Attack Graphs for Sensor Placement, Alert Prioritization, and Attack Response

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

Abstract

We describe the optimal placement of intrusion detection system (IDS) sensors and prioritization of IDS alarms, using attack graph analysis. Our attack graphs predict the various possible ways of penetrating a network to reach critical assets. In particular, automated analysis of network configuration and attacker exploits provides an attack graph showing all possible paths to critical assets. We then place IDS sensors to cover all these paths, using the fewest number of sensors. The sensor-placement problem we pose is an instance of the NP-hard minimal set cover problem, which we solve through a greedy algorithm. Through our approach, all traffic along vulnerable paths to critical resources is monitored, with no deployment of unnecessary sensors. Then, through our predictive vulnerability-based attack graphs, we prioritize IDS alarms based on their level of threat (attack graph distance) to critical assets. The predictive power of our attack graphs then provides the necessary context for appropriate attack response.

1. Introduction

In traditional network defense, IDS sensors are placed at network perimeters, and configured to detect every attempt at intrusion. But if an attacker manages to avoid detection at the perimeter, and gain a toehold into the network, his traffic on the internal network is unseen at the perimeter. Also, in today's highly distributed grid computing, network boundaries are no longer clear. Organizations have a desire to detect malicious traffic throughout their network, but may have limited resources for IDS sensor deployment. Moreover, IDS usually report all potentially malicious traffic, without regard to the actual network configuration, vulnerabilities, and mission impact. Given large volumes of network traffic, IDS with even small error rates can overwhelm operators with false alarms. Even when true intrusions are detected, the actual mission threat is often unclear, and operators are unsure as to what actions they should take.

To address these weaknesses, we focus on protecting the network assets that are mission-critical. By analyzing the network configuration, including topology, connectivity limiting devices such as firewalls, vulnerable services, etc., and matching it to known attacker exploits, we create a map of all possible attack paths through the network leading to compromise of mission-critical assets. This map (organized as an *attack graph*) provides the requirements for placing sensors within the attack paths to mission-critical assets. 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 attack graphs using known network vulnerabilities, we prioritize IDS alarms based on the level of threat they represent to critical assets. For example, we can give lower priority to alarms 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. Further, the attack graph 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.

2. Previous Work

One possible approach for automatic generation of attack graphs is to apply general-purpose symbolic model checking tools [1][2]. But model checkers have significant scalability problems, a consequence of the exponential complexity of the general state space they consider. Early graph-based approaches for analyzing attack combinations generally suffered the same exponential state space problem [3][4]. Subsequently, scalable attack graph models based on monotonic logic have emerged [5][6], as well as more efficient representations for fully-connected sub-graphs [7]. These advances help make automatic attack graph generation feasible for realistic sized networks. A review of various approaches to attack graph analysis is given in [8].

Initial applications of attack graphs focused on analyzing vulnerability to potential attacks. Attack graphs have subsequently been applied to IDS alert correlation [9]. Attack graphs are particularly powerful when predicted paths (based on vulnerabilities) are matched with actual detected attacks [10][11][12]. This helps eliminate false positives, enables prediction of missing alarms, makes alarm correlation faster, and provides the context for attack response.

Still, while integration of IDS alarms with predicted attack graphs has been proposed, a key step is missing, i.e., the actual placement of IDS sensors to cover the attack graphs. When IDS sensor placement is addressed in the literature, it is usually in the context of general architectures for distributed intrusion detection, such as [13]. At least one author has applied network attack modeling for sensor placement [14], using logic programming (Prolog) for system prototyping.

In our approach, we place alarms so as to cover all known paths of vulnerability potentially leading to compromise of critical network assets, based on comprehensive attack graph models. Our sensor placement is optimal, in that only a minimum number of sensors are needed. Then, once sensors are deployed, we use our predictive attack graph to prioritize the resulting alarms according to distance from critical assets.

