follow another in a path, this means both are needed to eventually reach the goal (conjunction), so their measures are multiplied. That is, given exploits E_1 and E_2, with corresponding normalized likelihoods $p(E_1)$ and $p(E_2)$, the combined conjunctive likelihood is

$$p(E_1 \text{ and } E_2) = p(E_1)p(E_2) \tag{7}$$

Such a conjunctive relationship also exists when postconditions of one or more exploits are all required as preconditions for a given exploit.

When a choice of paths is possible, *either* is sufficient for reaching the goal (disjunction). In this case, we have

$$p(E_1 \text{ or } E_2) = p(E_1) + p(E_2) - p(E_1)p(E_2) \tag{8}$$

Such a disjunctive relationship exists when any *one* of a set of exploit postconditions will satisfy an exploit precondition.

In general, paths coming into an exploit may form arbitrary logical expressions. In such cases, we propagate exploit measures through corresponding conjunctive/disjunctive combinations. Our model is agnostic with respect to input metrics, as long as units are consistent. For input metrics representing attack likelihood, suitably normalized from zero to one, the resulting attack graph metric is the overall likelihood that the attacker can reach a given attack goal.

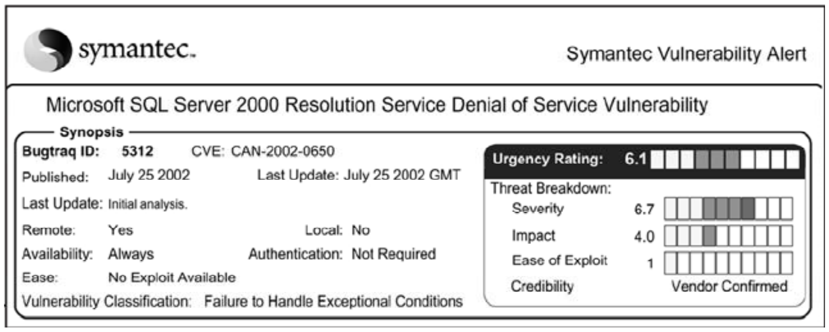
In our model, vulnerability scores in general could be Boolean, real-numbers, or distributions of values. In the case of Boolean scores, a vulnerability is either certain to exist (value of unity), or certain to not exist (value of zero), with a corresponding impact on conjunctive/disjunctive combinations.

Real numbers measure attack strength (likelihood, exploitability, impact, etc.) against an attack goal, in the given input metric units. Distributions are combined according to the same conjunctive/disjunctive formulas, point-wise for each possible value of input distributions. For example, when an input vulnerability has multiple possible metric values, a range of high and low values could be used for defining a distribution as input to our model.

One potential source of input vulnerability metrics for our model is CVSS [38]. CVSS is a widely adopted open framework for scoring the risk associated with information system vulnerabilities, part of the U.S. government's Security Content Automation Protocol (SCAP) [44]. CVSS provides a consistent quantitative score for each vulnerability, as well as underlying the qualitative vulnerability characteristics used to generate each score. A comprehensive set of CVSS scores is provided in the National Vulnerability Database (NVD) [45].

The CVSS Score is a composite of three groups of metrics: Base, Temporal, and Environmental. The Base Score represents the innate characteristics of each vulnerability, in terms of exploitability and impact. The Temporal Score covers any time-dependent vulnerability factors, including availability of exploit techniques/code, existence of patches, and degree of confidence in the report. The Environmental Score considers factors within a particular information system environment, such as collateral damage and proportion of vulnerable systems.

On the commercial front, one potential source of individual vulnerability risk scores as input to our model is the DeepSight Threat Management System [49]. DeepSight covers 18,000 distinct versions of 4,200 products from 2,200 vendors, based on 150 authoritative sources, at an average rate of 50 new vulnerabilities per week. It includes risk scores in the areas of urgency, severity, impact, and ease of exploitation, as shown in Figure 18.


symantec.

Symantec Vulnerability Alert

Microsoft SQL Server 2000 Resolution Service Denial of Service Vulnerability

Synopsis

Bugtraq ID: 5312 CVE: CAN-2002-0650

Published: July 25 2002 Last Update: July 25 2002 GMT

Last Update: Initial analysis.

Remote: Yes Local: No

Availability: Always Authentication: Not Required

Ease: No Exploit Available

Vulnerability Classification: Failure to Handle Exceptional Conditions

Urgency Rating: 6.1

Threat Breakdown:

Severity	6.7
Impact	4.0
Ease of Exploit	1
Credibility	Vendor Confirmed

Figure 18. DeepSight vulnerability scoring

DeepSight reports scope of access (remote/local) required for exploiting a vulnerability, as well as authentication needed by the attacker. It also reports on availability of software needed by the attacker. An Impact sub-score indicates the degree of compromise against the system. A Severity score combines Impact, Availability, Authentication, and Remote. Severity is then combined with indicators of ease of exploitation and report credibility, for an overall Urgency rating for each vulnerability.

Such reported vulnerability risk indicators can be useful as inputs to our attack graph metrics model. Our ongoing research includes understand the best ways that component values of such vulnerability scoring systems are best applied within the model.

However, it is important to stress again that the existing repositories of vulnerability data give scores for individual vulnerabilities alone. This says nothing about relationships among vulnerabilities that could be leveraged by an attacker. Our attack graph metrics propagate these individual vulnerability scores, combining them according to the logical relationships in the attack graph. The result is a measure of risk to the information system as a whole, rather than large numbers of disjointed measures for individual vulnerabilities.

4. RESULTS AND DISCUSSION

In this section, we describe the results of the methods, assumptions, and procedures carried out under this project, and discuss implications for network security. The remainder of this section is organized as follows:

- **Section 4.1:** Describes the building of network attack models, and generation of all possible network attack paths.
- **Section 4.2:** Describes correlation, prediction, and hypothesizing about network attacks via attack graph adjacency matrices.
- **Section 4.3:** Describes using predicted vulnerability paths for placing intrusion detection sensors, and prioritizing resulting intrusion alerts.
- **Section 4.4:** Describes the extraction of metrics from attack graphs that measure overall network security.
- **Section 4.5:** Describes results of independent testing and evaluation of our TVA tool.
- **Section 4.6:** Describes extensions to our TVA tool in the area of network model population, based on feedback from evaluations.
- **Section 4.7:** Summarizes key events through the course of this project.

4.1 Attack Modeling and Simulation

TVA decomposes attack graph generation into two phases: capturing an input network attack model, and using the model to simulate multi-step network penetration. The attack model represents the network configuration and potential attacker exploits. In attack simulation, the input model is analyzed to form an attack graph of causally interdependent exploits, according to user-specified constraints.

In TVA, the network attack model includes aspects of the network configuration relevant to attack penetration, as well as a set of potential attacker exploits that match attributes of the configuration. The TVA approach can apply to many different types of attack models, even non-cyber models, as long as a common schema is employed across the model.

Figure 19 is an example of one such schema for TVA network models. This schema simply shows the hierarchical relationships among model elements, i.e., a parent element "contains" its children. For clarity, the various attributes of the model elements are not shown, such as "name" attributes for machines and domains.

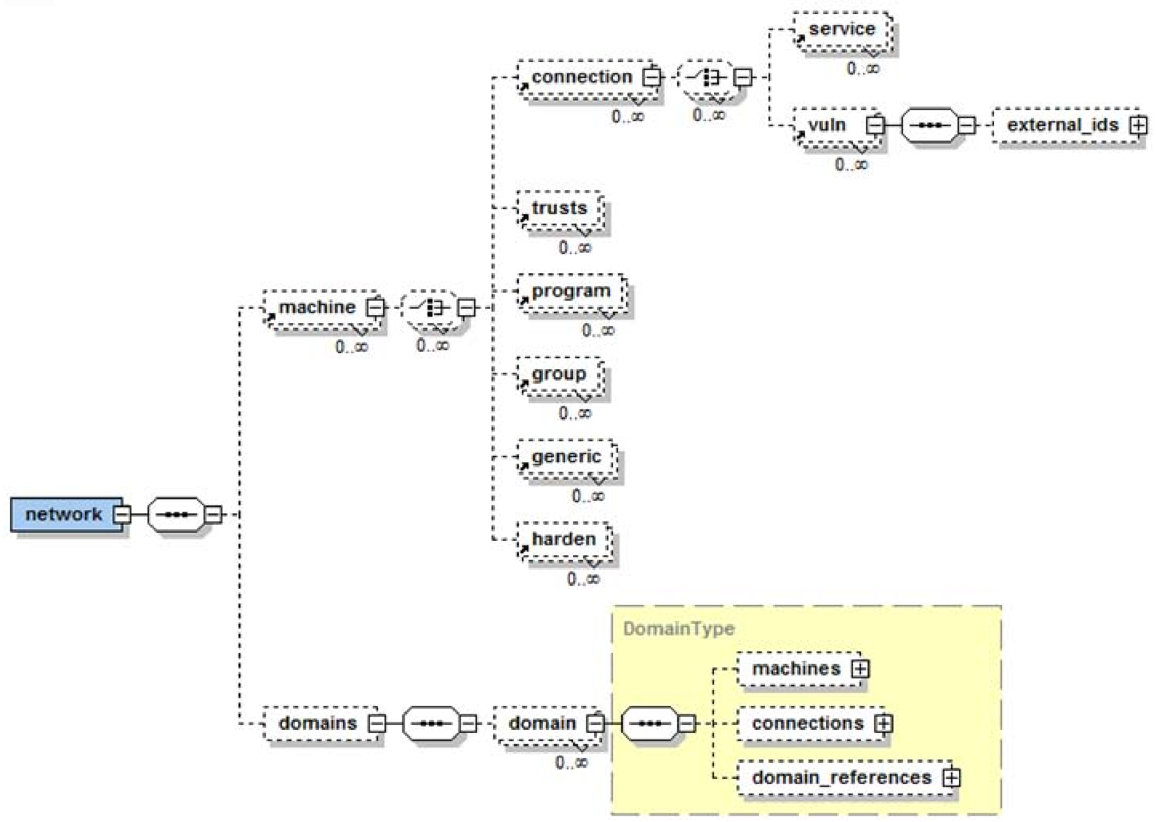


Figure 19. Example schema for TVA network models

In this model schema, a network is comprised of machines, and/or machines organized into protection domains. Protection domains capture the idea that the set of machines in a domain implicitly have unrestricted access to one another's vulnerable services. This abstraction is a scalable alternative to having a completely connected sub-graph within the attack graph. The domain reference allows for domains within domains, i.e., sub-domains.

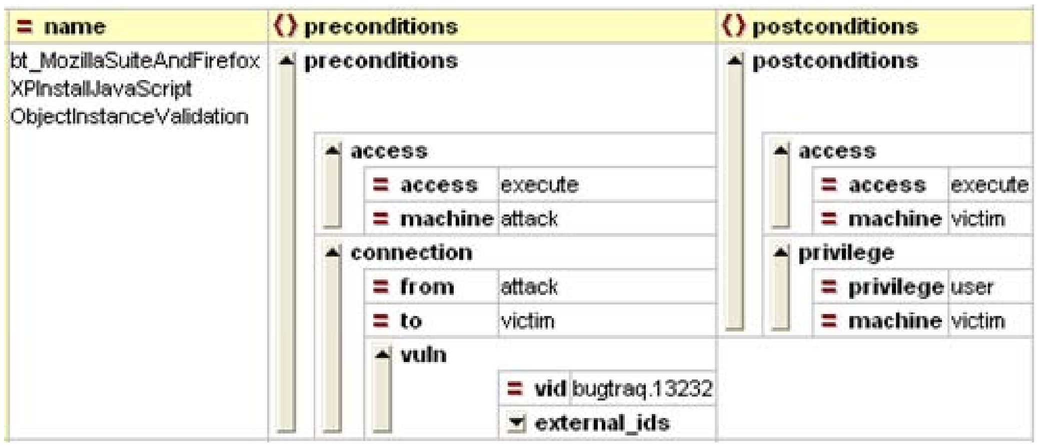
A machine includes sub-elements and attributes relevant for modeling network attack penetration (exploits). This includes operating system (an attribute of machine, not shown), connections to vulnerable services on other machines, sets of machines that are trusted, application programs on a machine, groups to which the machine belongs such as Windows New Technology (NT) domains, and user-defined generic attributes. A harden element defines the hardening of a vulnerability, i.e., exploitation of a given vulnerability on a given machine is omitted from the attack graph.

A connection describes how a machine connects to potentially vulnerable services across the network, to ports on other machines, or to its own ports. This mirrors the Transmission Control Protocol/Internet Protocol (TCP/IP) reference model, in which a layered connectivity structure represents the various network architectures and protocols [16]. A service connection indicates a running service on a destination machine, to which a source machine can connect.

Each connection is composed of a service or application type at the appropriate TCP/IP layer. For example, an HTTP connection specifies the web server name/version at the transport layer. Link-layer connectivity models exploits against the Address Resolution Protocol (ARP). This scopes attacks based on traffic sniffing, such as man-in-the-middle attacks based on ARP poisoning. Application-layer connectivity models exploits that rely on particular application configurations, trust relationships, or other high-level details.

To keep pace with emerging threats, we must continually monitor sources of reported vulnerabilities and add those to our database of modeled TVA exploits. We model an attacker exploit in terms of preconditions and postconditions, for generic attacker and victim machines, which are subsequently mapped to the target network. For convenience, we map vulnerable network connections to known standard vulnerability identifiers, such as Common Vulnerabilities and Exposures (CVE) [46] and Bugtraq [47].

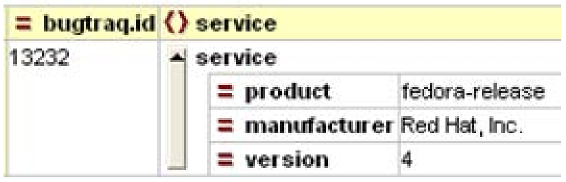
For populating models automatically, we map outputs of network scanning tools to our network schema, which in turn provide preconditions for attack graph exploits. Figure 20 shows example output data for Centennial Discovery [48], a network asset management tool. A Discovery agent deployed on a network host machine reports detailed host configuration data, i.e., product/manufacturer/version for each detected software component.



```
= name                                 () preconditions                    () postconditions
bt_MozillaSuiteAndFirefox              preconditions                       postconditions
XPInstallJavaScript                      access                              access
ObjectInstanceValidation                   = access   execute                  = access   execute
                                           = machine  attack                   = machine  victim
                                         connection                          privilege
                                           = from     attack                   = privilege user
                                           = to       victim                   = machine  victim
                                           vuln
                                                 = vid bugtraq.13232
                                                 external_ids
```

Figure 20. Software item reported by asset management tool

The discovered host software information is then mapped to preconditions for modeled exploits. Figure 21 shows preconditions and postconditions for exploitation of a Bugtraq vulnerability, in terms of generic attacker/victim machines. The preconditions are that the attacker can execute code on the attacking machine, and that there is a vulnerable connection from attacker to victim, identified as Bugtraq 13232.

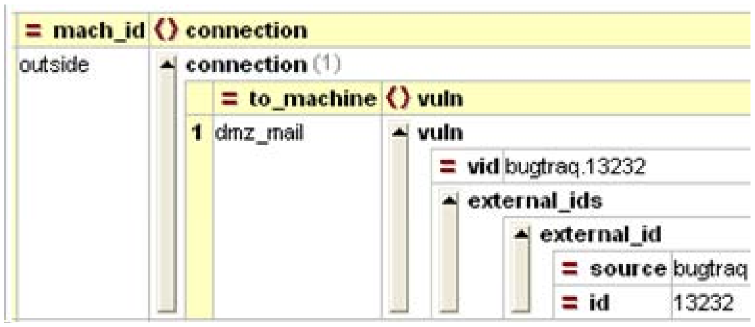


```
= bugtraq.id  () service
13232         service
                = product       fedora-release
                = manufacturer  Red Hat, Inc.
                = version       4
```

Figure 21. Example TVA modeled exploit

Symantec DeepSight [49], a web service direct feed of the Bugtraq database, gives the vulnerable software components for each reported vulnerability. Host configuration data gathered from an asset management tool such as Discovery generally differs from software descriptions in DeepSight.

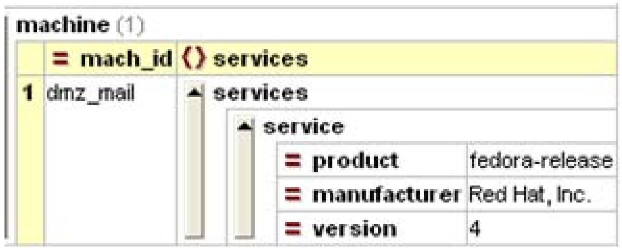
We map discovered host software components to corresponding vulnerability records, as in Figure 22. This shows a Discovery software description for Red Hat Fedora 4 mapped to Bugtraq vulnerability 13232. Symantec DeepSight has fields corresponding to product/manufacturer/service that help with this mapping, by matching against Discovery through regular expressions.



```
= mach_id  () connection
outside    connection (1)
             = to_machine  () vuln
           1 dmz_mail      vuln
                             = vid bugtraq.13232
                             external_ids
                               external_id
                                 = source bugtraq
                                 = id     13232
```

Figure 22. Software to vulnerability mapping

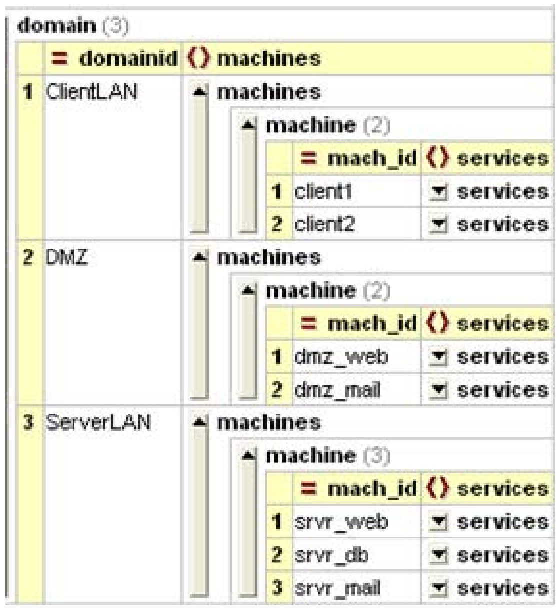
Figure 23 illustrates a resulting connection to vulnerable software (Bugtraq 13232) on the host machine. This connection is built into the attack model by mapping the discovered host software to a known vulnerability. Then, since a connection with Bugtraq 13232 is a precondition for a particular exploit, this exploit may be included in an attack graph for this network.



```
machine (1)
    = mach_id  () services
  1 dmz_mail   services
                 service
                   = product       fedora-release
                   = manufacturer  Red Hat, Inc.
                   = version       4
```

Figure 23. Network connection to vulnerable software

The Discovery asset management tool also defines protection domains, i.e., sets of machines with full connectivity to one another’s vulnerable services. This is shown in Figure 24. Each protection domain is identified, along with its member machines.



domain (3)

	domainid	machines		mach_id	services
1	ClientLAN	machine (2)	1	client1	services
			2	client2	services
2	DMZ	machine (2)	1	dmz_web	services
			2	dmz_mail	services
3	ServerLAN	machine (3)	1	srvr_web	services
			2	srvr_db	services
			3	srvr_mail	services

Figure 24. Protection domains reported by asset management tool

The purpose of modeling the network configuration in TVA is to support preconditions of modeled attacker exploits. As we have shown, we can map software components to their reported vulnerabilities. An alternative is to run remote vulnerability scans with tools such as Nessus, Retina [50], or FoundScan [51].

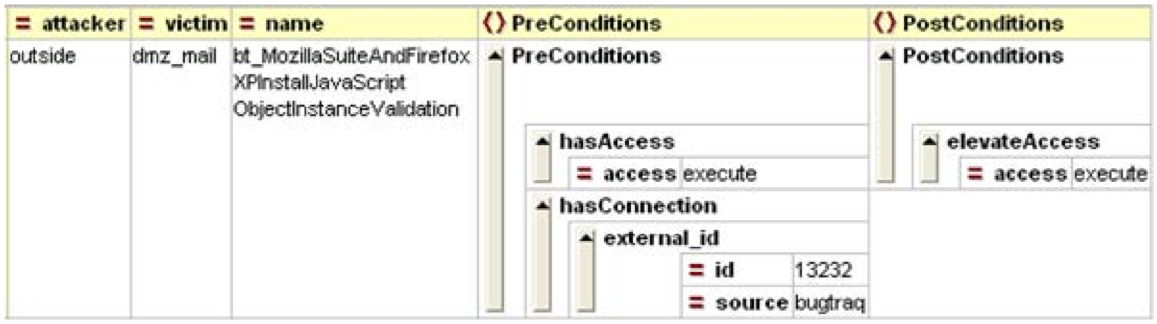
With this approach, the tool actively tests for the existence of host vulnerabilities. The scanner reports a detected vulnerability explicitly, using a standard vulnerability identifier, rather than reporting a particular software component. The corresponding exploit precondition is written in terms of this vulnerability identifier.

An advantage of this approach is that we can capture the effects of connectivity-limiting devices such as routers and firewalls. That is, we scan from different network vantage points, targeting hosts through firewalls. The idea is that the scanner takes the role of an attacker who has reached a certain point in the network. Thus we avoid creating any special firewall exceptions for the scanning machine, as is typically done for network vulnerability scans.

We then combine multiple scans from various network locations, building a complete map of connectivity to vulnerable services throughout the network. Alternatively, we can analyze firewall rules directly, adding the resulting vulnerable connections to the model. In this case, only local subnet scans are needed.

In TVA attack simulation, modeled exploits are matched against the network configuration model, forming an attack graph of causally interdependent exploits, according to user-specified simulation constraints. Because the model is pre-populated through network scans and vulnerability databases, all that remains is to define the attack scenario: e.g., the starting point, the attack goal, and any what-if changes to the network configuration.

In other words, given an input model of network configuration and attacker exploits, the exploits are instantiated for specific attacker/victim machine pairs in the network. Preconditions for instantiated exploits are tested, and resulting postconditions are matched with preconditions of other exploits. Figure 25 shows an exploit that has been instantiated for particular machines in the network model. The attacker and victim machines are no longer generic, i.e., they are defined for actual machines in the network.

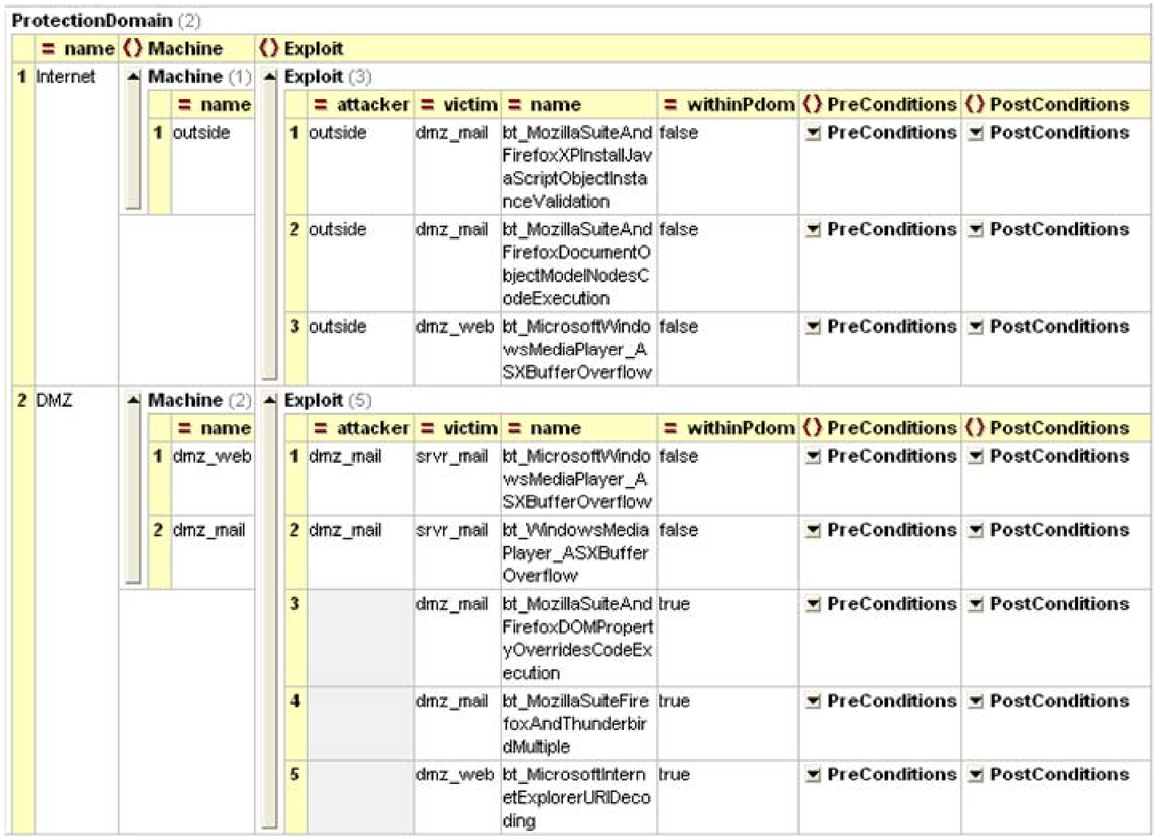


attacker	victim	name	PreConditions	PostConditions
outside	dmz_mail	bt_MozillaSuiteAndFirefox XPInstallJavaScript ObjectInstanceValidation	PreConditions hasAccess access: execute hasConnection external_id id: 13232 source: bugtraq	PostConditions elevateAccess access: execute

Figure 25. Exploit instantiated for particular network

An attack graph also needs to follow the structure of protection domains defined for the network. Within a protection domain, it is assumed that each machine has unrestricted connectivity to vulnerabilities on all other machines in the domain. This implies that the attack graph is completely connected with a domain.

Figure 26 shows example protection domains in attack graph data. Within each domain, the set of all member machines is specified, as well as exploits relevant to each domain. There are two possible types of exploits: within-domain and across-domain. Within-domain exploits are accessible to machines within the protection domain only. Thus it is sufficient to specify the victim machine only, since the attacking machines are implicit. Across-domain exploits are those that attack machines in other domains. Those exploits have both attacker and victim machines specified.


