EVA: A Framework for Network Analysis and Risk Assessment

Melissa Danforth
Department of Computer Science
California State University, Bakersfield
Bakersfield, CA 93311
mdanforth@csub.edu

Tags: security, research, attack graphs

Abstract

EVA is an attack graph tool that allows an administrator to assess and analyze a network in a variety of fashions. Unlike other attack graph tools which just focus on visualizing the network or recommending a set of patches to secure the network, EVA goes beyond these modes to fully explore the power of attack graphs for a multitude of administrative and security tasks. EVA can be used to derive a set of hardening measures for a network, to perform strategic analysis of a network, to design a more secure network architecture, to assist in forensic evaluations after a security event and to augment an intrusion detect system with information about the likely targets of an attack. This paper summarizes the framework used by EVA, provides real-world results of using EVA and shows how EVA is scalable to large networks.

1 Introduction

Securing a computer network against intrusion is a complicated task. The risk profile of the network depends not only on the configuration of individual machines, but also on the connectivity between machines. If an administrator only evaluates the risk profile of each machine individually, he will miss multi-stage attacks that propagate across the network. For example, an attacker might compromise a public web server and then use that server to compromise the database server. This is a classic "foothold" scenario whereby the attacker compromises one machine to use as a base for gaining access to internal networks he could not directly access. Such scenarios must be considered when evaluating a network.

Attack graphs [2, 4, 10, 11, 14, 15, 16, 17, 18, 19, 9, 21] and attack trees [6, 7] provide a method to discover and visualize such "foothold" scenarios in the network. Attack graphs and trees compute exploit paths that a theoretical attacker might take through the network, given knowledge of the vulnerabilities on each machine, the firewall rules in the network and the topology of the network. Attack graphs by themselves are purely just a method to represent and possibly visualize these paths. The true power of attack graphs lays in analyzing the attack graph.

EVA (Evolutionary Vulnerability Analysis) is an attack graph tool that supports a multitude of analysis modes. As shown in [5], it is scalable to large networks containing hundreds of hosts. This paper describes further improvements that increase the scalability to networks containing thousands of hosts. EVA is a policy driven model, which allows administrators to tune the analysis to the specific operating criteria or mission for their networks. The policy model is flexible so that the administrator does not need to provide extensive information to it.

Most prior work has focused on two modes of analysis: finding a set of hardening measures and performing "what if" scenarios. A set of hardening measures are typically patches or firewall rules that prevent the attacker from achieving one or more goals. The "what if" scenarios allow the administrator to pretend there are unknown vulnerabilities in the network. This allows an administrator to explore the consequences of unknown vulnerabilities, such as "zero-day" exploits. The "what if" mode essentially alters the input into the attack graph tool to support the scenario instead of the actual network. The resulting "what if" attack graph that can be analyzed using other modes of analysis. EVA supports these modes of analysis and uses the policy to guide the analysis.

EVA goes beyond these modes of analysis to further unlock the power of the attack graph model. It can also be used for network design, forensic evaluation and IDS monitoring. For the network design mode, the tool can be used in two ways. First, it can be given multiple prototype networks to evaluate and decide which has the best

security. The mode of analysis has also been used in GARNET [21]. The second use of the tool for network design is unique to EVA. Given a prototype network, it can automatically alter the connectivity and/or add IDS sensors to improve the security of the network. As with hardening measures, this analysis is guided by the policy for the network.

For forensic analysis, the evidence gathered during the course of the investigation is given to the tool. The tool then produces a list of resources that the attacker could have also compromised given the evidence. This gives direction to the forensic evaluators by pointing out likely paths the attacker took during the compromise. IDS monitoring uses a similar approach, but in real-time as opposed to after-the-fact. Theoretically, the list of potential exploit paths could be given to a intrusion response system to prevent the attacker from actually exploiting those paths.

These analysis modes have not been explored in other attack graph tools. This work describes how EVA can be expanded to supporting these new analysis modes. By supporting these modes, EVA has a much wider use than simply visualizing or securing the network. It can be used in multiple phases of operation for a variety of security purposes.