3. System Overview

Our approach is to capture the network configuration, from which we predict attack paths through the network, and use the predicted paths for sensor placement, alarm prioritization, and attack response. As shown in Figure 1, we scan the network to discover hosts, their operating system, application programs, and vulnerable network services. We also capture network connectivity, including the effects of devices such as firewalls and router access control lists (ACLs). With the resulting network model, a database of modeled attacker exploits, and a specification of threat origin and critical network assets, we compute an attack graph, as described in. This graph, computed in worst-case quadratic time [6], comprises all known attacks through the network.

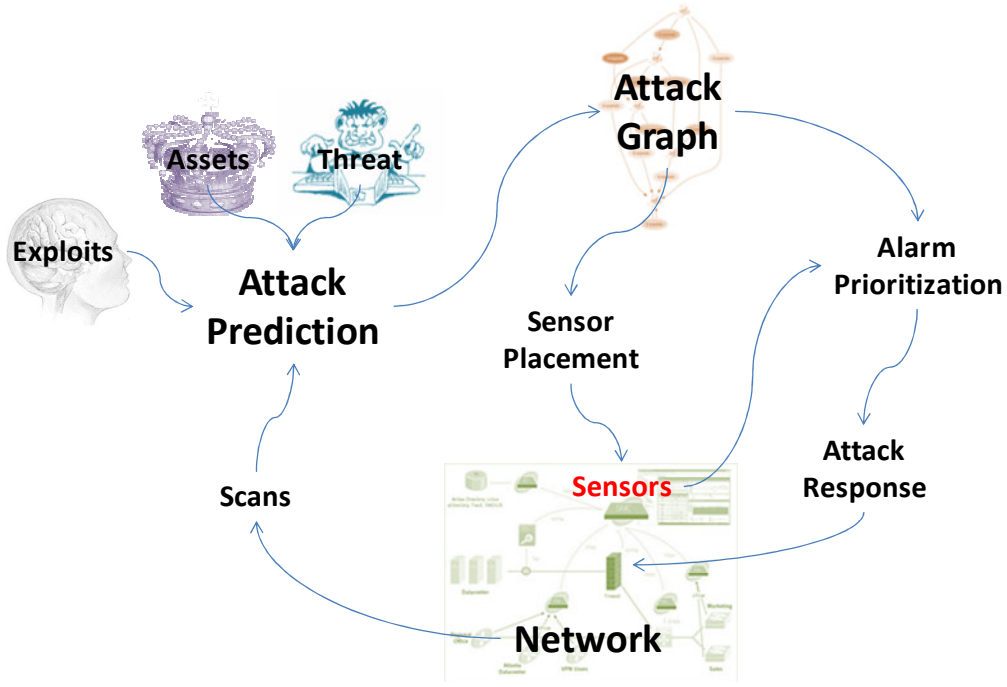

Figure 1: System for placing IDS sensors, prioritizing alerts, and responding to attacks.

As we describe in the next section (Section 4), the attack graph guides the placement of IDS sensors, for monitoring attacks against known vulnerabilities. In fact, our sensor placement solution is optimal, in the sense that the attack graph is entirely covered, using the fewest possible sensors. We then use the attack graph to correlate and prioritize any resulting IDS alarms, based on attack proximity to critical network assets, and to formulate optimal attack responses, as described in Section 5.

4. Optimal Sensor Placement

The problem we address is the following. Given the network configuration and vulnerabilities, where should we place IDS sensors so as to monitor all attack paths to critical network assets? Moreover, what placement will cover all critical paths with the fewest number of sensors?

For such optimal placement of sensors, we must first discover all attack paths to critical assets, i.e., the attack graph. These critical paths will define the sources and destinations of traffic to be monitored. Then, through analysis of network topology, we can identify sensor locations that cover the critical paths.

Consider the network on the left side Figure 2. There are 8 subnets, with various 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, assume there are firewalls, router ACLs, etc. in place to limit connectivity and help protect the network. Still, vulnerabilities remain on the network, and many are related, giving an attacker new vantage points for launching attacks and further penetrating the network.