ProtectionDomain (2)

	name	Machine	Exploit
1	Internet	Machine (1)	Exploit (3)
2	DMZ	Machine (2)	Exploit (5)

1 Internet — Machine (1)

	name
1	outside

1 Internet — Exploit (3)

	attacker	victim	name	withinPdom	PreConditions	PostConditions
1	outside	dmz_mail	bt_MozillaSuiteAndFirefoxXPInstallJavaScriptObjectInstanceValidation	false	PreConditions	PostConditions
2	outside	dmz_mail	bt_MozillaSuiteAndFirefoxDocumentObjectModelNodesCodeExecution	false	PreConditions	PostConditions
3	outside	dmz_web	bt_MicrosoftWindowsMediaPlayer_ASXBufferOverflow	false	PreConditions	PostConditions

2 DMZ — Machine (2)

	name
1	dmz_web
2	dmz_mail

2 DMZ — Exploit (5)

	attacker	victim	name	withinPdom	PreConditions	PostConditions
1	dmz_mail	srvr_mail	bt_MicrosoftWindowsMediaPlayer_ASXBufferOverflow	false	PreConditions	PostConditions
2	dmz_mail	srvr_mail	bt_WindowsMediaPlayer_ASXBufferOverflow	false	PreConditions	PostConditions
3		dmz_mail	bt_MozillaSuiteAndFirefoxDOMPropertyOverridesCodeExecution	true	PreConditions	PostConditions
4		dmz_mail	bt_MozillaSuiteFirefoxAndThunderbirdMultiple	true	PreConditions	PostConditions
5		dmz_web	bt_MicrosoftInternetExplorerURIDecoding	true	PreConditions	PostConditions

Figure 26. Protection domains in attack graph data

In TVA, an attack graph can be completely unconstrained; i.e., all possible attack paths regardless of assumed starting and ending points in the network. In such a scenario, the source of the threat is assumed unknown and no particular critical network assets are identified as specific attack goals. Figure 27 is an example of such an unconstrained attack graph.

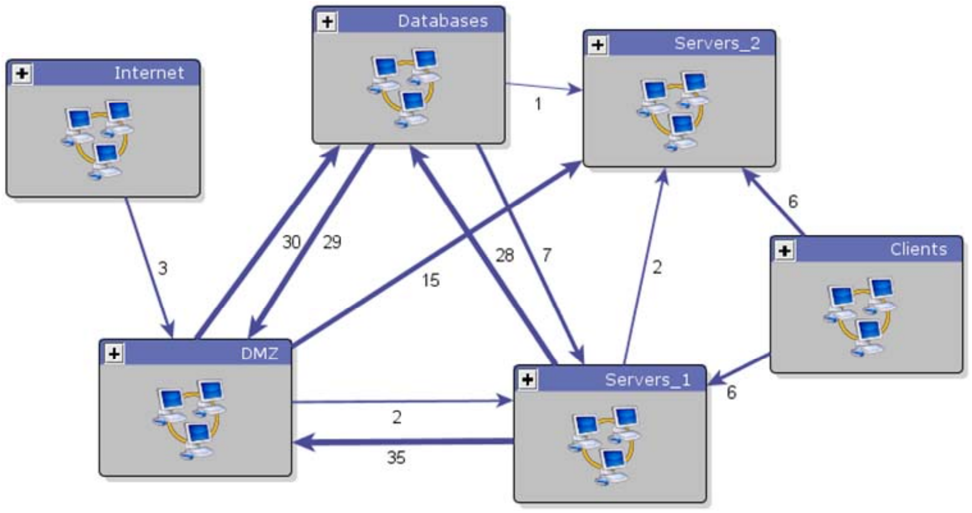


Figure 27. Unconstrained attack graph

Another option is to constrain the attack graph to a given starting point (or points) for the attack. The idea here is that the origin of the attack is assumed, and that only paths that can be reached from the origin are to be included. Figure 28 is an example attack graph in which the attack starting point (Internet) is specified.

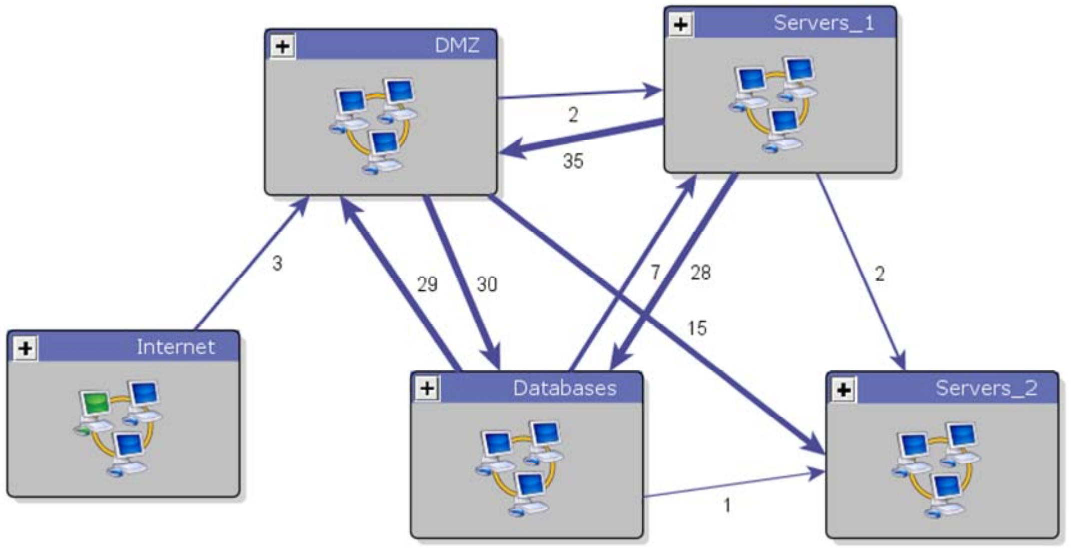


Figure 28. Attack graph with constrained starting point

Yet another option is to constrain the attack graph so that it ends at a given ending point (or points) serving as the attack goal. Here the idea is that certain critical network assets are to be protected, and only attack paths that reach the critical assets are to be included.

This option could be exercised alone, with an unconstrained starting point, or combined with a constrained starting point. Figure 29 is an example of the latter, in which the both the attack starting point (Internet) and attack ending point (Databases) are specified.

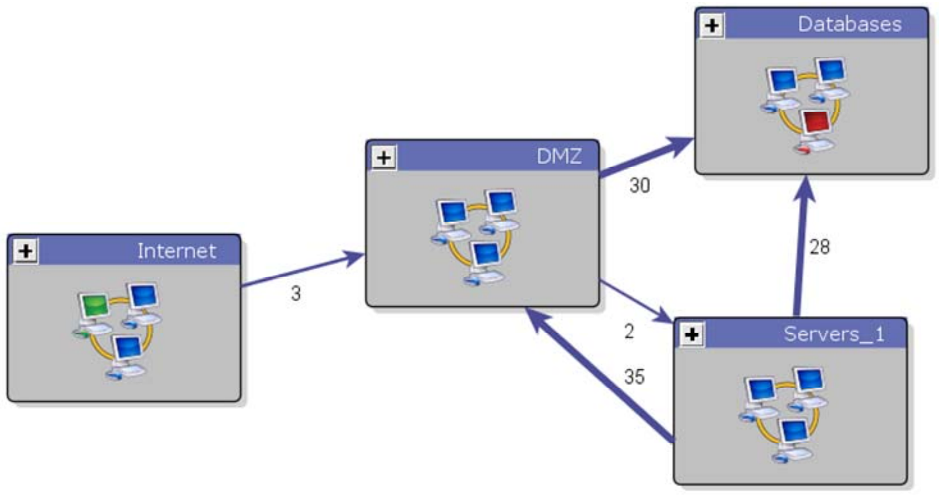


Figure 29. Attack graph with constrained starting and ending points

The motivation for constraining the attack graph is to reduce the scope of the graph to expected attack scenarios, eliminating unnecessary clutter. For example, in Figure 29, the outgoing edges from the Database protection domain are omitted. If the primary goal is to protect the databases, then attacks away from there are less important, i.e., the databases have already been compromised. Similarly, any attacks into the starting point could be omitted, since the attacker already has control of it.

Attack paths particularly important to consider are the most direct ones, i.e., shortest paths from attack start and/or attack goal. This is shown in Figure 30. Two scenarios are considered. In Figure 30(a), the graph shows direct (shortest) paths from a given starting point. In Figure 30(b), both the attack starting point and goal points are given. The graph shows all direct paths from the starting point to the goal point.

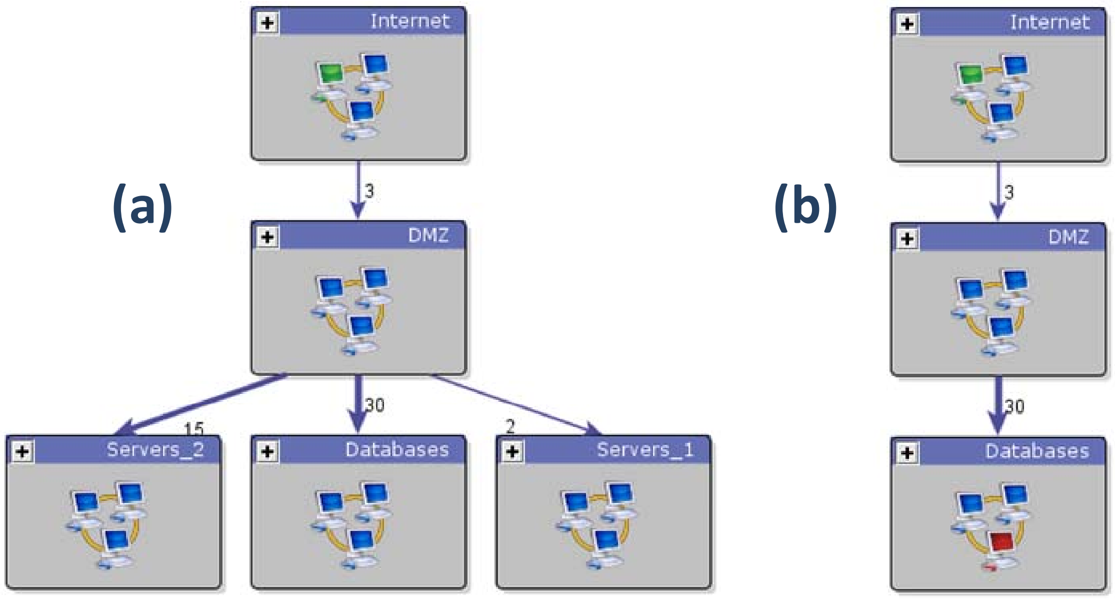


Figure 30. Attack graph constrained to direct attacks

Again, the idea is to identify the most critical paths and vulnerabilities, for pre-attack network hardening as well as real-time alarm correlation, prediction, and response. Thus, given the assumed threat sources, attacker behavior, and critical network resources, we can tailor our analysis and defensive measures accordingly.

Through sophisticated visualization, graphs can be rolled up or drilled down as the graph is explored. Figure 31 shows a visualization interface for attack graph exploration and analysis. The main view of the graph shows all possible paths through the network, based on the user-defined attack scenario. In this view, the analyst can expand or collapse graph clusters (protection domains) as desired, rearrange graph elements, and select elements for further details. In the figure, two domains are expanded to show their specific hosts and exploits between them.

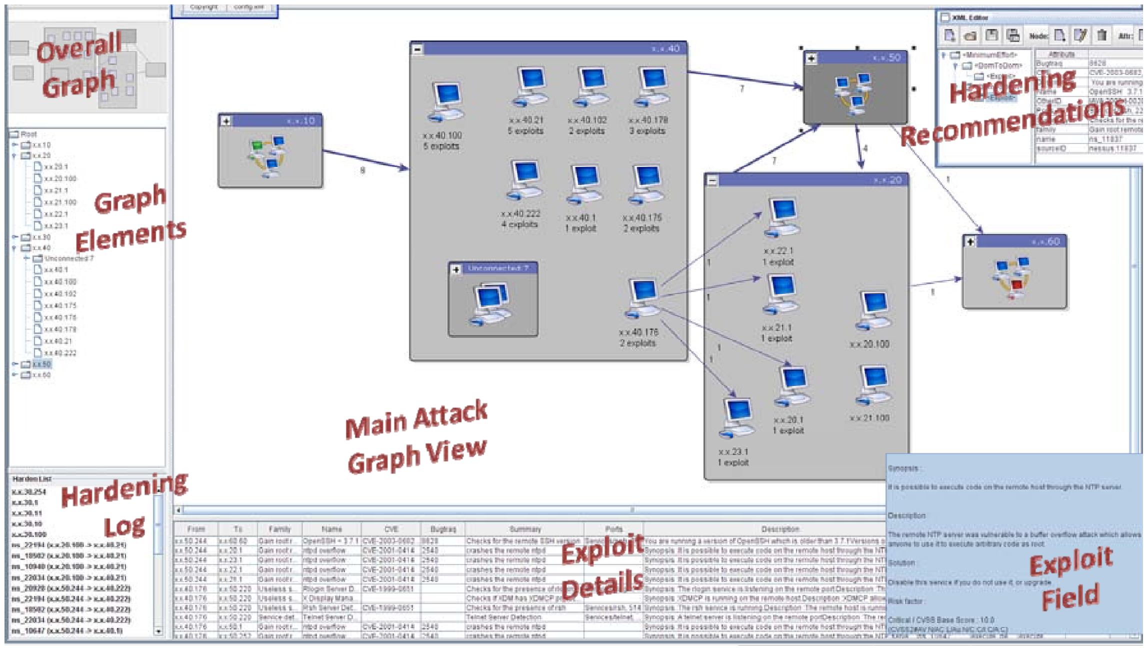


Figure 31. Attack graph visualization interface

When an edge (set of exploits) is selected in the main view, details for the corresponding exploits are provided. Each exploit record contains a number of relevant fields describing the underlying vulnerability. A hierarchical (tree) directory of all attack graph elements is provided, linked to other views. A view of the entire graph is constantly maintained, providing overall context as the main view is rescaled or panned. Automated recommendations for network hardening are provided, and specific hardening actions taken are logged.

The visualization interface in Figure 31 provides an abstract, purely cyber-centric view of network attacks. But in some situations, understanding the physical location of possible attacks may be important, as for assessing mission impact. Given the locality of network elements, we can embed the attack graph into a geo-spatial visualization.

This is illustrated in Figure 32. Here, elements of the attack graph are clustered around major centers of the network, and graph edges show exploits between centers. Interactive visualization capabilities can support drilldown for further details at a desired level of resolution.

We face sophisticated attackers who may combine multiple vulnerabilities to penetrate networks with devastating impact. Assessment of attack risk must go well beyond simply counting the number of vulnerabilities or vulnerable hosts. Metrics like percentage of patched systems ignore interactions among network vulnerabilities; such metrics are limited, because vulnerabilities in isolation lack context.

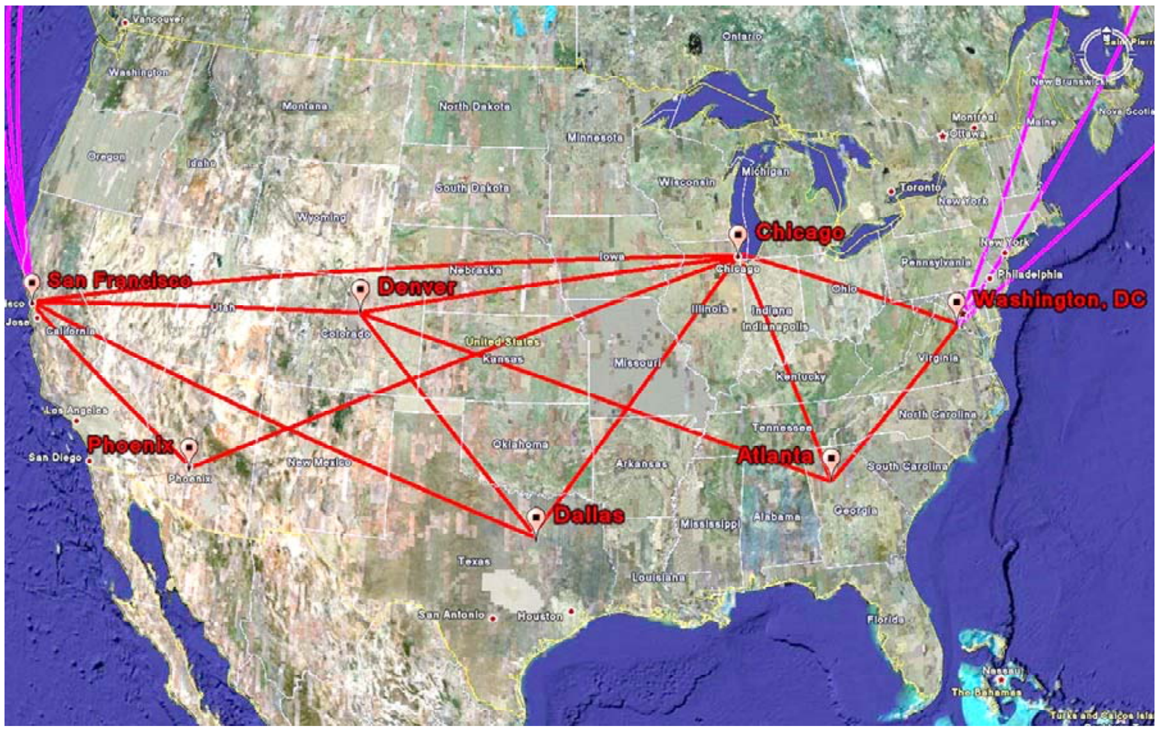


Figure 32. Geo-spatial attack graph user interface

The TVA attack graph is generated in advance of attack, based on proactive network scans. After the network is hardened to the extent possible, any residual vulnerability paths are used to correlate and prioritize alerts and log events (currently not implemented). The attack graph provides the context needed for planning optimal responses to attack.

TVA attacks graphs can also guide the optimal deployment and operation of intrusion detection systems, tailored to our network and its critical assets. During deployment, we must decide where to place detection sensors within the network.

Traditionally, intrusion detection sensors are placed at network perimeters, with the idea of detecting attacks from the outside. But with this deployment, traffic in the 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. We should strike a balance, in which we cover known residual vulnerability paths, using the fewest sensors necessary. TVA attack graphs provide this balance.

Consider the attack graph in Figure 33. Assume that this is the residual attack graph after network hardening measures have been applied. So now the goal is to map this attack graph to the network topology, and embed intrusion detection sensors in the network to cover all the vulnerability paths (with the fewest sensors).

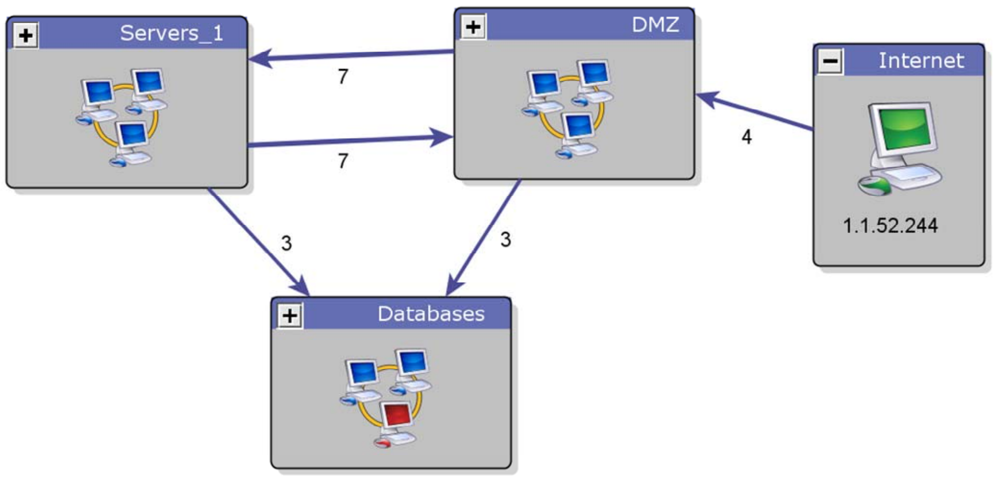


Figure 33. Residual attack graph

Figure 34 shows the topology for this network, overlaid by the attack paths from Figure 33. This is a simplified network diagram, for illustrating the problem of sensor placement for attack graph coverage. The diagram omits firewalls, which limit connectivity as reflected by the attack graph. Also, the elements labeled Router A, Router B, and Subnet n are abstract network devices that are capable of monitoring traffic through them, e.g., via SPAN ports.

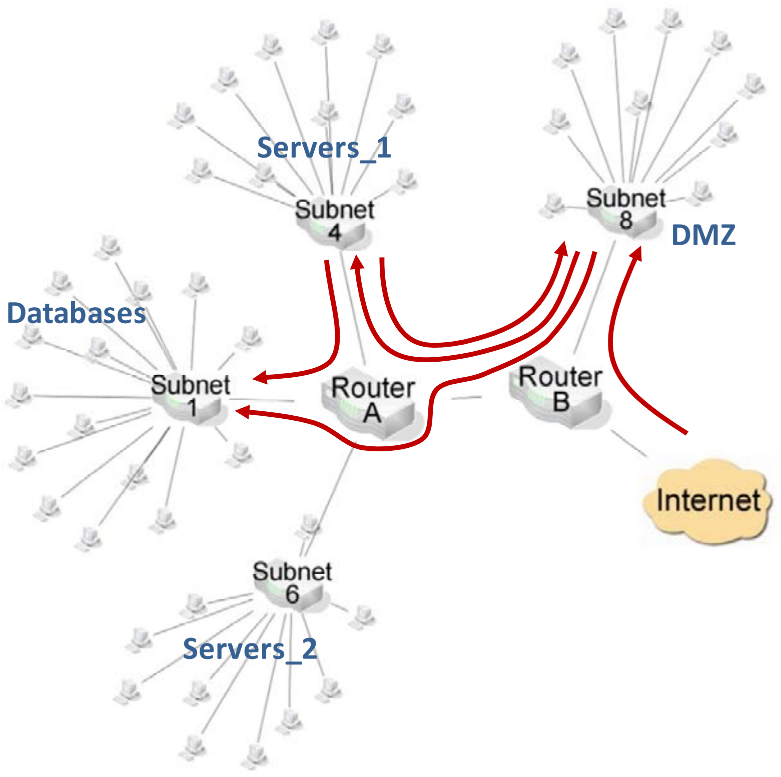


Figure 34. Intrusion detection sensor deployment

Analysis of the joint topology/attack representation in Figure 34 shows that detection sensors placed at Router A and Router B cover all vulnerability paths, with the fewest sensors. An alternative is to place sensors at Subnet 1, Subnet 4, and Subnet 8, which also covers all paths but requires three (versus two) sensors.

In this network, deploying a sensor at the perimeter alone (Router B) will miss attack traffic from Servers_1 to Databases. In the opposite extreme, we might decide to deploy sensors at each of the four Subnet n devices, to catch all potential attack traffic. But TVA shows that in fact there are no critical vulnerability paths involving Subnet 6, so deploying a sensor there is wasteful, including continually monitor alerts generated from there. Again, sensors deployed at Router A and Router B are sufficient to cover all vulnerable paths.

For enterprise networks, performing this kind of analysis requires the kind of automated support provided by TVA. Our attack graphs bring together information from a variety of sources over multiple network layers into a concise map. While the sensor placement problem itself is NP-hard, there is a heuristic algorithm that scales well and provides near-optimal solutions [6].

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

In this regard, specifications such as Intrusion Detection Message Exchange Format (IDMEF) [52] or the ArcSight [53] event log format define data formats for information sharing between intrusion detection systems and TVA. For example, one implementation option is the IDMEF plug-in [54] for the popular intrusion detection system Snort [55]. This plug-in allows Snort to output alerts in the IDMEF message format. Data exchanges in IDMEF are in eXtensible Markup Language (XML), with the format being enforced through a formal schema.

Figure 35 shows the structure of an IDMEF alert. The IDMEF model is intended to represent alerts in an unambiguous fashion, while explicitly assuming that alert information is heterogeneous. Alerts from different tools may have varying amounts and types of information about an event, which the IDMEF data model is designed to accommodate.

For TVA, the critical data are source and target (attacker and victim) network addresses and an alarm identifier that can be mapped to a vulnerability in the TVA model. In IDMEF, these are supported by the Source, Target, and Classification elements (respectively).

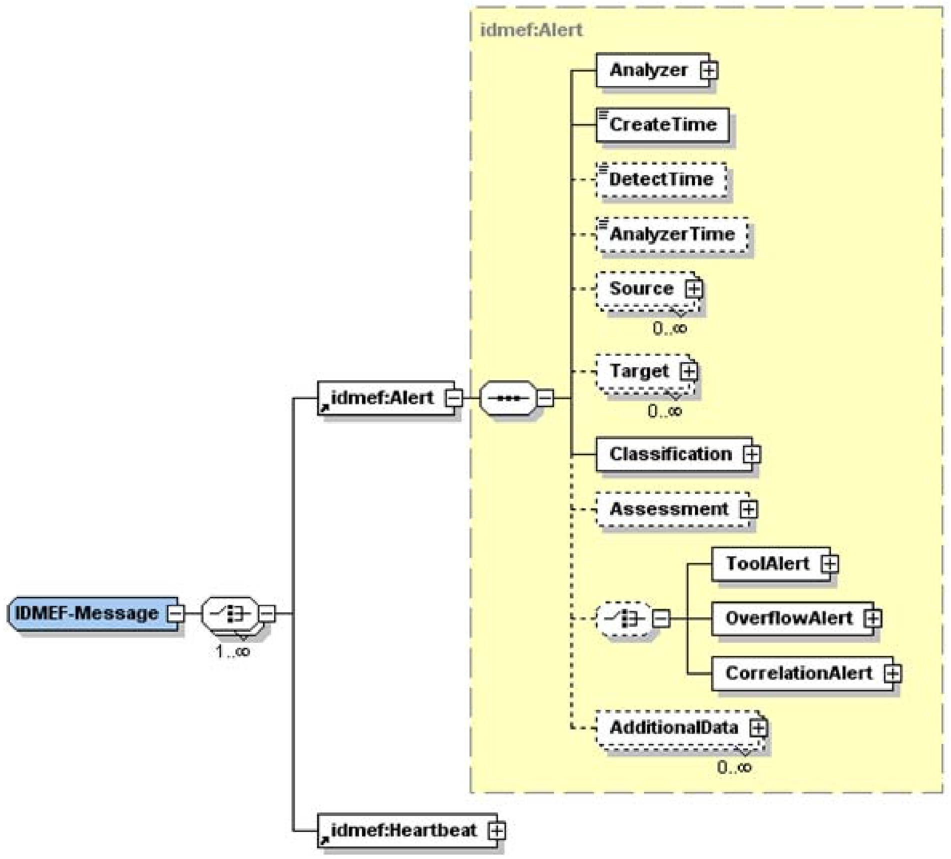


Figure 35. IDMEF alert structure

When intrusion alarms are generated, TVA attack graphs provide the necessary context for correlating and prioritizing them. First, we can place a high priority on alarms that lie on vulnerability paths through our network. We can prioritize them even further based on their 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 attack graph analysis is highly precise, taking all relevant facts into account. We determine 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 an advanced form of false-alarm reduction, e.g., restriction to alarms along critical paths.

It is important to model network vulnerability as we do. Multi-step alarm correlation that does not take real network vulnerabilities into account is limited [14]. Pre-computing vulnerability-based attack graphs in advance of attack has the additional advantage of rapid correlation, i.e., faster than an intrusion detection system can generate them [10][12].

Further, the predictive capabilities of TVA attack graphs enable 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 enables us to predict missed events (false negatives), helping to mitigate inaccuracies in our intrusion detection systems.

To illustrate some of these ideas, consider Figure 36. This is the same residual attack graph as in Figure 33, with relevant protection domains expanded to show additional details. This attack graph provides considerable insight for correlating and prioritizing any alarms generated for this network and for responding to these potential attacks.