Section 2 describes prior works in attack graphs and attack trees. This section highlights how EVA differs from these prior works. Section 3 details the attack graph model used by EVA. In Section 4, the methodology used to generate the attack graphs is given. Section 5 describes the genetic algorithm used for analyzing attack graphs. Section 5 also details the policy model and the various modes of analysis. Section 6 provides some experimental results of using EVA on our student lab network and on simulated networks. Section 7 talks about future work to improve this tool.

2 Related Work

Several prior works [11, 15, 17, 18] have shown that determining a set of hardening measures is in NP. Philips and Swiler [15] also shows that the problem of placing sensors to maximize coverage of the exploit paths an attacker could take is in NP as well. Given this, most prior works have focused on non-adaptive approximation methods to find a set of hardening measures.

Philips and Swiler [15] allow an administrator to secure one resource at a time by computing shortest paths to that resource. This does not actually provide a set of hardening measures, but instead trims the attack graph to just the most likely paths an attacker would take. Their method requires extensive administrator interaction to actually determine the hardening measures and to secure all the resources on the network.

Other groups have proposed non-adaptive approximation methods to derive a set of hardening measures. Noel, *et al.* [10, 14] derive an algebraic expression of the initial conditions that allow an attacker to compromise a single resource. Sheyner, *et al.* [11, 17, 18] use a greedy algorithm to protect a given resource. Ammann, *et al.* [2] compute the hardening measures for a single resource based on information added to each node during the attack graph generation. These methods only compute the set of hardening measures for a single "goal" at a time. They must be repeated for each resource the administrator wishes to protect. This requires not only more processing time, but most likely will result in repeating computational steps when two resources share a portion of their exploit paths. EVA on the other hand derives a set of hardening measures to protect all the resources the administrator has marked as critical.

Dewri, *et al.* [7] uses a genetic algorithm to compute a set of hardening measures for one or more resources. Their algorithm also supports each hardening measure having a different cost. This is similar to the approach used by EVA, but there are several critical differences, as detailed in [5]. First, their cost model is not very flexible. It requires the administrator to assign a cost and weight for every single possible hardening measure. Since the number of hardening measures increases dramatically as the size of the network increases, Dewri's method would require extensive user input before being able to compute the set of hardening measures for larger networks. EVA uses a default cost for most hardening measures, but allows the administrator to adjust the cost for any hardening measure. The administrator also has flexibility in this adjustment. One can adjust a measure globally, such as "do not allow port 80 to be disabled", or one can adjust a measure for a specific machine. Thus, the administrator only has to specify costs for those measures deemed desirable or undesirable for the network.

Second, the genetic algorithm used in [7] is not very scalable to large networks, as shown in [5]. This is because they use a multi-objective genetic algorithm that treats the security provided by the set of hardening measures and the cost of that set as equals. As shown in [5], this leads to their algorithm maintaining a set of low cost but also low security hardening measures. Most of these low cost solutions turn out to be evolutionary dead-ends because they provided very little security. By maintaining them, the genetic algorithm in [7] is essentially wasting memory and computational time on untenable solutions. The genetic algorithm used by EVA uses a priority based method which first prioritizes on securing the network and then looks at minimizing the cost of the set of hardening measures. The experimental results shown in [5] show that this is a far more suitable approach for the attack graph problem.

