Minimizing Attack Graph Data Structures

Peter Mell
National Institute of Standards and Technology
Gaithersburg, MD United States
email:peter.mell@nist.gov

Richard Harang
U.S. Army Research Laboratory
Adelphi, MD United States
email:richard.e.harang.civ@mail.mil

Abstract— **An attack graph is a data structure representing how an attacker can chain together multiple attacks to expand their influence within a network (often in an attempt to reach some set of goal states). Restricting attack graph size is vital for the execution of high degree polynomial analysis algorithms. However, we find that the most widely-cited and recently-used 'condition/exploit' attack graph representation has a worst-case quadratic node growth with respect to the number of hosts in the network when a linear representation will suffice. In 2002, a node linear representation in the form of a 'condition' approach was published but was not significantly used in subsequent research. In analyzing the condition approach, we find that (while node linear) it suffers from edge explosions: the creation of unnecessary complete bipartite sub-graphs. To address the weaknesses in both approaches, we provide a new hybrid 'condition/vulnerability' representation that regains linearity in the number of nodes and that removes unnecessary complete bipartite sub-graphs, mitigating the edge explosion problem. In our empirical study modeling an operational 5968-node network, our new representation had 94 % fewer nodes and 64 % fewer edges than the currently used condition/exploit approach.**

Keywords- attack graph; complexity analysis; data structures; minimization; representation; security.

I. INTRODUCTION

An attack graph is a representation of how an attacker can chain together multiple attacks to expand their influence within a network (often in an attempt to reach some set of goal states) [1]. Among other things, an attack graph can be used to calculate the threat exposure of critical assets, prioritize vulnerability remediation, optimize security investments, and as a tool to guide post-compromise forensic activities. Restricting attack graph size is vital for both human visualization of sub-graphs and the execution of high degree polynomial analysis algorithms. Early attack graph research used a 'state enumeration' representation [2] that would record all possible orderings by which an attacker could exploit a set of vulnerabilities, and hence grow at a factorial rate (exponential with some modifications). This rapid growth was mitigated in 2002 by a 'condition-oriented' approach providing a linear number of data objects with a quadratic worst-case number of relationships (with respect to the number of hosts in the original network) [3]. A major tenet of this approach was the assumption of 'monotonicity', which stated that an attacker would never lose a privilege once it was gained and any increase in privilege would not negate other gained privileges. This removed the need to account for the order in which attacks are initiated, and so reduced the complexity of the representation.

Subsequent research modified this model to make it attack- focused and enable humans to visually follow the steps within an attack more easily [2]. This hybrid 'condition/exploit' model [4] has been used extensively since 2003 for attack graph research. Unfortunately, we find that this model results in redundant data encodings, under which the worst-case graph size become quadratic in the number of nodes. Thus, over a decade of attack graph research has used a quadratic representation when a linear one was available in the literature.

However, even had the node linear condition-oriented approach been adopted, we find that it suffers from avoidable edge explosions (the creation of unnecessary complete bipartite sub-graphs) under certain conditions. These edge explosions add a quadratic factor to worst-case edge growth based on the maximum number of attacker privileges on any one host.

These size complexity issues are not always readily apparent from visual inspection of small example graphs. Example attack graphs in the literature to date have often contained a small number of dissimilar hosts with limited per-host attack surfaces and thus do not reveal the worst-case growth in both nodes and edges that we have observed. In large enterprise networks, however, where hosts have both large and diverse attack surfaces and are vulnerable to high-level compromise (thus yielding a high number of post-conditions), these complexity issues become much more evident.

To address these weaknesses, we provide a new hybrid 'condition/vulnerability' representation that regains linearity in the number of nodes and that does not suffer from the edge explosion problem. In our empirical study of a network model derived from an operational 5968-node network, our new representation had 94 % fewer nodes and 64 % fewer edges than the most widely cited recent approach in our surveyed literature (the condition/exploit model).

This reduced graph size will increase the speed of automated analysis, even making some algorithms tractable under certain scenarios. This can be true for higher polynomial complexity algorithms such as the cubic algorithms in [3] and certainly for metrics derived from attack graphs that grow either exponentially or with high polynomial degree [5]-[8] (as often occurs when graphs containing cycles must be rendered acyclic for the purposes of probabilistic modeling). The reduced size can also aid in human visualization and analysis of select sub-graphs of interest.

The development of the work is as follows. In Section 2, we survey past attack graph representations and describe them in terms of four major categories (while citing minor variations). For each category, we provide a description, theoretical analysis of worst-case growth, and an example graph from previously published work. Section 3 then provides our two new representations that improve upon the worst-case node and edge growth of the other representations. Section 4 provides a theoretical analysis of set of analyzed approaches. Section 5 provides empirical results using a network model based on an operational network where we compare the attack graph sizes using the different approaches. Section 6 concludes the paper.

II. SURVEY OF ATTACK GRAPH REPRESENTATIONS

Papers discussing some abstraction of the attack graph idea began appearing as early as 1996 [9]-[12]. The first widely used representation was the ‘state enumeration’ approach ([2] and [13]-[16]) that had the unfortunate characteristic that it could grow faster than exponential. In 2002, the ‘condition-oriented’ approach was published with the express purpose of reducing the graph representation size down to polynomial complexity [3]. In 2003, the ‘exploit-oriented’ approach was published [17] to enhance the human readability of the graphs compared to the condition-oriented approach [2]. While not discussed in the literature, we will show that the exploit-oriented approach suffers high polynomial growth rates. This may explain why, in the same year, our surveyed literature moved to a more efficient hybrid ‘condition/exploit-oriented’ approach [4] (which we find still has a growth rate higher than that of the condition-oriented approach). Our literature survey shows that this approach has been used extensively since 2003 and is the predominant representation (e.g., [1], [4], [7], and [18]-[20]). The following subsections discuss each of these approaches in detail. It should be noted that each representation encapsulates the same underlying knowledge but using a different abstraction. For each type of representation, we analyze the worst-case growth rate of a resulting attack graph with respect to the number of nodes and edges. We define *h* to be the number of hosts in the associated physical network, *v* to be the maximum number of vulnerabilities on any one host, and *c* to be the maximum number of attacker privileges that can be achieved on any one host from the set of available vulnerabilities. For the graph size complexity analyses, we assume that there is a unique set of pre-conditions for each attack.