For example, suppose an alarm is raised for an attack between two machines in the DMZ, say, from DMZ_1 to DMZ_2. From just a single alarm in the DMZ, we might wait before responding. On the other hand, if an alarm is raised from Internet into DMZ, followed by an alarm within the DMZ, it is a much stronger indicator that the attack may be a real security breach. Remember that false alarms are common with intrusion detection, and erroneously blocking traffic in response to false alarms is a denial of service.

From an alarm within the DMZ, another approach might be to block traffic from DMZ_3 to DB_1 and DB_2. Because of the possibility of denial of service, such an action is not usually taken. But we can we limit the blocking to the vulnerable ports on DB_1 and DB_2 only, specifically from DMZ_3, so that any non-vulnerable services on those machines could remain unblocked.

We might then keep traffic from DMZ_3 into Servers_1 machines unblocked, since those machines are one less attack step (i.e., three steps) from critical machine DB_4. In other words, we could still wait to see if an alarm is raised from the DMZ into Servers_1, at which point we block the vulnerable baths from Servers_1 to Databases.

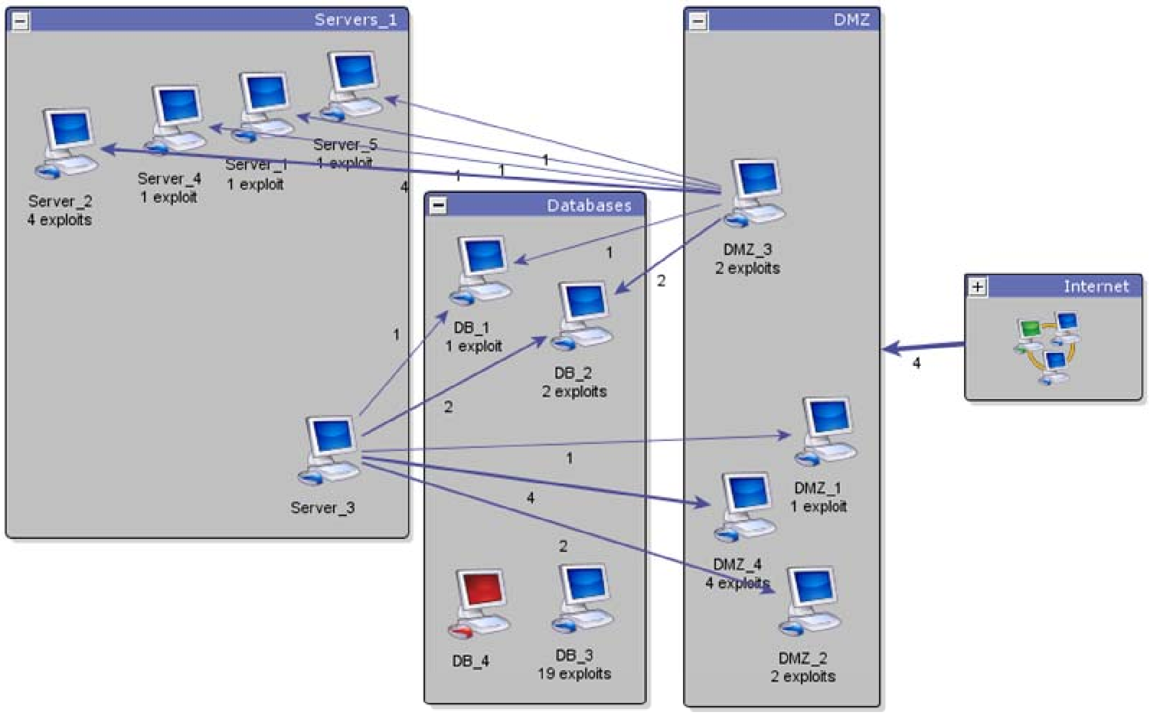


Figure 36. Attack prediction and response

An even more aggressive response to an alarm within the DMZ is to block outgoing traffic from the DMZ to vulnerable services in Servers_1 and Databases. Again, there is the potential for denial of service, but we still limit our response to vulnerable connectivity. Without attack graph analysis, the only response to a serious attack is to block all traffic from the DMZ, rather than just vulnerable connectivity.

Further, one could surmise that an alarm in the DMZ is follow-on from a missed intrusion from the Internet into the DMZ. This could guide further investigation into traffic logs into the DMZ, looking for missed attacks, especially against the four vulnerable paths into the DMZ.

If an attack was detected within Servers_1 (e.g., from Server_1 to Server_2), a similar set of responses is indicated. As a precaution, one could block traffic from Server_3 to vulnerable ports on DB_1 and DB_2. But blocking traffic from Server_3 into the DMZ is less indicated because it is leading away from the critical Databases domain. Similarly, any alerts from Server_3 into the DMZ are lower priority, especially if they are not against vulnerable DMZ services.

Thus TVA provides a range of reasonable responses, ranked by severity or actual likelihood of attack. Here, severity is in terms of lying on critical vulnerability paths, especially close to critical assets, and likelihood is increased by causal correlation of alerts. Multiple options are available that enable us to fine tune responses as potential attacks unfold, based on proactive response plans.

4.2 Matrix Analysis and Visualization

Our clustered adjacency matrices reveal the underlying regularities in network attack graphs. This approach is particularly attractive because it avoids the edge clutter usually associated with literal drawings of attack graphs. The adjacency matrix is concise, representing each graph edge with a single matrix element.

Our approach places no particular restrictions on the form of the attack graph. It therefore applies to attack graphs based on network vulnerabilities, detected intrusions, or combinations thereof, and well as attack graphs with aggregated vertices, e.g., aggregated by network machine. Our approach has low-order polynomial complexity overall, for scalability to larger networks.

The information-theoretic clustering algorithm we apply reorders rows and columns of the adjacency matrix so that rectangular blocks of similarly-connected attack graph elements emerge. This clustering algorithm is fully automatic, parameter-free, and scales linearly with problem size.

We further transform the attack graph adjacency matrix by raising it to higher powers, to represent multiple attack steps. We can thus show attacker reachability across the network within any number of attack steps. We combine these per-step reachability matrices into a single matrix that shows the minimum number of steps between any pair of vertices in the attack graph. We also summarize reachability for all number of steps via transitive closure.

Through our general approach, we are able to correlate, predict, and hypothesize about network attacks. For example, we can provide a concise summary of changes in an attack graph resulting from changes in the network configuration.

We can place intrusion alarms in the context of the vulnerability-based attack graph for categorizing alarms. We can step forward from an attack, to predict its impact and prioritize defensive responses according to the number of steps required to reach victim machines. We can also step backward from an attack, to predict its origin.

When attack graphs for realistic networks are drawn in their full detail, the resulting display can be overwhelming. This is demonstrated in Figure 37. This is an attack graph for about 50 machines, in a network of 3 subnets. This picture includes every detail of every part of every attack path through the network.

In particular, it shows each exploitable vulnerability, with all of their preconditions and postconditions, between each applicable machine. Clearly the attack graph is overwhelming when shown in this level of detail. The challenge is to convey all attack paths in a way that is comprehensive but easily understandable.

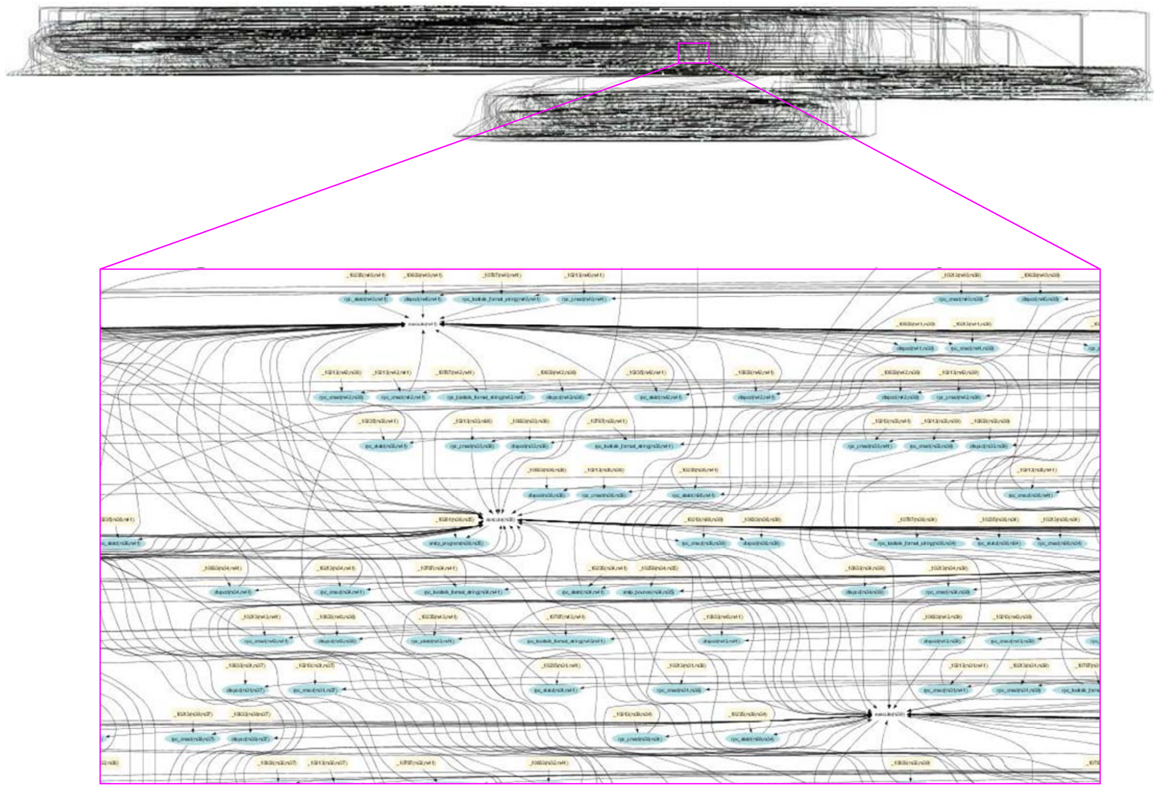


Figure 37. Example attack graph in its full complexity

Figure 38 shows the same attack graph as in Figure 37. Now the low-level security conditions have been aggregated to machine vertices, and exploits have been aggregated for pairs of machines.

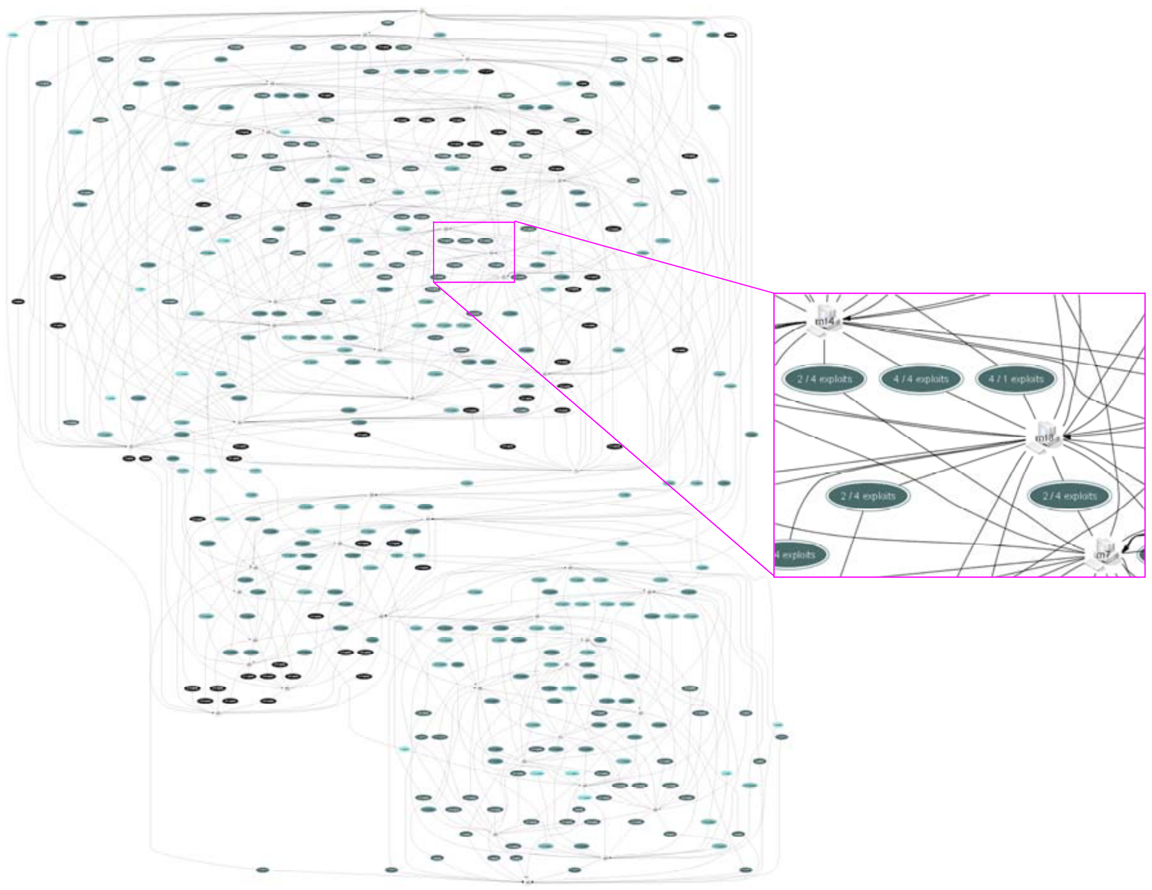


Figure 38. Attack graph aggregated to individual machines

Despite the fact that this attack graph has been aggregated to the level of machines, the drawn graph is still cluttered with edges and is hard to follow. Some clustering is apparent in this drawing, but the exact nature of the clusters, such as their boundaries and cross-cluster relationships, is not readily apparent.

Figure 39 shows the same attack graph as Figure 37 and Figure 38, this time represented as an adjacency matrix. In the matrix, rows represent exploits from a particular machine, and columns represent exploits to a particular machine. The presence at least one exploit between a pair of machines is indicated by black matrix element, and the absence is indicated by white.

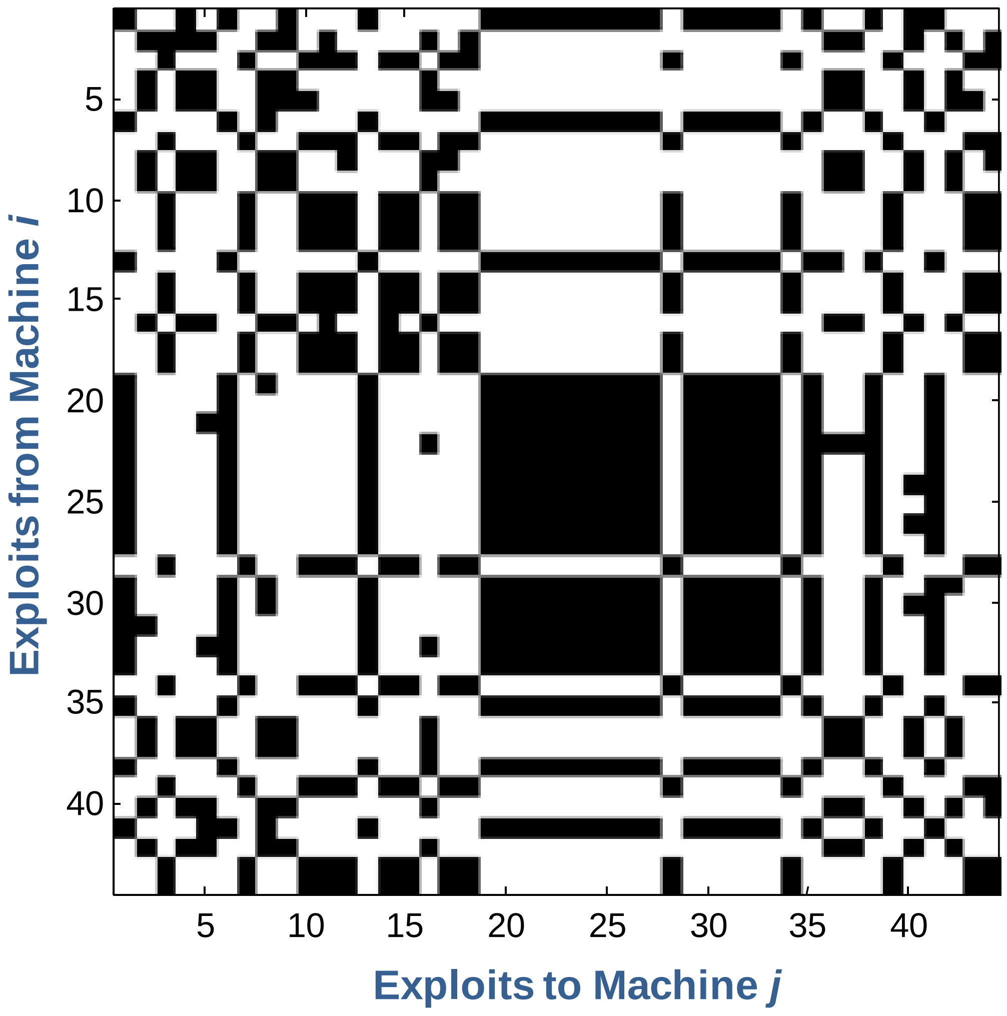


Figure 39. Unclustered adjacency matrix for attack graph in Figure 38

In Figure 39, the rows and columns of the adjacency matrix are in arbitrary order. In general, there is no a priori way of ordering the matrix rows and columns to form meaningful clusters. That is, from the ordering in Figure 39, the underlying structure of the attack graph is obscured by this matrix visualization.

Mathematically, the rows and columns of an adjacency matrix could be placed in any order, without affecting the structure of the attack graph the matrix represents. But orderings that capture regularities in graph structure are clearly desirable. In particular, we seek orderings that tend to cluster graph vertices (adjacency matrix rows and columns) by common edges (non-zero matrix elements). This allows us to treat such clusters of common edges as a single unit as we analyze the attack graph (adjacency matrix).

To extract the underlying graph structure, we apply of matrix clustering. This clustering is designed to form homogeneous blocks of matrix elements, so that within each block, there is a similar pattern of attack graph edges (adjacency matrix elements). For example, protection domains and interactions between them are formed automatically, without prior knowledge.

More generally, our approach detects densely-connected attack graph clusters, even if they are not fully-connected subgraphs (protection domains). This particular clustering algorithm requires no user intervention, has no parameters that need tuning, and scales linearly with graph size.

The clustering algorithm optimally reorders rows and columns to form regions of high and low densities in the attack graph adjacency matrix. The algorithm also provides an information-theoretic measure of cluster optimality, based on ideas from data compression.

It employs the Minimum Description Length principle, in which regularity in the data is used to describe it in fewer symbols. Intuitively, one can say that by compressing the data (describing it in fewer symbols) we better understand it, in the sense that we have captured the regularities in its structure.

In Figure 40, we have clustered the attack graph adjacency matrix *A* in Figure 39. The underlying structure of the attack graph is now clear. The clustering has identified 9 clusters (rectangular blocks) of homogeneous graph edges.

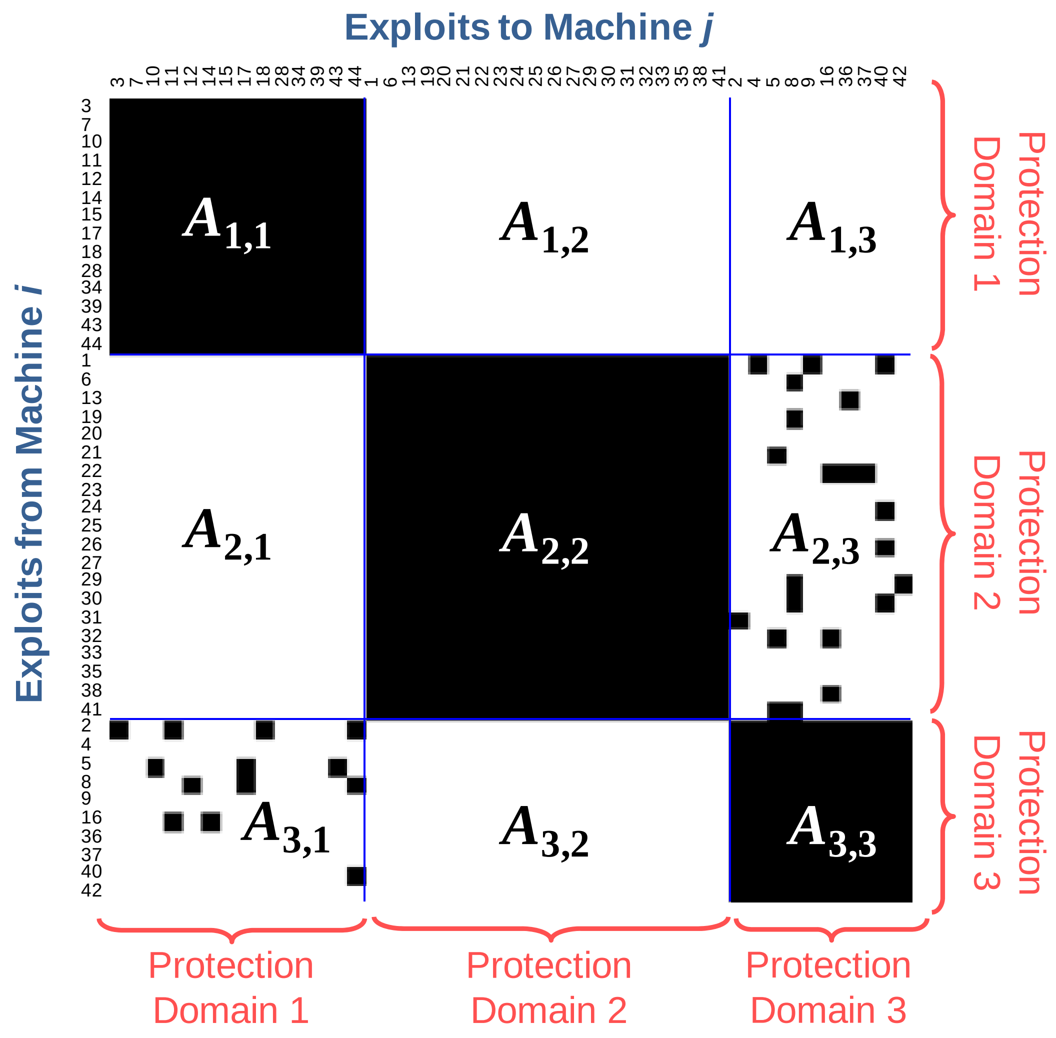


Figure 40. Clustered adjacency matrix for attack graph in Figure 38

The 3 blocks on the main diagonal (blocks $A_{1,1}$, $A_{2,2}$, and $A_{3,3}$) are solid black, indicating full attack graph connectivity within each block. That is, within one of these main-diagonal blocks, every machine can attack every other machine (through at least one exploit).

These blocks constitute TVA protection domains, which have been detected automatically by the clustering algorithm. Block $A_{2,3}$ of the clustered adjacency matrix A shows exploits launched from the 2nd to 3rd protection domains (from block $A_{2,2}$ to $A_{3,3}$). Similarly, block $A_{3,1}$ shows exploits from the 3rd to first protection domains (from block $A_{3,3}$ to $A_{1,1}$).

In Figure 40, matrix rows and columns are labeled with their original indices. This shows exactly how they were reordered by the clustering algorithm. In practice, we might use more meaningful labels such as IP addresses.

Figure 41 shows the square of the clustered attack graph adjacency matrix in Figure 40. Here we have used the arithmetic product, as opposed to the Boolean product. This shows not only whether there exists at least one 2-step attack from one machine to another, but also the actual count of all possible 2-step attacks.

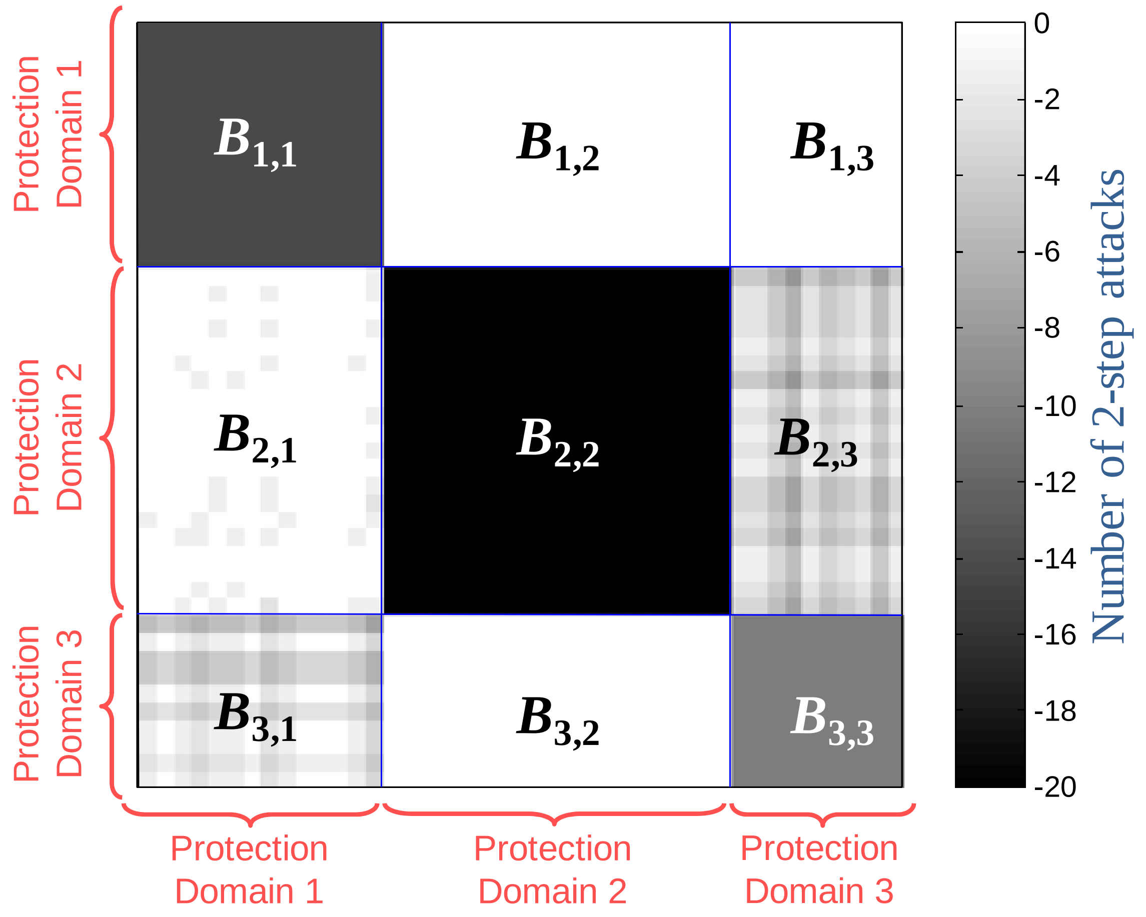


Figure 41. Clustered matrix for attack graph in Figure 38 (2-step attacks)

For example, we see that within the 2nd protection domain (block $A_{2,2}$), there are 20 possible 2-step attacks between each pair of machines, corresponding to the 20 machines in that protection domain. We see corresponding numbers of 2-step attacks within the other 2 protection domains (blocks $A_{1,1}$ and $A_{3,3}$). There are relatively fewer 2-step attacks *across* protection domains (blocks $A_{2,3}$ and $A_{3,1}$).