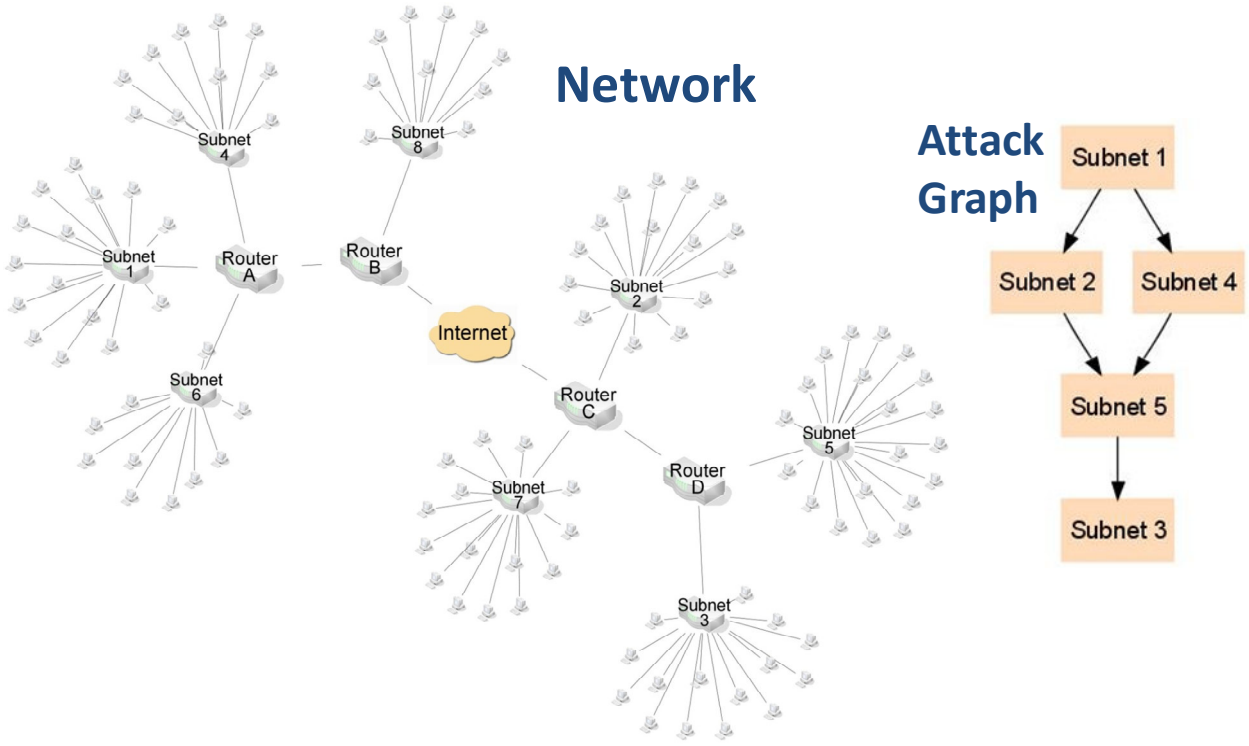

Figure 2: Example network and a high-level summary of its attack graph.

The right side of Figure 2 is a high-level overview of attacks through this network. Here, we assume that critical network assets reside in Subnet 3. The attack graph then shows all possible paths into Subnet 3, at the subnet-to-subnet level. Again, the attack graph is discovered through exhaustive analysis of network vulnerabilities and their interdependencies. At the level of detail shown (on the right side of the figure), a subnet box represents the collection of machines in a subnet, and an arrow means there is at least one vulnerable connection from one subnet to another. Of course, there may be much more connectivity through the network than shown in the attack graph, and many of these other connections may in fact be vulnerable to attack. But by definition the attack graph includes all paths leading to compromise of the given critical assets (in this case, in Subnet 3), so any other vulnerabilities can be safely ignored.

Now, given the knowledge of critical paths through the network, we can analyze the network topology for placing sensors to cover all paths. To minimize costs, we seek to cover all critical paths (the attack graph) using the least number of sensors. This optimal sensor placement is an instance of the classical *minimum set cover* problem [15]. Finding such minimum cover is NP-hard, so exhaustive search is needed for guaranteed optimal solutions. Fortunately, there is a polynomial-time greedy algorithm for minimum set cover that gives good results in practice.