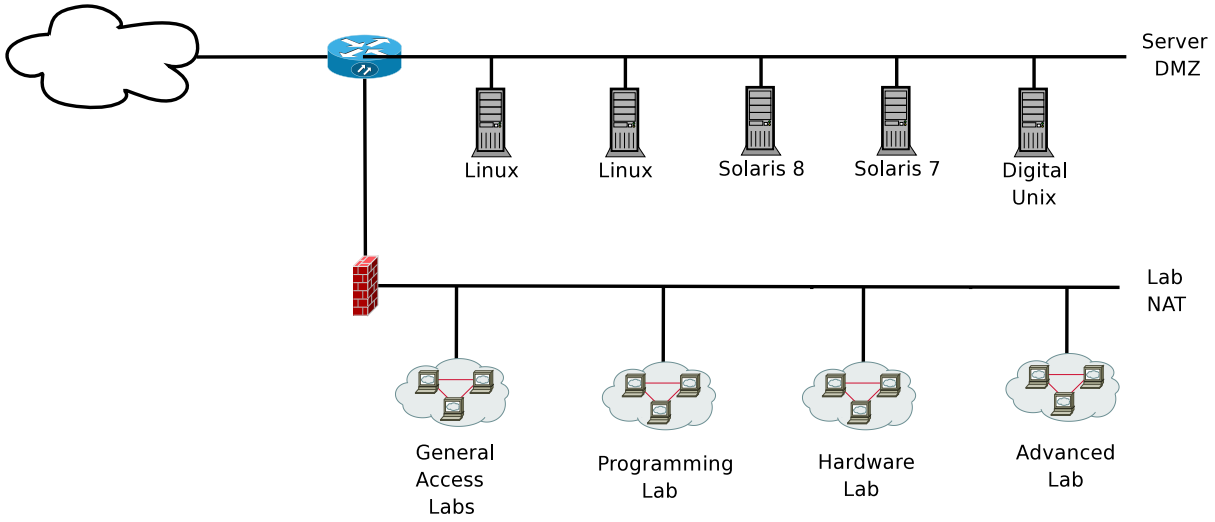


Figure 1: The Computer Science instructional network that was scanned for modeling in EVA.

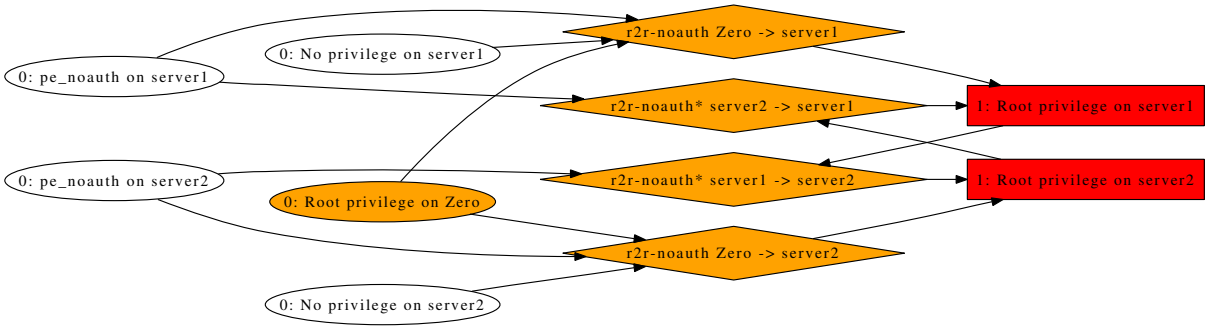
NetSPA [9] and its graphical front-end GARNET [21] are the closest competitors to EVA in the market today. NetSPA is a project out of MIT Lincoln Labs that was awarded $10k in MIT's 2008 Entrepreneurship Competition to form a startup company based around NetSPA called CyberAnalytix [13]. While NetSPA is similar to EVA, there are several key differences between NetSPA and EVA. First, NetSPA uses a different technical approach to the attack graph problem than EVA. NetSPA focuses on the data structure of the attack graph and post-processing the attack graph to reduce complexity. EVA uses a classic adjacency-list data structure for the attack graph and focuses on pre-processing the network using an abstract exploit model described in Section 3.1 and a meta-machine model described in Section 3.2 to reduce the complexity of the network. Both approaches provide scalability, but are fundamentally different in nature. Second, NetSPA uses a non-adaptive algorithm to compute the set of hardening measures while EVA uses an adaptive genetic algorithm that incorporates the site's policy when computing the set of hardening measures. By incorporating the policy, EVA is able to provide recommendations tuned to the site's mission or operating criteria. Third, NetSPA and GARNET focus on providing a set of hardening measures and visualizing the network for both actual networks and theoretical ("what if") scenarios. EVA supports these modes and also adds modes for network design, forensic evaluation and IDS monitoring. This gives EVA more versatility.