Figure 42 shows successive powers (A^2, A^3, and A^4) of the clustered adjacency matrix, this time employing Boolean matrix multiplication. This shows attacker reachability between each pair of machines, within 2, 3, and 4 steps, respectively. We see that within 4 steps, machines in the 2nd block can successfully attack all machines (i.e., all columns) in the network. Also within 4 steps, machines in the 1st block can be successfully attacked from all machines (i.e., from all rows) in the network.

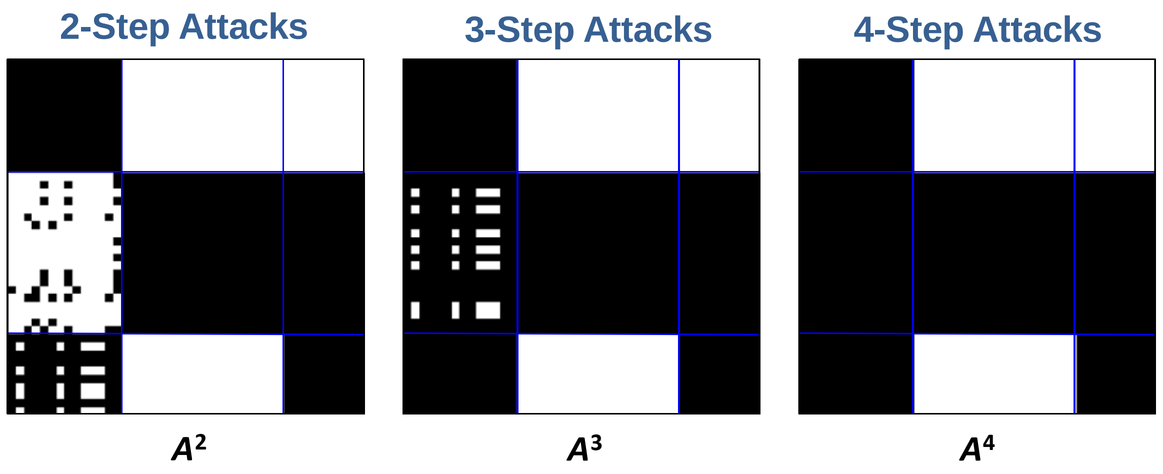


Figure 42. Reachability for 2, 3, and 4 steps for attack graph in Figure 38

In Figure 43, we combine the reachability matrices in Figure 42 (and the adjacency matrix in Figure 40) into a single matrix, as defined by Equation (6). All the information in the separate per-step reachability matrices can now be seen together. We apply this multi-step reachability matrix for prediction of attack origin and impact.

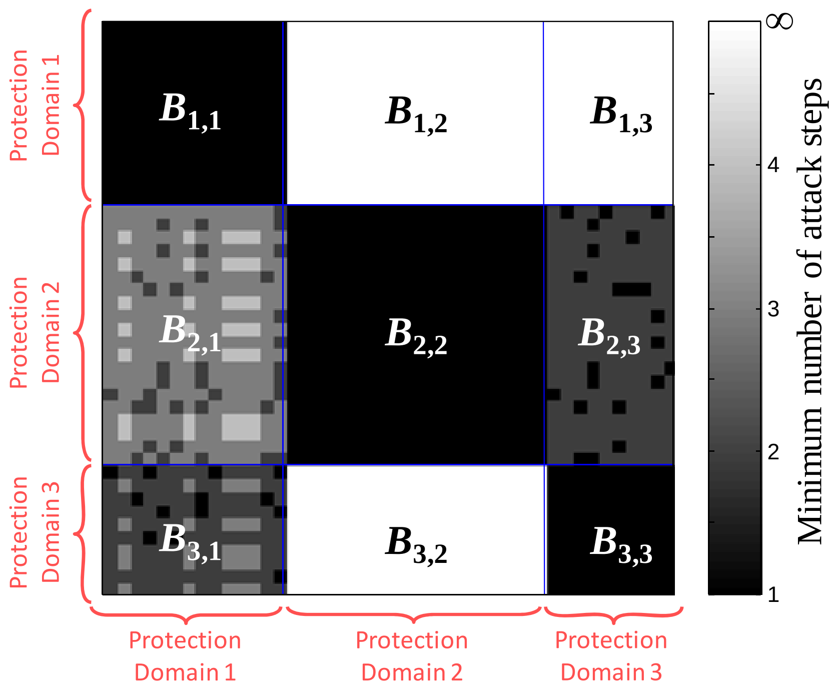


Figure 43. Multi-step reachability for attack graph in Figure 38

Figure 44 shows an attack graph adjacency matrix for a network of 730 machines. The corresponding drawn attack graph would be cluttered and difficult to understand. Again, we have aggregated low-level security conditions to machines and sets of exploits between them, so that rows and columns are machines, and non-zero entries mean at least one exploit between a pair of machines. Here, a priori ordering of machines (rows and columns) is sufficient, so that we do not apply matrix clustering.

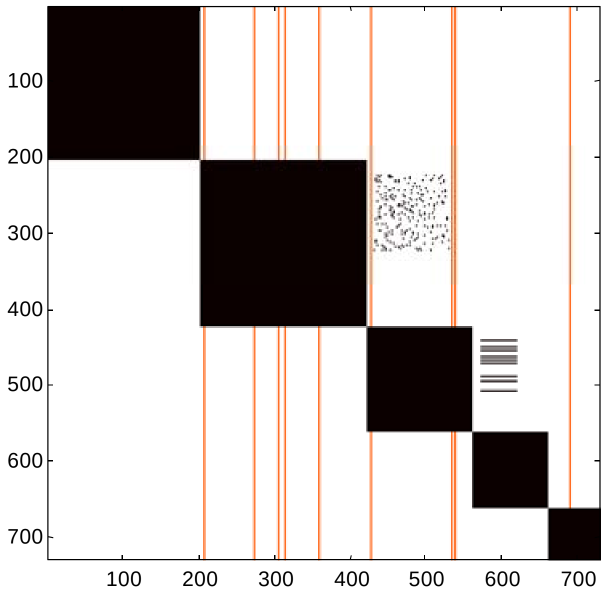


Figure 44. Attack graph adjacency matrix for baseline and changed network.

In Figure 44, black indicates the attack graph for a baseline network configuration, before any network configuration changes. Orange then indicates new attack graph edges, resulting from changes to the network configuration. In this case, the vertical orange lines (columns) indicate vulnerable web servers that have been added to the network, with a policy that all machines in the network can connect to these vulnerable servers.

Figure 45 shows the resulting transitive closure (all-step reachability) for the baseline (black) and changed (black+orange) network. Here we see that before deployment of the web servers (black), attacks are only possible within the main diagonal blocks, from block $A_{2,2}$ to blocks $A_{3,3}$ and $A_{4,4}$, and from block $A_{3,3}$ to block $A_{4,4}$.

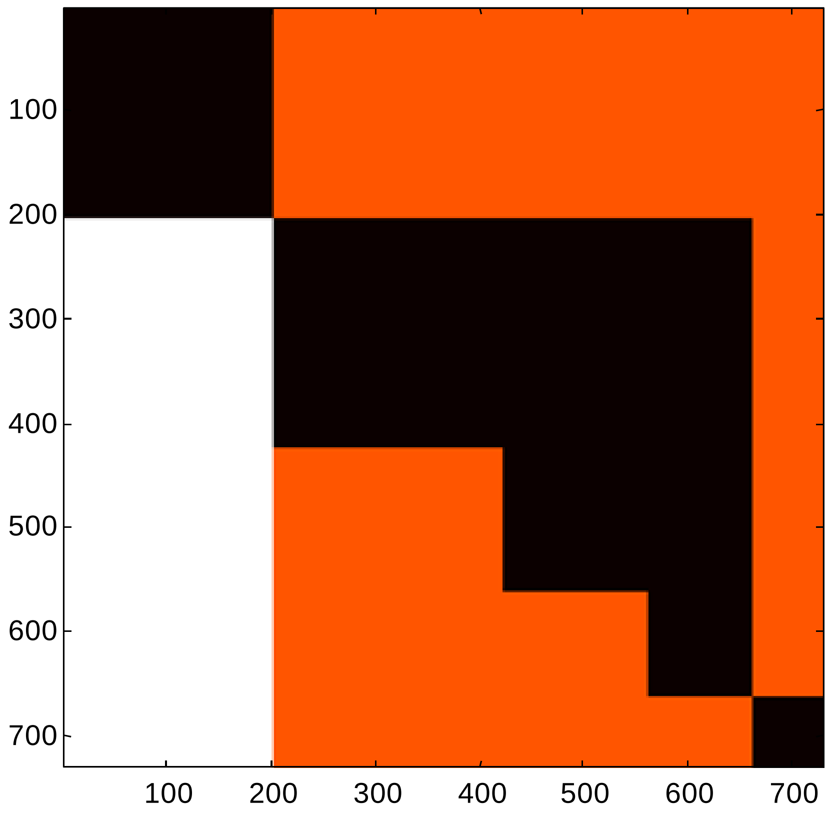


Figure 45. Transitive closure for baseline and changed network

But after deployment of the web server (black+orange), machines in block $A_{1,1}$ can reach all machines in the network, and all machines in the network can reach the machines in blocks $A_{2,2}$, $A_{3,3}$, $A_{4,4}$, and $A_{5,5}$. That is, only machines in block $A_{1,1}$ are safe from attacks outside their block.

Figure 46 shows same the multi-step vulnerability-based reachability matrix from Figure 43, but this time with intrusion alarms associated with elements of the matrix. Thus, knowing attack reachability across the network, we can categorize and correlate detected intrusions in terms of it.

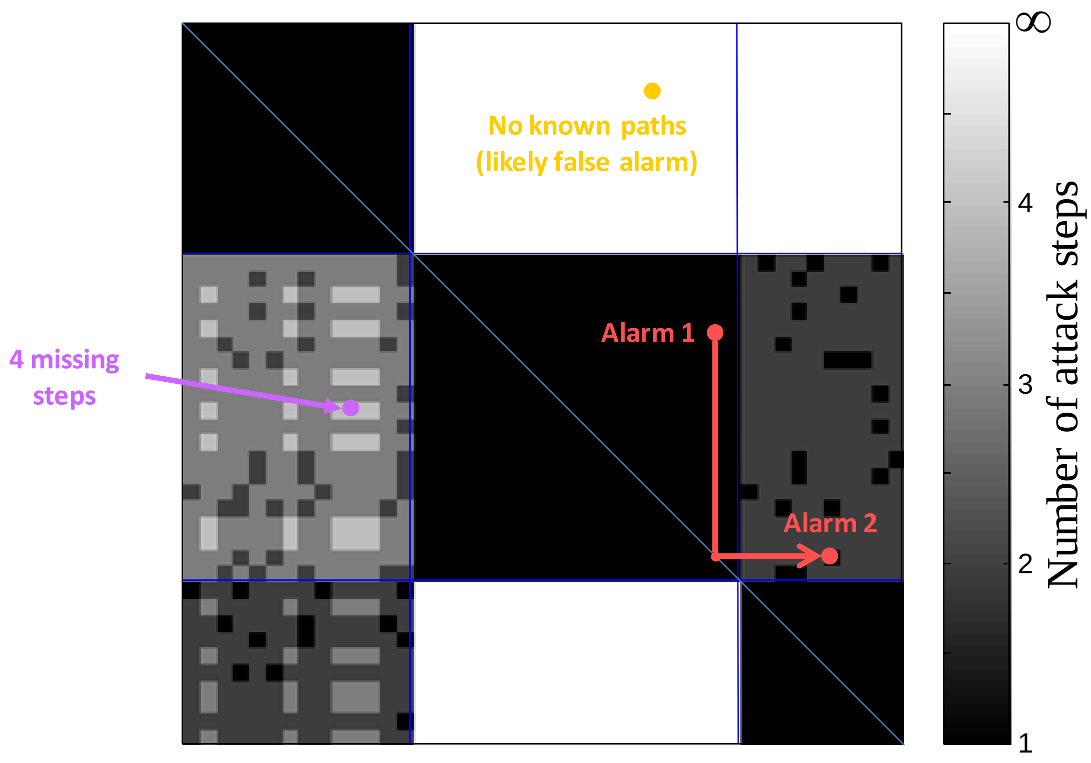


Figure 46. Correlating intrusion alarms via attack graph reachability

In Figure 46, an intrusion alarm (in yellow) occurs between a pair of machines that are known to be unreachable from one to the other. In this case, we might consider it to be a false alarm. Similarly, an intrusion alarm (in purple) occurs between a pair of machines, which, according to the attack graph, should require at least 4 attack steps to reach from one to the other. If we were to rely on traditional attack graph drawings, we would need to trace many edges before coming to this conclusion.

There are 2 other intrusion alarms in Figure 46 that according to the attack graph, are each possible one-step attacks. In fact, if we project (along a column) from Alarm 1 to the main diagonal, we find Alarm 2 on the projected row, indicating that according to the attack graph, Alarm 2 follows immediately after Alarm 1. In this case, we correlate the two alarms based on the likelihood that they are part of a coordinated attack.

Figure 47 shows another intrusion alarm associated with the multi-step reachability matrix from Figure 43. This time, we project to the main diagonal in each direction, i.e., along columns and rows, to explore forward and backward steps from this alarm. In this way, we can predict the origin of the attack (from the backward direction) and the impact of the attack (from the forward direction).

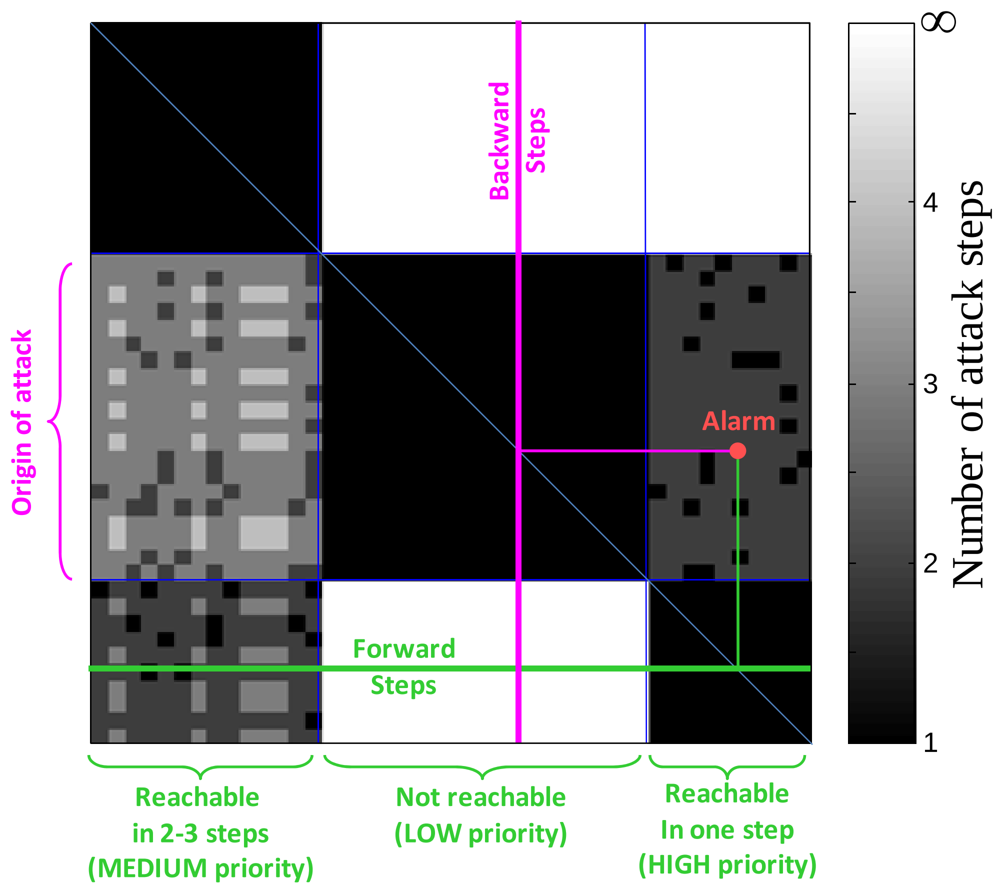


Figure 47. Predicting attack origin and impact

Projecting along the row to the main diagonal, we reach a column that has non-zero entries only within the 2nd main-diagonal block (block $A_{2,2}$). In fact, these non-zero entries are all of unity value, indicating that these are all one-step attacks. In other words, only an attack from one of the machines in block $A_{2,2}$ could have led to the detected intrusion, and that could have happened within one attack step.

In Figure 47, when we project the detected intrusion to the main diagonal along its column, the row of intersection shows all possible forward steps from the detected event. In this case, we see that machines in the 2nd block cannot be reached by the attacker, since block $A_{3,2}$ is zero-valued. For this reason, no attack response is necessary for defending machines in these columns.

However, because block $A_{3,1}$ shows reachability within 2 to 3 steps, measures might be taken to defend machines in these columns, although not at the highest priority. The highest priority should be for machines in the columns of block $A_{3,3}$, because those machines are all reachable from the detected event within a single step.

4.3 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 intrusion detection.

Now, given our knowledge of the network configuration and residual paths of vulnerability, where should we place intrusion detection 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.

Consider the testbed network in Figure 48, 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.

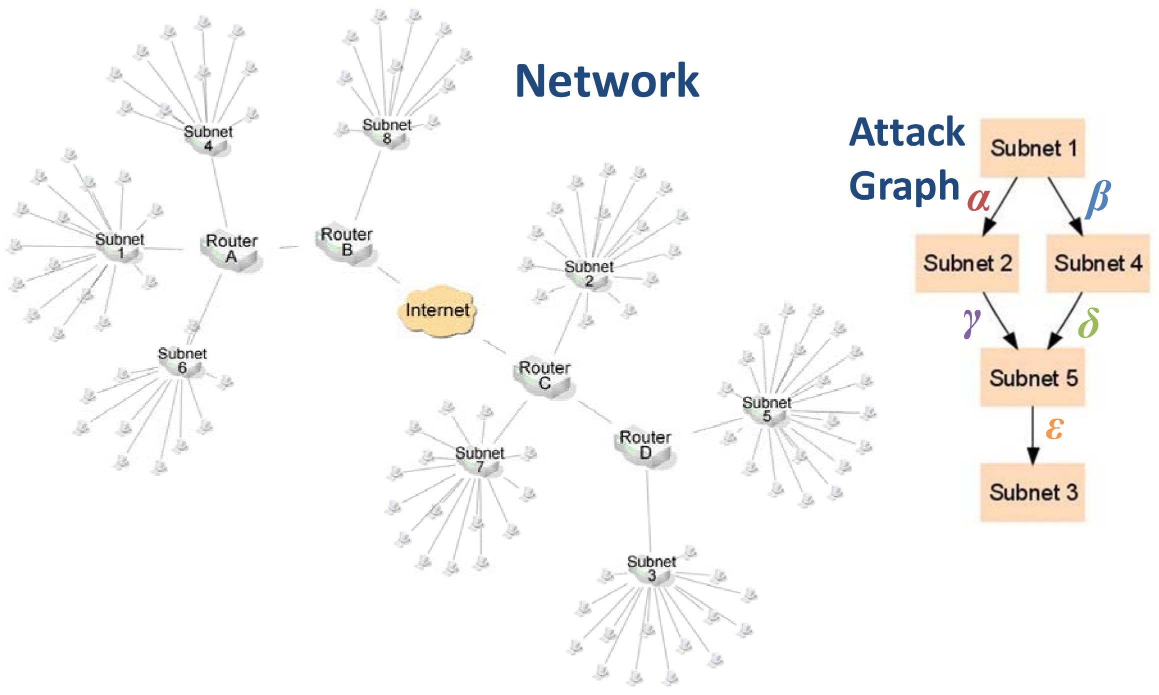


Figure 48. Testbed network and its high-level attack graph

The right side of Figure 48 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 intrusion detection system.

Now, given our knowledge of vulnerable 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. 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 intrusion detection sensors deployed on particular network devices. Each intrusion detection sensor monitors a given set of edges, i.e., can see the traffic between the given attacker/victim machines.

Consider Figure 49, which builds from the testbed network in Figure 48. 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.

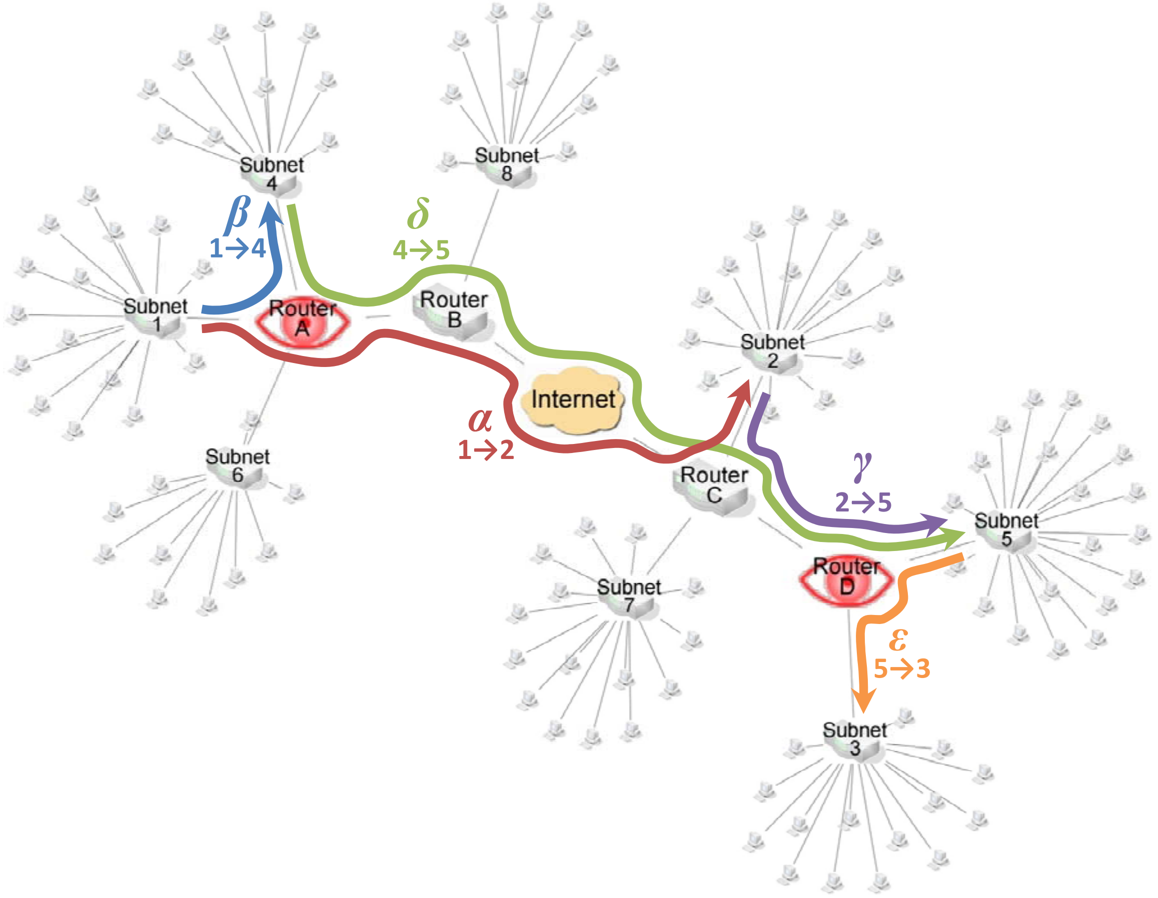


Figure 49. Optimal sensor placement for testbed network

Set cover is known to be computationally hard, one of Karp's original 21 NP-complete problems [56]. Fortunately, there is a well known polynomial-time greedy algorithm for set cover that gives good results in practice [43]. 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)} = {α, β, δ}
- Router B covers {(1,2), (4,5)} = {α, δ}
- Router C covers {(1,2), (4,5), (2,5)} = {α, δ, γ}
- Router D covers {(2,5), (4,5), (5,3)} = {γ, δ, ε}

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 [57]. 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 [58] and evolutionary algorithms [59].

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 report, our formulation of intrusion detection sensor placement as an instance of set covering places our proposed approach on firm ground.

Once intrusion detector 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 50 shows attack graph details for the testbed network in Figure 48, 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).

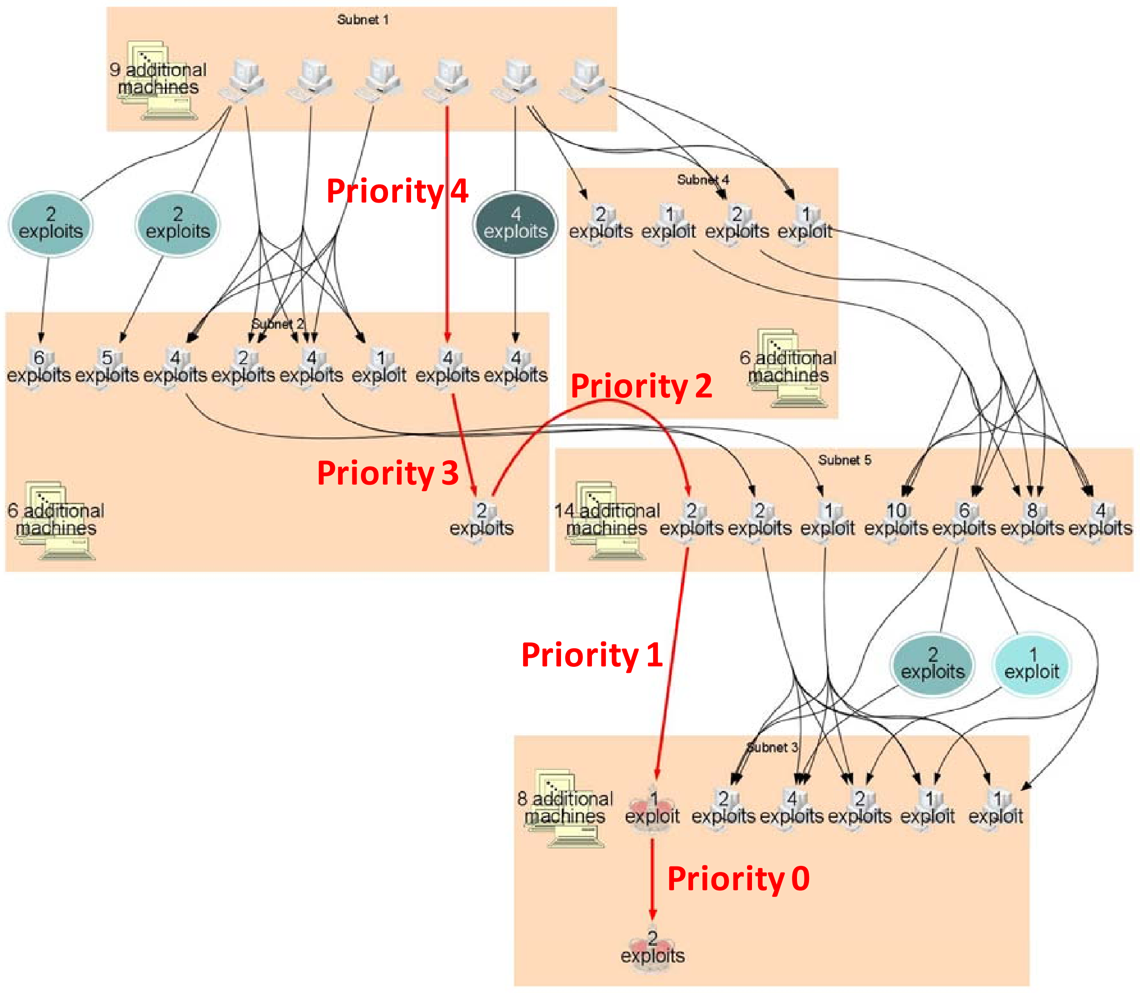


Figure 50. Priority of alerts for testbed network

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.