Consider Figure 3, which is taken from the example in Figure 2. Here, through the network topology, we trace the routes of each subnet-to-subnet edge of the attack graph. For example, the vulnerable connections 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 connections in the attack graph.

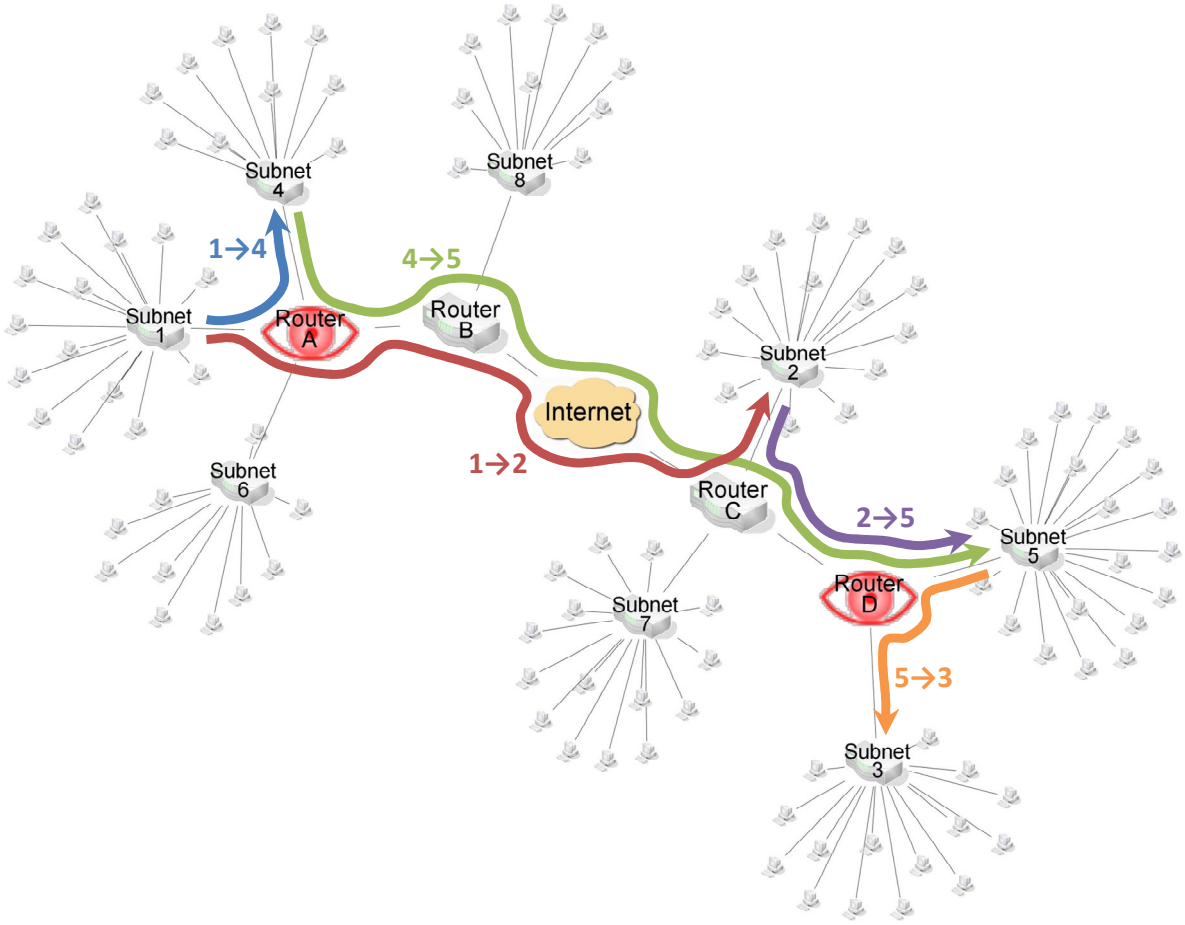

Figure 3: Optimal sensor placement.

