Multiple Coordinated Views for Network Attack Graphs

Steven Noel Michael Jacobs Pramod Kalapa Sushil Jajodia

Center for Secure Information Systems, George Mason University

ABSTRACT

While efficient graph-based representations have been developed for modeling combinations of low-level network attacks, relatively little attention has been paid to effective techniques for visualizing such attack graphs. This paper describes a number of new attack graph visualization techniques, each having certain desirable properties and offering different perspectives for solving different kinds of problems. Moreover, the techniques we describe can be applied not only separately, but can also be combined into coordinated attack graph views. We apply improved visual clustering to previously described network protection domains (attack graph cliques), which reduces graph complexity and makes the overall attack flow easier to understand. We also visualize the attack graph adjacency matrix, which shows patterns of network attack while avoiding the clutter usually associated with drawing large graphs. We show how the attack graph adjacency matrix concisely conveys the impact of network configuration changes on attack graphs. We also describe a novel attack graph filtering technique based on the interactive navigation of a hierarchy of attack graph constraints. Overall, our techniques scale quadratically with the number of machines in the attack graph.

CR Categories and Subject Descriptors: K.6.5 [Management of Computing and Information Systems]: Security and Protection – unauthorized access

Keywords: Network attack modeling, network attack graphs, topological vulnerability analysis, graph visualization, graph clustering, graph filtering

1 INTRODUCTION

The utility of organizing multi-step network attacks into graphs is well established. Such attack graphs allow one to see, step by step, the various ways an attacker can incrementally penetrate a network. A network vulnerability considered in isolation may not appear to pose a significant threat. But the interdependency of vulnerabilities and the connectivity of a network make such analysis incomplete. Even well administered networks may be vulnerable to attacks, because of the security ramifications of offering a variety of services.

Attack graphs allow the security analyst to assess the true vulnerability of critical network resources, and to understand how vulnerabilities in individual network services contribute to overall vulnerability. Attack graphs can function in defensive as well as offensive modes, in applications as diverse as network hardening and what-if analysis, fine tuning and alarm correlation for intrusion detection systems, enemy course-of-action prediction, automated attack response, and penetration testing.

Attack graphs have traditionally been created manually, e.g., by security red teams. But more recently, significant progress has been made toward generating attack graphs automatically from models of networks and attacker exploits, notably [1–12]. Given the creation of such models from network discovery tools, vulnerability databases, etc., the resulting attack graphs are potentially huge. Despite not generally being reported in the literature, there are many practitioners today dealing with very large attack graphs.

While it is currently possible to generate very large and complex attack graphs, relatively little work has been done for managing their complexity. Information visualization techniques have begun to be applied to information security, particularly for visualizing network traffic [13–16] and intrusion detection events [17–18]. These techniques are useful for visual clustering, pattern recognition, anomaly detection, etc. for situational awareness. But they generally treat attack events independently of one another, as opposed to forming sequences of events. As such, they do not address the particular needs of attack graph complexity management.

In this paper, we describe a number of new techniques for managing attack graph complexity through visualization. We introduce a form of visual clustering with reduced complexity and improved cognitive flow for a previously described attack graph representation based on the protection domain abstraction [19]. This approach is particularly advantageous as the number of exploits across protection domains increases.

We also show how direct visualization of the attack graph adjacency matrix avoids the clutter often associated with the drawing of large graphs. Here we rely on known attributes of network machines to help cluster graph edges in the adjacency matrix visualization. For example, when adjacency matrix rows and columns are sorted by internet protocol (IP) address, protection domains (in this case, subnets) are evident from the full intra-domain connectivity and relatively sparse inter-domain connectivity. We show how the adjacency matrix lends itself well to visualizing differences between attack graphs, e.g., as a result of network configuration changes.

We introduce a novel interactive attack graph filtering scheme using an attack graph constraint hierarchy (tree). This allows the user to interactively navigate the tree hierarchy to specify filtering constraints that control which subset of the attack graph is currently visible. In our approach, there are navigational links within the constraint tree itself, as well as bidirectional links between the constraint tree and the filtered attack graph. We also show how intrusion alarms associated with vulnerability-based attack graph exploits can be linked to the constraint tree and highlighted on the attack graph.

The techniques we describe can be applied not only individually, but also in combination, to provide coordinated multiple views of an attack graph. Each view has certain strengths, and when combined, this provides great flexibility for

Center for Secure Information Systems, George Mason University, 4400 University Drive, Mail Stop 5B5, Fairfax, VA 22030-4444. E-mail: [snoel, mjacobs1, pkalapa, jajodia]@gmu.edu.

Workshop on Visualization for Computer Security
October 26, Minneapolis, MN, USA
0-7803-9477-1/05/$20.00 ©2005 IEEE.

user interaction. These techniques apply to general attack graphs, such as ones generated from network vulnerability data, intrusion detection alarms, etc. They also apply regardless of whether the attack graph is constrained to particular starting points(s) and/or attack goal(s). Overall, the worst-case complexity of these techniques is quadratic in the number of network machines.

In the next section, we review related work in this area. In Section 3, we show how visual clustering can reduce attack graph complexity using the protection domain abstraction. Section 4 shows how attack graph adjacency matrix visualization, when combined with ordering by IP address, provides a view of attack connectivity that avoids the clutter of large drawn graphs. This section also shows the value of the adjacency matrix visualization for representing attack graph changes. Section 5 introduces a novel approach for interactive visual attack graph filtering, including navigational links and highlighting for intrusion alarms. In Section 6, we summarize our work and present our conclusions.