3 Attack Graph Model

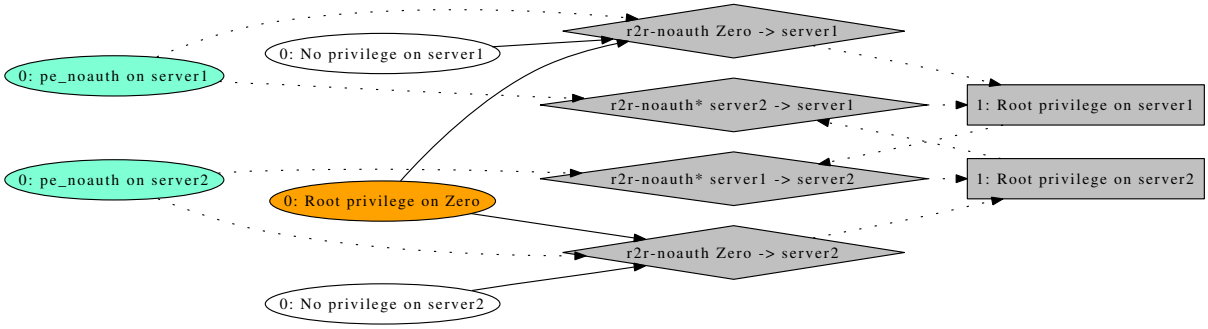
The attack graph model used by EVA was first described in [4]. The attack graph itself is an adjacency-list matrix that describes the exploit paths an attacker could take through the network. The inner nodes of the graph represent various states the attacker has achieved, such as "user privilege on $host_5$". The initial nodes of the graph represent the initial state of the network, such as what vulnerabilities are present and what privileges the attacker has initially. The edges of the graph represent exploits the attacker has executed. An attack graph for the network evaluated in Section 6 is shown in Figure 2.

The primary underpinning of the model is a set of exploit templates that describe exploits an attacker could use in the network. These templates are represented in a "requires/provides" [12] format. The "requires" portion of the template specifies what conditions must exist for the exploit to occur. The "provides" portion of the template states the consequences of the exploit, such as new privileges the attacker gains from the exploit. An attack graph is built by matching templates to the current knowledge about the network. When all of the "require" conditions are met for a template, it is executed and all of its "provide" conditions are added to the attack graph. For purposes of the representation, each condition is tied to an individual node in the attack graph.

The initial nodes of the attack graph are derived from several sources: a model of the network connectivity, a list of vulnerabilities present on all machines in the network and an attacker model. The model of the network connectivity describes the firewall and/or routing rules that would prevent two hosts from communicating with one another. By default, EVA assumes two hosts can



(a) Unpatched



(b) Patched

Figure 2: The attack graph for the basic network configuration and the patched attack graph after analyzing it. The color scheme for the graphs is as follows. The orange oval is the attacker's starting point. Red boxes represent machines where the attacker has obtained root privileges. Yellow boxes are machines where the attacker has gained user privileges. Orange diamonds are the attacks executed against the network. Clear ovals are the initial conditions in the network. In the patched graph, disabled attacks and nodes are grey while the patched vulnerabilities are aqua. For this network, the recommended patches prevent the attacker from getting root on both machines, since both root nodes have been disabled in the patched graph.

communicate on a given port. The connectivity model only needs to specify denied connectivity. When referring to TCP connections, the denied connectivity is assumed to be a denied SYN packet. The denied connectivity is directional, just as firewall rules are. For example, if you deny $host_1$ from connecting to port 443 on $host_4$, this does not prevent $host_4$ from connecting to port 443 on $host_1$. This connectivity matrix can be derived from knowledge of the network's firewall and routing rules.