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, the sets are defined by the routers along critical paths:

- Router A covers {(1,2), (1,4), (4,5)}
- Router B covers {(1,2), (4,5)}
- Router C covers {(1,2), (4,5), (2,5)}
- Router D covers {(2,5), (4,5), (5,3)}

Here, the element (*x*, *y*) means vulnerable connections 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 (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., (2,5) and (5,3). At this point, we have covered all 5 elements (vulnerable connections in the attack graph). Our sensor-placement solution is thus complete, shown in Figure 3 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. Indeed, for such a small example, we could easily solve it through exhaustive search. In general, the greedy algorithm will do no worse than log(*n*) times the optimal solution, for *n* elements to be covered, though it

usually does much better than this. Using appropriate data structures, the greedy algorithm can be implemented in $O(r)$, where r is the size of the input representation (number of elements over all input sets). Better solutions for set cover may be possible through more sophisticated algorithms (typically with longer run times), e.g., simulated annealing [16] and evolutionary algorithms [17].

5. Alarm Prioritization and Attack Response

Once IDS sensors are in place and alarms are generated, we can use the attack graph to correlate alarms, prioritize them, predict future attack steps, and respond optimally. Figure 4 shows attack graph details for the example in Figure 2, where edges are exploited vulnerabilities between machines. Paths in the graph all lead to specified critical assets (shown as crowns).