As an example operational scenario, in Figure 50, assume that a security operator sees the Priority-4 intrusion 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 intrusion detection sensors.

4.4 Security Metrics for Risk Analysis

Removing attack paths reduces options for an attacker, but at what cost to the defender? For example, in Figure 16, blocking ssh or rsh traffic removes certain initial network conditions, preventing exploits that depend on these vulnerable connections, thereby reducing the number of attack paths.

But what is lacking is a clear measure of exactly how security has been improved in each case. For example, which is better, blocking ssh or blocking rsh? Are these significant or only marginal improvements over the current system configuration? These are the kinds of questions that are answered though our attack graph metric.

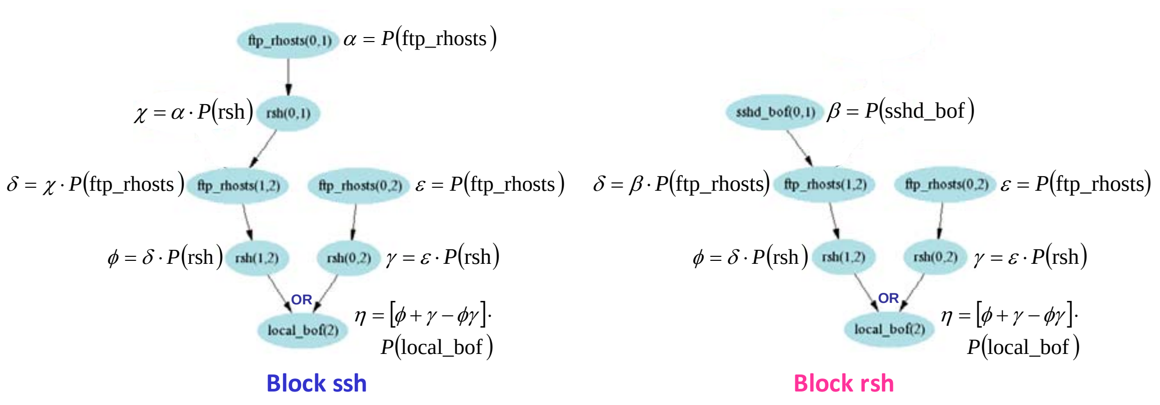
For our risk analysis, we simulate attack graphs through Monte Carlo methods. This allows us to handle uncertain input values, by modeling their distributions through statistics such as numerical ranges, mean values, etc. We then propagate attacks through the attack graph, from inputs generated according to our assumed input probability distributions. This allows us to make optimal choices for complex cyber-attack models that cannot otherwise be easily found.

Monte Carlo methods are ideal for such problems with highly complex logical (non-linear) relationships with many input variables. For optimum performance, the attack graph is computed once, and then evaluated for each Monte Carlo input sample.

In Figure 16, the network hardening options are to block either ssh or rsh traffic (to the file server) through the firewall. Here, we assume that no software patches or other forms of mitigation are available for the ftp and buffer overflow vulnerabilities on the inner network. Thus the available options are to block these services.

In this example, the remote shell service is working correctly (even though it is being leveraged in the attack), and needs no patching. Let us assume that the workstation (*Machine 1*) needs the ftp services on the inner network, i.e., we avoid blocking ftp traffic through the firewall. Further, we assume that the workstation needs at least one type of shell access (either remote shell or secure shell) on the inner network. This means we can block either rsh or ssh traffic, but not both.

Figure 51 shows the residual attack graphs resulting from blocking rsh or ssh traffic. In other words, for each network hardening choice, a particular set of attack paths still remains. We then use this residual attack graph to compute our attack graph metric. That is, we propagate likelihoods of vulnerability exploitation according to the logical relationships in the attack graph. The result is an overall likelihood of reaching the attack goal, for each of our two choices of network hardening.



ftp_rhosts(0,1)
$\alpha = P(\text{ftp\_rhosts})$
$\chi = \alpha \cdot P(\text{rsh})$
rsh(0,1)
$\delta = \chi \cdot P(\text{ftp\_rhosts})$
ftp_rhosts(1,2)
ftp_rhosts(0,2)
$\varepsilon = P(\text{ftp\_rhosts})$
$\phi = \delta \cdot P(\text{rsh})$
rsh(1,2)
rsh(0,2)
$\gamma = \varepsilon \cdot P(\text{rsh})$
OR
local_bof(2)
$\eta = [\phi + \gamma - \phi\gamma] \cdot P(\text{local\_bof})$
Block ssh
sshd_bof(0,1)
$\beta = P(\text{sshd\_bof})$
$\delta = \beta \cdot P(\text{ftp\_rhosts})$
ftp_rhosts(1,2)
ftp_rhosts(0,2)
$\varepsilon = P(\text{ftp\_rhosts})$
$\phi = \delta \cdot P(\text{rsh})$
rsh(1,2)
rsh(0,2)
$\gamma = \varepsilon \cdot P(\text{rsh})$
OR
local_bof(2)
$\eta = [\phi + \gamma - \phi\gamma] \cdot P(\text{local\_bof})$
Block rsh

Figure 51. Residual attack graphs for network configuration choices

In the figure, the cumulative likelihood of each exploit is shown, as propagated through the attack graph. For example, the likelihood of *rsh(0,2)* occurring is the conjunctive combination (Boolean AND) of *ftp_rhosts(0,2)* and *rsh(0,2)*, while the likelihood of *local_bof(2)* is the disjunctive combination (Boolean OR) of *rsh(0,2)* and *rsh(1,2)*. Since *local_bof(2)* yields the overall goal for this attack scenario (compromise of the database server), the likelihood of *local_bof(2)* occurring is the overall attack graph metric.

In Figure 51, the model inputs *P*(•) represent the inherent chance of each exploit occurring, independent of other exploits. For example, these could be relative frequencies of events observed on a network over a period of time. Or, they could be scores from a source such as NVD (CVSS scores) or Symantec DeepSight.

In this particular case, let us assume we have a minimum and maximum likelihood value for each exploit. We model these likelihoods as uniformly distributed between their minimum and maximum values, i.e., *ftp_rhosts* and *rsh* between [0.5, 0.6], and *sshd_bof* and *local_bof* between [0.1, 0.2]. We can interpret such input distributions as likelihoods of successful exploit execution over a given period of time (month, year, etc.).

Figure 52 shows the result of Monte Carlo simulations of the attack graph model in Figure 51. Each choice of network configuration (block ssh, block rsh, and no change) has a corresponding residual attack graph.

In the simulations, we generate large samples of inputs (individual vulnerability scores) randomly according to the given distributions. We then propagate these vulnerability scores according to the logical relationships of the corresponding residual attack graph. In this way, we measure the overall likelihood of system compromise, for the given choice of network hardening.

For example, at the top of Figure 52, the input for *ftp_rhosts(0,1)* is a uniform distribution $U_1[0.5, 0.6]$. When this is combined conjunctively (Boolean AND) with the input for *rsh(0,1)*, which is $U_2[0.1, 0.2]$, the resulting $U_1 \bullet U_2$ is the triangle distribution $T[0.24, 0.36]$ shown as the combined likelihood for the attacker accomplishing *rsh(0,1)*.

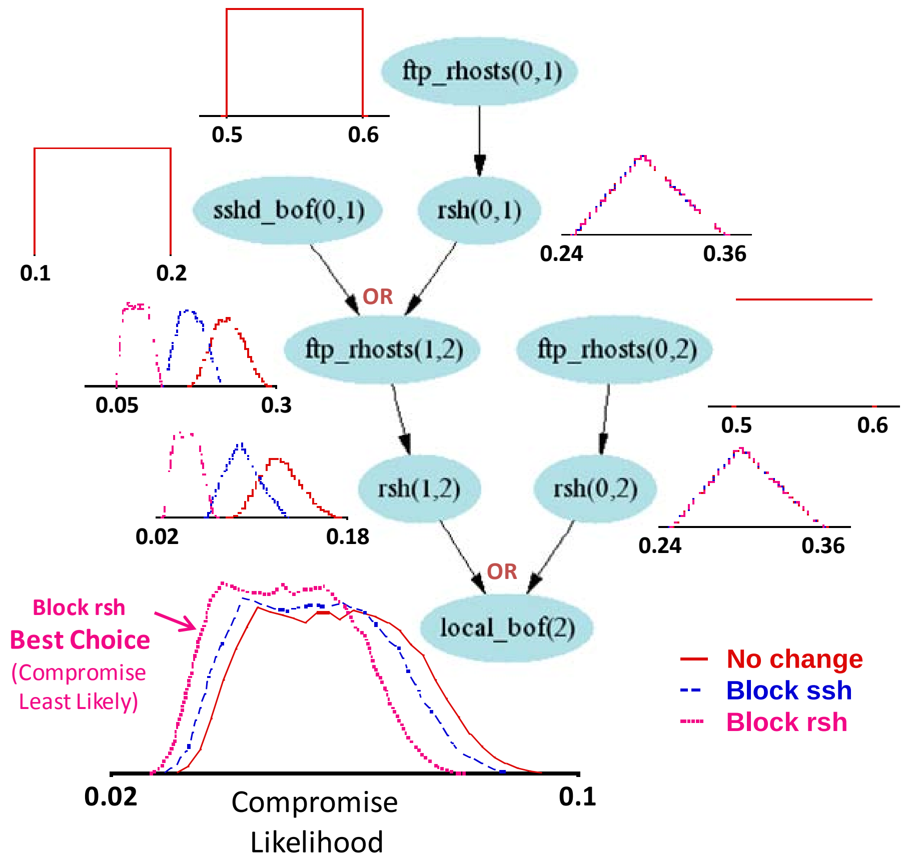


Figure 52. Attack-graph metrics for each network configuration choice

The simulation output (bottom of Figure 52) is the overall attack-graph metric distributions. This shows the overall likelihood of database compromise (the attack goal) for each option of network configuration. This shows that blocking rsh traffic provides the best overall protection for this system. In particular, with this network hardening choice, there is lower likelihood of attack compromise.

Simply comparing representative (e.g., mean) values gives is an immediate determination of the best choice. Or we could gain further insight by comparing such features as uncertainty spread.

Through our attack graph simulations, we analyze risks in terms of attack likelihoods. In this section, we extend this risk analysis to include associated operational costs, attack impact costs, etc. This is useful for economic analysis, and supports informed decision making about return on security investment. In other words, we can quantify whether expenditures on additional security measures are justified by reduction in expected losses from security breaches.

Figure 53 shows a model for such return-on-investment analysis. We first compute the metric for likelihood of attack compromise, i.e., the distribution of resulting attack graph metrics shown in Figure 52. This metrics distribution feeds a cost analysis, which combines compromise likelihood from the attack graph with the projected costs of an actual compromise. This is then weighed against the costs of actually implementing network changes to mitigate the risk.

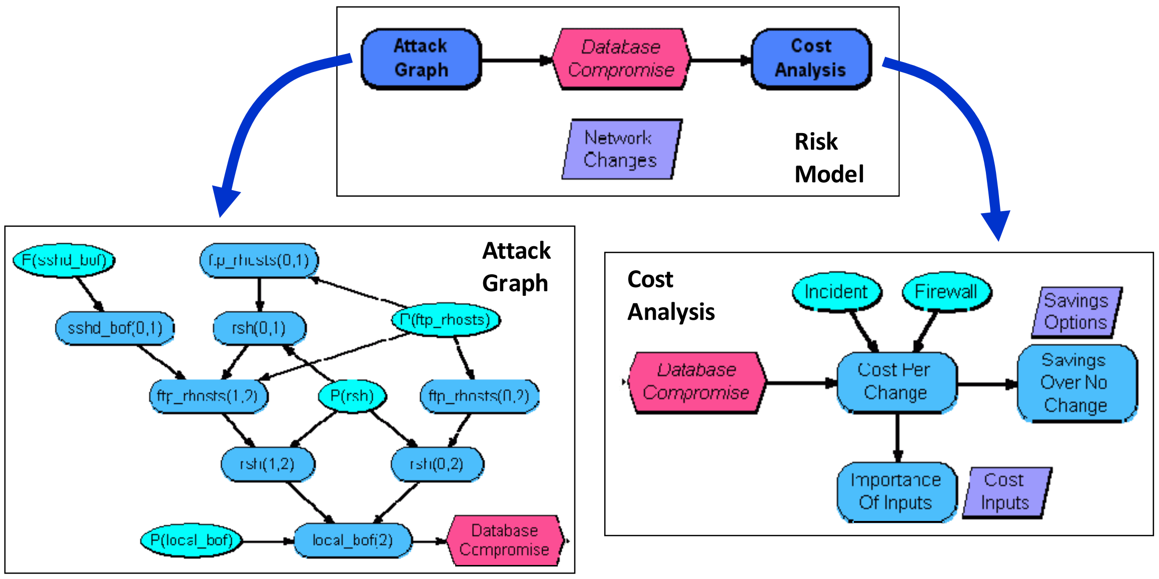


Figure 53. Security return-on-investment model

In other words, each network security option has a particular implementation cost. Each option in turn reduces the risk of a security breach by a certain amount, and there is an assumed loss for a breach actually occurring. The overall cost for each security option is then

Overall Cost = Implementation Cost + Cost of Security Breach × Likelihood of Breach

We seek the security solution that optimizes overall costs. In other words, the cost of implementing a security measure is offset by a corresponding reduction in expected value of cost for a breach, in which the cost of a breach is weighed by its likelihood of actually occurring.

In this example, we assume the estimated cost of recovering from a database breach (Incident) is $200,000, with standard deviation of $20,000. Each network configuration has an associated likelihood of a breach occurring, with a corresponding multiplicative reduction (from the full $200,000) in expected loss.

The estimated cost for implementing firewall changes (*Firewall*) is $1,000, with standard deviation of $100. So the question is whether the firewall implementation costs are justified in terms of reduced risk (expected loss), versus simply making no change to the network.

Figure 54 shows the resulting probability distributions (density and cumulative) of overall costs for each network configuration. In terms of overall cost, blocking ssh traffic is now seen to actually cost more than simply making no change to the network. This is because the slight decrease in breach likelihood (expected loss) is outweighed by the cost of implementing the firewall change. On the other hand, blocking rsh traffic reduces breach likelihood more, making its overall costs lower versus leaving the network unchanged.

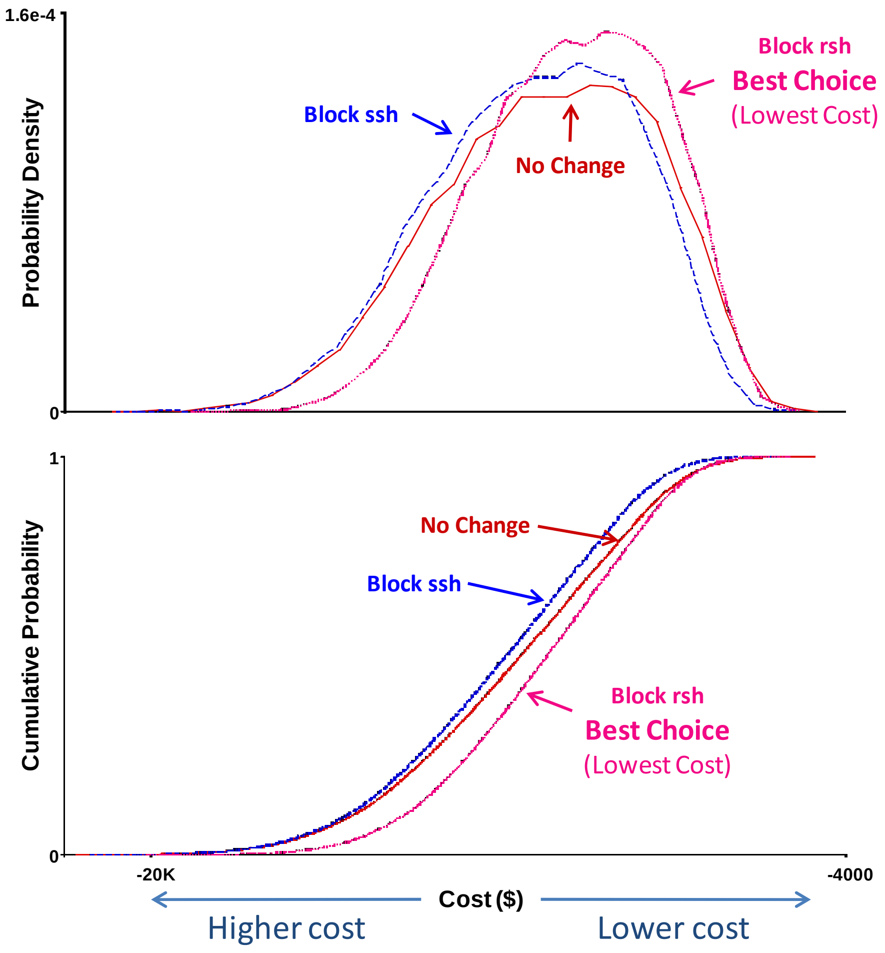


Figure 54. Cost of each network change based on attack-graph metrics

In our risk model in Figure 53, the overall cost for each network configuration (*Cost Per Change*) is input to *Savings Over No Change*. This directly compares the costs of blocking rsh and ssh (respectively) to the cost of no change, for each Monte Carlo sample. In particular, we use the "no change" option as the baseline, and compute differences in mean, minimum, and maximum costs for each hardening option.

The resulting model comparisons are shown in Figure 55. We see that blocking rsh traffic can provide almost $2,500 in reduced cost (maximum difference from no change), while blocking ssh traffic can incur almost $1,000 additional cost (minimum difference from no change).

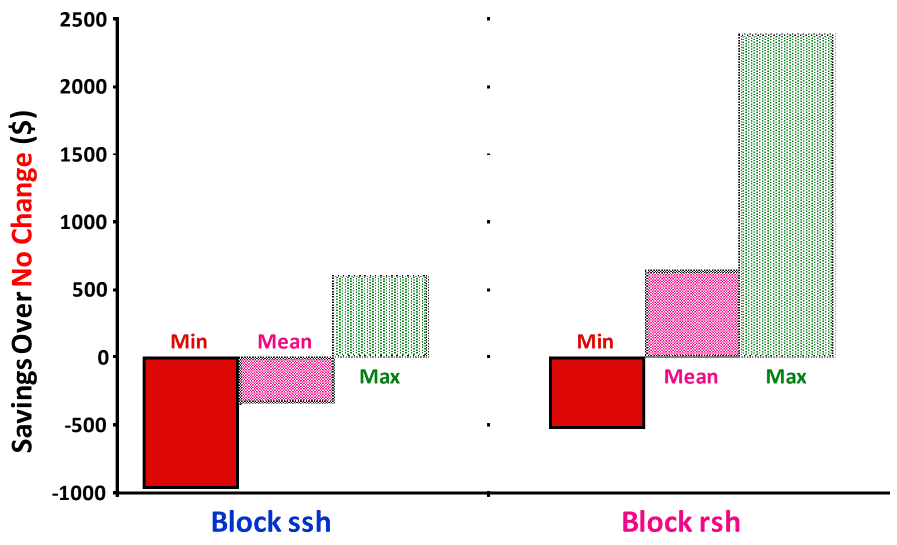


Figure 55. Comparative savings (versus no change)

To quantify the confidence in our analysis, we begin by measuring how strongly the model output is correlated with each model input. This shows the relative importance of each input variable, i.e., how strongly its uncertainty translates to uncertainty in the output result. In our risk model (Figure 53), this is *Importance Of Inputs*, which correlates *Cost Per Change* (Figure 54) with each of the model inputs (indexed by *Cost Inputs*).

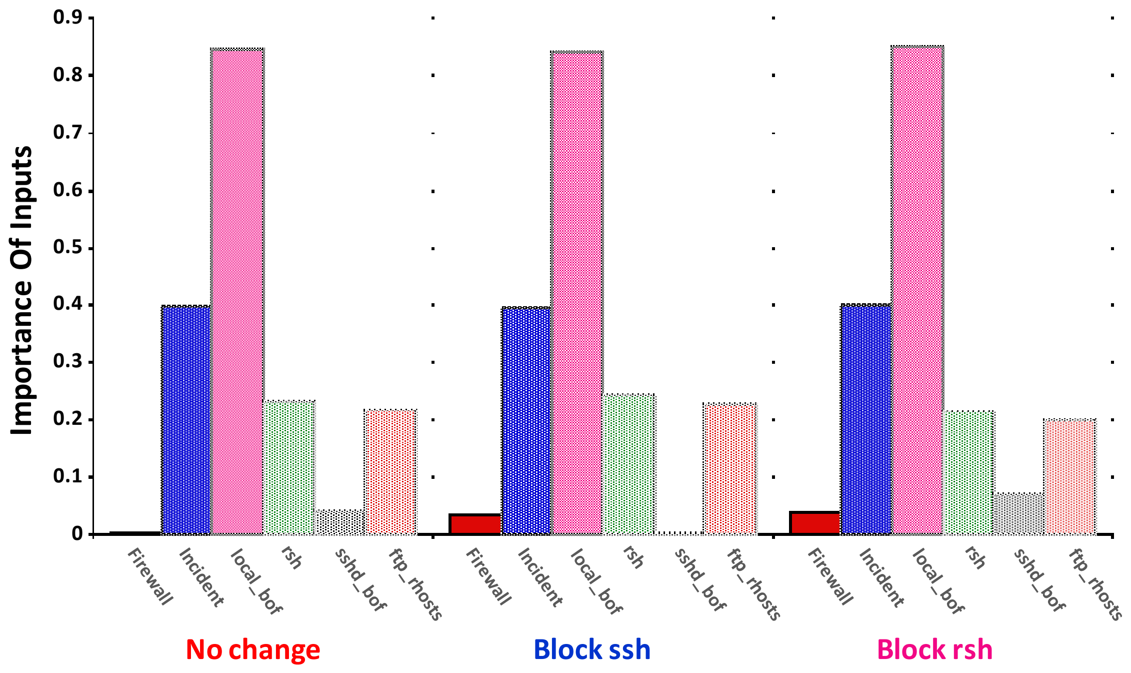


Figure 56. Relative importance of model inputs

Figure 56 shows the resulting input/output correlations, for each network configuration (block rsh, block ssh, or no change). For all configurations, the output uncertainty is dominated by the likelihood of *local_bof*, a critical exploit that is included in all attack paths. The next most important input is the cost of a database compromise incident, followed by likelihoods of the *rsh* and *ftp_rhosts* exploits. Uncertainty in firewall costs or the likelihood of *sshd_bof* have little impact on uncertainty of results.

We gain further confidence by testing how model input changes might potentially alter our choice of the best network configuration. In particular, we calculate how overall costs (*Cost Per Change*) vary as each model input changes.

Figure 57 is the variation in *Cost Per Change* as a function of each input individually, with all other inputs held at their nominal (mean) values. To test robustness, we vary input values well outside their actual ranges in the model. The goal of this analysis is to verify that our conclusions are valid throughout the entire range of possible model input values.

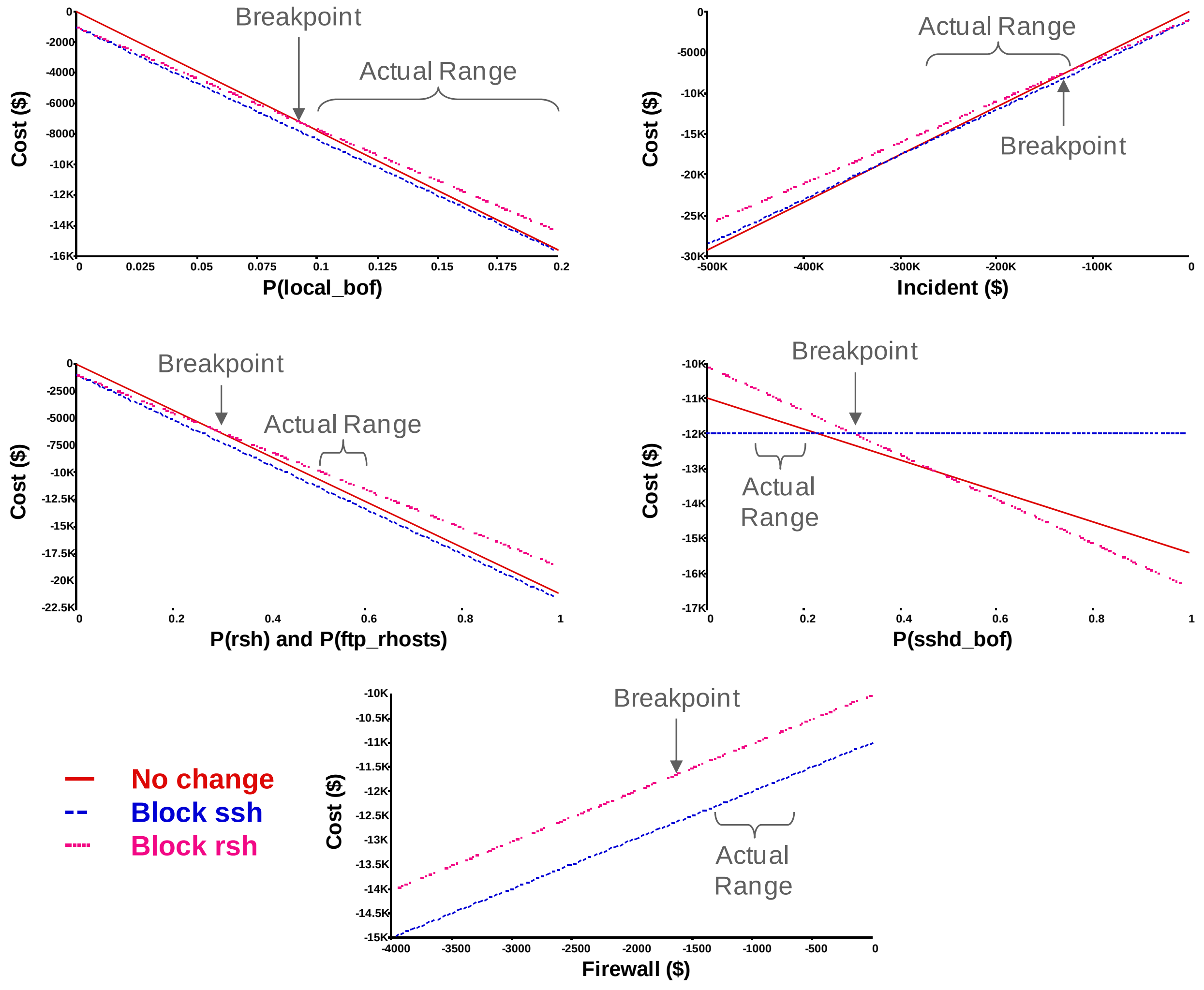


Figure 57. Cost dependency on individual inputs

The confidence results in Figure 57 validate our conclusions. The upper left of the figure shows how model output depends on the likelihood of the *local_bof* exploit. If the likelihood of *local_bof* were less than the breakpoint value of 0.09, then making no change to the network would be the best solution. But in our model, the range of *local_bof* likelihood is actually between 0.1 and 0.2. Blocking rsh traffic remains the best solution throughout this range.

