Understanding Complex Network Attack Graphs through Clustered Adjacency Matrices

Steven Noel and Sushil Jajodia
Center for Secure Information Systems, George Mason University
{snoel, jajodia}@gmu.edu

Abstract

We apply adjacency matrix clustering to network attack graphs for attack correlation, prediction, and hypothesizing. We self-multiply the clustered adjacency matrices to show attacker reachability across the network for a given number of attack steps, culminating in transitive closure for attack prediction over all possible number of steps. This reachability analysis provides a concise summary of the impact of network configuration changes on the attack graph. Using our framework, we also place intrusion alarms in the context of vulnerability-based attack graphs, so that false alarms become apparent and missed detections can be inferred. We introduce a graphical technique that shows multiple-step attacks by matching rows and columns of the clustered adjacency matrix. This allows attack impact/responses to be identified and prioritized according to the number of attack steps to victim machines, and allows attack origins to be determined. Our techniques have quadratic complexity in the size of the attack graph.

1. Introduction

The utility of organizing combinations of network attacks as graphs is well established. Traditionally, such attack graphs have been formed manually by security red teams (penetration testers). But significant progress has been made recently in generating attack graphs automatically, based on models of network security conditions and attacker exploits, created from network scans, vulnerability databases, etc. By representing dependencies among attacker exploits rather than explicitly enumerating attack states, exponential graph complexity can be avoided.

In the current state of practice, it is thus possible to efficiently compute attack graphs for realistic networks. But the resulting graphs can still pose serious challenges for human comprehension. This is compounded by the fact that attack graphs are usually communicated by literal drawings of graph vertices and edges. While graph drawing has been studied extensively, the problem is ill-posed in the sense that many possibilities exist for what constitutes a good graph drawing. Also, finding optimal placement of graph vertices according to many of the desired criteria is NP-complete. For the relatively dense attack graphs often found in practice (e.g., within a trusted internal network), graph drawing techniques are largely ineffective, producing overly cluttered drawings for graphs of larger than moderate size.

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

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

The adjacency matrix is a concise graph representation, but alone it can be insufficient. That is, without the proper ordering of matrix rows and columns, the underlying attack graph structure is not necessarily apparent. We therefore apply an information-theoretic clustering technique [1] that reorders the adjacency matrix so that blocks of similarly-connected attack graph elements emerge. The clustering technique is fully automatic, parameter-free, and scales linearly with graph size.

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

retain the reordering induced by clustering, so that patterns in the attack graph structure are still apparent.

The general approach of clustering attack graph adjacency matrices (and raising them to higher powers) provides a framework for correlating, predicting, and hypothesizing about network attacks. The approach applies to general attack graphs, regardless of what the particular graph vertices and edges represent. For example, such attack graphs could have been formed from models of network vulnerability, or from causal relationships among intrusion detection events. Attack graph vertices could also represent aggregated sub-graphs, such as aggregation by machines and exploits between them. Overall, the techniques we describe have quadratic complexity in the size of the attack graph, for scalability to larger networks.

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

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

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

In the next section, we review related work in this area. In Section 2, we describe our general approach for clustered attack graph adjacency matrices, including raising them to higher powers for multi-step reachability. Section 4 applies our approach in a number of ways for network attack protection, detection, and response. In Section 5, we summarize our work and draw conclusions.

2. Related Work

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

Still, when attack graphs are generated for realistic networks, using comprehensive sets of modeled attacker exploits, the resulting attack graphs can be very large. Previous approaches generally use graph drawing algorithms [10], in which vertices and edges between them are drawn according to particular aesthetic criteria. While large graphs have been successfully drawn, these have generally been relatively sparsely connected. But network attack graphs can be both large and exhibit very dense connectivity. For example, for only 200 machines, with each machine having 4 vulnerable services, within a trusted internal network with unrestricted connectivity, the resulting fully-connected attack graph has $(4 \times 200)^2 = 640{,}000$ edges.

An approach has been proposed for managing attack graph complexity through hierarchical aggregation [4], based on the formalism of clustered graphs [11]. The idea is to collapse subsets of the attack graph into single aggregate vertices, and allow interactive de-aggregation. A disadvantage of this approach is that lower-level details of the attack graph are hidden until they are de-aggregated, and the process of interactive de-aggregation is potentially tedious. In contrast, in our approach, all graph details are visible in a single view.