The list of vulnerabilities on all machines in the network can be obtained via vulnerability scanner reports. The vulnerability name must match the naming structure used in the exploit templates. Currently, EVA can translate certain Nessus [20] plugin IDs to a vulnerability name. The machines must also be given unique names in this list and these names must match the names used in the connectivity model. IP addresses or domain names are a very logical name to use for this model. One can also rename machines in both the vulnerability list and connectivity model to any name of the administrator's choosing.

The attacker model describes what initial privileges the attacker has in the network and where the attacker is located in the network. For example, one can model an attacker that is outside of the network and who has no initial privileges in the network (the "outsider" problem). One can also model an attacker who has a machine inside the network under his control or who has certain privileges inside the network (the "insider" problem). One can also use any combination of these two problems, although currently EVA assumes a single-attacker model, so it cannot distinguish between the nodes achieved by two or more attackers. In other words, if you model both an insider and outsider, EVA assumes them to be the same person. Allowing multiple attackers is a planned future refinement (see Section 7).

One of the issues with attack graphs that was described in more detail in [4] is that the number of edges in the graph, in other words the number of exploits executed by the attacker, is dominated by the number of exploit templates in the model and the number of machines in the network. If a is the number of exploit templates and n is the number of machines, then the number of edges is $O(an^2)$. To achieve scalability, one must reduce the number of exploit templates and the number of machines in the network in such a way that it does not affect the functionality of the attack graph. To do this, EVA uses two approaches: an abstract model of exploit classes and clustering of identical machines.

3.1 Abstract Model of Exploit Classes

When looking at the early literature on attack graphs, particularly the work of Sheyner, *et al.* [11, 17, 18], two things became clear. First, if one were to model each and every exploit that existed in the world, the exploit templates would quickly grow to an enormous size. Second, many exploits shared characteristics and only varied by the name of the vulnerability and/or the port number used in the exploit. One could greatly reduce the number of exploit templates required by coming up with abstracted templates that apply to a variety of actual exploits.

The difficulty with this approach is creating abstract templates that retain the ability to model different types of exploits while still grouping multiple exploits together. Essentially, a classification system had to be developed for exploits. The details of this classification system are given in [4]. In brief, exploit classes such as "remote to user" or "remote to root" were developed. Most of the classes focus on privilege escalations, client-side privilege escalations (such as a browser exploit), username/password guessing, password cracking, information leaks, bypassing firewall rules or altering router rules. The model currently does not support denial of service, but it could be extended to do so by writing a new set of rules for that class.

Rewriting the exploit templates is only part of the abstraction process. The vulnerability list also must be translated from actual vulnerabilities to abstract vulnerabilities. This is done currently by comparing the Nessus [20] plugin ID to a mapping that converts known Nessus plugin IDs to their corresponding abstract vulnerability. This mapping is currently maintained by hand. The translation of the vulnerability list is done during the pre-processing stage, before generating the graph. For each machine in the vulnerability list, its set of vulnerabilities are translated to the abstract vulnerability class. If two or more vulnerabilities for that machine map to the same abstract class, the duplicates are discarded. When post-processing the reports generated by the analysis tool, this process is reversed.

Likewise, the port numbers given in the model of network connectivity must also be abstracted. This is a slightly more complex process, since any given port may be used for more than one abstract exploit class. Again, a mapping of port number to abstract port name is used, except this mapping supports one-to-many mappings where one port number might be associated with several abstract exploit classes.

There are two major advantages to having an abstract model for the exploit templates. The first advantage is that this greatly reduces the size of the template set. By reducing the number of templates, the number of edges in the graph are also reduced, as detailed above. This increases the scalability of the model since, as described in [4], the number of edges are a prime indicator of the complexity of the attack graph. The second advantage is reduced administrative overhead. One does not have

to alter the exploit templates every time a new exploit comes to light. Instead, the administrator can see if that exploit is part of an existing abstract class. If so, the preprocessing mappings can be altered to support this new exploit. If not, the model allows an administrator to write templates for specific exploits that are not covered by the abstract templates.