A. *State Enumeration*

State enumeration was the first widely used attack graph approach. Its distinguishing feature is that it explicitly accounts for the different orderings in which an attacker may launch attacks. There are two types. One type uses nodes to represent attacks and edges to represent post-conditions resulting in attacker privilege [13]-[15]. The other type uses nodes to represent attacker privilege and edges to represent attacks ([2] and [16]).

The graph design contains multiple layers of nodes, regardless of the particular node and edge semantics. The initially available set of conditions or attacks are presented at the top layer. Each subsequent lower layer represents the possible decisions that an attacker could make. Directed edges connect the nodes at one layer to the available nodes at the next lower layer. There are no edges between nodes at a particular layer. An attack scenario begins at the top layer and works its way down with the node chosen in each layer representing an attacker’s next decision.

1) *Complexity Analysis*

Assume that *v* and *c* equal 1. At the top layer (labelled 0), any of h hosts may be selected as the first node in the attack path. At layer 1, there are *h*-1 hosts that can be attacked from each of the h start nodes. For each node at layer 2 there are *h*-2 hosts that can be attacked, and so forth. This creates a rapidly expanding tree where the number of nodes and edges increases as O(*h*!), yielding a worst-case factorial growth rate for state enumeration graphs.

Most approaches attempt to prune this tree to focus only on subtrees of interest (e.g., those leading to some goal state). This can limit the growth, allow limited reuse of nodes, and can migrate the structure from a tree to a hierarchical directed graph with no loops (see Figure 1). Despite such optimizations, the growth rate of this approach is still worst-case exponential [2].

From an empirical perspective, such graphs become too large to be practical. For example, a network with just 5 hosts and 8 available exploits produced an attack graph with 5948 nodes and 68 364 edges [15].

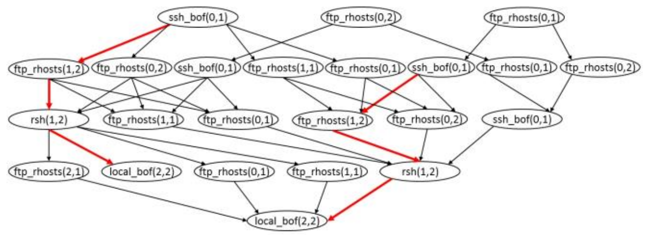

Figure 1. Example State Enumeration Attack Graph

Figure 1 shows an example pruned state enumeration attack graph, from [2], derived from an example with 2 target hosts running 2 services each, a single attacking host, and 4 unique attack types. Notice that the representation uses the previously discussed modifications to avoid factorial growth rate. The large red arrows highlight a remaining inefficiency by showing an example of path duplication in the graph (discussed in [2]).

Additional variations include [21]-[24].

B. *Condition-Oriented*

The condition-oriented approach was introduced in [3] as a method to reduce the representational complexity of the state enumeration approach by using the assumption of attacker privilege ‘monotonicity’. This assumption implies that an attacker never loses a privilege once it is gained, and any increased privilege will not negate any other privileges. In practice, it means that (unlike in the state enumeration approach) there is no longer any need to track the order in which attacks are initiated. This assumption has been found to be reasonable in most cases [2] and has been adopted by almost all subsequent attack graph representations.

The work of [2] introduced a graphical representation to the approach. Each ‘condition node’ represents some state of attacker privilege (e.g., execute as superuser on host x). An edge from node a to node b with label c represents that pre-condition a is necessary (but not necessarily sufficient) for attack c to provide privilege b. Sufficiency to gain privilege b is obtained if the pre-conditions of all edges into node b, labeled with c, are satisfied. By grouping the in-edges to each node by their edge labels, we can see that each group represents a possible attack, and each node is thus expressed in a variant of disjunctive normal form (DNF): in-edges within a group provide conjunctional logic and the distinct groups form logical disjunctions. Note that unlike general DNF statements, negation is not represented.

One limitation of the approach in [3] is that a single attack on a host that can be instantiated by different sets of pre-conditions must be represented by multiple attack instances named differently (using the edge names) to enable disjunctional logic. Distinct attack instances, involving different hosts, may be named similarly with no ambiguity.

1) *Complexity Analysis*

The condition-oriented approach achieves linearity in the number of nodes, significantly reducing the complexity compared to the state enumeration approach. An attacker may have up to c distinct condition states on each of the h hosts, and thus the number of nodes is bounded by hc. Unfortunately, as each of the hc condition nodes can be connected to all hc other nodes, and each connection between two nodes may have up to v edges to represent exploiting each available vulnerability, we obtain up to $hc \times hc \times v = h^2c^2v$ edges in the graph. Normally, h is much larger than c or v for a large network (as c and v are the maximums per node, not totals) and thus we can treat c and v as constants for the complexity analysis. Thus, the condition approach is $O(h)$ in nodes and $O(h^2)$ in edges, representing an enormous improvement over the exponential state enumeration approach. However, note the multiplicative c^2 term in the number of edges. This is the result of unnecessary complete bipartite sub-graphs forming under certain conditions. We analyze these edge explosion situations in more detail in Section 4.

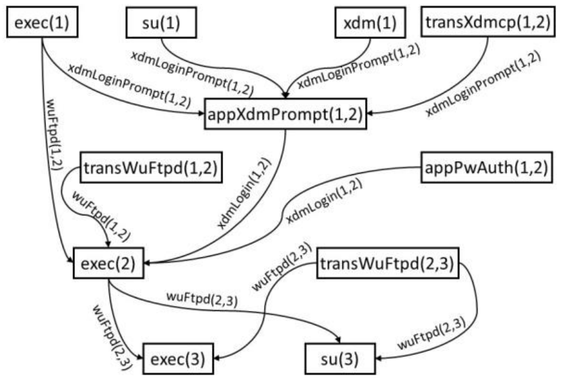

Figure 2. Example Condition-Oriented Attack Graph

Figure 2 shows an example condition-oriented attack graph, from [2], derived from a network with 2 target hosts, a single attacking host, and 3 unique attack types. Note the reduced complexity compared to Figure 1 although, as stated previously, these small example graphs are primarily intended to illustrate the methods, and do not demonstrate the differences in worst-case size complexities.

Additional variations include [21] and [25].