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

Our approach applies information-theoretic clustering to the attack graph adjacency matrix [1]. This clustering rearranges rows and columns of the adjacency matrix to form homogeneous groups. In this way, patterns of common connectivity within the attack graph are clear, and groups (attack graph subsets) can be considered as single units. This clustering technique is fully automatic, is free of parameters, and scales linearly with graph size.

There have been approaches that view network traffic in the form of a matrix [12][13], where rows and columns

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

The multi-step reachability matrix in our approach corresponds to the attack graph exploit distances in [3], although those distances are computed through graph traversal as opposed to our matrix multiplication. However, in the previous approach, exploit distances are not clustered or visualized; rather, they are used to correlate intrusion detection alarms. While the previous approach considers multiple steps to handle missing alerts and build attack scenarios, it does not predict attack origin and intent as in our approach.

3. General Approach

In this section, we describe our general approach for applying adjacency matrices to network attack graphs. Sub-Section 3.1 describes how adjacency matrices can be created for various types of attack graphs. In Sub-Section 3.2, we describe a matrix clustering algorithm that finds homogenous groups in the attack graph adjacency matrix. Sub-Section 3.3 then describes how the (possibly clustered) attack graph adjacency matrix can be transformed to represent multi-step attacks. Sub-Section 3.4 then describes how detected intrusions can be placed in the context of attack graph reachability matrices for predicting attack origin and impact.

3.1 Attack Graph Adjacency Matrix

Our approach begins with the creation of a network attack graph, through some means, based on some representation of network attacks. There are really no particular restrictions on the exact form of the attack graph for our approach to apply. For example, the graph could be based on hypothetical attacker exploits generated from knowledge of vulnerabilities, network connectivity, etc., as in [2][6][7][8]. Or, the graph could be constructed from causal relationships among intrusion detection system alarms, as in [5][9]. We can also handle intrusion alarms placed within the context of vulnerability-based attack graphs, as was done (implicitly) in [3].

It was pointed out in [3] that attack graphs can be created with specified starting and goal points (to constrain the graph to regions of interest), or with starting and goal points unspecified (e.g., for intrusion alarm correlation). In [4], it was pointed out that there are dual attack graph representations in which either network security conditions or attacker exploits could be the graph vertices, with the other being the graph edges. Also in [4], subgraphs of the attack graph were aggregated to single vertices. Our approach handles all of these situations.

Consider a simple example in where there is a set of network machines having no connectivity limitations among them, so that the attack graph is fully connected. For such a set of 200 machines, with just one vulnerable network service on each machine (vertex), there are 200^2 = 40,000 exploits (edges) that must be displayed. If such a graph were drawn with lines for edges, it would not be apparent from the resulting mass of lines that this indeed represents a fully connected attack graph. We therefore employ an adjacency matrix visualization, in which each attack graph edge is represented by a matrix element rather than by a drawn line. In our example of 200 fully connected machines each having one vulnerable service, the attack graph adjacency matrix would simply be a 200-square matrix of all ones.

Formally, for n vertices in the attack graph, the adjacency matrix A is an $n \times n$ matrix where element $a_{i,j}$ of A indicates the presence of an edge from vertex i to vertex j. In attack graphs, it is possible that there are multiple edges between a pair of vertices (mathematically, a multigraph), such as multiple conditions between a pair of exploits or multiple exploits between a pair of machines. In such cases, we can either record the actual number of edges, or simply record the presence (0, 1) of at least one edge. The adjacency matrix records only the presence of an edge, and not its semantics, which can be considered in follow-on analysis.

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

3.2 Adjacency Matrix Clustering

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

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

But in general, we cannot rely on *a priori* ordering of rows and columns to place the adjacency matrix into meaningful clusters. We therefore apply a particular matrix clustering algorithm [1] that is designed to form homogeneous rectangular blocks of matrix elements (row and column intersections). Here, homogeneity means that within a block, there is a similar pattern of attack graph edges (adjacency matrix elements). This clustering algorithm requires no user intervention, has no parameters that need tuning, and scales linearly with problem size.