3.2 Clustering

The second approach to reduce complexity and increase scalability is to group identical machines into a cluster. In [4], this cluster was modeled as one meta-machine. This has been updated to model each cluster as two machines, so that the interactions between machines in a cluster can be observed.

The process of clustering is similar to what was described in [4]. After the connectivity model and list of vulnerabilities has been pre-processed for the abstract template model, it is further pre-processed to discover the clusters. On the first pass, all machines with identical vulnerabilities are put into a proto-cluster. On the second pass, each proto-cluster is subdivided into the final clusters based on the connectivity. Each final cluster contains machines with identical vulnerabilities and identical connectivity. Each cluster is assigned a name and the members of that cluster are recorded. Then the vulnerability list and connectivity model are updated as follows. If a cluster contains only one machine, that machine is left as-is in both the vulnerability list and the connectivity model. If a cluster contains two or more machines, all machines in the cluster are removed from both the vulnerability list and connectivity model. Then two machines whose names are based on the cluster name are added to both the connectivity model and vulnerability list. These two cluster machines have all the vulnerabilities and connectivity rules specified by the original machines in the cluster. Clustering is currently done with a Perl script to parse and alter the input files.

For a network which has large segments of identical machines, clustering can greatly improve the performance of EVA by reducing the number of machines modeled in the attack graph. Since the members of the cluster are recorded, it is easy in post-processing to augment all reports about a cluster with the list of machines in that cluster. The administrator can then tell that hardening measures need to be applied to all machines in the cluster.

4 Generation of Graphs

As described in Section 3, the exploit templates are in a "requires/provides" format. This makes them well-suited to be encoded as rules in an expert system. The expert system JESS [8] is used by EVA. The abstract exploit templates are encoded as rules in the expert system. These rules use the CLIPS [1] syntax, so the ruleset could be exported to other expert systems that support this syntax. The network connectivity model, the list of vulnerabilities and the attacker model are encoded as initial facts to the expert system. From these initial facts, the "requires" portion of zero or more templates is satisfied. The "provides" portion of the template asserts more facts into the expert system. This in turn may satisfy other templates.

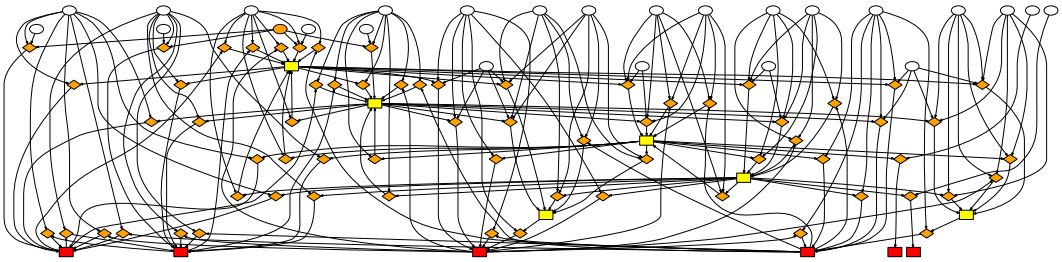
Unlike some prior works [11, 17, 18, 16] which only see if the attacker can achieve a specific goal, such as "get root on the web server", EVA uses an exploratory approach to seek out all possible exploit paths the attacker could take through the network. The matching of facts to exploit templates continues until the newly asserted facts cause no more templates to be satisfied. Thus all avenues of attacks that can be described given the initial facts and the exploit templates are explored.

The expert system also records each exploit template rule that is activated, the facts that caused it to be satisfied and the facts that are asserted as a consequence of it being activated. This is equivalent to one edge in the attack graph. The nodes in the attack graph are equivalent to the facts in the expert system, which are also recorded. A Perl script translates the output of the expert system into two formats: a visualization format and the genetic algorithm format. The visualization format uses the DOT syntax of the Graphviz project [3]. From DOT, one can produce images in a variety of formats such as EPS and GIF. The genetic algorithm format is a list of annotated edges used to construct the adjacency-list matrix for analysis.