2 Previous Work

Various approaches have been described for generating attack graphs. These generally start with models of network security conditions and sets of rules (attacker exploits) that induce new conditions based on existing ones. Valid sequences of exploits are then generated, and organized as a graph. The various methods fall under the major categories of logic-based approaches, e.g., symbolic model checking [1–3], and graph-theoretic approaches [4–12].

Early approaches to generating attack graphs generally had serious scalability problems, because of the state explosion problem. More recent approaches represent dependencies among state transitions (exploits) [8–12] rather than explicitly enumerating states. This reduces complexity from exponential to low-order polynomial. Despite the complexity reduction of the exploit-dependency representation, such attack graphs can still be difficult to manage for larger, less secure networks. The particular problems of attack graph complexity management (from a usability standpoint) have been largely unexplored.

An approach has been proposed [19] that reduces attack graph complexity within so-called protection domains (cliques of an attack graph) from quadratic to linear. However, in that approach, the visual clustering of machines into protection domains is done via graph edges, which introduces unnecessary complexity and makes the attack flow more difficult to follow. We show how the same information can be better conveyed through visual clustering as opposed to additional graph edges. We also reduce complexity by removing redundant exploits, i.e., when a set of exploits from an inter-domain machine is the same as from the intra-domain machines. Like [19], our approach employs clustered graphs, first introduced in [20], with more recent work being represented by [21].

While our new visual representation of protection domains reduces complexity (in comparison to [19]), for larger and more complex graphs, edge clutter can still dominate. We therefore apply adjacency matrix visualizations to attack graphs. We also visualize these in a novel way by highlighting attack graph changes caused by changes to the network configuration.

Adjacency matrix visualizations have been applied in other domains, such as browsing web hyperlinks [22], viewing networked information spaces [23], assessing project requirements [24], and managing large software projects [25]. A number of advantages of adjacency matrix visualizations pointed out in previous work apply to attack graphs as well. Adjacency matrices address the edge-clutter problem inherent in traditional node-link graph visualizations, especially for larger, less secure networks. They also place a strong emphasis on graph edges, in this case, attacker exploits. This allows one to access at a glance where network attacks are possible. With adjacency matrices, groups of edges can be manipulated as a single unit, which is particularly convenient for our matrix row/column ordering based on IP addresses. They also avoid the typically expensive computations associated with graph layout algorithms, which are better suited to less dense graphs.

We also go beyond [19] by applying attack graph filtering as a complementary approach to complexity management. Rather than employing a general-purpose graph filter (e.g., [26]), we define filtering constraints based on the network attack problem. We arrange these constraints in novel forward and backward pointing trees, which allow interactive constraint specification. In this way, one can navigate constraints (which uniquely identify attack graph elements), to interactively select sub-graphs of interest.

3 Visually Clustered Protection Domains

We now show how visual clustering can reduce complexity and improve cognitive flow for attack graphs using the protection domain abstraction. Typically, the greatest cause of high attack graph complexity is unrestricted connectivity (e.g., no firewall filtering) among a set of machines. When such so-called protection domains (attack graph cliques) are known, complexity can be greatly reduced.

In the standard exploit dependency representation, attack graph complexity is $O(scn^2)$, for n machines in the attack graph. Here, s is the average number of exploits against a machine, independent of any particular attacking machine, which might be in the range of say one to 30. The factor c is the average number of security conditions per machine, which in practice is perhaps one to five.

Employing the protection domain abstraction, complexity is reduced to $O(scm^2)$, where m the number of machines that have exploits launched *across* protection domains only. That is, exploits among within-subnet machines are implicitly rather than explicitly represented. Worst-case complexity is thus determined by the number of machines m involved in across-subnet exploits only. Clearly $m < n$ in all cases, but for networks in which there is significant connectivity limitation among protection domains (e.g., via firewalls), it may be that $m << n$, resulting in dramatic improvements in complexity. For example, for a 200-machine protection domain, with a single exploitable vulnerability on each machine, complexity is reduced from 200^2 = 40,000 to 200, a reduction of 2 orders of magnitude.

Figure 1 shows an example low-level attack graph, in which no aggregation of exploits and security conditions has been applied. This attack graph was generated from a network model created from Nessus [27] vulnerability scans, which includes the effects of firewall filtering. In particular, 16 network machines are distributed among four subnets, with firewall filtering among subnets, and no filtering within subnets. The subnets thus each define protection domains in our model.

Figure 2 shows the same attack graph as Figure 1, this time applying the protection domain abstraction. Security conditions for each machine are aggregated to a single vertex, as are sets of exploits between each pair of machines. Each machine vertex is labeled with the number of intra-domain exploits against it, and within a protection domain all machines can execute those exploits against each another. Starting and goal machines are colored green and red, respectively.

In Figure 2, domain membership is indicated by graph edges from machine to domain, as in [19]. In contrast, consider Figure 3, in which protection domain membership is indicated by visual clusters rather than graph edges. Graph complexity is reduced by avoiding domain member ship edges, i.e., for n machines there are n fewer edges. With the visual clusters, it is much easier to see domain membership by not having to follow graph edges to domain nodes.