C. *Exploit-Oriented*

The exploit-oriented approach represents attacks as nodes and states of attacker privilege as edges ([2] and [17]) to ease visual analysis compared to the condition-oriented approach. Note that each ‘attack node’ is labeled with the host launching the attack and the host receiving the attack (which can be the same host for local attacks). This dual labeling on attack nodes will cause significant representational inefficiencies. The in-edges to a node represent the pre-conditions for launching an attack and the out-edges represent the post-conditions of the exploit. All out-edge post-conditions of a node are satisfied if and only if all the in-edge pre-conditions are satisfied.

Explicit representation of disjunction is not available and so in the presentation in the literature, attacks that can be instantiated by distinct sets of pre-conditions must be represented by multiple nodes. However, by applying a similar edge grouping approach as in the condition-oriented approach, this duplication of nodes can be avoided. Since the edges in this approach are already labeled with the post-condition names they must be additionally labeled with a grouping name (a name for the group of pre-conditions that will enable exercising the related exploit). While this modification is not discussed in the literature, we assume this optimization to represent the approach as efficiently as possible. Without this optimization, the number of nodes would increase by a factor of c.

1) *Complexity Analysis*

We now examine the worst-case growth rate of some arbitrary attack graph. With respect to nodes, the graph can grow as large as h^2v. Each of the h hosts can attack all h hosts with v different attacks (assuming just one attack per vulnerability) resulting in h^2v nodes. With respect to edges, the graph can grow as large as h^3v^2. Consider a single node where b represents the attack target. This node can have a connection to each node where the attack source is b. There will be hv nodes with attack source b. Each connection though can be made up of c edges. Thus, each node can create hvc out-edges. Since the number of nodes is h^2v, this leads us to $h^2v \times hvc = h^3v^2c$ edges. Treating c and v as constants compared to h, the exploit approach is $O(h^2)$ in nodes and $O(h^3)$ in edges. This is an enormous increase in graph size from the condition approach, however it still outperforms the exponential state enumeration approach.

Figure 3 shows an example exploit-oriented attack graph, from [2], derived from the same network as Figure 2. The example condition-oriented graph has 11 nodes and 12 edges while the example exploit-oriented graph as 6 nodes and 13 edges. Despite the reduction in graph size demonstrated in this example, we will empirically show that the exploit-oriented graphs can grow much larger than the condition-oriented graphs in enterprise networks.

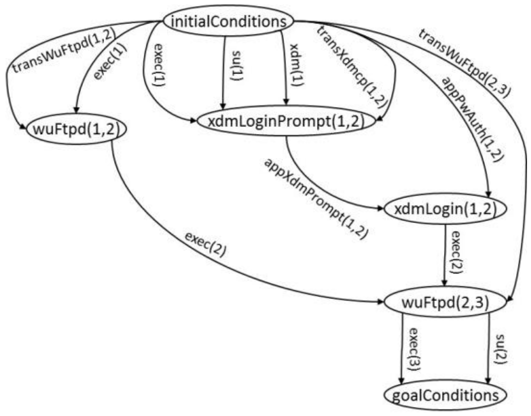

Figure 3. Example Exploit-Oriented Attach Graph

D. *Hybrid Condition/Exploit-Oriented*

The hybrid condition/exploit-oriented approach uses two distinct types of nodes ([1], [4], [7], and [18]-[20]) representing both attacks and the states of attacker privilege, while the edges are unlabeled. The 'attack nodes' and 'condition nodes' have the same semantics as those in the exploit-oriented graphs and the condition-oriented graphs, respectively.

This structure produces a directed bipartite graph, with attack nodes having edges to condition nodes and vice versa. However, the interpretation of the in-edges varies per type of node. Attack nodes and their post-condition out-edges are all satisfied if and only if all in-edges are satisfied (conjunction as with the exploit-oriented graphs). Condition nodes and their out-edges are satisfied if at least one in-edge is satisfied (disjunction as with multiple groups of in-edges in the condition-oriented graphs).

As in the exploit-oriented representation, the problem of a single exploit that can be instantiated with multiple sets of pre-conditions is not well addressed in the literature. In a naïve implementation, a single exploit must be divided into multiple attack node instances, one for each distinct set of pre-conditions. However, by applying the same optimization as before (again, not previously presented in the literature) where we allow condition node to attack node edges to be labeled with a group name, we can avoid this multiplication, and we assume this optimization throughout. As in the condition approach, the interpretation is disjunction among the groups and conjunction within a group. Without this optimization, the worst-case number of attack nodes would increase by a factor of c (as with the exploit approach).

1) *Complexity Analysis*

We now examine the worst-case growth rate of some arbitrary attack graph. With respect to nodes, the graph can grow as large as $hc+h^2v$. There will be hc condition nodes as derived in Section 2.2.1 and there will be h^2v attack nodes as derived in Section 2.3.1. With respect to edges, the condition and attack nodes form a directed bipartite graph. We first explore the set of attack to condition node edges. Each attack node has a single target host as discussed in Section 2.3. Each attack node then can at most activate c condition nodes on the target host where each activation creates an attack to condition node edge. Since there are up to h^2v attack nodes, we can then have up to $h^2v \times c = h^2vc$ attack node to condition node edges. Similarly, each condition node is mapped to a host, say a, and thus may have an edge to any of the hv exploit nodes where the attack source is a. Since there are hc condition nodes and up to hv edges per node, we get a total of $hc \times hv = h^2vc$ condition to exploit node edges. Summing the two types of edges, we get $h^2vc + h^2vc = 2h^2vc$.

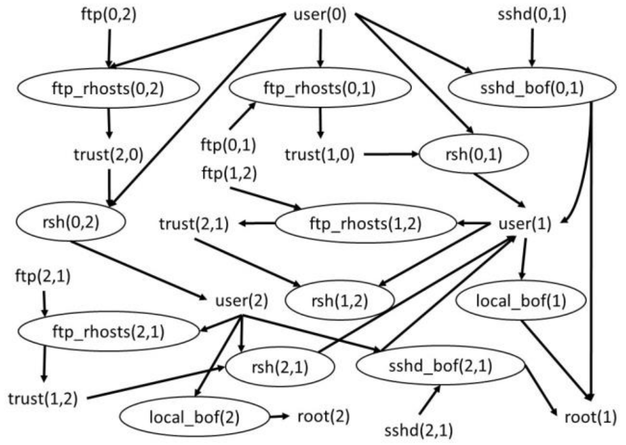

Figure 4. Example Condition/Exploit-Oriented Attack Graph

Figure 4 shows a condition/exploit-oriented attack graph from an example provided in [4]. This was derived from the same network as in Figure 1. The circled nodes are the attack nodes and the un-circled nodes are the condition nodes. Notice the relative simplicity in comparison with the state enumeration approach in Figure 1. Again though, the size of these small examples doesn't demonstrate complexity growth on large enterprise networks.