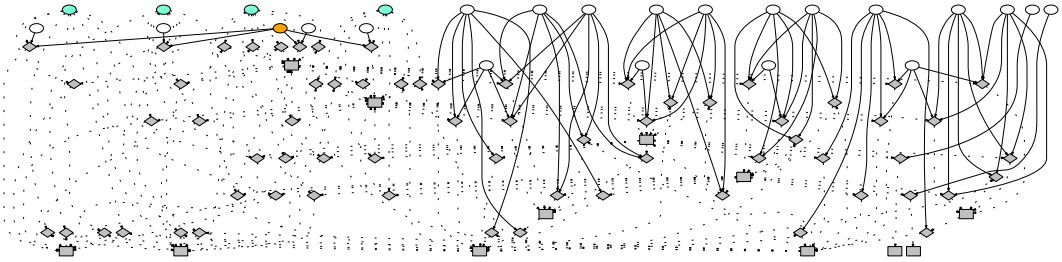
5 Evolutionary Analysis

In order to determine a set of hardening measures, one must first specify what is considered to be the "bad" states in the attack graph, i.e. what the administrator does not want the attacker to achieve. For example, the administrator might want to prevent the attacker from gaining root-level privileges on all hosts. When deriving the hardening set, one then seeks to disconnect the attacker from these undesirable states by applying a hardening measure. The "bad" states correspond to a set of nodes in the attack graph. This can be given specifically, such as "prevent root access on $host_8$", or generally, such as "prevent root access on all hosts". These bad states are referred to collectively as the goal nodes since they represent the goals of the attacker.

Related to finding a set of hardening measures, one can also analyze the network to assess its risk profile. To do so, one simply measures how many of these "bad"



(a) Unpatched



(b) Patched

Figure 3: The attack graph and analyzed attack graph for the scenario where a user visits a malicious website with a vulnerable web browser. This is a classic outsider scenario where the attacker gains a foothold in the network then uses this foothold to further compromise the network. The color scheme is as described in Figure 2.

states the attacker has obtained and output that as a risk metric or a risk profile. Again, this can be tuned to the particular needs of a given network by changing the set of "bad" states to reflect what is undesirable for that particular network.

The hardening measures supported by EVA are patching a vulnerability, adding a firewall rule and placing an IDS sensor. Priority is given to each hardening measure based on the policy model and the mode of analysis. Each measure has two attributes associated with it: the cost of that measure and the security provided by that measure. Both attributes can be manipulated by the policy and by the mode of analysis. When the mode of analysis is to derive a set of hardening measures, the default costs in order from cheapest to most expensive are patches, firewalls and IDS sensors. The default behavior is to have patches and firewall rules confer more security than IDS sensors. Any of these defaults can be changed by the policy model. One can also tell the genetic algorithm to only consider a subset of hardening measures, such as to just consider patches.

A genetic algorithm was chosen as the means of doing the analysis. As described in [5], finding a set of hardening measures directly is computationally infeasible. One cannot "brute force" the solution. Genetic algorithms are an approximation method that allows one to start with random solutions and then refine those solutions into better solutions via an evolutionary process. This is essentially a guided search of the solutions space. Each solution is referred to as a chromosome. A group of solutions being evaluated are called a population. The evaluation continues iteratively for several rounds, with each round being called a "generation". Initially, in the first generation, the population is randomly generated. Then the "fitness" of each chromosome is evaluated. The fitness function determines how well a given solution works for the problem. The most fit chromosomes are then selected as parents and recombined, with the hopes of creating even better solutions. Finally, a few chromosomes are randomly mutated. In EVA, a mutation flips the bit, so if a hardening measure was in use, it would no longer be used and vis versa. After recombination and mutation, the population moves on to the next generation, where it begins with evaluating the fitness of the chromosomes. The population will keep passing through the fitness evaluation, recombination and mutation steps until the programmed maximum number of generations has elapsed.