The upper right of Figure 57 shows the how the model output depends on the cost of a database compromise incident. In the model, the cost per incident is between about $130,000 to $275,000. Except for a small portion of this range (up to about $135,000), blocking rsh traffic remains the best solution.

Figure 57 also shows the model dependencies on the remaining input variables. For all these variables, blocking rsh traffic is the best solution within the full range of actual model values. Overall, we have a strong measure of confidence that blocking rsh traffic is the best choice for providing return on security investment.

4.5 Formal Evaluations

Various activities were performed for testing the TVA tool, and evaluating its new capabilities. Preliminary testing was performed by independent analysts at AFRL/Rome, in preparation for a more in-depth formal evaluation. Here we summarize some of the results of preliminary testing.

As part of preliminary testing, a network was developed to serve as a testbed environment for evaluating TVA and other network security tools. As part of its design for testing TVA, the intention is for the network to have sufficient complexity for evaluating multiple-step attack paths.

This means that a number of subnets are needed, with connectivity-limiting devices like firewalls in place. Testing numerous types of vulnerabilities requires a range of machine platforms, operating systems, services, host applications, etc. Overall, the intention is simulate environmental conditions that might be encountered within a DoD network.

Figure 58 shows the structure of the testbed network for preliminary testing. There are 7 subnets for testing TVA, with a total of about 70 host machines. Of the 7 total subnets, 3 subnets were combined into one TVA protection domain, since they are known to have unrestricted connectivity among them. This results in 5 protection domains for the TVA network model.

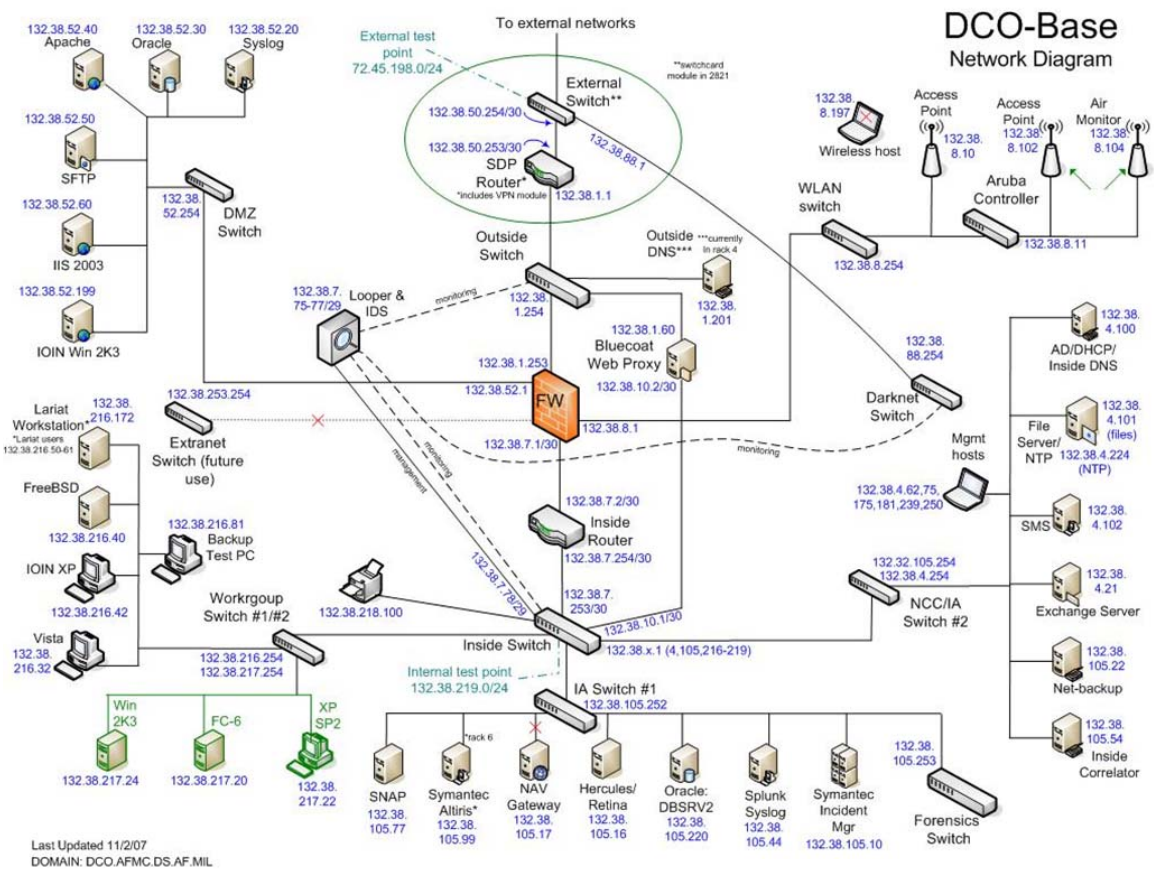


Figure 58. Testbed network for preliminary testing

To build the network model for input to TVA, Nessus scans were performed from each of the testbed subnets (5 protection domains). Each scan targeted the entire testbed network. This provides vulnerable connectivity from different network vantage points, as for an actual attacker at different locations. TVA then merges all the scan data into a single network model that reflects connections to vulnerable services throughout the network.

Figure 59 shows the resulting attack graph of all possible attacks through the network. This shows that the graph is nearly fully connected, i.e., from any point in the network, there are direct attacks (or at most 2-step attacks) to anywhere else in the network. This suggests that an operational network having this configuration is fully open to attack.

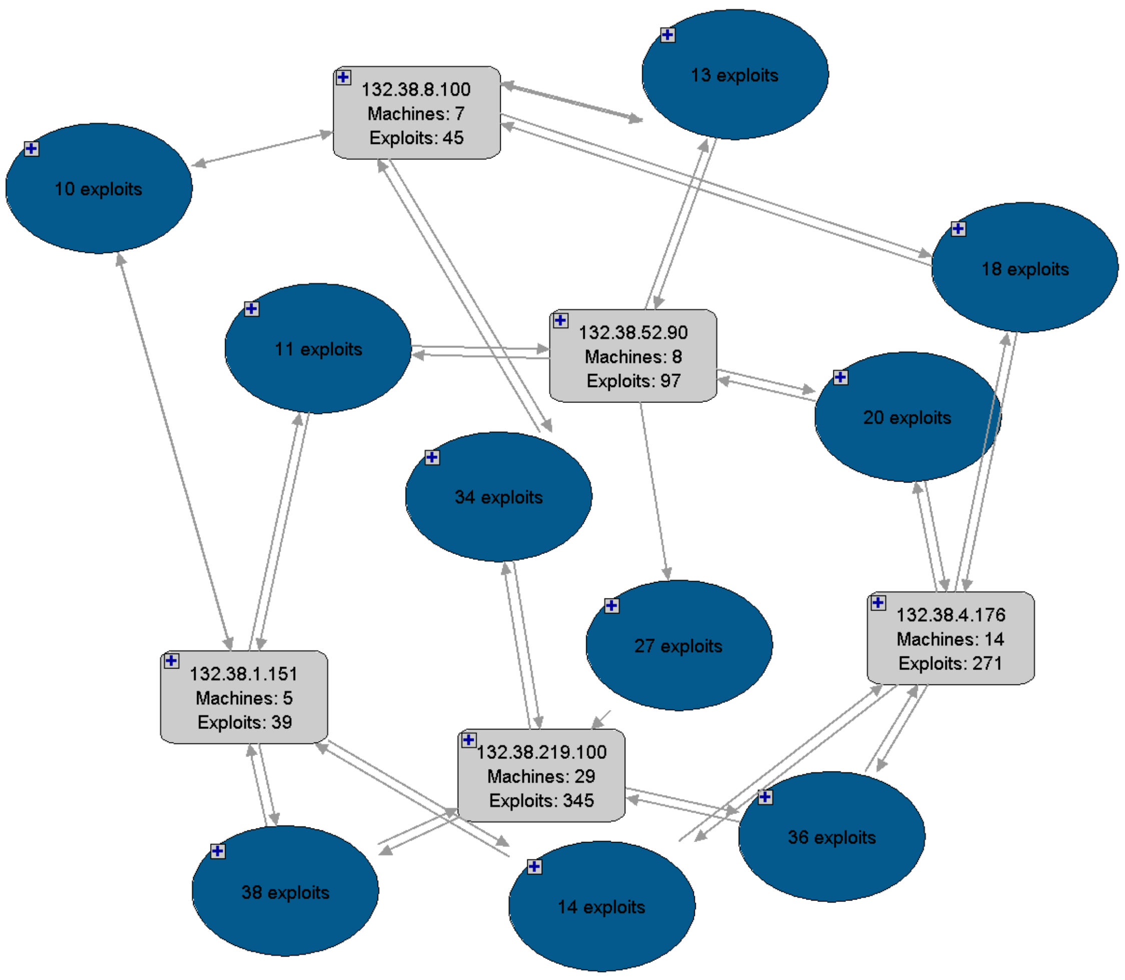


Figure 59. Attack graph for preliminary testing

Figure 60 shows the same attack graph as Figure 59, this time with the protection domains expanded to show the machines in each domain. This shows that there is a direct one-step attack from/to nearly every machine in the network.

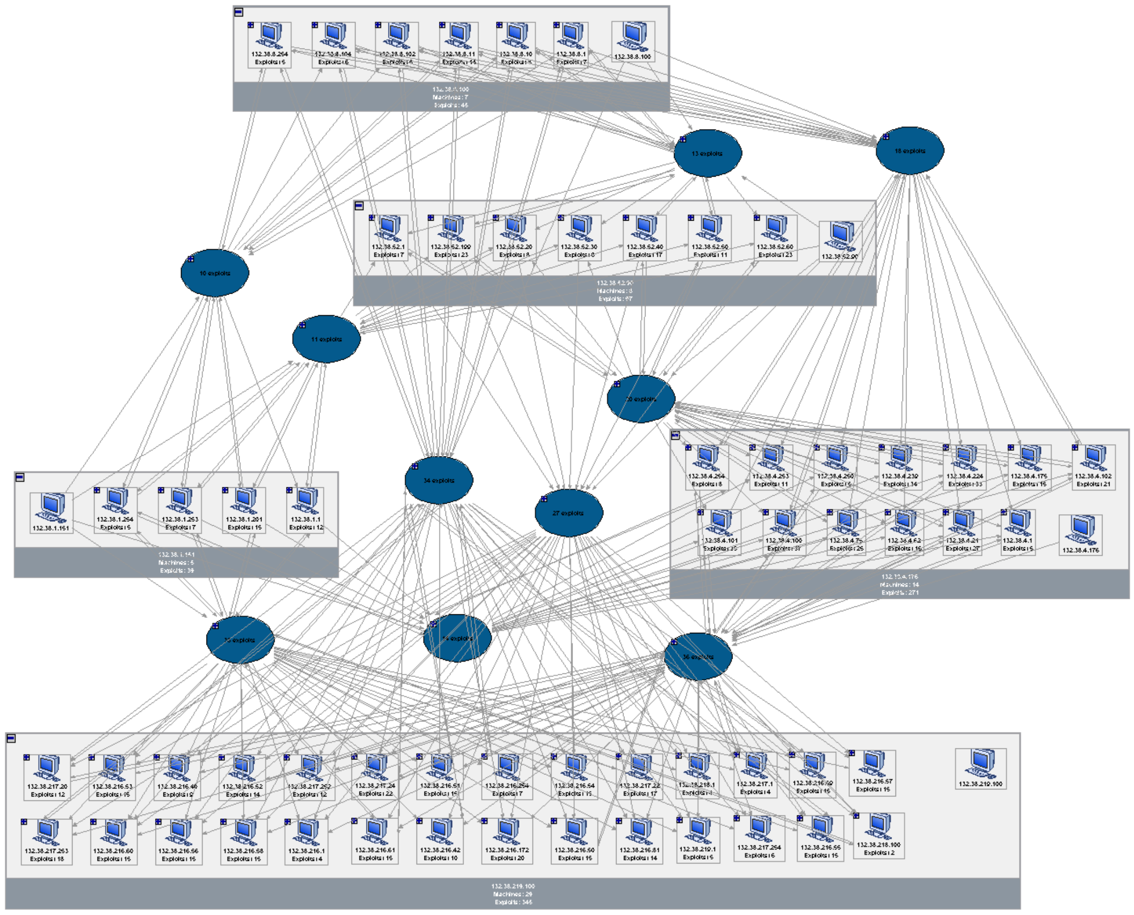


Figure 60. Attack graph for preliminary testing (expanded)

Figure 61 shows attacks specifically between 2 protection domains. This shows that for every victim machine in the lower domain, only a subset of all its vulnerabilities is reachable (through the firewall) from the upper domain. For example, the machine on the far right has 25 total vulnerabilities (reachable within the protection domain itself). But only 4 of those 25 vulnerabilities are reachable from the other domain.

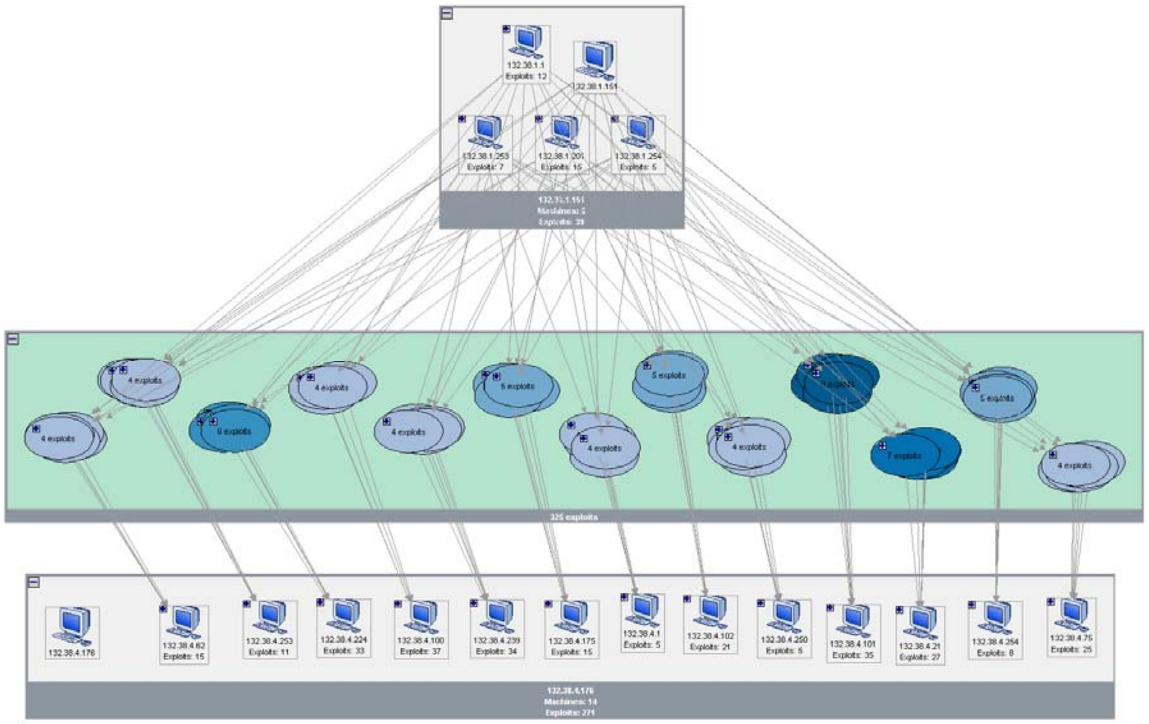


Figure 61. Attacks between a pair of protection domains

In Figure 61, for each victim machine in the lower domain, the attack graph shows exploitable vulnerabilities from all machines in the upper domain. This is the result of an implied policy during the import of Nessus scan data by the TVA tool. In particular, we assume that the scan from a subnet represents vulnerabilities that are reachable from *all other* machines in that subnet.

This is a worst case assumption that, in the absence of additional information, the firewall rules are liberal. We subsequently changed this policy to show only reachable vulnerabilities that are actually detected by Nessus scans. The presence of additional reachability can then be inferred by looking at the attack graph. This in turn makes the attack graph less complex and easier to understand. Also, we supplement vulnerability scan data with firewall rule data to provide the additional vulnerable connectivity in the network model.

A more formal evaluation of our TVA was then performed by independent analysts at AFRL/Rome, a testbed network environment. Here we summarize some of the results of that evaluation.

The testbed network was further developed after preliminary testing. The testbed network design includes sufficient subnet complexity and range of hosts, services, firewalls, host applications, etc. The intention was simulate the type of environmental conditions for security testing that might be encountered within a DoD network.

Figure 62 shows the structure of the testbed network for evaluation of the TVA tool. Functionally, 9 subnets were used for building a TVA network model. This involved deploying a Nessus scanner in each of the 9 subnets. Then each scanner was configured to target the full set of 9 subnets. The resulting 9 Nessus scan files were then imported into TVA.

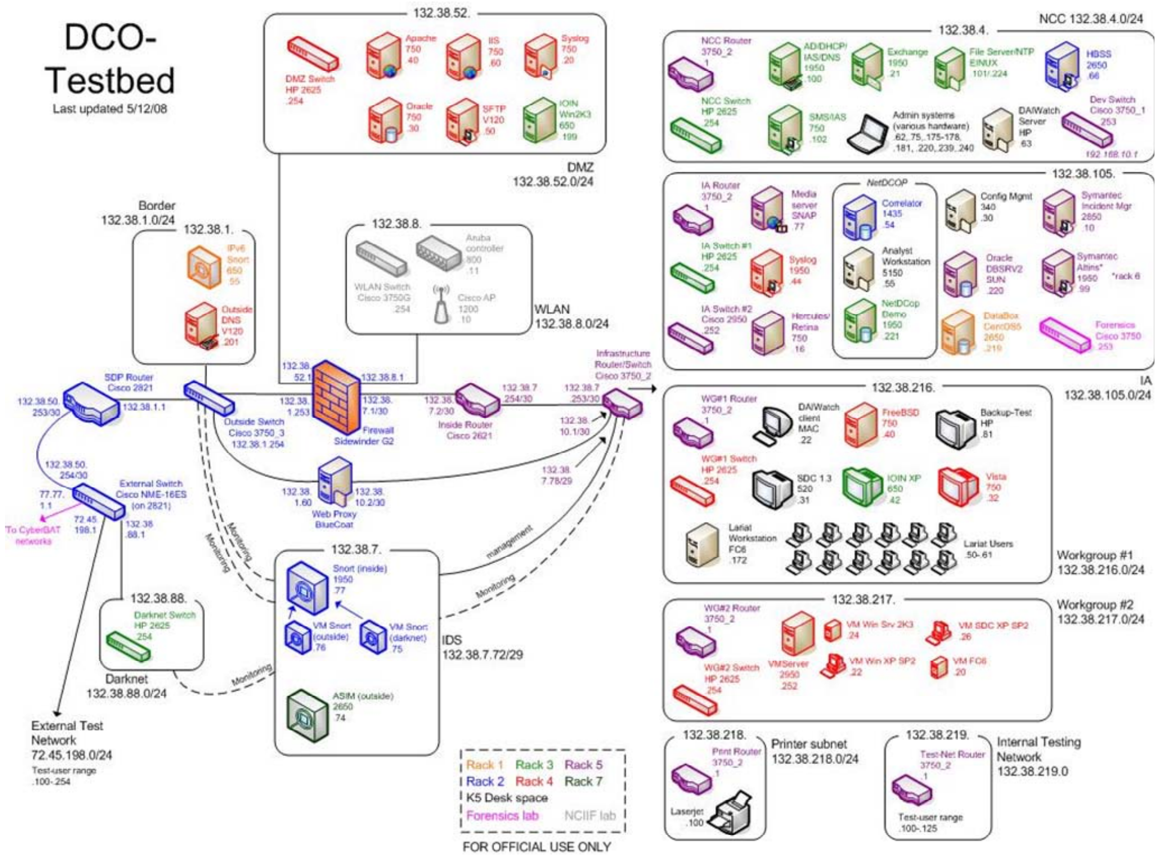


Figure 62. Testbed network for TVA tool evaluation

From this, we generate a baseline attack graph. This is shown in Figure 63. This attack graph is configured to generate attacks from all possible attack starting points, and all possible attack goals. In other words, it shows all possible attack paths, unconstrained by start or goal. Because the Nessus scans were run from different parts of the network, TVA is able to infer firewall effects.

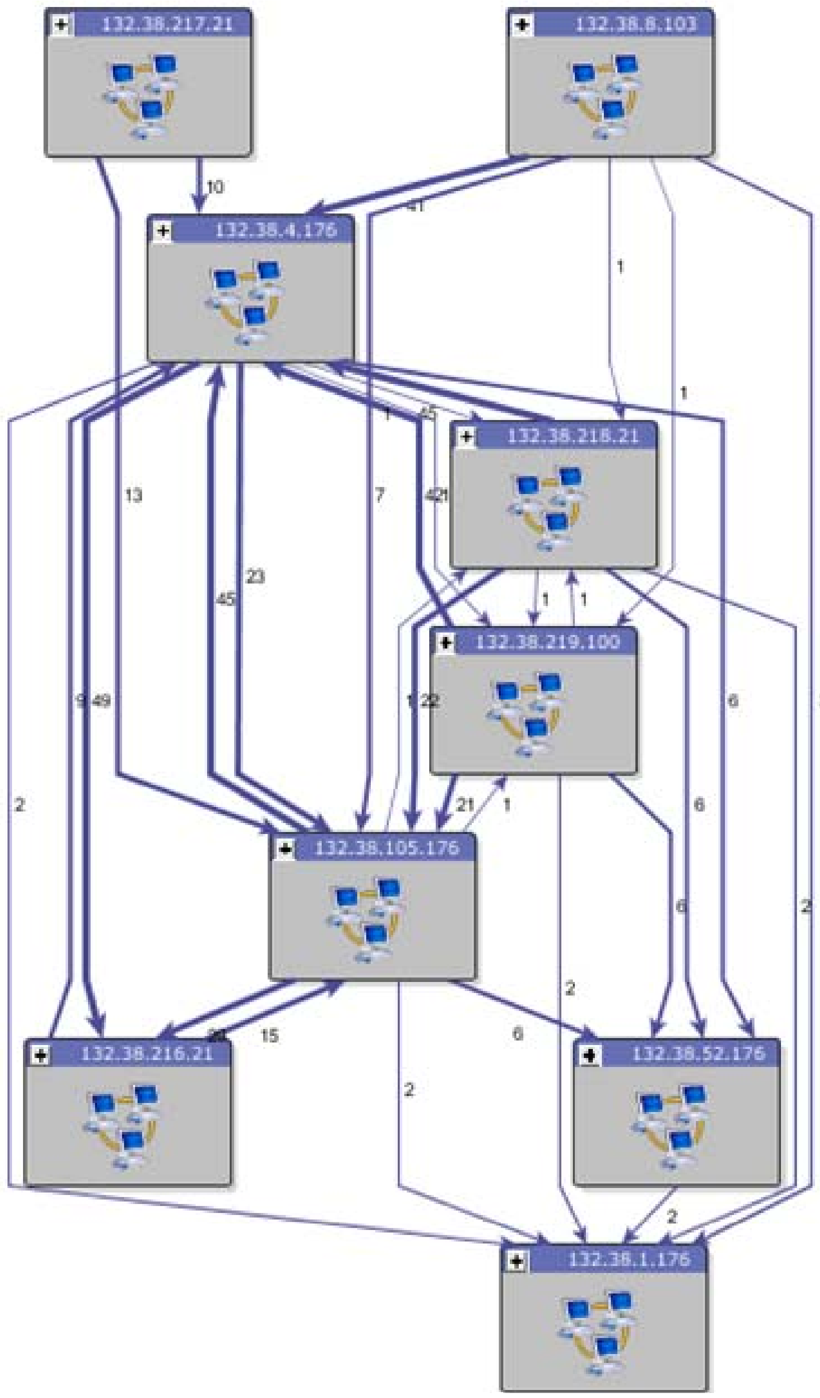


Figure 63. Baseline attack graph for Nessus scan data

Some interesting aspects of the attack graph in Figure 63 are that the .217 subnet and .8 subnet are attack sources, but no attacks lead into them. Conversely, the .1 subnet is an attack sink, with no attacks away from it. The .4 subnet has the highest density of connections to other subnets.

Figure 64 shows the same attack graph as Figure 63. Here the subnets in the attack graph are repositioned to show certain attack relationships more clearly.

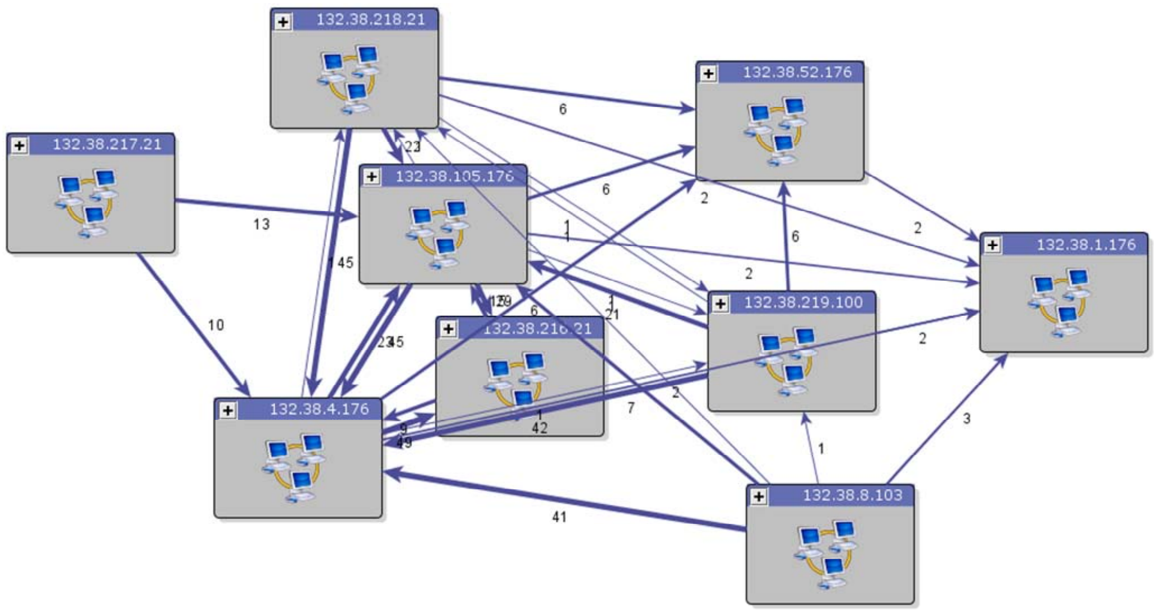


Figure 64. Repositioned baseline attack graph

The Nessus scan data was supplemented with data from the Sidewinder firewall in the testbed network. In particular, a new component of the TVA import process is able to parse Sidewinder data and apply it to the TVA network model. Figure 65 shows the resulting attack graph.