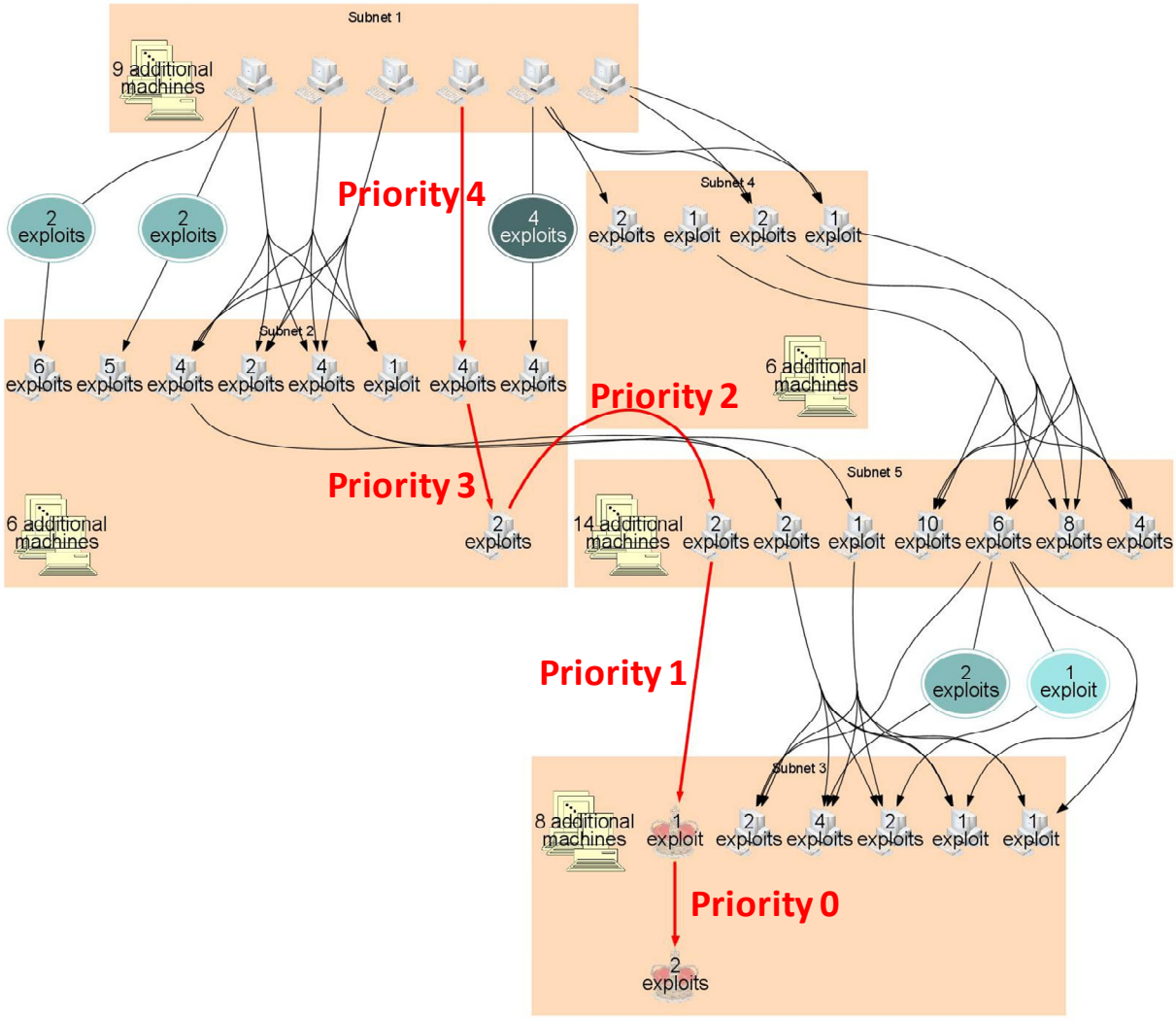

Figure 4: Priority of alarms via attack graph distance to critical assets.

We can prioritize alarms 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.

For example, in Figure 2, assume that the Priority-3 alert (within Subnet 2) is raised. At that point, we know that the attacker could next move anywhere within Subnet 2, or could launch an attack from the victim machine against one particular machine in Subnet 5. Thus, to prevent penetration towards the given critical network assets, traffic could be blocked from the victim machine to the Subnet-3 machine, for the specific ports corresponding to the two predicted exploits against it. This is the kind of highly focused attack response capability provided by our predictive attack graphs.

6. Summary

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

Acknowledgements

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

References

1. R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, California, 2000.
2. O. Sheyner, J. Haines, S. Jha, R. Lippman, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2002.
3. D. Zerkle, K. Levitt, "Netkuang – A Multi-Host Configuration Vulnerability Checker," in *Proceedings of the 6th USENIX Unix Security Symposium*, San Jose, California, 1996.
4. L. Swiler, C. Phillips, D. Ellis, S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, 2001.
5. P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, 2002.
6. S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Springer, 2005.

7. S. Noel, S. Jajodia, "Managing Attack Graph Complexity through Visual Hierarchical Aggregation," in *Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, 2004.
8. R. Lippmann, K. Ingols, An *Annotated Review of Past Papers on Attack Graphs*, Technical Report ESC-TR-2005-054, MIT Lincoln Laboratory, 2005.
9. P. Ning, Y. Cui, D. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington DC, 2002.
10. S. Noel, E. Robertson, S. Jajodia, "Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances," in *Proceedings of the 20th Annual Computer Security Applications Conference*, Tucson, Arizona, 2004.
11. S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, "Multiple Coordinated Views for Network Attack Graphs," in *Proceedings of the Workshop on Visualization for Computer Security*, Minneapolis, Minnesota, 2005.
12. S. Noel, S. Jajodia, "Understanding Complex Network Attack Graphs through Clustered Adjacency Matrices," in *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, Arizona, 2005.
13. C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné, A. Thomas, "A Hardware Platform for Network Intrusion Detection and Prevention," in *Proceedings of 3rd Workshop on Network Processors & Applications*, Madrid, Spain, 2004.
14. M. Rolando, M. Rossi, N. Sanarico, D. Mandrioli, "A Formal Approach to Sensor Placement and Configuration in a Network Intrusion Detection System," in *Proceedings of the ACM International Workshop on Software Engineering for Secure Systems*, Shanghai, China, 2006.
15. T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press and McGraw-Hill, 2001.
16. S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, 1997.
17. R. Kalapala, M. Pelikan, A. Hartmann, *Hybrid Evolutionary Algorithms on Minimum Vertex Cover for Random Graphs*, MEDAL Report No. 2007004, University of Missouri–St. Louis, 2007.