This algorithm finds the number of row and column clusters, along with the assignment of rows and columns to those clusters, such that the clusters form regions of high and low densities. Numbers of clusters and cluster assignments provide an information-theoretic measure of cluster optimality. This is based on ideas from data compression, including the Minimum Description Length principle [14], in which regularity in the data can be used to compress it (describe it in fewer symbols). Intuitively, one can say that the more we compress the data, the better we understand it, in the sense that we have better captured its regularities.

3.3 Multi-Step Reachability

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

For a square ($n \times n$) adjacency matrix A and a positive integer p, then A^p is A raised to the power p, i.e., A multiplied by itself $p - 1$ times. Here, matrix multiplication is in the usual sense, i.e., an element of A^2 is

$$\left(A^2\right)_{ij} = \sum_k a_{ik} \cdot a_{kj} . \quad (1)$$

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

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

$$A \vee A^2 \vee A^3 \vee \ldots \vee A^{n-1} \quad (2)$$

tells us, for each pair of vertices, whether the attacker can reach one attack graph vertex to another over all possible numbers of steps. This Boolean sum is known as the transitive closure of A. The classical Floyd-Warshall algorithm computes transitive closure in $O(n^3)$, although there are improved algorithms, e.g., [15], that come closer to $O(n^2)$. Frequently in practice, elements of A^p monotonically increase as p increases. In such cases, we can distinguish the minimum number of steps required to reach each pair of attack graph vertices by computing the multi-step reachability matrix

$$A + A^2 + A^3 + \ldots + A^{n-1}, \quad (3)$$

where the matrix multiplication is Boolean and the summation is simply arithmetic. Since elements of A^p increase monotonically from zero to one (under Boolean matrix multiplication), the elements of the reachability matrix in Equation (3) give the minimum number of steps required to reach one attack graph vertex to another.

A fundamental property of attack graphs is how well connected the various graph vertices (exploits, machines, etc.) are. For example, attack graphs that have few or weak (large multi-step only) connections are easier to defend against, and those with more and stronger connections are more difficult to defend against. Knowing the numbers and depths of attacks (e.g., through higher powers of the adjacency matrix) helps us understand large-scale tendencies across the network. Individual vertices' roles within the attack graph are also described by their numbers and depths of attacks to other vertices. For example, vertices (e.g., machines) with many attack paths through them might bear closer scrutiny. Or, critical vertices could be identified as "bottlenecks" in the attack graph.

3.4 Attack Prediction

In our approach, one can place detected intrusions within the context of vulnerability-based attack graphs. We first compute a vulnerability-based attack graph from knowledge of the network configuration, attacker exploits, etc. We then form the adjacency matrix A for the attack graph, perform clustering on A, and compute

either the transitive closure of A or the multi-step reachability matrix in Equation (3). Then, when an intrusion alarm is generated, if we can associate it with an edge (e.g., exploit) in the attack graph, we can thus associate it with the corresponding element of any of the following:

1. The adjacency matrix A (for single-step reachability)
2. The multi-step reachability matrix in Equation (3) (for multi-step reachability)
3. The transitive closure of A (for all-step reachability)

From this, we can immediately categorize alerts based on the numbers of associated attack steps. For example, if an alarm occurs within a zero-valued region of the transitive closure, we might conclude it is a false alarm, i.e., we know it is not possible according to the attack graph. Or, if an alarm occurs within a single-step region of the reachability matrix, we know that it is indeed one of the single-step attacks in the attack graph. Somewhere in between, if an alarm occurs in a p-step region, we know the attack graph predicts that it takes a minimum of p steps to achieve such an attack.

By associating intrusion alarms with a reachability graph, we can also predict the origin and impact of attacks. That is, once we place intrusion alarm on one of the vulnerability-based reachability graphs, we can navigate the graph to do attack prediction. The idea is to project to the main diagonal of the graph, in which row and column indices are equal. Vertical projection (along a column) leads to attack step(s) in the forward direction. That is, when one projects along a column to the main diagonal, the resulting row gives the possible steps forward in the attack. We can predict attack origin and impact either (1) one step away, (2) multiple steps away with the number of steps distinguished, or (3) over all steps combined. Here are those 3 possibilities:

1. When using the adjacency matrix A, non-zero elements along the projected row show all possible single steps forward. Projection also can be done iteratively, to follow step-by-step (one at a time) in the attack.
2. When using the multi-step reachability matrix in Equation (3), the projected row shows the minimum number of subsequent steps needed to reach another vertex. One can also iteratively project, either choosing single-step elements only, or "skipping" steps by choosing multi-step elements.
3. When using the transitive closure, the projected row shows whether a particular vertex can be subsequently reached in any number of steps. Here, iterative projection is not necessary, since transitive closure shows reachability

From the preceding discussion, we see that projection along a column of a reachability matrix predicts the impact (possible steps forward) of an attack. Correspondingly, we can project along a row (as opposed to a column) of such a matrix to predict attack origin (possible steps backward). In this case, when one projects along a row to the main diagonal, the resulting column gives the possible steps backward in the attack. As before, we can predict attack origin using either (1) the adjacency matrix, (2) the multi-step reachability matrix, or (3) the transitive closure. Just as for forward projection, this gives either (1) single-step reachability, (2) multi-step reachability, or (3) all-step reachability, but this time in a backward direction for predicting attack origin.

4. Applications

In this section, we apply our general approach to a number of different network security situations. Sub-Section 4.1 shows the application of clustered adjacency and reachability matrices to a vulnerability-based attack graph. In Sub-Section 4.2, we show how reachability matrices give a concise summary of attack graph changes driven by changes to the network configuration. Sub-Section 4.4 shows how our approach can help categorize intrusion alarms and predict attack origin and impact.

4.1 Vulnerability-Based Attack Graphs

Figure 1 shows an example drawn attack graph, in which low-level security conditions have been aggregated to machine vertices, and exploits have been aggregated for pairs of machines. This attack graph was generated from a network model created from Nessus [16] vulnerability scans. Despite the fact that this attack graph has been aggregated to the level of machines, the drawn graph is still cluttered with edges and is hard to follow. Some clustering is apparent in this drawing, but the exact nature of the clusters, such as their boundaries and cross-cluster relationships, is not readily apparent.

Figure 2 shows the same attack graph as Figure 1, this time represented as an adjacency matrix. In the matrix, rows represent exploits from a particular machine, and columns represent exploits to a particular machine. The presence at least one exploit between a pair of machines is indicated by black matrix element, and the absence is indicated by white. Here, there is no *a priori* way of ordering the matrix rows and columns to form meaningful clusters. That is, from the given ordering, the underlying structure of the attack graph is obscured.

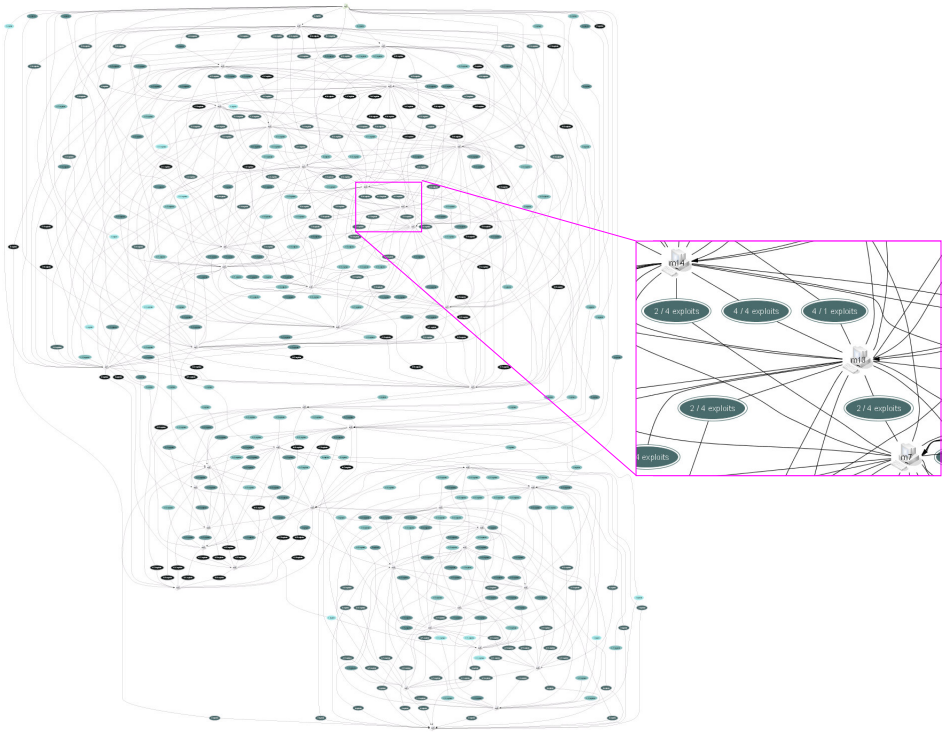

Figure 1: Example drawn attack graph.