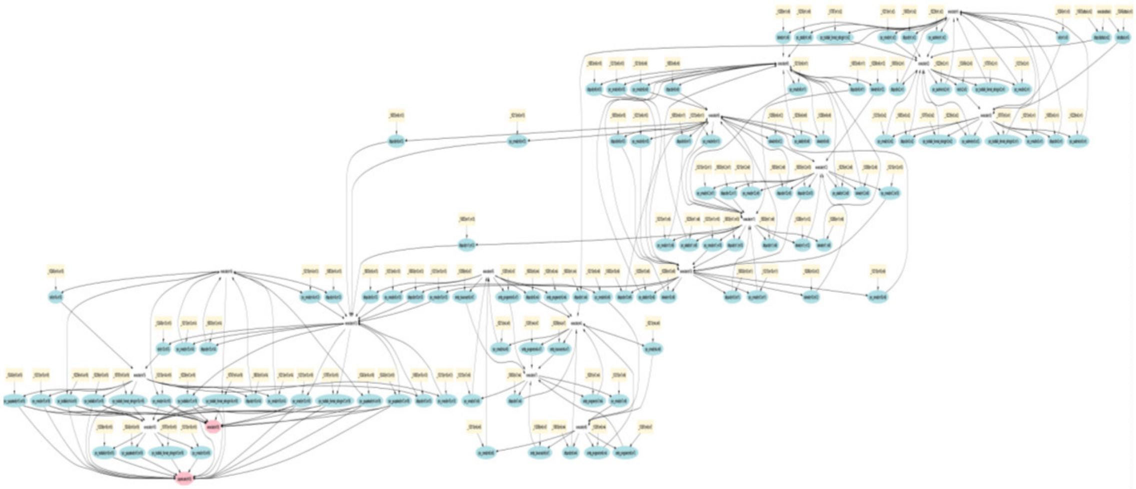


Figure 1: Low-level attack graph.

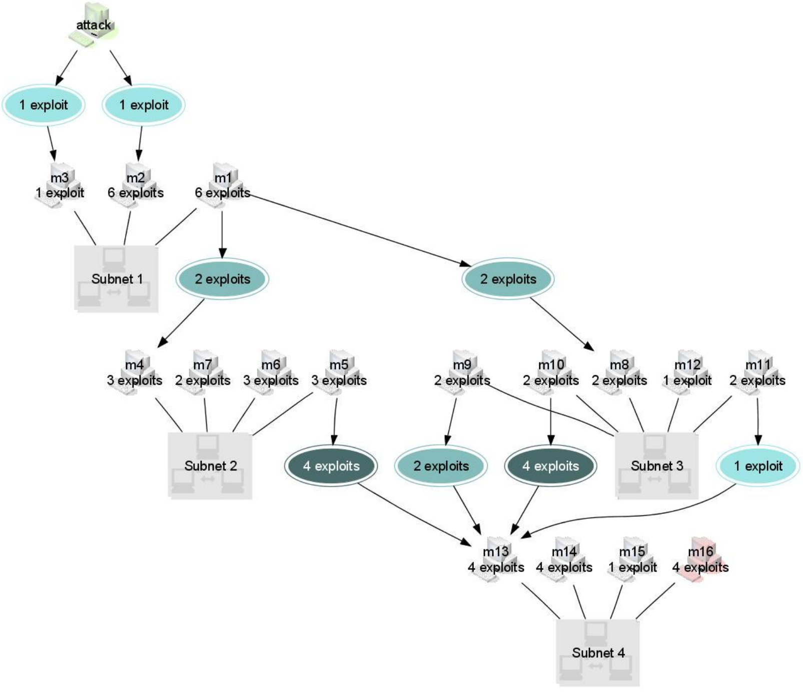


Figure 2: Protection domain membership via graph edges.

Figure 3 includes another improvement over the visual representation in [19], in the form of a redundancy reduction. In particular, for instances in which the exploit set across a protection domain matches the intra-domain exploits for the victim machine, the exploit set is omitted, since it can be inferred from the machine label.

The advantage of protection domain membership via visual clustering is even more obvious as the number of exploits across protection domains increases. In Figure 4, we change the firewall filtering in the network model so that there are significantly more exploits across protection domains. In comparison to Figure 2, the complexity in Figure 4 has increased considerably.

Figure 5 indicates protection domain membership with visual clusters as opposed to graph edges, for the same network as in Figure 4. While complexity has increased considerably for domain membership via graph edges, it has increased comparably much less for membership via visual clusters (compare Figure 3 and Figure 5). Figure 6 shows protection domain visual clustering applied to an even larger network, i.e., 75 machines in 5 subnets, with a significant number of across-subnet exploits. For this attack graph, the starting and goal machines are unspecified, and machines without inter-domain exploits have been aggregated.

4 Attack Graph Adjacency Matrix Visualization

Despite the complexity reduction provided by protection domain visual clusters, there are fundamental scalability problems with drawing graphs directly (so-called node-link visualizations). As larger graphs are drawn, edge clutter begins to dominate, and it becomes increasingly difficult to trace edges and distinguish them from one another.

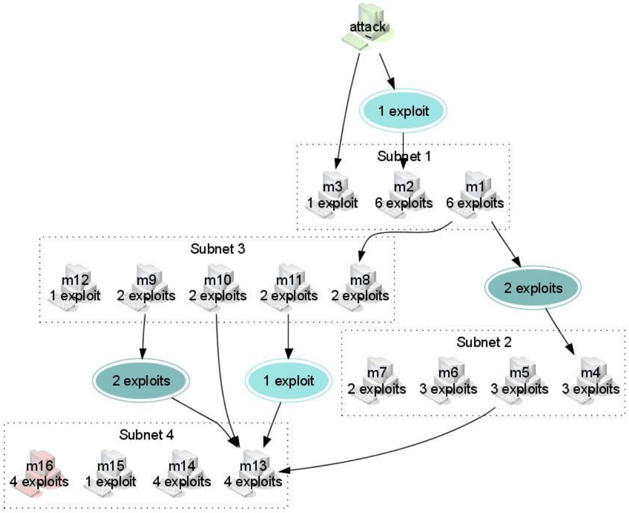


Figure 3: Protection domain membership via visual clustering (compare to Figure 2).