Additional variants include [26]-[28].

III. VULNERABILITY-BASED REPRESENTATIONS

Our contribution is the idea of representing the vulnerabilities on specific hosts explicitly within attack graphs. From a visualization point of view, this makes it easy to see how a chain of vulnerabilities (and hosts) can be compromised. From a graph complexity point of view, the vulnerability nodes replace the use of the attack nodes thereby lowering node complexity to linear (from quadratic). Intuitively, where exploit nodes must contain references to both the source and target hosts in an attack step, and thus grow potentially quadratically in highly connected networks, the vulnerability nodes only reference the exploitable host, and so grow only linearly. We represent attacks within the edges where they can take advantage of the fact that edges inherently have sources and targets (similar to the condition approach).

This approach leads to two new representations that build upon one another. We first describe a vulnerability-oriented approach where we replace the attack nodes from the exploit-oriented approach with vulnerability nodes. This reduces node complexity from quadratic to linear and edge complexity from cubic to quadratic. We then point out how the vulnerability-oriented approach suffers from similar quadratic edge explosion scenarios as the condition approach (but this time relative to v, not c).

We then build upon the vulnerability approach to provide a hybrid condition/vulnerability representation that has a linear number of nodes, quadratic number of edges, and that does not suffer from quadratic edge explosion (in either v or c). It also, like the hybrid condition/exploit approach, improves on the human readability of the condition-oriented approach.

A. *Vulnerability-Oriented*

Our first approach is analogous to the exploit-oriented approach except that we replace the attack nodes with vulnerability nodes. A vulnerability node is labeled with a vulnerability name and the relevant location in the network (usually but not necessarily a hostname). Edges represent attacker privilege, just like in the exploit-oriented approach. In cases where multiple sets of pre-conditions can activate a particular vulnerability, we handle it with the edge grouping optimization we've presented previously. A set of pre-conditions that will activate a vulnerability are represented by a set of in-edges to a node that all have a common group name. Thus we represent attacks using groups of commonly named edges just like in the condition approach. Each node is thus expressed in DNF: in-edges within a group provide conjunctional logic and the distinct groups form logical disjunctions.

1) *Complexity Analysis*

We now examine the worst-case growth rate of some arbitrary attack graph. With respect to nodes, the graph can grow as large as hv because each of the h hosts can have v vulnerabilities. With respect to edges, each node can have hv outgoing connections to other nodes. Each connection can be made up of at most c edges. Thus, the total number of edges is at most $hv \times hv \times c = h^2v^2c$.

A disadvantage of this approach is the v^2 term in the number of edges. This is the result of unnecessary complete bi-partite sub-graphs forming under certain conditions. We analyze this edge explosion in more detail in Section 4.

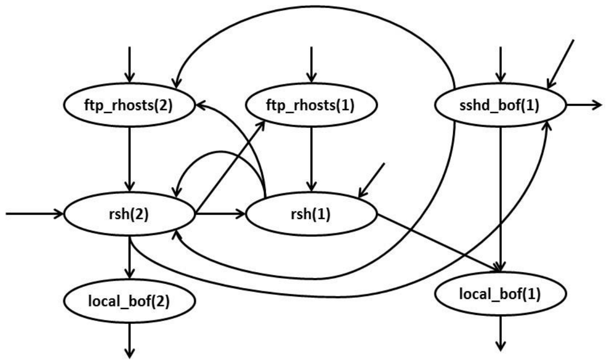

Figure 5. Example Vulnerability-Oriented Attack Graph

Figure 5 shows an example of a vulnerability-oriented graph, derived from the representation in Figure 4. For the sake of readability, edge grouping labels are omitted. Note that the number of nodes is reduced with respect to Figure 4, and all vulnerabilities are in exactly one node instead of replicated over multiple nodes with different attack sources (e.g., the attacks targeting the ftp rhosts vulnerability on node 1 in Figure 4).

B. *Hybrid Condition/Vulnerability-Oriented*

Our second novel approach is a hybrid approach combining condition nodes with our new vulnerability nodes. The condition nodes are analogous to those in the condition-oriented approach. The vulnerability nodes are identical to those in our vulnerability-oriented representation. We represent attacks by labeling the condition node to vulnerability node edges with the attack instances being used (including source and destination hostnames/IPs where applicable); this edge labeling is similar to that in the condition-oriented graphs. We then connect the vulnerability nodes to the condition nodes with unlabeled edges showing which post-conditions emerge as a result of exploiting the relevant vulnerability instance.

As with the hybrid condition/exploit graph, this structure produces a directed bipartite graph. For condition nodes, they and all of their attack labeled out-edges are satisfied if at least one in-edge is satisfied (disjunction). For vulnerability nodes, they and all of their unlabeled out-edges are satisfied if and only if a group of identically labeled in-edges are satisfied (conjunction within a group and disjunction between the groups). This distributed implementation of disjunctions and conjunctions enables the same DNF logic of the condition-oriented approach.

A single attack that can be instantiated with multiple sets of pre-conditions can be represented by using a different group name for each set of instantiating pre-conditions.

1) *Complexity Analysis*

We now examine the worst-case growth rate of an arbitrary attack graph. With respect to nodes, the graph can grow as large as $h(c+v)$. There will be hc condition nodes as derived in Section 2.2.1 and hv vulnerability nodes as derived Section 3.1.1. Thus, there are at most $hc+hv$ nodes total. With respect to edges, there can be as many as $h^2cv+hvc$. First, for the set of edges that point from vulnerability nodes to condition nodes, each of the hv vulnerability nodes can activate up to c condition nodes (since each vulnerability pertains to a host with up to c conditions), creating hvc edges. In the other direction, each of the hc condition nodes could allow an attack to all hv vulnerability nodes, producing $hc*hv$ edges. Thus, the total number of edges is $hvc+hc*hv=h^2cv+hvc$.