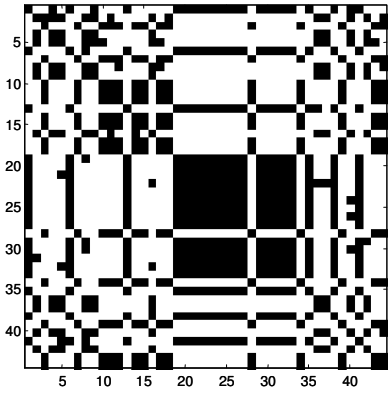

Figure 2: Unclustered adjacency matrix for attack graph in Figure 1.

In Figure 3, we have clustered the attack graph adjacency matrix in Figure 2. The underlying structure of the attack graph is now clear. The clustering algorithm has identified 9 clusters (rectangular blocks) of homogeneous graph edges. The 3 blocks on the main diagonal (blocks $A_{1,1}$, $A_{2,2}$, and $A_{3,3}$) are solid black, indicating full attack graph connectivity within each block. That is, within one of these main-diagonal blocks, every machine can attack every other machine (through at least one exploit). Thus in the terminology of [4], these blocks constitute protection domains, which have been detected automatically by the clustering algorithm. Block $A_{2,3}$ of the clustered adjacency matrix shows exploits launched from the 2nd to 3rd protection domains (from block $A_{2,2}$ to $A_{3,3}$). Similarly, block $A_{3,1}$ shows exploits from the 3rd to 1st protection domains (from block $A_{3,3}$ to $A_{1,1}$). Note that Figure 3 shows how matrix rows and columns were reordered by the clustering algorithm, i.e., rows and columns are labeled with their original indices. In practice, we might use more meaningful labels, such as IP addresses.

Figure 4 shows the square of the clustered attack graph adjacency matrix in Figure 3. Here we have used the arithmetic product (as opposed to the Boolean product). This shows not only whether there exists at least one 2-step attack from one machine to another, but also the actual count of all possible 2-step attacks. For example, we see that within the 2nd protection domain (block $A_{2,2}$), there are 20 possible 2-step attacks between each pair of machines, corresponding to the 20 machines in that protection domain. We see corresponding numbers of 2-step attacks within the other 2 protection domains (blocks $A_{1,1}$ and $A_{3,3}$). There are relatively fewer 2-step attacks across protection domains (blocks $A_{2,3}$ and $A_{3,1}$).

Figure 5 shows successive powers (A^2, A^3, and A^4) of the clustered adjacency matrix, this time employing

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

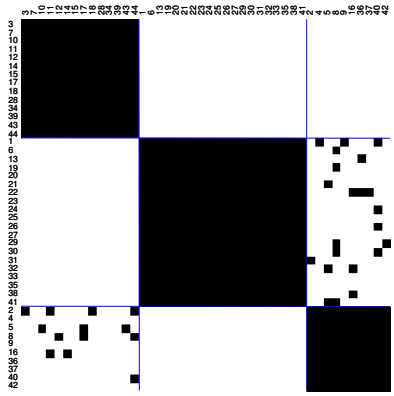

Figure 3: Clustered adjacency matrix for attack graph in Figure 1.

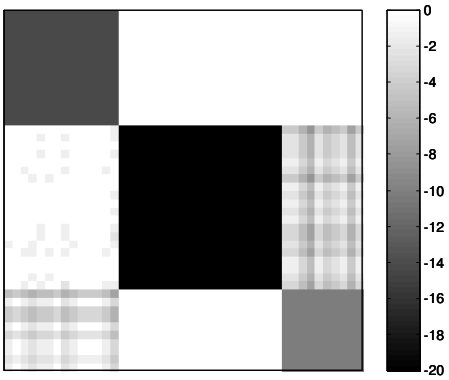

Figure 4: Clustered adjacency matrix for attack graph in Figure 1, raised to 2nd power for 2-step attacks.