An alternative is to visualize the attack graph adjacency matrix. There are several factors that can make attack graphs larger and denser, so that adjacency matrices become more attractive. An obvious factor is the size of the network under analysis. Our society has become increasingly dependent on networked computers, and the trend towards larger networks will continue. For example, there are enterprises today consisting of tens of thousands of machines. Also, less secure networks clearly have larger attack graphs. There is a tendency for network administrators to protect their domain borders, but to leave machines in their domain relatively unsecured against each other. This may result in attack subgraphs that are very densely or even fully connected. Machines might each have several exploitable vulnerabilities (we have seen cases of as many as 20 per machine), and exploits are generally linked quadratically. When considered across an enterprise, especially given global internet connectivity, attack graphs are potentially huge.

Adjacency matrix visualization could be done for any level of attack graph aggregation, by assigning individual matrix rows/columns for any such aggregate nodes. In general, adjacency matrix rows and columns can be put in arbitrary orders. But orderings that tend to cluster graph nodes by common edges are clearly desirable. We could then treat such clusters of common connectivity as a single unit.

Figure 7 is the adjacency matrix **A** for the 75-machine attack graph in Figure 6. Here, the matrix rows and columns are attacker and victim machines, respectively. An element $a_{i,j}$ of **A** is true (black) when machine *i* launches at least one exploit against machine *j* in the attack graph, and is otherwise white. We sort the rows and columns of **A** according to machine IP address.

From IP address ordering, machines in the same subnet appear in consecutive rows and columns of the adjacency matrix. The unrestricted connectivity within subnets (protection domains) thus causes the fully-connected blocks of machines on the main diagonal.

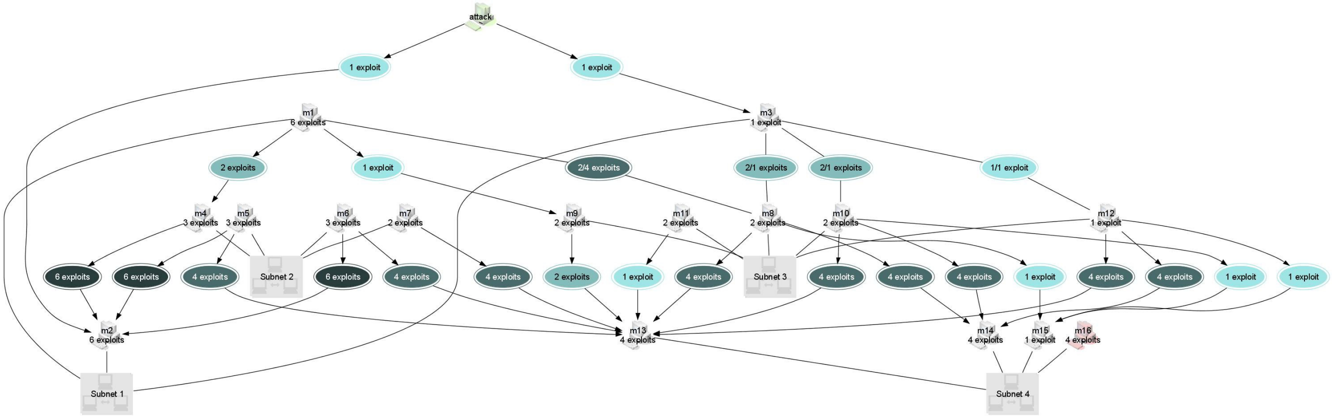


Figure 4: Protection domain membership via graph edges (additional inter-domain exploits).

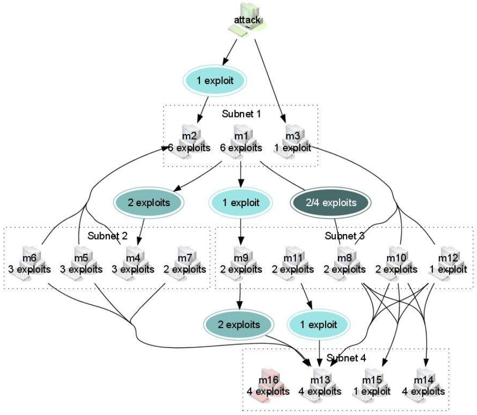


Figure 5: Additional inter-domain exploits (compare to Figure 4).

In Figure 7, block $\mathbf{A}_{i,j}$ of the adjacency matrix indicates exploits launched from machines in Subnet *i* to machines in Subnet *j*. There are a number of apparent features in the adjacency matrix. For example, there are solid blocks of machines (3 in block $\mathbf{A}_{1,2}$, and one each in blocks $\mathbf{A}_{2,3}$ and $\mathbf{A}_{3,2}$) with consecutive ranges of attacking/attacked IP addresses. In blocks $\mathbf{A}_{1,4}$, $\mathbf{A}_{2,5}$, and $\mathbf{A}_{4,5}$, IP addresses are scattered rather than consecutive. In block $\mathbf{A}_{4,1}$, machine x.x.4.12 attacks machine x.x.1.6.

Figure 7 also shows how multi-step attacks can be traced on the adjacency matrix. As an example, it shows two possible multi-step attacks against Subnet 5 (block $\mathbf{A}_{5,5}$), starting from Subnet 1 ($\mathbf{A}_{1,1}$). One attack follows the sequence $\mathbf{A}_{1,1}$, $\mathbf{A}_{1,4}$, $\mathbf{A}_{4,4}$, $\mathbf{A}_{4,5}$, $\mathbf{A}_{5,5}$. The other attack follows the sequence $\mathbf{A}_{1,1}$, $\mathbf{A}_{1,2}$, $\mathbf{A}_{2,2}$, $\mathbf{A}_{2,5}$, $\mathbf{A}_{5,5}$.