Treating c and v as constants compared to h, the condition/vulnerability approach is $O(h)$ in nodes and $O(h^2)$ in edges. This is a major improvement over the quadratic growth in the number of nodes caused by attack nodes in the exploit and condition/exploit approaches. While the complexity of the number of nodes is of the same order between the condition, vulnerability, and condition/vulnerability graphs, we note that there is no $v2$ or c^2 term in the edge growth equation for the condition/vulnerability graph. This is indicative of the fact that this approach does not suffer from the quadratic edge explosion problem of the condition and the vulnerability-oriented approaches in either *v* or *c*. This is discussed in detail in Section 4.

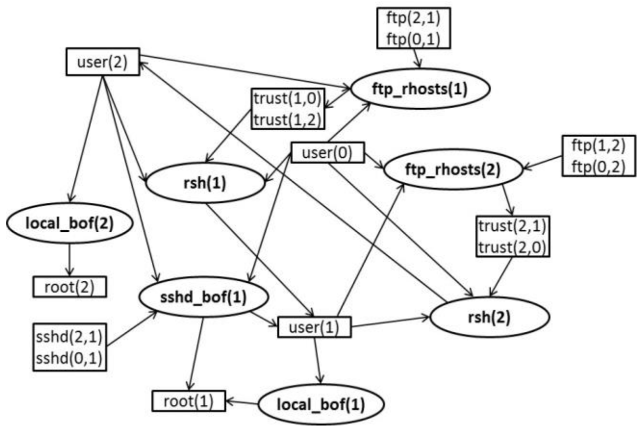

Figure 6. Example Condition/Vulnerability-Oriented Attack Graph

Figure 6 shows an example of the condition/vulnerability graph displaying the same attack graph data as in Figure 4 and Figure 5. In this example with just 3 nodes and relatively diverse attack surfaces, this data actually has a more compact representation in the vulnerability graph format. We demonstrate in Section 5 on larger scale real-world data that this advantage is not general, and in fact the condition/vulnerability graph provides significant size advantages in such cases.

IV. Theoretical Analysis

In this section, we provide a theoretical analysis of the worst-case behavior of the representations. Note that we do not analyze the state enumeration approach given its known exponential growth rate (previously discussed). We begin with a review of the big-O complexity of each graph type and show the advantages of the condition, vulnerability, and condition/vulnerability approaches. We then follow with an analysis of the terms within the actual worst-case growth equations. These equations reveal quadratic terms in both *v* and *c* that cause the edge explosion scenarios in the condition and vulnerability approaches, respectively. We then explore in more detail when and why these avoidable edge explosion scenarios occur. We end the section with a discussion of edge explosion scenarios that are unavoidable in all of our analyzed representations (and that may reflect an inherent limit in reducing graph sizes).

A. *Big-O Complexity Graph Growth Comparisons*

In an attack graph, *h* is expected to grow much larger than *v* or *c* for a typical enterprise network. Note that *v* and *c* are the maximums per host (not the total number of vulnerability and conditions) and thus are usually miniscule compared to *h*. For this reason, we treat *v* and *c* as constants to derive overall complexity of each representation. These big-O measurements were derived in Sections 2 and 3 and are summarized in Table 1. The calculations showing the largest growth rates are bolded. Note, if *h* is not large relative to *v* or *c*, use the below Table 2 instead of Table 1 to determine the most efficient representation.

TABLE 1. COMPLEXITY MEASUREMENT OF ATTACK GRAPH REPRESENTATION

Representation	Nodes	Edges
Condition	O(h)	O(h^2)
Exploit	**O(h^2)**	**O(h^3)**
Vulnerability	O(h)	O(h^2)
Condition/Exploit	**O(h^2)**	O(h^2)
Condition/Vulnerability	O(h)	O(h^2)

The quadratic node growth of the exploit and condition/exploit approaches is much larger than the linear node growth of the condition and condition/vulnerability approaches. With respect to edges, the cubic edge growth of the exploit approach is larger than the quadratic growth of the other approaches. Thus, the condition, vulnerability, and condition/vulnerability approaches are the best approaches in limiting worst-case graph growth with respect to *h*.

To intuitively understand why the exploit and condition/exploit node growth is quadratic, consider that an exploit node must necessarily contain two host name labels: the attack source and the attack target. If a set of hosts A can attack a set of hosts B using a single attack, then the number of exploit nodes representing this will be |A|*|B| (i.e., quadratic growth). Contrast this to the condition/vulnerability approach where there will be only a single vulnerability node per host in B resulting in |B| vulnerability nodes (i.e., linear growth).

One optimization is to reduce graph size by consolidating groups of identical hosts (those with identical security value, vulnerabilities, and permitted connectivity) into single hosts when building the attack graph. This essentially reduces *h* and thus minimizes the graph size, but we note that it does not change the big-O complexity results nor the outcome of our comparative analyses.

B. *Worst-Case Equations Graph Growth Comparisons*

We now look at the actual worst-case equations that we derived in Sections 2 and 3 in order to further refine our comparison between the approaches. These equations are summarized in Table 2. The terms specifically discussed in our analysis are bolded.

TABLE 2. WORST-CASE GROWTH OF ATTACK GRAPH REPRESENTATIONS

Representation	Nodes	Edges
Condition	hc	h^2v**c^2**
Exploit	h^2v	h^3v^2c
Vulnerability	hv	h^2**v^2**c
Condition/Exploit	$hc+h^2v$	$2h^2vc$
Condition/Vulnerability	**hc+hv**	h^2**vc**$+hvc$

We focus our analysis on the condition, vulnerability, and condition/vulnerability approaches since the other two approaches were shown to have larger big-O node complexities in Table 1.

With respect to node growth, the condition and vulnerability approaches will always have fewer nodes than the condition/vulnerability approach due it having the sum of the former two. However, it should be noted that this increase is a small linear factor.

With respect to edge growth, however, both the condition and vulnerability graphs grow quadratically in *c* and *v*, respectively. It is these quadratic terms (not present in the condition/vulnerability edge equation) that reflect edge explosion scenarios where unnecessary complete bipartite sub-graphs are formed. We claim that they are unnecessary because the condition/vulnerability approach provides a linear representation of the same data. Visually, the condition and vulnerability approaches use complete bi-partite sub-graphs where, for the same data, the condition/vulnerability approach uses star topologies (explained in detail in the next section).