In Figure 6, we have combined the reachability matrices in Figure 5 (and the adjacency matrix in Figure 3) into a single matrix, as defined by Equation (3). All the information in the separate per-step reachability matrices can now be seen together. This multi-step reachability matrix is used again in Section 4.4 for attack origin and impact prediction.

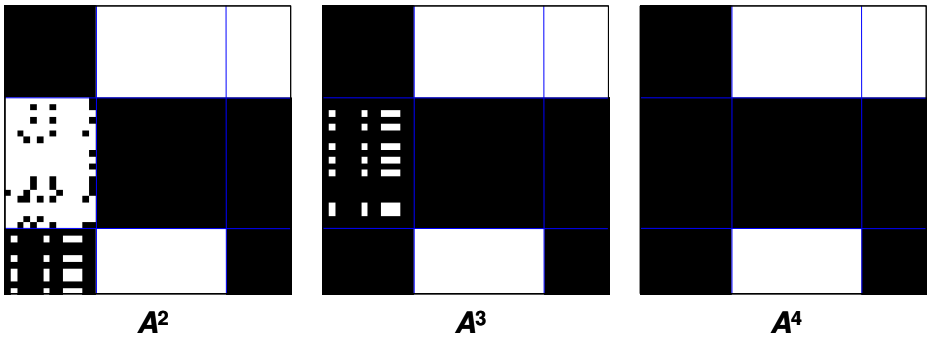

Figure 5: Reachability for 2, 3, and 4 steps for attack graph in Figure 1.

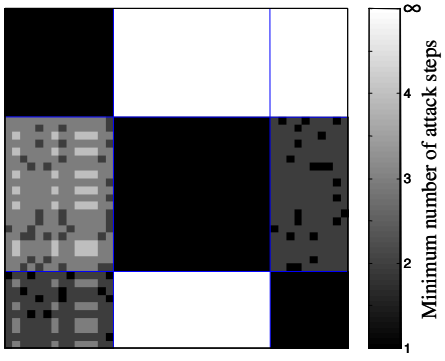

Figure 6: Multi-step reachability for attack graph in Figure 1.

4.2 Reachability for Attack Graph Changes

Figure 7 shows an attack graph adjacency matrix for a network of 730 machines. While we do not show it, the corresponding drawn attack graph would be exceedingly cluttered and difficult to understand. Again, we have aggregated low-level security conditions to machines and sets of exploits between them, so that rows and columns are machines, and non-zero entries mean at least one exploit between a pair of machines. Here, *a priori* ordering of machines (rows and columns) is sufficient, so that we do not apply matrix clustering. In the figure, black indicates the attack graph for a baseline network

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

Figure 8 shows the resulting transitive closure (all-step reachability) for the baseline (black) and changed (black+orange) network. Here we see that before deployment of the web servers (black), attacks are only possible within the main diagonal blocks, from block $A_{2,2}$ to blocks $A_{3,3}$ and $A_{4,4}$, and from block $A_{3,3}$ to block $A_{4,4}$. But after deployment of the web server (black+orange), machines in block $A_{1,1}$ can reach all machines in the network, and all machines in the network can reach the machines in blocks $A_{2,2}$, $A_{3,3}$, $A_{4,4}$, and $A_{5,5}$. That is, only machines in block $A_{1,1}$ are safe from attacks outside their block.

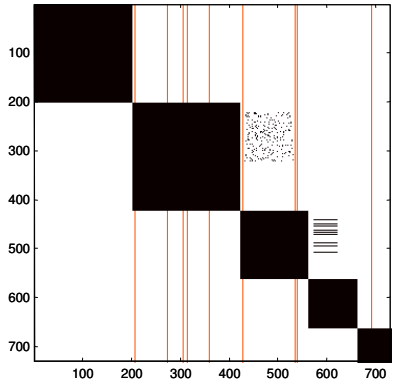

Figure 7: Attack graph adjacency matrix for baseline (black) and changed (orange) network.