The adjacency matrix visualization is particularly well suited to showing differences between attack graphs. This is illustrated in Figure 8 and Figure 9. Figure 8 shows a baseline attack graph, before any changes have been made to the network configuration. Other than the main-diagonal subnets, there are 2 other major features. The vertical lines extend over all matrix rows, representing vulnerable web servers scattered through the network and accessible from all other machines. Also, there are exploitable vulnerabilities from Subnet 2 to Subnet 3.

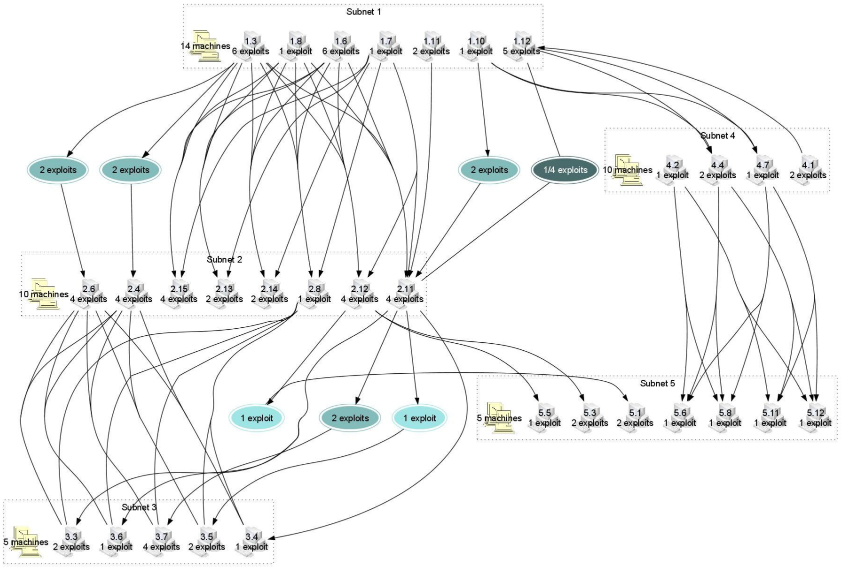


Figure 6: Protection domain membership via visual clusters (75-machine network).

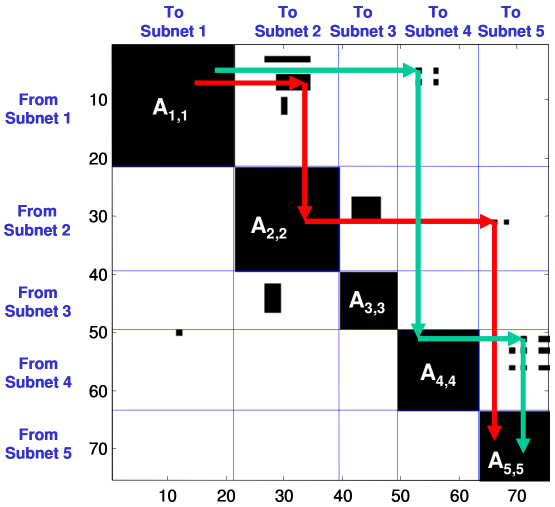


Figure 7: Adjacency matrix visualization (75-machine network).

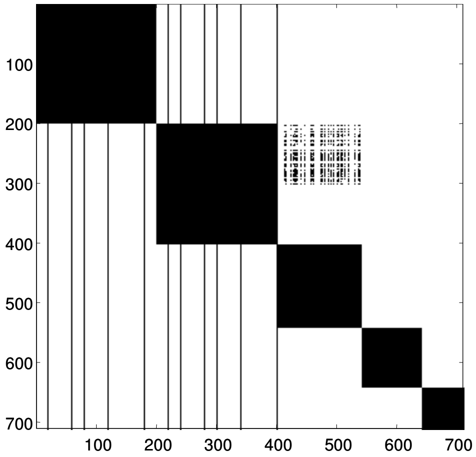


Figure 8: Baseline attack graph.

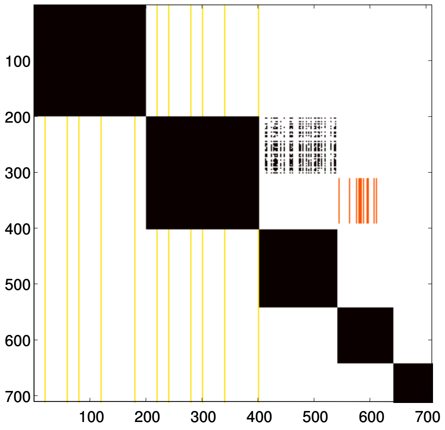


Figure 9: Attack graph differences.

Figure 9 is a difference matrix that represents a set of attack graph changes from the baseline. Here, the vulnerable web servers have been patched, thus removing the exploits against them (indicated in yellow). Also, new vulnerabilities have been introduced from Subnet 2 to Subnet 4, with the corresponding new exploits marked in orange.

5 Interactive Attack Graph Filtering

A complementary approach to attack graph aggregation is to allow the user to interactively filter the graph according to selected criteria. We introduce a novel hierarchical organization of attack graph filtering constraints (the constraint tree), which the user can navigate via a tree widget.

This approach fulfills a user requirement of showing only an attack subgraph of interest. For example, one may be comparing the attack graph to actual exploits carried out on the network, step by step, in support of penetration testing. Or, one may be concerned only with the portion of the attack graph in the neighborhood of detected intrusions. In this sense, our interactive filtering provides focus, while the other visualizations (clustered graphs and adjacency matrices) provide context.