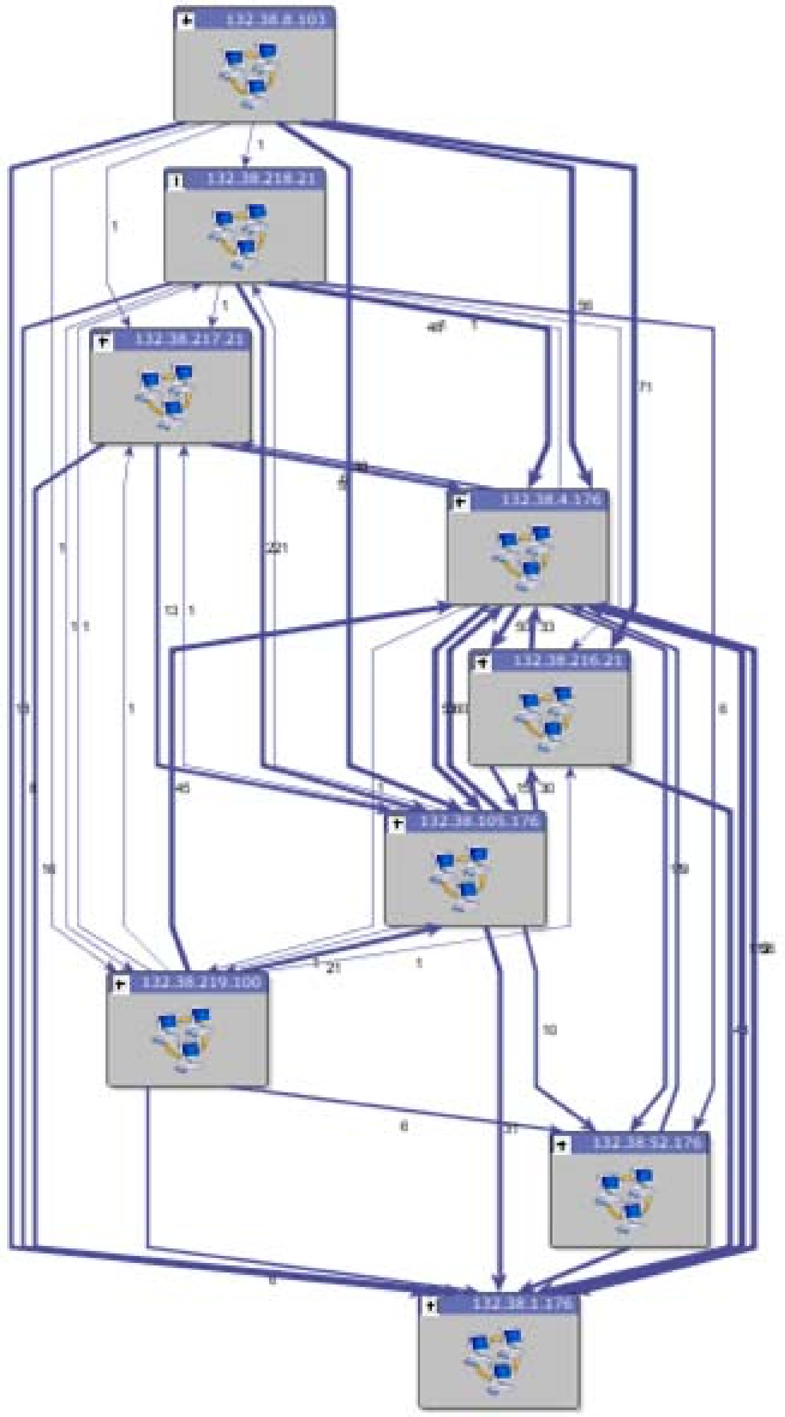


Figure 65. Attack graph with Sidewinder firewall rules data added

Compare the attack graph in Figure 65 with the baseline in Figure 63. The .217 subnet is no longer solely an attack source, because of the vulnerable connection introduced from the .8 subnet to the .217 subnet. The .1 subnet is still an attack sink. But overall, the attack graph is significantly more densely connected.

The addition of Sidewinder data in the model has revealed vulnerable connections that are not in the Nessus scan data. This is because the Nessus scans were performed from a single source host in each subnet. The firewall may have host-specific rules that block certain source hosts but allow others. This information could also be obtained from additional Nessus scans from different source hosts in each subnet.

Figure 66 shows the same attack graph as Figure 65. Again, the subnets in the attack graph are repositioned to show certain attack relationships more clearly.

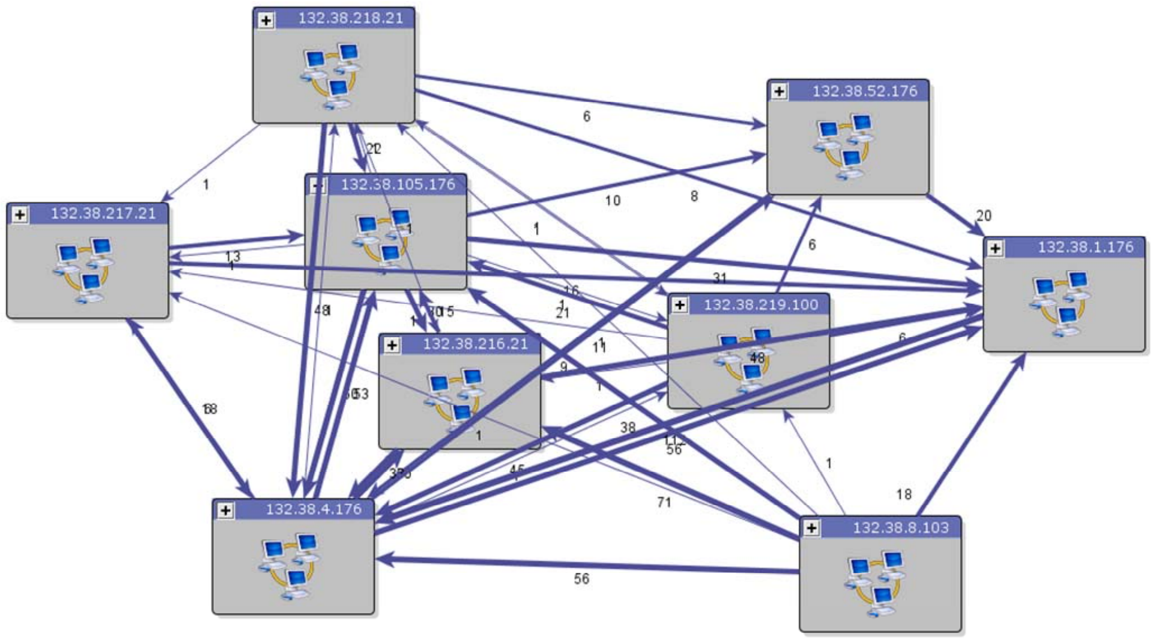


Figure 66. Repositioned attack graph for added firewall data

The "direct paths" option of our TVA tool constrains the attack graph to include only the most direct paths from attack start to attack goal. In general this may be multiple-step paths, if those are the most direct. This option highlights the most critical vulnerability paths. Figure 67 shows the attack graph for the testbed network, using direct paths.

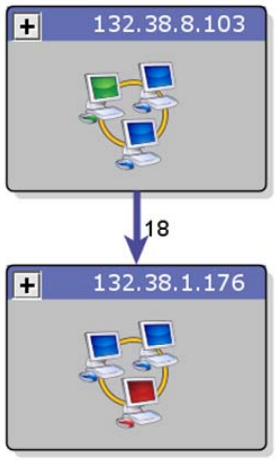


Figure 67. Direct path showing single-step attack from start to goal

Here we use a machine in the .8 subnet as the attack starting point, and a machine in the .1 subnet as an attack goal. In general, our TVA supports an arbitrary choice of attack start and goal, even for direct paths. The attack graph in Figure 67 clearly shows that there are 8 vulnerabilities that are directly exploitable (in one attack step) from the .8 subnet to the .1 subnet.

Figure 68 shows an attack graph for the same network model, this time using the .52 subnet rather than the .1 subnet as the attack goal (critical network resource to protect). In this case the attacker can reach the critical resource in 2 attack steps.

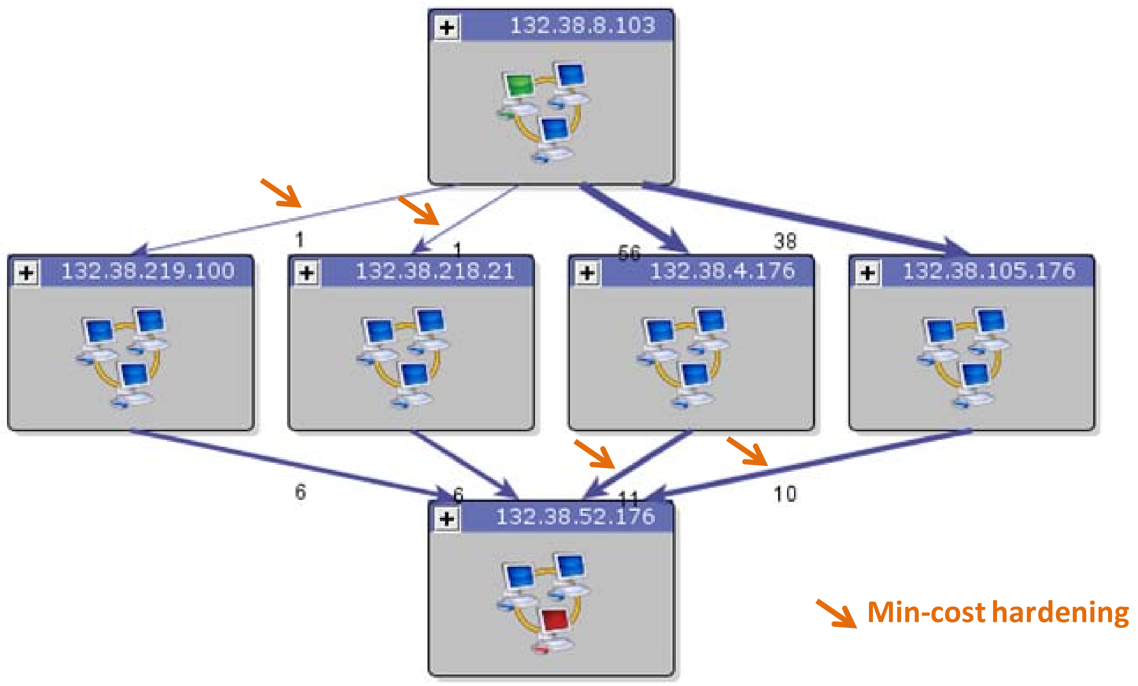


Figure 68. Direct paths to a different attack goal

Figure 68 also shows the minimum-cost hardening recommendation that TVA computes for this attack scenario. In particular, hardening this particular minimum set of vulnerabilities ensures that the attacker must exploit at least 3 vulnerabilities to reach the critical resource.

Figure 69 shows the attack graph for the same network, with the same attack starting point and attack goal as in Figure 68. This time the direct-paths option is turned off so that all possible attacks from start to goal are shown.

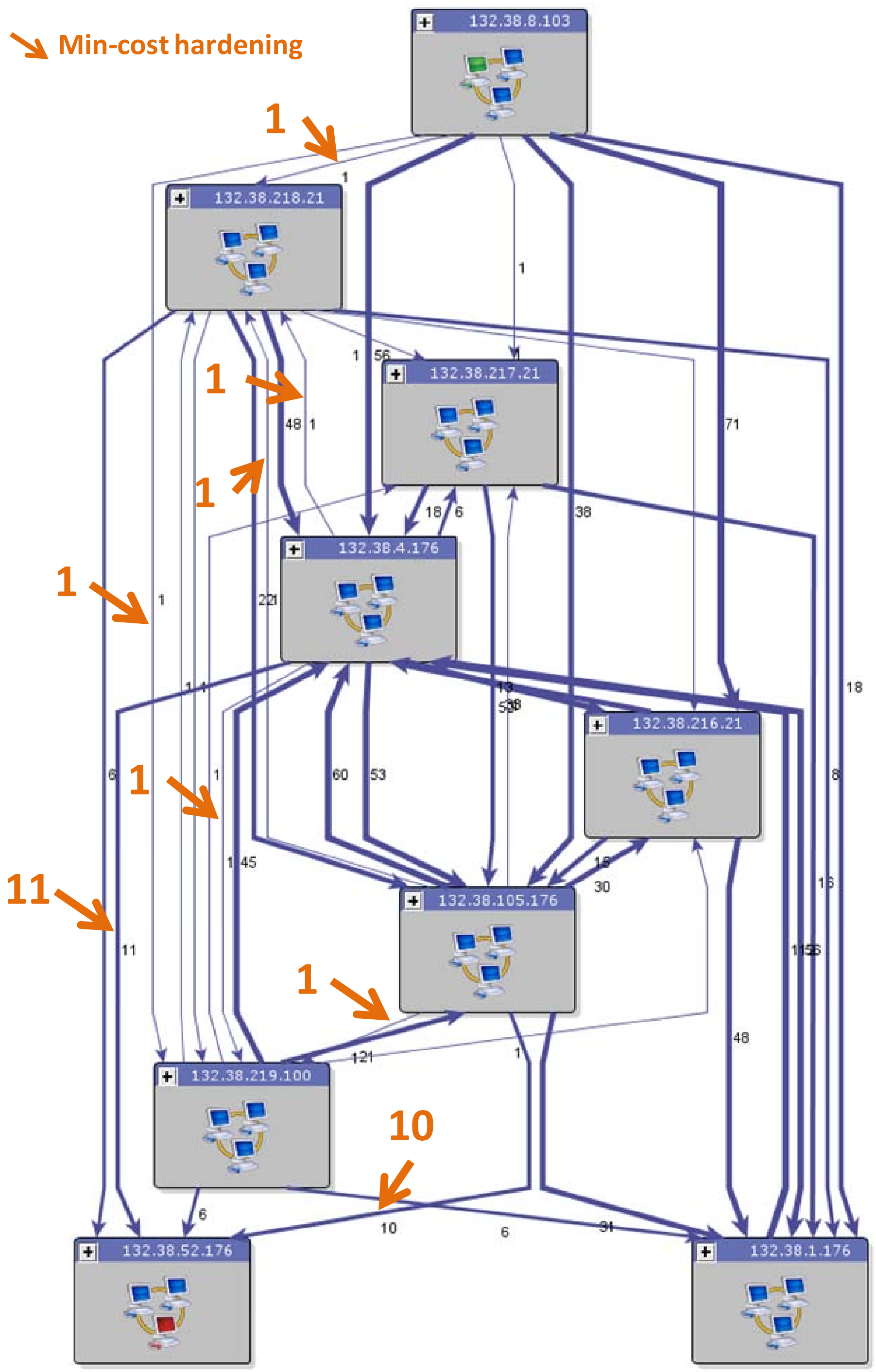


Figure 69. All attack paths, with minimum-cost hardening recommendation

Figure 69 also shows the minimum-cost hardening recommendation for all possible attack paths from start to goal. In this case, hardening this particular set of 27 total vulnerabilities prevents the attacker from reaching the critical resource. Figure 70 shows the resulting attack graph after these 27 vulnerabilities have been hardened.

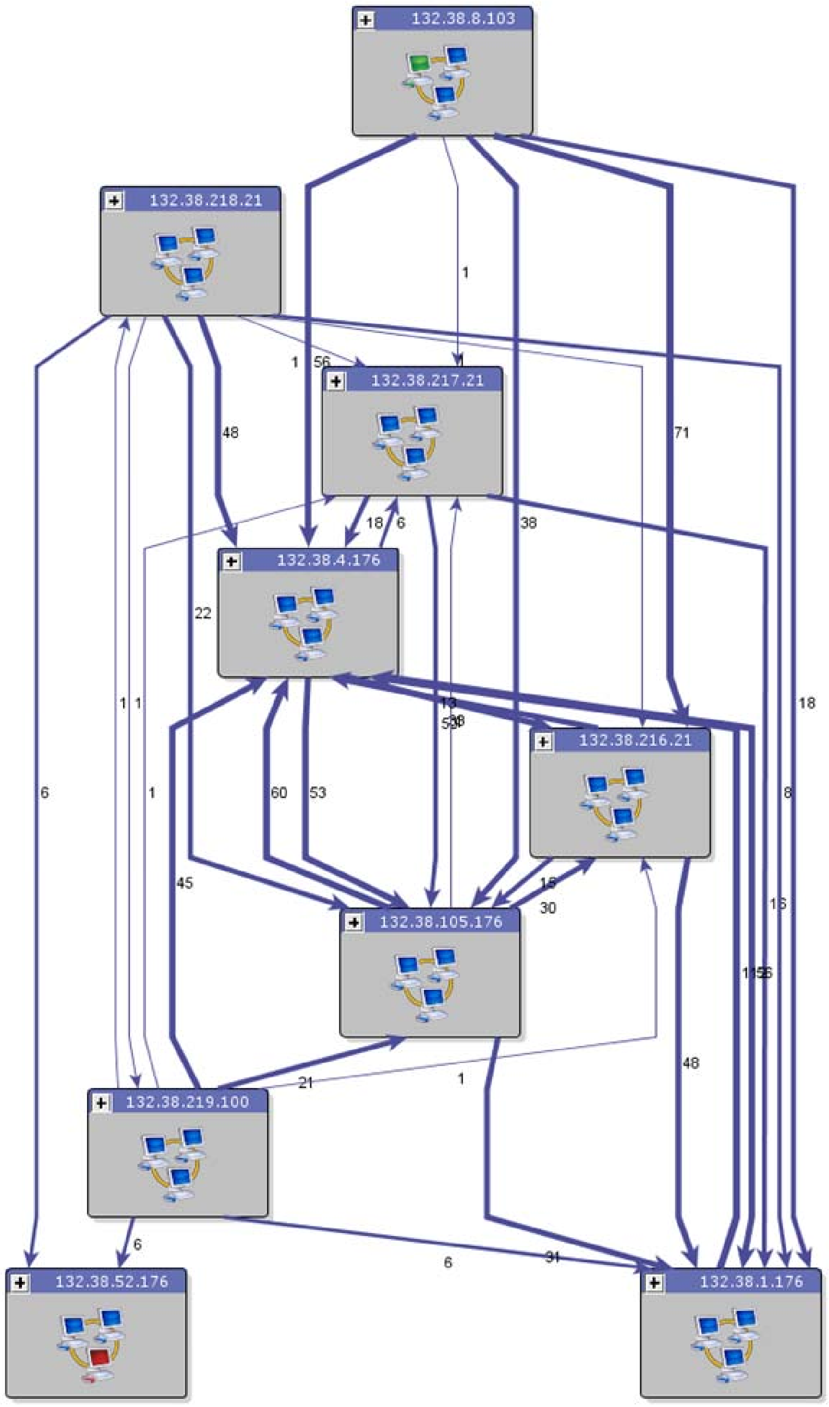


Figure 70. Application of minimum-cost hardening

Figure 71 shows the same attack graph as Figure 70, with the subnets repositioned to show attack relationships more clearly. This more clearly shows that while there are exploitable vulnerabilities from .218 and .219 into the goal .52 subnet, there are no incoming attacks to these from elsewhere in the network.

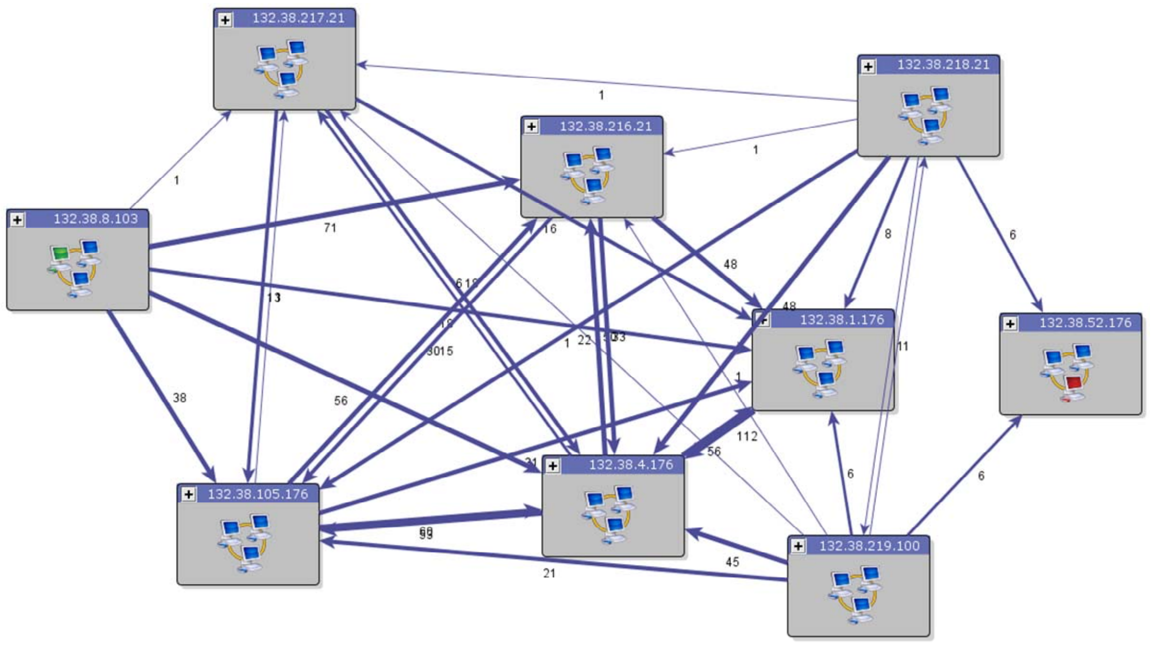


Figure 71. Repositioned attack graph after minimum-cost hardening

4.6 Model Population Extensions

Key feedback from the testing and evaluation of our TVA tool includes the following:

- The TVA tool supported the Nessus vulnerability scanner, but not Retina. The Retina vulnerability scanner is in widespread use in the Department of Defense (DoD).
- Alternatives to multiple vulnerability scans from different network vantage points should be considered.

Figure 72 shows the architecture of our TVA tool, in which importers for various input sources can be integrated. The importers for Retina and Sidewinder are highlighted. This also shows that the TVA tool has a FoundScan importer, although that is outside the scope of this project.

The architecture diagram in Figure 72 shows that the result of the TVA tool importers is a network model for consumption by the attack graph analysis engine. This canonical vendor-neutral network representation decouples the importer code from the analysis engine code.

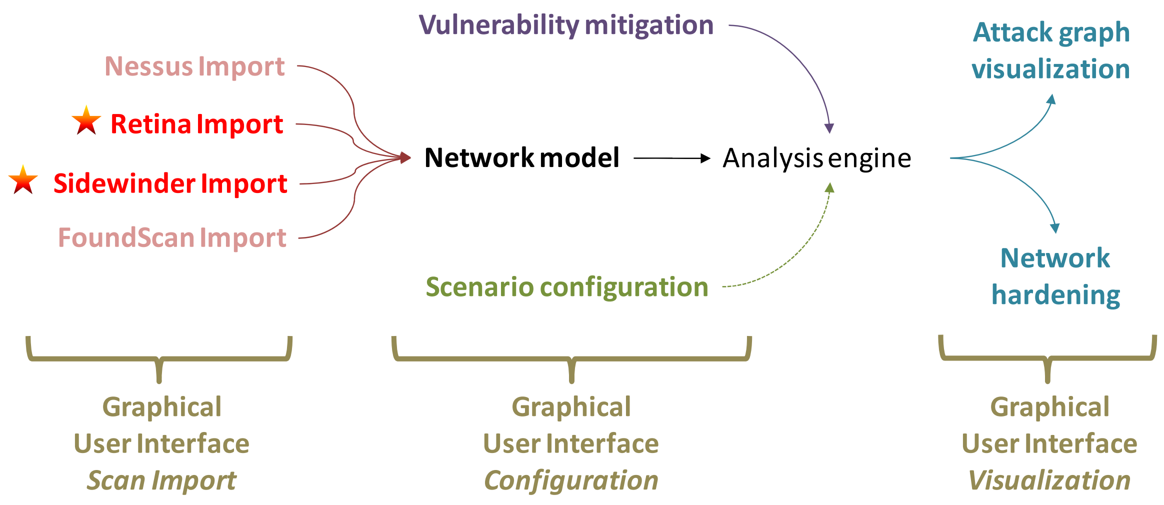


Figure 72. TVA tool architecture

Figure 73 shows the structure of our TVA network model. This shows that a network is compose of a number of protection domains, each containing a number of machines. Each machine has a number of vulnerabilities that are exploitable within the local domain. Each machine also has a number of connections to vulnerabilities on machines in other domains.

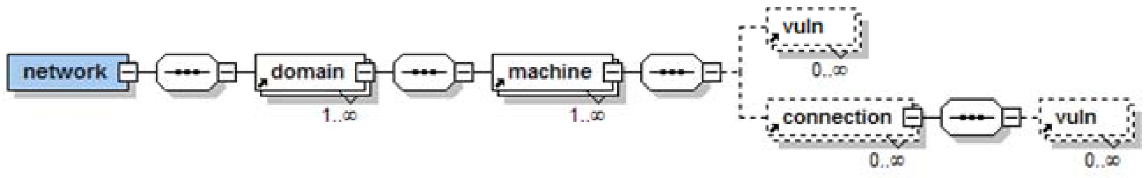


Figure 73. Structure of TVA network model

In the TVA architecture, the importer for Retina has a preprocessor written in eXtensible Stylesheet Language (XSL) [60]. XSL is a powerful pattern-matching language for performing transformations on XML data.

As shown in Figure 74, the Retina preprocessor converts native Retina data to a generic scanner format. This format represents scan data for TVA, independent of a particular scanner vendor's format. The vendor-neutral format is the provided as an input to the scan import module that analyses scan data and builds the network model.