Thus overall, our theoretical analysis indicates that our condition/vulnerability approach will result in achieving the most compact attack graphs. The exploit approach was the worst (excluding the naïve state enumeration approach), suffering from cubic edge growth. The condition/exploit approach (the most commonly used in recent research in our surveyed literature) suffered from quadratic node growth in our largest term, *h*. Finally, the condition and vulnerability approaches suffered from quadratic edge explosions (in *c* and *v*, respectively) as a result of the creation of complete bi-partite sub-graphs that our condition/vulnerability approach converts to linear growth star topologies. We now explore in detail the edge explosion scenarios.

C. Avoidable Edge Explosion Scenarios

We define an edge explosion as the creation of a complete bipartite sub-graph within an attack graph due to some specific scenario. Some scenarios cause edge explosions regardless of which of our analyzed attack graph representations is used. We call these scenarios ‘unavoidable’ (with respect to our set of representations) and thus they are not useful for a comparative analysis. We discuss such unavoidable scenarios in the next section.

This section focuses on avoidable scenarios that create quadratic edge explosions in the condition and vulnerability representations, which are converted to linear star representations in the condition/vulnerability approach. The condition/exploit approach has similar representational advantages with respect to edge explosion scenarios. However, we do not specifically analyze this for the condition/exploit approach due to its worse quadratic node complexity but we do note that it is the dual node type representations that enable the reduction (i.e., the ‘hybrid’ node design).

Avoidable edge explosion can occur in both the condition and vulnerability graphs as shown by their worst-case quadratic growth in *c* and *v*, respectively. These upper bounds are approached in condition graphs whenever a single attack has multiple pre-conditions and multiple post-conditions. This also happens in vulnerability graphs when some set of vulnerabilities on a host allows subsequent exploitation of some other set of vulnerabilities (on the same or other hosts).

Both cases can be viewed in terms of directed hyperedges, representing the multi-way relationships. For example, an attack with multiple pre-conditions and post-conditions can be represented by a single directed hyperedge with the pre-conditions at the tail of the edge and the post-conditions at the head. In standard directed graphs, representing this requires the creation of a complete directed bipartite sub-graph with the pre-conditions in the edge ‘tail’ set and the post-conditions in the ‘head’ set. The representation of these hyperedges by the corresponding complete bipartite graphs is then responsible for the quadratic explosion in the number of edges. Similarly, given a set of vulnerabilities on a host where each one enables an attacker to exploit some other common set of vulnerabilities can be represented by a single directed hyperedge. In a vulnerability graph, this hyperedge would require a directed complete bipartite graph with the enabling vulnerabilities in the edge ‘tail’ set and the newly available vulnerability nodes in the edge ‘head’ set.

Hybrid node graphs, such as the condition/vulnerability and the condition/exploit graphs, represent these directed hyperedges more efficiently (i.e., linearly). To see this, consider that in the condition/vulnerability graph, each vulnerability node may represent a directed hyperedge that links multiple pre-conditions to multiple post-conditions. Each condition node may also represent a directed hyperedge that links multiple vulnerabilities on a single host to a set of common target vulnerabilities. This representational approach of a directed hyperedge forms a star graph for each hyperedge. It is then easy to see that star graphs grow linearly in the number of edges while the complete bipartite sub-graphs grow quadratically, thus enabling the size complexity advantage of the hybrid node approaches.

For the purposes of illustration, consider Figure 7 below. Conditions C_1 to C_4 form the tail of a hyperedge corresponding to a vulnerability V_a, while conditions C_5 to C_8 for the head. The resulting condition graph is complete bipartite, as each of C_1 to C_4 must be linked to each of C_5 to C_6 (Figure 7, left); by contrast, using a separate class of node to represent the vulnerability-related hyperedge in the condition/vulnerability approach allows for a much more compact representation in the form of a star graph (Figure 7, right).

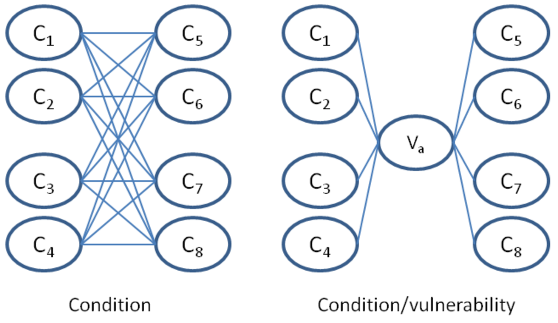

Figure 7. Unnecessary complete bipartite structures in the condition-oriented graph

Vulnerability graphs have a directly analogous representation, where a condition node may represent a hyperedge, as shown in Figure 8.

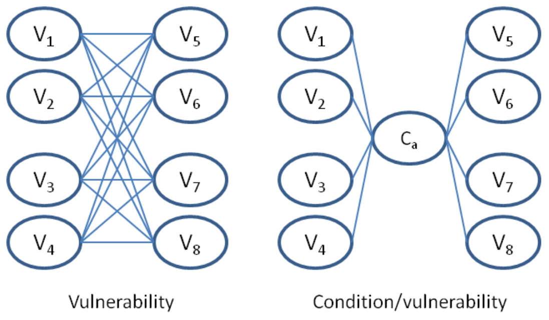

Figure 8. Unnecessary complete bipartite structures in the vulnerability-oriented approach

In the vulnerability graph, each vulnerability (V_1 to V_4) on a single host must have an edge to each vulnerability that is now accessible for attack (V_5 to V_8). In the condition/vulnerability graph, a single condition node C_a represents the attacker privileges gained by exploiting V_1 to V_4. Condition node C_a then allows exploitation of V_5 to V_8. The addition of C_a creates a linear growth star graph in place of the quadratic complete bipartite graph in the vulnerability representation.

D. *Unavoidable Edge Explosion Scenarios*

There are cases where the condition/vulnerability graph will still contain complete bipartite components and it is direct to see that the related condition or vulnerability graph will also exhibit such a component. Thus, such scenarios are unavoidable (with our set of analyzed representations). While we can't prove nonexistence of a linear representation here, we believe it unlikely and that we are pushing against inviolable data representational boundaries in trying to further reduce the size of the attack graph.

Consider a scenario where multiple distinct hyperedges have identical head or tail sets in either the condition- or vulnerability- oriented approach. This will naturally result in complete bipartite components in the condition/vulnerability representation; however, it is straightforward to see that such cases also produce complete bipartite graphs in the condition and vulnerability representations as well.