In our approach, the smallest unit of attack graph constraint is a pair of exploits linked by a common pre/post condition. Each exploit in such a pair is parameterized by attacker and victim machines. Since the exploits are linked by a common condition (on a particular machine), the victim machine in the "from" exploit is the attacker machine in the "to" exploit.

Figure 10 demonstrates our attack graph constraint approach. The constraint tree is in the upper left panel. There are 6 levels in the tree (from left to right):

1. Attacker machine in the "from" exploit
2. Victim machine in the "from" exploit
3. Specific "from" exploit between attacker and victim
4. Particular postcondition of "from" exploit
5. Victim machine in the "to" exploit
6. Specific "to" exploit to victim

To illustrate this, in Figure 10 the link from exploit `telnet(m12,m8)` to exploit `rpc_cmsd(m8,m10)` is circled, and the corresponding nodes in the constraint tree are identified. Note here that the attacker machine in the "from" exploit (Level 1 above) is clipped from view in the figure.

Traversing the tree from the root to a leaf uniquely identifies a (forward-pointing) condition link between a pair of exploits. Non-leaf nodes represent sets of links that meet the criteria up to that level. In other words, a node's children represent steps in the attack graph that occur *after* it. Similarly, one can define a backward oriented tree that represents links *into* rather than *away from* a selected node.

In our approach, the constraint tree controls which portion of the full attack is visible. That is, (right) clicking on a tree node commits a constraint specification according to that node. Higher (leftmost) nodes in the tree correspond to less specified constraints, resulting in larger subsets displayed.

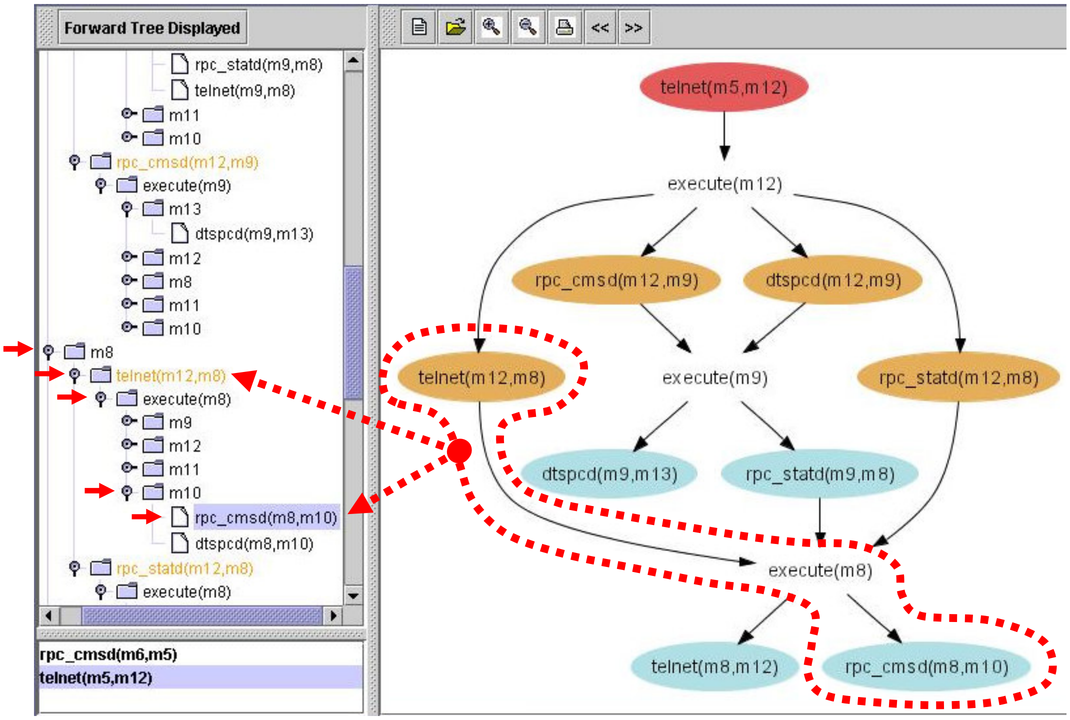


Figure 10: Attack graph constraints.