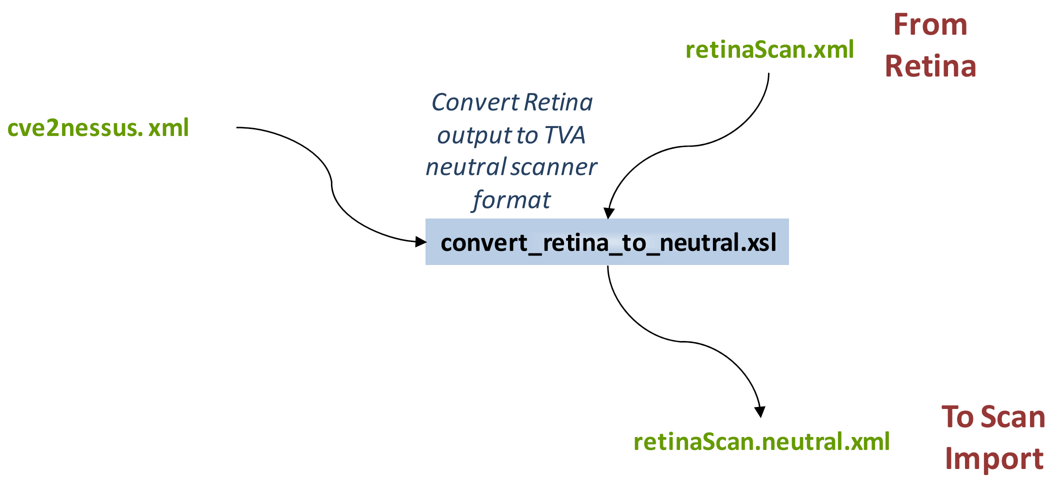

retinaScan.xml
From Retina
cve2nessus. xml
Convert Retina output to TVA neutral scanner format
convert_retina_to_neutral.xsl
retinaScan.neutral.xml
To Scan Import

Figure 74. Preprocessing of Retina scan data

The preprocessing in Figure 74 takes native Retina scan XML documents as input. Figure 75 shows the structure of native Retina scan data.

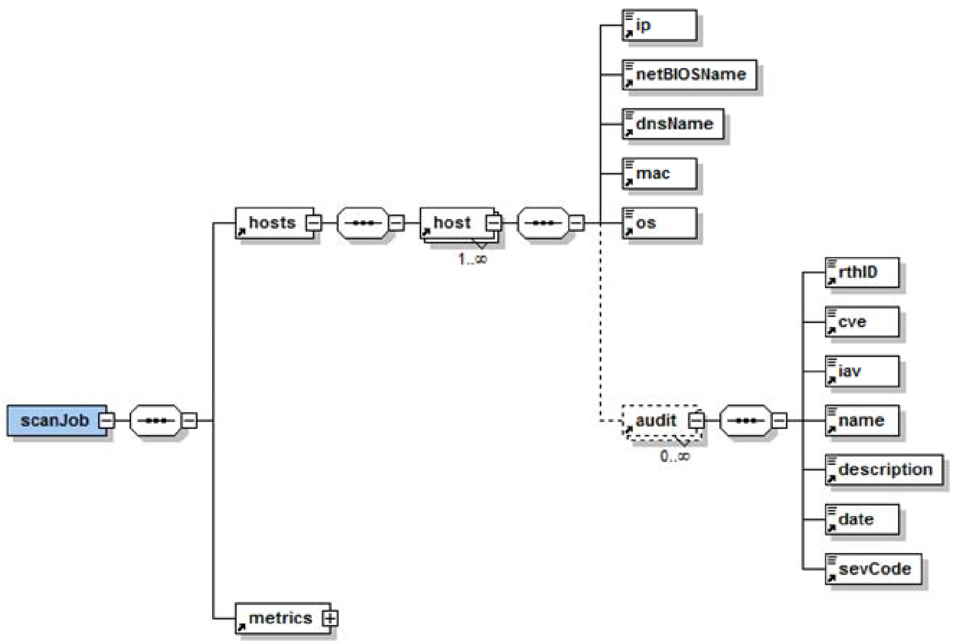


Figure 75. Structure of Retina native scan data

Figure 76 shows the structure of the resulting vendor-neutral TVA scan data format. This includes only the data needed for building the input network model for TVA. In particular, a network scan contains a set of hosts. Each host has a set of ports, where each port represents a vulnerability detected by the scanner (in this case, Retina).

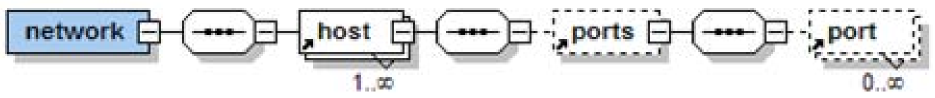


Figure 76. Structure so TVA scan data

The scan import preprocessing in Figure 74 shows an additional input, i.e., a set of mappings from CVE to Nessus identifiers. This allows TVA to share vulnerabilities modeled for Nessus with vulnerabilities detected by Retina. This correlation is based on common CVEs between vendors. Figure 77 shows the structure of this mapping file, which simply maps a CVE to a set of corresponding Nessus identifiers.

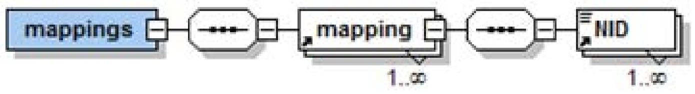


Figure 77. Mapping from CVE to Nessus identifier

As originally designed, to capture the network model for TVA, vulnerability scans are performed from each subnet (or group of subnets comprising a protection domain). Each scan is configured to scan all machines within the subnet and all machines in neighboring subnets. The idea is to capture the visibility of all machines to all other machines across all subnets.

In this design, one scan per subnet, correctly configured, provides the input for TVA. In other words, the scans need to include not only intra-subnet, but also inter-subnet. One could also augment the TVA model with additional scans from a particular subnet, to capture firewall rules that are source-host dependent.

As an example, Figure 78 shows two subnets connected by a firewall. For this network, one would perform these 2 vulnerability scans:

- Scan 10.10.1.1-254 and 10.10.2.1-254 from machine 10.10.1.1
- Scan 10.10.1.1-254 and 10.10.2.1-254 from machine 10.10.2.1

CAULDRON can then merge the resulting scan files, and build a network model that includes connectivity (to vulnerable services) among all machines. Each of the scan results in a single output XML file per scan.

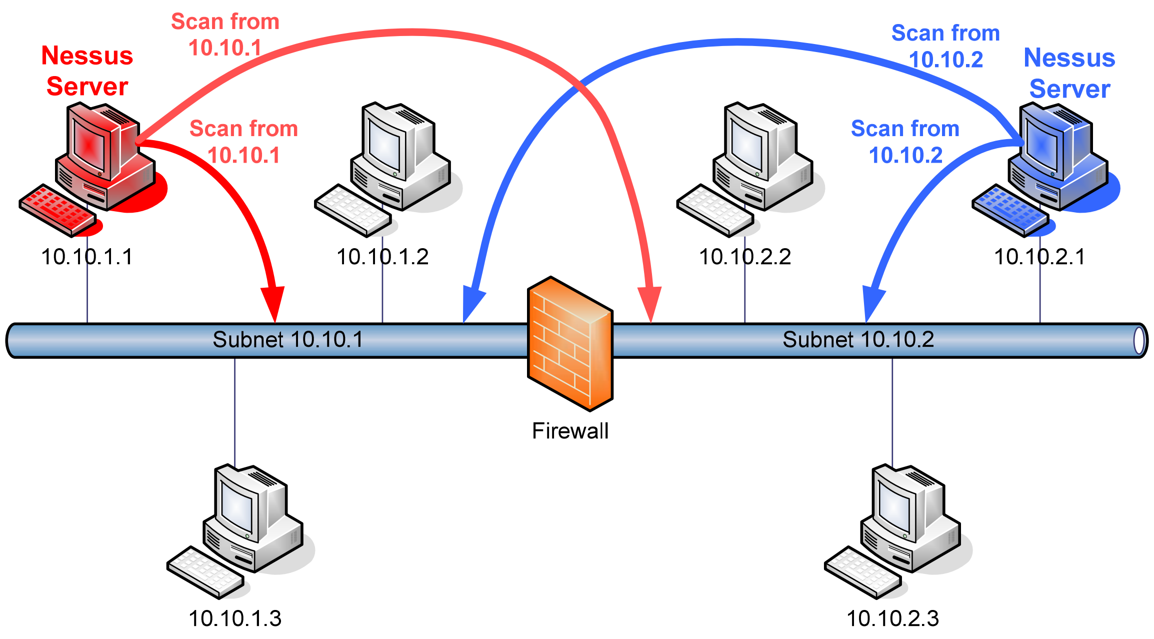


Figure 78. Vulnerability scans for two subnets

Figure 79 shows a similar example, this time with 3 subnets connected by firewalls. For this network, one would perform these vulnerability scans:

- Scan {10.10.1.1-254, 10.10.2.1-254, 10.10.3.1-254} from 10.10.1.1
- Scan {10.10.1.1-254, 10.10.2.1-254, 10.10.3.1-254} from 10.10.2.1
- Scan {10.10.1.1-254, 10.10.2.1-254, and 10.10.3.1-254} from 10.10.3.1

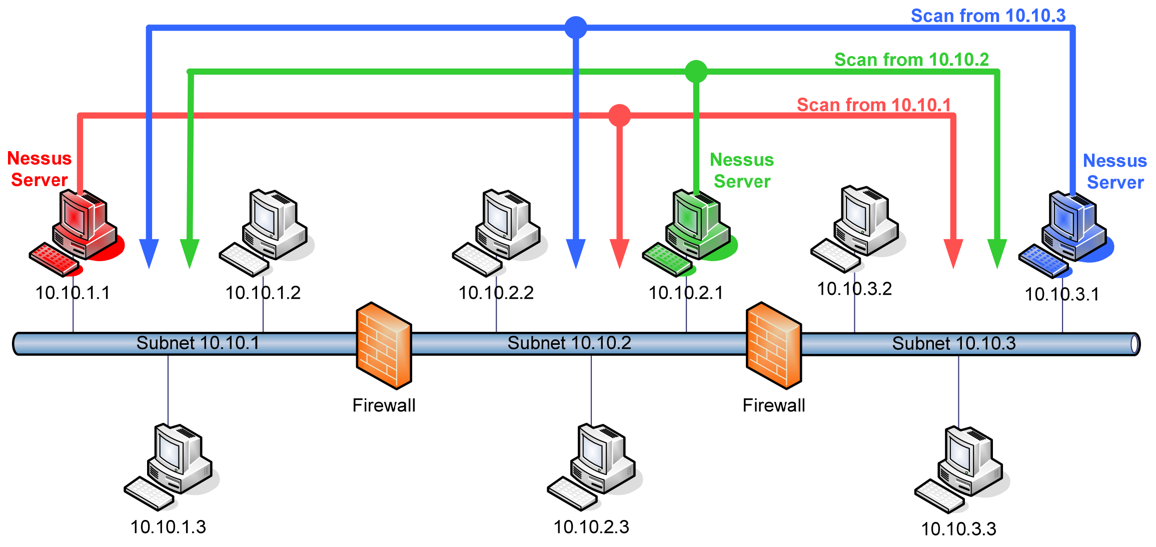


Figure 79. Vulnerability scans for three subnets

Again, the TVA tool will merge the three resulting scan XML files. As before, each scan (with multiple targets) results in a single XML file per scan. Each XML scan file is then preprocessed to convert to a vendor-neutral format, and then merged into a single TVA input network model.

A new capability was developed for augmenting the TVA network model by importing firewall rules directly. Any connections to vulnerable services in the firewall data are added to any vulnerable connections created from vulnerability scans to remote subnets. Our TVA tool currently supports the import of Sidewinder firewall native data.

One could rely entirely on firewall data for capturing vulnerable connectivity across subnets. In this case, scan would be done locally in individual subnets for detecting vulnerable services on hosts in each local subnet. The firewall data then determines connections to vulnerable services *across* subnets. One could also scan locally *and* remotely, and use firewall data to discover any additional vulnerable connectivity, e.g., host-specific firewall rules.

Figure 80 shows the structure of generic (vendor-neutral) firewall data for import to TVA. Firewall data is a set of ALLOW rules, each rule having a source and destination that is allowed. Each source or destination has attributes for IP address, subnet mask, protocol, and port.

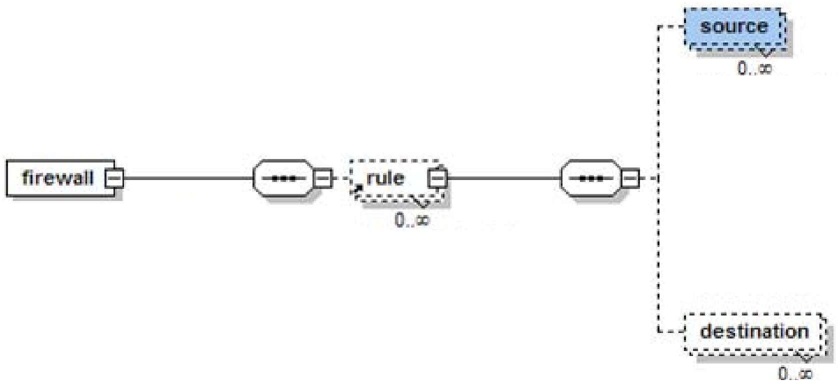


Figure 80. Structure of TVA firewall rule data

4.7 Project Events

The following is a summary of project events:

- Kickoff meeting at Air Force Research Laboratory (AFRL) in Rome, New York (November 17, 2006).
- Presented at US Secret Service Electronic Crimes Task Force Meeting, Miami, Florida (February 13, 2007).
- Demonstrated at Department of Homeland Security (DHS) Identity Theft Technology Council, Menlo Park, California (February 27, 2007).
- Presented at AFRL (Rome) Cyber Defense Conference, May 22, 2007.
- Technical Meeting: With Frances Rose (AFRL), at George Mason University, Fairfax, Virginia, January 8, 2008.
- Technical Meeting: With Thomas Parisi (AFRL), Gilberto Garza (Air Force), et al, at Lackland Air Force Base, San Antonio, Texas, February 12, 2008.
- Invited Talk: DHS System Integrator Forum, Arlington, Virginia, February 21, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, March 18, 2008.
- Invited Talk: Air Force Scientific Advisory Board, March 28, 2008.
- Technical Meeting: With Thomas Parisi (AFRL), Gilberto Garza (Air Force), et al, at AFRL, Rome, New York, June 5, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, April 23, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, May 8, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, June 27, 2008.
- Telephone Conference: With Anita Bhat (AFRL) and Thomas Parisi (AFRL), regarding planned deployment of project software, August 12, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, July 14, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, July 30, 2008.
- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, August 8, 2008.

- Provided Project Software: To Jason Fingerman (AFRL), in support of attack graph testbed evaluation, September 4, 2008.
- Received Test Scan Data: From Jason Fingerman (AFRL), in support of attack graph testbed evaluation, September 5, 2008.
- Technical Meeting: With Anita Bhat (AFRL), regarding planned deployment of project software, Arlington, Virginia, October 17, 2008.
- Provided Technical Briefing Material: To Jason Fingerman (AFRL), November 12, 2008.
- Reviewed Technical Report: From Jason Fingerman (AFRL), attack graph tools comparative analysis, November 19, 2008.
- Technical Meeting: With Thomas Parisi (AFRL), Gilberto Garza (Air Force), and Jason Fingerman (AFRL) at AFRL, regarding evaluation of project software, Rome New York, November 24, 2008.
- Provided Technical Briefing Material: To Jason Fingerman (AFRL), December 9, 2008.
- Reviewed Evaluation Data: From Jason Fingerman (AFRL), detailed testbed data from evaluation of project software, December 8, 2008.
- Technical Support: To Michael Pepin of United States CENTral COMmand (CENTCOM), responded to technical questions about deployment of project software, January 5, 2009.
- Software Validation: From Michael Pepin (CENTCOM), indicated that project software passed Gold Disk validation, January 22, 2009.
- Technical Discussions: With Jason Fingerman (AFRL), regarding alternative firewall device for possible future input to project software, January 23, 2009.
- Analyzed Firewall Data: From Jason Fingerman (AFRL), configuration data for firewall device on attack graph testbed network, January 26, 2009.
- Technical Support: To Anita Bhat (AFRL), responded to technical questions about project software deployment, February 3, 2009.
- Tool Demonstration: By Jason Fingerman (AFRL) to Major General Senty. Resulted in action item to define a strategy for transfer of project software to (Kelly USA) for operational evaluation, with technical integration support from AFRL (Brian Spink and Col Dewey Parker), March 6, 2009.
- Technical Discussions: With Jason Fingerman (AFRL), regarding support for integrating with Retina vulnerability scanner for input to project software, March 9, 2009.

- Technical Discussions: With Jason Fingerman (AFRL), regarding new format for expressing generic firewall rules data for input to project software, March 9, 2009.
- Reviewed Technical Report: From Gilberto Garza (Air Force), review of 2008 attack graph tools evaluation conducted by AFRL Rome," March 25, 2009.
- Technical Discussions: With Jason Fingerman (AFRL), regarding improvements to integration with SideWinder firewall device as input to project software, March 26, 2009.
- Provided Project Software: To Jason Fingerman (AFRL), imports Sidewinder firewall data, April 8, 2009.
- Provided Project Software: To Jason Fingerman (AFRL), supports recent Nessus vulnerability scanner output format, April 23, 2009.
- Network Data Analysis: Analyzed network data from Jason Fingerman (AFRL), recent Nessus scans and Sidewinder rule export for attack graph testbed network, May 4, 2009.
- Provided Project Software: To Jason Fingerman (AFRL), handles resolution of domain references in Sidewinder data, May 6, 2009.

5. CONCLUSIONS

We demonstrate a new approach for visualization, correlation, and prediction of complex multi-step cyber attack graphs through networks. This approach considers software vulnerabilities across all network hosts, network connectivity, firewall effects, and potential attacker exploits. This approach augments traditional graph-centric representations with graph adjacency matrices.

This approach helps make complex attack graphs manageable, while including all known network attack paths. We show application of this approach to network hardening, attack correlation, and attack origin/impact prediction.

Our analysis combines cyber vulnerabilities in ways that real attackers might do, discovering all possible known attack paths through a network. This comprehensive result provides network defense in depth, showing multiple options (and optimal recommendations) for mitigating potential attacks.

From TVA attack graphs, we compute recommendations for optimal network hardening. We also demonstrate sophisticated visualization capabilities for attack graph exploration.

Our TVA tool was subjected to testing and validation by an independent team. Feedback from the evaluation led to two important extensions to our TVA tool: additional scanner support (Retina), and firewall support (Sidewinder). Support for these is important because of their widespread deployment throughout the DoD.

From the attack graphs predicted by TVA, we demonstrate optimal deployment of intrusion detection sensors. This covers all known vulnerability paths using the minimum number of sensors. Our optimal sensor placement is an instance of the NP-hard minimal set cover problem, which we solve through an efficient greedy algorithm.

Once sensors are deployed and alerts are raised, the predictive attack graph allows us to correlate isolated intrusion alerts into multi-step attacks. It also supports prioritization of alerts based on attack graph distance to critical assets. Overall, this helps us formulate the best options for responding to attacks.

By propagating individual vulnerability metrics through our attack graphs, we compute a new metric that measures the cyber security of a network. We use this metric to compare risk mitigation options in terms of maximizing security and minimizing cost. Our flexible new attack graph metric model quantifies overall security of networked systems, e.g., for cost/benefit tradeoffs for analyzing return on security investment.

6. REFERENCES

[1] 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, December 2005.

[2] S. Jajodia, S. Noel, "Topological Vulnerability Analysis," in *Proceedings of the Army Research Office Cyber Situational Awareness Workshop,* S. Jajodia, C. Wang, V. Swarup, P. Liu (eds.), Springer, 2009.

[3] S. Noel, S. Jajodia, "Proactive Intrusion Prevention and Response via Attack Graphs," in *Practical Intrusion Analysis: Prevention and Detection for the Twenty-First Century*, R. Trost (ed.), Addison-Wesley Professional, 2009.

[4] S. Noel, S. Jajodia, "Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs," Critical Issues in Command, Control, Communications, Computers, Intelligence (C4I), Armed Forces Communications and Electronics Association (AFCEA) Solutions Series, Lansdowne, Virginia, May 2009.

[5] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, K. Prole, "Advances in Topological Vulnerability Analysis," in *Proceedings of the Cybersecurity Applications & Technology Conference For Homeland Security*, Washington, DC, March 2009.

[6] S. Noel, S. Jajodia, "Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs," *Journal of Network and Systems Management*, special issue on Security Configuration Management, acceptance ratio 4/27, September 2008.

[7] S. O'Hare, S. Noel, K. Prole, "A Graph-Theoretic Visualization Approach to Network Risk Analysis," in *Proceedings of the Workshop on Visualization for Computer Security*, Cambridge, Massachusetts, September 2008.

[8] S. Jajodia, S. Noel, "Topological Vulnerability Analysis: A Powerful New Approach for Network Attack Prevention, Detection, and Response," in *Algorithms, Architectures, and Information Systems Security*, B. Bhattacharya, S. Sur-Kolay, S. Nandy, and A. Bagchi (eds.), World Scientific Press, 2007.

[9] S. Noel, S. Jajodia, "Attack Graphs for Sensor Placement, Alert Prioritization, and Attack Response," Cyberspace Research Workshop, Air Force Cyberspace Symposium, Shreveport, Louisiana, November 2007.

[10] L. Wang, S. Noel, S. Jajodia, "Minimum-Cost Network Hardening Using Attack Graphs," *Computer Communications*, 29(18), 3812-3824, November 2006.

[11] 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, 2005.

[12] 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, December 2004.

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

[14] 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.

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

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

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

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

[19] G. Di Battista, P. Eades, R. Tamassia, I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.

[20] J. Heer, S. Card, J. Landay, "Prefuse: A Toolkit for Interactive Information Visualization," in *Proceedings of the Conference on Human Factors in Computing Systems*, Portland, Oregon, April 2005.

[21] B. Shneiderman, *Treemaps for Space-Constrained Visualization of Hierarchies*, http://www.cs.umd.edu/hcil/treemap-history/, last updated June 25, 2009.

[22] L. Williams, R. Lippmann, K. Ingols, "An Interactive Attack Graph Cascade and Reachability Display," in *Proceedings of the Workshop on Visualization for Computer Security*, 2008.

[23] D. Chakrabarti, S. Papadimitriou, D. Modha, C. Faloutsos, "Fully Automatic Cross-Associations," in *Proceedings of the 10th International Conference on Knowledge Discovery & Data Mining*, Seattle, Washington, August 2004.

[24] P. Eades, Q.-W. Feng, "Multilevel Visualization of Clustered Graphs," in *Proceedings of the Symposium on Graph Drawing*, September 1996.

[25] K. Lakkaraju, W. Yurcik, A. Lee, "NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness," in *Proceedings of the Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, October 2004.

[26] J. McPherson, K.–L. Ma, P. Krystosek, T. Bartoletti, M. Christensen, "PortVis: A Tool for Port-Based Detection of Security Events," in *Proceedings of the Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, October 2004.

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

[28] 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 International Workshop on Software Engineering for Secure Systems*, Shanghai, China, 2006.

[29] 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.

[30] R. Deraison, Nessus, http://www.nessus.org, last retrieved June 2008.

[46] MITRE, *CVE – Common Vulnerabilities and Exposures*, http://cve.mitre.org/, last retrieved October 2008.

[47] *Bugtraq Vulnerabilities*, http://www.securityfocus.com/vulnerabilities, last retrieved October 2008.

[48] Centennial Software, *Discovery Asset Management*, http://www.centennial-software.com/products/discovery/, last retrieved September 2008.

[49] Symantec Corporation, *Symantec DeepSight Threat Management System*, https://tms.symantec.com/Default.aspx, last retrieved August 2008.

[50] eEye Digital Security, *Retina Network Security Scanner*, http://www.eeye.com/html/Products/Retina/index.html , last retrieved July 2008.

[51] Foundstone, FoundScan, http://www.foundstone.com/us/index.asp, last retrieved September 2008.

[31] D. Zerkle, K. Levitt, "Netkuang – A Multi-Host Configuration Vulnerability Checker," in *Proceedings of the 6th USENIX Unix Security Symposium*, 1996.

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

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

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

[35] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, R. Cunningham, "Validating and Restoring Defense in Depth Using Attack Graphs," in *Proceedings of the Military Communications Conference*, 2006.

[36] W. Li, *An Approach to Graph-Based Modeling of Network Exploitations*, PhD dissertation, Department of Computer Science, Mississippi State University, 2005.

[37] M. Swanson, N. Bartol, J. Sabato, J Hash, L. Graffo, *Security Metrics Guide for Information Technology Systems*, Technical Report 800-55, National Institute of Standards and Technology, 2003.

[38] Forum of Incident Response and Security Teams (FIRST), *Common Vulnerability Scoring System (CVSS)*, http://www.first.org/cvss/, last retrieved June 2008.

[52] Internet Engineering Task Force, *The Intrusion Detection Message Exchange Format (IDMEF)*, http://www.ietf.org/rfc/rfc4765.txt, last retrieved November 2008.

[53] ArcSight, *Enterprise Security Management*, http://www.arcsight.com/, last retrieved October 2008.

[54] SourceForge, *Snort IDMEF Plugin*, http://sourceforge.net/projects/snort-idmef, last retrieved July 2008.

[55] Sourcefire, *Snort – The De Facto Standard for Intrusion Detection/Prevention*, http://www.snort.org/, last retrieved September 2008.

[39] A. Gilat, *MATLAB: An Introduction with Applications*, 3rd Edition, Wiley, 2008.

[40] P. Grünwald, "A Tutorial Introduction to the Minimum Description Length Principle," in *Advances in Minimum Description Length: Theory and Applications*, P. Grünwald, I. Myung, M. Pitt (eds.), MIT Press, 2005.

[41] *Matrix Decomposition*, http://en.wikipedia.org/wiki/Matrix_decomposition, last modified September 13, 2009.

[42] E. Nuutila, *Efficient Transitive Closure Computation in Large Digraphs*, Ph.D. dissertation, Acta Polytechnica Scandinavica, Helsinki, 1995.

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

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

[57] "A Threshold of Ln N for Approximating Set Cover," *Journal of the Association of Computing Machinery*, 45(4), 1998.

[58] S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, 1997.

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

[44] *Security Content Automation Protocol (SCAP)*, http://scap.nist.gov/, last retrieved September 2009.

[45] *National Vulnerability Database (NVD)*, http://nvd.nist.gov/, last retrieved September 2009.

[60] *The Extensible Stylesheet Language Family (XSL)*, http://www.w3.org/Style/XSL/.

7. LIST OF ACRONYMS

Acronym	Meaning
AFCEA	Armed Forces Communications and Electronics Association
AFRL	Air Force Research Laboratory
ARP	Address Resolution Protocol
C4I	Command, Control, Communications, Computers, Intelligence
CENTCOM	CENTral COMmand
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DoD	Department of Defense
DHS	Department of Homeland Security
DMZ	Demilitarized Zone
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IDMEF	Intrusion Detection Message Exchange Format
IIS	Internet Information Server
IP	Internet Protocol
LAN	Local Area Network
NIST	National Institute of Standards and Technology
NT	New Technology
NVD	National Vulnerability Database
RSH	Remote SHell
SCAP	Security Content Automation Protocol
SSH	Secure SHell
TCP	Transmission Control Protocol
TVA	Topological Vulnerability Analysis
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language