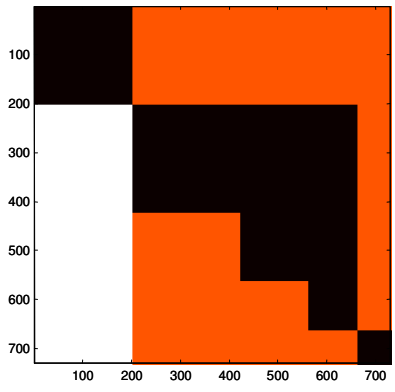

Figure 8: Transitive closure for Figure 7, for baseline (black) and changed (black+orange) network.

4.3 Prediction of Attack Origin and Impact

Figure 9 shows the multi-step vulnerability-based reachability matrix from Figure 6, but this time with intrusion alarms associated with elements of the matrix. Thus, knowing attack reachability across the network, we can categorize and correlate detected intrusions in terms of it. In Figure 9, an intrusion alarm (in yellow) occurs between a pair of machines that are known to be unreachable from one to the other. In this case, we might consider it to be a false alarm. Similarly, an intrusion alarm (in purple) occurs between a pair of machines, which, according to the attack graph, should require at least 4 attack steps to reach from one to the other. If we were to rely on traditional attack graph drawings, we would need to trace many edges before coming to this conclusion.

There are 2 other intrusion alarms in Figure 9 that according to the attack graph, are each possible one-step attacks. In fact, if we project (along a column) from Alarm 1 to the main diagonal, we find Alarm 2 on the projected row, indicating that according to the attack graph, Alarm 2 follows immediately after Alarm 1. In this case, we correlate the two alarms based on the likelihood that they are part of a coordinated attack.

Figure 10 shows another intrusion alarm associated with the multi-step reachability matrix from Figure 6. This time, we project to the main diagonal in each direction, i.e., along columns and rows, to explore forward and backward steps from this alarm. In this way, we can predict the origin of the attack (from the backward direction) and the impact of the attack (from the forward direction). Projecting along the row to the main diagonal, we reach a column that has non-zero entries only within the 2nd main-diagonal block (block $A_{2,2}$). In fact, these non-zero entries are all of unity value, indicating that these are all one-step attacks. In other words, only an attack from one of the machines in block $A_{2,2}$ could have led to the detected intrusion, and that could have happened within one attack step.

In Figure 10, when we project the detected intrusion to the main diagonal along its column, the row of intersection shows all possible forward steps from the detected event. In this case, we see that machines in the 2nd block cannot be reached by the attacker, since block $A_{3,2}$ is zero-valued. For this reason, no attack response is necessary for defending machines in these columns. However, because block $A_{3,1}$ shows reachability within 2 to 3 steps, measures might be taken to defend machines in these columns, although not at the highest priority. The highest priority should be for machines in the columns of block $A_{3,3}$, because those machines are all reachable from the detected event within a single step.

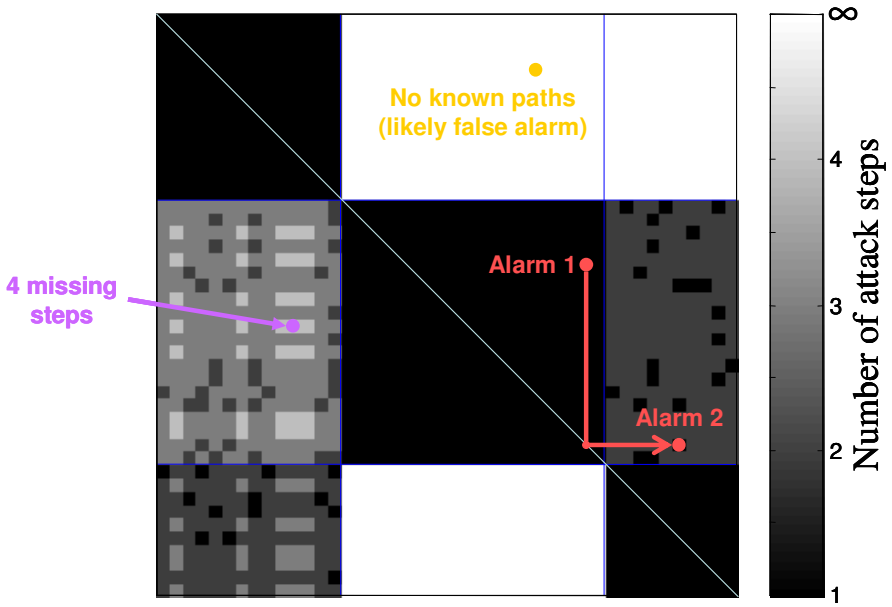

Figure 9: Categorizing and correlating intrusion alarms via attack graph reachability.