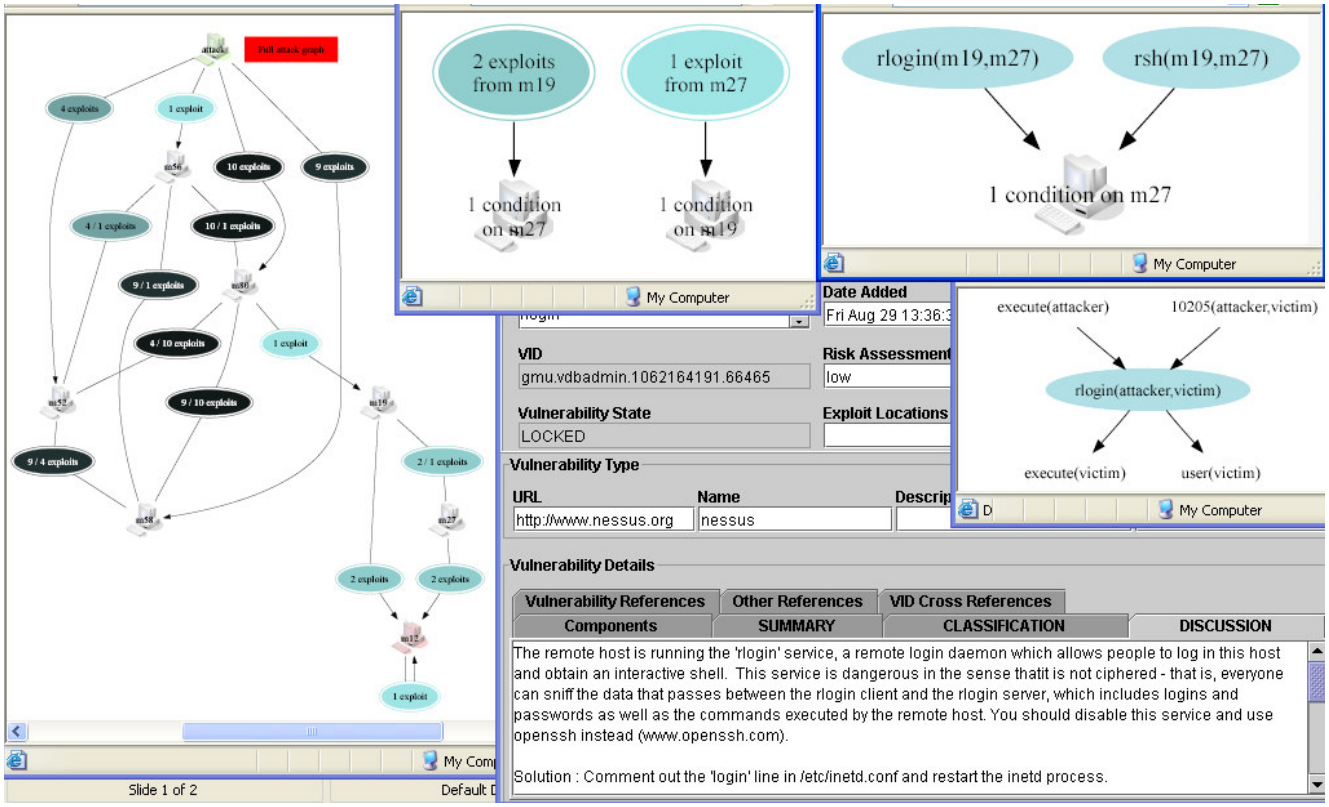
Figure 10 also includes a panel (lower right) for intrusion detection events, which have navigational links to constraint tree nodes. Left-clicking an intrusion detection event leads to the corresponding "from" exploit in the constraint tree. From there, for example, one could immediately right click to view all possible exploits (against all possible machines) that directly follow the intrusion event. Or one could navigate lower into the tree, at each level specifying more specific constraints. Once the desired level is reached, right-clicking commits the constraint specification.

In Figure 10, the filtered attack graph is shown in the right panel. The full-detail view is shown here, although in general this panel starts with a higher level overview (such as protection domains) with detail drilldown. In the figure, intrusion detection events are highlighted red, and one-step attack responses are highlighted in orange (remaining exploits are blue).

Additional navigational controls help the user efficiently explore the attack graph. When a leaf node is reached (in which a "to" exploit has been specified), ctrl-g brings focus to that part of the tree in which that "to" exploit is now a "from" exploit. In this way, the user can continue to grow the attack graph in either a forward or backward direction, depending on whether the forward or backward tree is being used. There are also navigational links from the (filtered) attack graph itself back to the constraint tree, i.e., to the point where the selected exploit is a "from" exploit.

We note that there is a very close correspondence between our attack graph constraint tree and the adjacency matrix visualization described earlier. In particular, sequences of selected points in the adjacency matrix correspond to sequences of corresponding leaf nodes in the constraint tree. In fact, these 2 interactive views can be linked and used interchangeably. This brings all of the views we have described into a coordinated set, each giving the user different perspectives on the attack graph.

Figure 11 shows additional drilldown capabilities for our approach. Beginning on the left, and continuing clockwise, one starts at a higher-level aggregate view, e.g., machines and exploit sets between them. Clicking on an exploit set moves to individual exploit sets for each direction of attack between the 2 machines.



Date Added
Fri Aug 29 13:36:3

VID
gmu.vdbadmin.1062164191.66465

Risk Assessment
low

Vulnerability State
LOCKED

Exploit Locations

Vulnerability Type

URL	Name	Descrip
http://www.nessus.org	nessus	

Vulnerability Details

The remote host is running the 'rlogin' service, a remote login daemon which allows people to log in this host and obtain an interactive shell. This service is dangerous in the sense thatit is not ciphered - that is, everyone can sniff the data that passes between the rlogin client and the rlogin server, which includes logins and passwords as well as the commands executed by the remote host. You should disable this service and use openssh instead (www.openssh.com).

Solution : Comment out the 'login' line in /etc/inetd.conf and restart the inetd process.

Figure 11: Attack graph drilldown.

Selecting one of the exploit sets in Figure 11 in shows all exploits in the set, with all relevant conditions included in the attack graph. Clicking an exploit shows its full set of preconditions and postconditions. From there, one can query a vulnerability database for details about specific conditions.

6 Implementation Details

There are a number of key challenges in implementing the techniques we describe here. These include (1) collecting and managing data about networks and their vulnerabilities, (2) building network attack models in terms of security conditions and attacker exploits, (3) analyzing the models through simulated attacks to produce attack graphs, (4) aggregating and filtering the attack graphs, (5) drawing the graphs, and (7) providing interactive controls for attack graph navigation.

Our implementation employs a layered architecture. Data collection is handled in two (parallel) layers. Data about the network is collected via vulnerability scanning tool(s) such as Nessus [27]. Data about reported vulnerabilities in various hardware and software components (e.g., Bugtraq [28] and many others) is collected and maintained in a relational database. From the vulnerability scanner data, network models are created automatically. From the data on reported vulnerabilities, exploits are modeled. Exploit modeling is generally done manually, for assured accuracy, although when included as part of the standard process for accessing reported vulnerabilities this usually takes little extra effort.