See, for example, the condition/vulnerability graphs in Figure 9. The leftmost panel depicts a situation in which two distinct vulnerabilities have identical head and tail sets (such as two identical vulnerabilities on two different hosts that each enable a common set of vulnerabilities on a third); the center depicts a situation in which the head sets are distinct but the tail sets are identical (perhaps granting host-specific post-conditions), and the rightmost pane depicts identical head sets with distinct tail sets (such as would be expected from host-specific pre-conditions with global post-conditions). In each case, it is straightforward to see that a path exists from each pre-condition to each post-condition, and so the resulting condition-oriented graph will be a complete bi-partite graph.

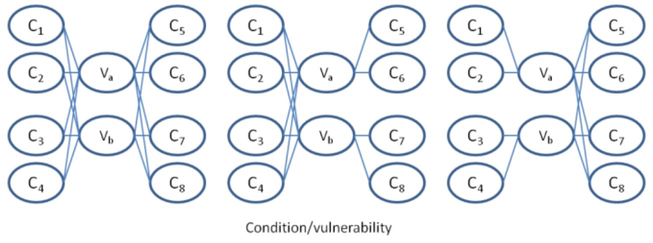

Figure 9. Condition/Vulnerability Graph Scenarios

The situation is functionally identical for vulnerability-oriented graphs. Similar complete bipartite sub-graphs within a condition/vulnerability graph will result in a completely connected bipartite sub-graphs in the related vulnerability graph.

Looking at the underlying equations, note that in the condition/vulnerability graph the number of edges is bounded as the product of *c* and *hv*, rather than being quadratic in *c*, thus requiring both *v* and *c* to grow simultaneously for a comparable edge explosion. Note that this doesn't reflect a unique weakness for the condition/vulnerability approach as both the condition and vulnerability approaches also contain *h*, *c*, and *v* in their edge equations (but there with a quadratic *c* or *v*). This worst-case scenario is only realized in the leftmost panel of Figure 9, while all three panels result in complete bipartite sub-graphs in the case of the condition-oriented graph.

V. Empirical Results

We now provide an example to illustrate the performance between the different approaches using a network model. Our network model (derived in part from data from an operational network) has 5968 hosts and 7825 vulnerabilities. The vulnerabilities consist of 41 distinct types mapped to two different severity levels. We mapped 7791 vulnerability instances to confidentiality breaches and 34 instances to providing user level access.

With respect to attack post-conditions, a vulnerability was modeled as producing two post-conditions: the severity level mapped to the host name and a designator indicating that the host had some specific vulnerability exploited. This models the situation where a single attack can produce multiple post-conditions.

With respect to connectivity, we modeled all nodes as being logically connected to each other. For a start node in the attack graph, we designated one of the hosts as hostile (using one with no vulnerabilities) to represent an insider threat situation.

Table 3 provides the empirical results given the above stated scenario. To derive these results, we created an attack graph simulator using Python 2.7.6 that calculates the graph sizes using all of our analyzed representations. Note that these results are not based on the equations from Table 2 as those equations represent worst-case attack graph sizes. Here we analyze the actual sizes given the network model described above.

TABLE 3. EMPIRICAL RESULTS

Graph Type	Nodes	Edges
Condition	5140	436 290
Exploit	218 146	7 189 929
Vulnerability	7825	272 920
Condition/Exploit	223 285	654 435
Condition/Vuln.	12 964	233 795

As expected from the theoretical analysis, the number of nodes for the exploit and condition/exploit representations was much larger than the other approaches due to the O(h^2) growth rate. The number of edges in the condition graph is almost twice that of the condition/vulnerability graph, attributable to the O(c^2) growth rate of the edges. Thus based on these empirical results, the vulnerability and condition/vulnerability approaches appear the best for our scenario and are comparable (with the vulnerability approach having fewer nodes and the condition/vulnerability approach having fewer edges).

Note how in this example our condition/vulnerability approach had 94 % fewer nodes and 64 % fewer edges than the widely cited and commonly used condition/exploit approach. This illustrates how an adjustment in representation can have dramatic results in graph size.

However, if we model each attack as producing exactly one post condition, then the advantages of the condition/vulnerability approach disappear relative to the condition graph (see Table 4).

TABLE 4. SINGLE POST CONDITION EMPIRICAL RESULTS

Graph Type	Nodes	Edges
Condition	2584	218145
Exploit	218146	7189929
Vulnerability	7825	272920
Condition/Exploit	220728	436290
Condition/Vuln.	10408	225970

Here the condition graph has an advantage on the number of nodes while roughly matching the number of edges of the conditional/vulnerability approach. Thus, use of the vulnerability/condition approach does not guarantee a smaller graph than the condition representation. However, it guarantees a linear growth rate with respect to c, allowing for tighter representations given arbitrary scenarios.

Note that the vulnerability approach statistics stay the same in both Table 3 and Table 4. This is because the removed post conditions were not ones that enabled an attack to be launched (we just removed the flag that a host had a specific vulnerability exploited).

The widely-cited and used condition/exploit model was much larger in all of our scenarios because it suffers from both the O(h^2) growth rate in the nodes (this is true also of the exploit approach). Had we modeled a network where the logical connectivity of the hosts was much more restricted, the node disadvantage of the condition/exploit approach would have been minimized. However, many operational networks (including this one) have large numbers of hosts with significant logical connectivity (e.g., approaching complete sub-graphs).

VI. CONCLUSIONS

For the last decade, the condition/exploit-oriented approach was the most commonly used representation in our literature survey. However, we found it to have node growth quadratic in the number of hosts on the network. This will slow down analysis algorithms that have a high polynomial degree while making visualization for humans more difficult (simply from the increased size). Interestingly, we found the previously published condition approach provided a much more compact linear node representation, but it wasn’t widely adopted. This may have been because it was confusing to visually analyze since attacks are represented by collections of edges. We also discovered that it suffers from quadratic edge explosions based on the number of possible attacker privileges on a host.

To address these problems, we proposed using a vulnerability-based approach for nodes in attack graphs. This eliminates the inefficiency of the attack nodes (taking us from a quadratic to a linear node representation) while it makes the graph more intuitive to read compared to the condition approach (since any attack results in compromising a single vulnerability node as opposed to activating multiple condition nodes). Surprisingly, we found this approach to also contain an edge explosion problem but this time relative to the number of vulnerabilities on a host.

We thus developed the hybrid condition/vulnerability approach with the following advantages: linear node growth, elimination of avoidable edge explosion issues, and an easy to understand representation (due to the use of the vulnerability nodes). For arbitrary graphs, our condition/vulnerability approach provides better size guarantees with respect to edge growth while only having a small linear penalty on node growth.