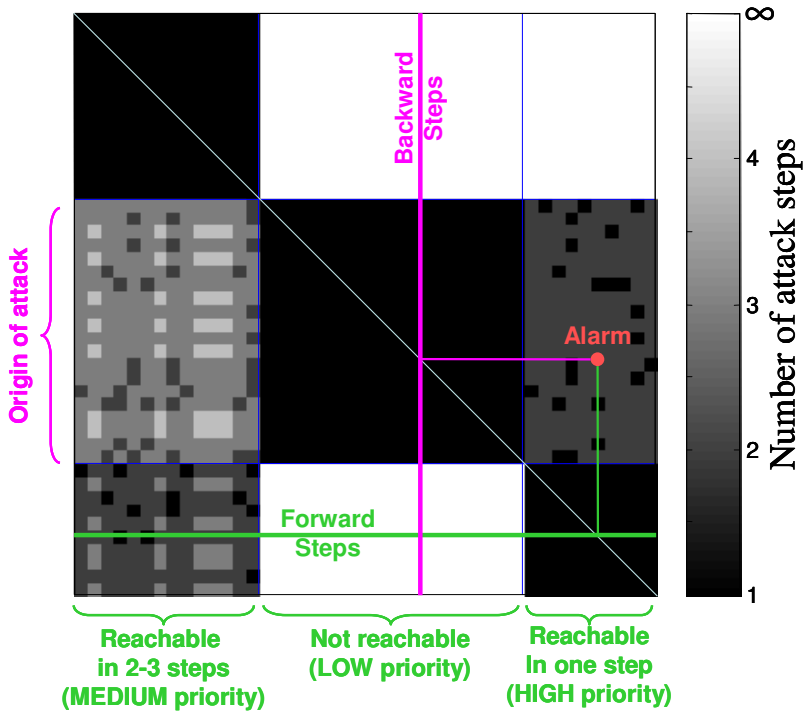

Figure 10: Predicting attack origin and impact.

5. Summary and Conclusions

This paper shows how clustered adjacency matrices reveal the underlying regularities in network attack graphs. Our approach is particularly attractive because it avoids the edge clutter usually associated with literal drawings of attack graphs. The adjacency matrix is concise, representing each graph edge with a single matrix element. Our approach places no particular

restrictions on the form of the attack graph. It therefore applies to attack graphs based on network vulnerabilities, detected intrusions, or combinations thereof, and well as attack graphs with aggregated vertices, e.g., aggregated by network machine. Our approach has low-order polynomial complexity overall, for scalability to larger networks.

The information-theoretic clustering algorithm we apply reorders rows and columns of the adjacency matrix so that rectangular blocks of similarly-connected attack graph elements emerge. This clustering algorithm is fully automatic, parameter-free, and scales linearly with problem size. We further transform the attack graph adjacency matrix by raising it to higher powers, to represent multiple attack steps. We can thus show attacker reachability across the network within any number of attack steps. We combine these per-step reachability matrices into a single matrix that shows the minimum number of steps between any pair of vertices in the attack graph. We also summarize reachability over all number of steps via transitive closure.

Through our general approach, we are able to correlate, predict, and hypothesize about network attacks. For example, we can provide a concise summary of changes in an attack graph resulting from changes in the network configuration. We can place intrusion alarms in the context of the vulnerability-based attack graph for categorizing alarms. We can step forward from an attack, to predict its impact and prioritize defensive responses according to the number of steps required to reach victim machines. We can also step backward from an attack, to predict its origin.

6. Acknowledgements

This work was partially supported by the Air Force Research Laboratory Rome under grant F30602-00-2-0512 and by the Army Research Office under grants DAAD19-03-1-0257 and W911NF-05-1-0374.

7. References

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

[2] 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.

[3] 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.

[4] 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, VA, October 2004.

[5] 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.

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

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

[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] 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.

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

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

[12] 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.

[13] 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.

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

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

[16] Nessus open source vulnerability scanner project, web page, http://www.nessus.org/, last accessed May 2005.