Creation of attack graphs (from the network attack model) is carried out by a custom intelligent analysis engine. This analysis engine simulates incremental network penetration through the application of modeled exploits against the network model. Throughout analysis, the engine maintains the network security state as it evolves under incremental attacker penetration, and forms the attack graph by matching exploit preconditions and postconditions.

In the aggregation/filtering layer, elements of the attack graph are clustered according to machine, exploits between pairs of machines, etc. Each cluster is in turn mapped to the original subgraph it represents. To handle attack graph constraints, graph edges are stored in hierarchical data structures that mirror the forward and backward pointing constraint trees. Graph drawing is performed via AT&T Graphviz [29]. The resulting graph images are rendered, and embedded with interactive links from graph clusters to their corresponding subgraphs. These results are all web content, which can be shared with others who do not have the tool themselves

Our attack graph generation and visualization technology has been transitioned to a commercial product – Secure Element's *C5 Attack Predictor (C5 AP)* [30]. C5 AP is designed for security experts (e.g., penetration testers) who want to use predictive modeling for non-invasive vulnerability analysis on client networks, and includes daily updates on new vulnerabilities and exploits.

7 Summary and Conclusions

In this paper, we have described a complementary collection of coordinated views for network attack graphs. Each view provides different perspectives, and the views can be linked to increase their effectiveness. While most previous work has focused on attack graph generation, we develop ways to visually analyze the potentially very large graphs that may result.

We reduced the visual complexity for a previously described attack graph representation based on the protection domain (attack graph clique) abstraction. We showed how this visual representation is particularly advantageous as the number of inter-domain exploits increases.

We also described visualization of the attack graph adjacency matrix, in which we took advantage of IP address ordering to help cluster adjacency matrix rows and columns. We showed how the adjacency matrix is ideal for highlighting the impact of network configuration changes on the attack graph.

We introduced a novel interactive attack graph filtering technique, based on a hierarchy of attack graph constraints. It allows the user to interactively navigate the constraint hierarchy to control the visible subset of the attack graph. We also link intrusion detection events to the constraint tree, which allows exploration of intrusion origin/impact.

The techniques we described apply to general attack graphs, including ones based on network vulnerabilities and/or intrusion detection alarms. They also apply to either fully unconstrained graphs, or graphs constrained to by starting/goal conditions. Overall, these techniques have quadratic worst-case complexity.

8 Acknowledgements

We wish to thank John Monastra for introducing the idea of attack graph filtering based on arbitrary user-defined constraints. We also thank Robert Weierbach for his support, including vulnerability data collection/analysis and exploit modeling. This work was partially supported by the Air Force Research Laboratory, Rome under the grant F30602-00-2-0512, the Army Research Office under the grant DAAD19-03-1-0257, and the National Science Foundation under the grants IIS-0430402 and IIS-0242237.

References

[1] C. Ramakrishnan, R. Sekar, "Model-Based Analysis of Configuration Vulnerabilities," in *Proceedings of the 7th ACM Conference on Computer and Communication Security*, November 2000.

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

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

[4] R. Baldwin, *Kuang: Rule Based Security Checking, Technical Report*, MIT Lab for Computer Science, May 1994.

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

[6] C. Phillips, L. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," in *Proceedings of the New Security Paradigms Workshop*, Charlottesville, VA, 1998.

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

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

[9] S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher, 2005.

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

[11] 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.

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

[14] 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 CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, VA, October 2004.

[15] X. Yin, W. Yurcik, M. Treaster, Y. Li, K. Lakkaraju, "VisFlowConnect: NetFlow Visualizations of Link Relationships for Security Situational Awareness," in *Proceedings of the CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, VA, October 2004.

[16] S. Lau, "The Spinning Cube of Potential Doom," *Communications of the ACM*, 47(6), June 2004.

[17] S. Axelsson, "Combining A Bayesian Classifier with Visualization: Understanding the IDS," in *Proceedings of the CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, VA, October 2004.

[18] "Secure Decisions," web page, http://www.securedecisions.com/, last accessed May 2005.

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

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

[21] M. Raitner, *Maintaining Hierarchical Graph Views for Dynamic Graphs*, Technical Report, MIP-0403, University of Passau, January, 2004.

[22] C. Kunz, V. Botsch, "Visual Representation and Contextualization of Search Results – List and Matrix Browser," in *Proceedings of the International Conference on Dublin Core and Metadata for E-Communities*, Florence, Italy, 2002.

[23] J. Ziegler, C. Kunz, V. Botsch, J. Schneeberger, "Visualizing and Exploring Large Networked Information Spaces with Matrix Browser," in *Proceedings of the IEEE Conference on Information Visualization*, London, England, 2002.

[24] M. Graham, J. Kennedy, "Exploring and Examining Assessment Data via a Matrix Visualisation," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, Gallipoli, Italy, 2004.

[25] F. van Ham, "Using Multilevel Call Matrices in Large Software Projects," in *Proceedings of the IEEE Symposium on Information Visualization*, Seattle, WA, 2003.

[26] E. Gansner, S. North, "An Open Graph Visualization System and Its Applications to Software Engineering," *Software – Practice and Experience*, 30(11), 2000.

[27] "Nessus Open Source Vulnerability Scanner Project," web page, http://www.nessus.org/, last accessed July 2005.

[28] "Security Focus," web page, http://www.securityfocus.com/bid, last accessed July 2005.

[29] "Graphviz – Graph Visualization Software," web page, http://www.graphviz.org/, last accessed July 2005

[30] "C5 Attack Predictor Overview," web page, http://www.secure-elements.com/products/c5ap/, last accessed July 2005.