Despite this, the condition and vulnerability approaches are still viable representation options (linear in node growth and quadratic in edge growth). Even with the quadratic edge explosion possibilities, they can be used when it is known that a particular scenario will not suffer significantly from this problem. Perhaps the best argument for using these two approaches is simply that they have a single interpretation for the nodes. This should facilitate the application of standard graph algorithms for analysis, something not available with the currently used hybrid condition/exploit approach or our hybrid condition/vulnerability approach. Given the simple interpretation of our vulnerability approach, it is a candidate for exploration in this area, which may be addressed in future work.

Lastly and most importantly, we emphasize that the research community should move away from using attack nodes (as found in both the exploit representation and the hybrid condition/exploit representation) since the attack nodes add a quadratic factor to the worst-case node growth equations. Moving to a much more compact node linear representation (regardless of the specific choice) may catalyze the research community by opening the door to previously intractable algorithmic analyses and facilitating human analysis of specific features.

References

[1] A. Singhal and X. Ou, "Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs," National Institute of Standards and Technology Interagency Report 7788, 2011.

[2] S. Noel and S. Jajodia, "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation," in Workshop on Visualization and Data Mining for Computer Security, Fairfax, 2004, pp. 109-118.

[3] P. Ammann, D. Wijesekera and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in ACM Conference on Computer and Communications Security, Washington, D.C., 2002, pp. 217-224.

[4] S. Noel, S. Jajodia, B. O'Berry and M. Jacobs, "Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs," in Computer Security Applications Conference, Las Vegas, 2003, pp. 86-95.

[5] M. Frigault, L. Wang, A. Singhal and S. Jajodia, "Measuring Network Security Using Dynamic Bayesian Network," in Proceedings of the 4th ACM Workshop on Quality of Protection, 2008, pp. 23-30.

[6] L. Wang, T. Islam, T. Long, A. Singhal and S. Jajodia., "An Attack Graph-Based Probabalistic Security Metric," in Data and Applications Security XXII, Springer, 2008, pp. 283-296.

[7] N. Idika and B. Bhargava, "Extending Attack Graph-Based Security Metrics and Aggregating Their Application," IEEE Transactions on Dependable and Secure Computing, vol. 9, no. 1, 2012, pp. 75-85.

[8] J. Homer, X. Ou and D. Schmidt, "A Sound and Practical Approach to Quantifying Security Risk in Enterprise Networks," Kansas State University Technical Report, 2009.

[9] M. Dacier, Y. Deswarte and M. Kaaniche, "Quantitative Assessment of Operational Security: Models and Tools," LAAS Research Report 96493, 1996.

[10] I. Moskowitz and M. Kang, "An Insecurity Flow Model," in New Security Paradigms Workshop, 1997, pp. 61-74.

[11] C. Meadows, "A Respresentation of Protocol Attacks for Risk Assessment," Network Threats, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 38, 1998, pp. 1-10.

[12] C. Phillips and L. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," in Proceedings of the 1998 Workshop on New Security Paradigms, Charlottesville, 1998, pp. 71-79.

[13] R. Ortalo, Y. Deswarte and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," IEEE Transactions on Software Engineering, vol. 25, no. 5, 1999, pp. 633-650.

[14] L. Swiler, C. Phillips, D. Ellis and S. Chakerian, "Computer-Attack Graph Generation Tool," in DARPA Information Survivability Conference, Anaheim, 2001, pp. 307-321.

[15] O. Sheyner, J. Haines, S. Jha, R. Lippman and J. Wing, "Automated Generation and Analysis of Attack Graphs," in IEEE Symposium on Security and Privacy, Washington D.C., 2002, pp. 273-284.

[16] S. Jha, O. Sheyner and J. Wing, "Two Formal Analyses of Attack Graphs," in IEEE Computer Security Foundations Workshop, Cape Breton, 2002, pp. 49-63.

[17] S. Jajodia, S. Noel and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in Managing Cyber Threats: Issues, Approaches and Challenges, Kluwer Academic Publisher, 2003, pp. 247-266.

[18] S. Noel and S. Jajodia, "Measuring Security Risk of Networks Using Attack Graphs," International Journal of Next-Generation Computing, vol. 1, no. 1, 2010, pp. 135-147.

[19] J. Pamula, P. Ammann, S. Jajodia and V. Swarup, "A Weakest-Adversary Security Metric for Network Configuration Security Analysis," in Workshop on Quality of Protection, Alexandria, 2006, pp. 31-38.

[20] L. Wang, S. Noel and S. Jajodia, "Minimum-Cost Network Hardening Using Attack Graphs," Computer Communications, 2006, pp. 3812-3824.

[21] B. Schneier, "Attack trees," Dr. Dobb’s journal, 1999, pp. 21-29.

[22] B. Kordy, S. Mauw, S. Radomirović and P. Schweitzer, "Foundations of attack–defense trees," in Formal Aspects of Security and Trust, Springer , 2011, pp. 80-95.

[23] V. Gorodetski and I. Kotenko, "Attacks against computer network: Formal grammar-based framework and simulation tool," in Recent Advances in Intrusion Detection, 2002, pp. 219-238.

[24] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in 2000 IEEE Symposium on Security and Privacy, 2000, pp. 156-165.

[25] J. Dawkins and J. Hale, "A systematic approach to multi-stage network attack analysis," in Proceedings, Second IEEE International Information Assurance Workshop, 2004, pp. 48-56.

[26] N. Poolsappasit, R. Dewri and I. Ray, "Dynamic security risk management using bayesian attack graphs," IEEE Transactions on Dependable and Secure Computing, 2012, pp. 61-74.

[27] D. Koller and N. Friedman, Probabilistic graphical models: principles and techniques, MIT Press, 2009.

[28] S. J. Templeton and K. Levitt, "A requires/provides model for computer attacks," in Proceedings of the 2000 Workshop on New Security Paradigms, 2001, pp. 31-38.

[29] R. Lippmann, K. Ingols, C. Scott and K. Piwowarski, "Validating and Restoring Defense in Depth Using Attack Graphs," in Military Communications Conference, Washington, D.C., 2006, pp. 1-10.

[30] S. Nanda and N. Deo, "A Highly Scalable Model for Network Attack Identification and Path Prediction," in SoutheastCon, Richmond, 2007, pp. 663-668.