More details about the genetic algorithm can be found in [5]. The code has been updated since that time to be

multi-threaded when evaluating the fitness of the population. Since each chromosome in the population has its own fitness, this point of the evaluation is well-suited to multi-threading. The population is broken down into sub-groups and each sub-group spawns a thread to evaluate the fitness of the chromosomes in its sub-group. The number of threads is selected when the program is compiled. Currently, four threads are spawned. The main program waits for each thread to complete before moving on to the recombination step.

The chromosome in the genetic algorithm corresponds to a proposed set of hardening measures. During fitness evaluation, each measure in the chromosome is applied to the attack graph. Each node and edge in the attack graph records how it is affected by the measure. A patch disables an initial node, which corresponds to a vulnerability on a machine, and all edges leading out of that node, which correspond to attacks enabled by that vulnerability. A firewall rule disables an edge, which corresponds to the attack that the firewall rule blocks. An IDS sensor watches an edge. This indicates that the attack represented by that edge will be detected if it is executed. After applying the hardening measures, a cascade effect takes place throughout the graph, as described below.

Edges, which correspond to one specific attack, will disable themselves if any incoming node to that edge is disabled. This is because the incoming nodes correspond to preconditions required for the attack to succeed. If any precondition becomes disabled, the attack can no longer succeed, so the edge disables itself. It does not disable the other incoming nodes though since those have not been affected by the fact that the attack can no longer succeed. Similarly, if any of the incoming nodes for an edge are watched, the edge marks itself as watched. This indicates that one of the preconditions for the attack is enabled by an attack that the IDS can detect. This will only occur when several attacks are needed in order for the attacker to reach a goal. While the IDS may not detect the attack corresponding with this edge, it has detected an early attack that is required for this edge's attack to succeed. Thus, this edge will mark itself as watched.

Internal nodes will disable themselves when all their incoming edges are disabled. This means that all attacks which lead to that state have been disabled. When a node disables itself, all edges leading out from that node will disable themselves due to the behavior of edges described above. Similarly, when all edges coming into a node are watched or disabled, the node will mark itself as watched. This indicates that all possible paths to the privilege or condition represented by the node have been covered by IDS sensors. The attacker cannot reach this node without triggering an IDS alarm, so the node is marked as watched. This will then trigger all edges leaving that node to mark themselves as watched, for the reasons described above. If a node or edge is marked as both watched and disabled, the disabled state takes priority.

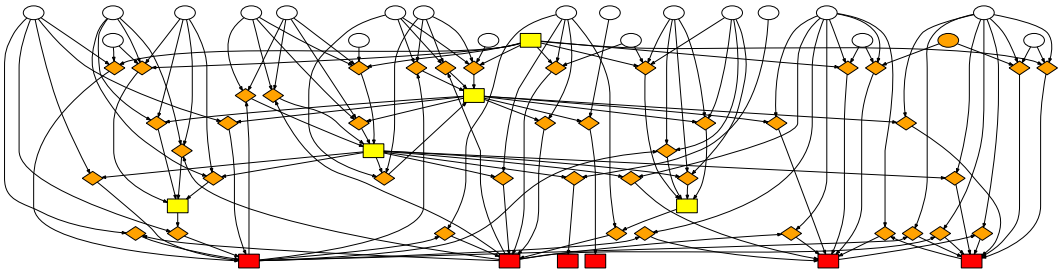
At the end of applying all the proposed hardening measures and this cascade effect, each goal node is checked. The preferable result is that all the goal nodes have been disabled. For each node that is not disabled, its risk metric is calculated based on if it is being watched by an IDS sensor and how many enabled edges can still reach it. The sum of the risk metrics for each goal node is the overall risk that is still present with that proposed set of hardening measures. The genetic algorithm fitness function first seeks to minimize this risk and then attempts to minimize the cost of the measures in the hardening set.

The primary advantage to using a genetic algorithm for analysis are that the direction of the search can be easily changed by altering the nature of the chromosome or the fitness function. For example, if one is just concerned with finding a set of patches to apply, the chromosome can be redefined as just the set of hardening measures corresponding to patches. The same genetic algorithm described above will still work even with this redefinition. EVA's flexibility in analysis comes from this flexibility that genetic algorithms provides.

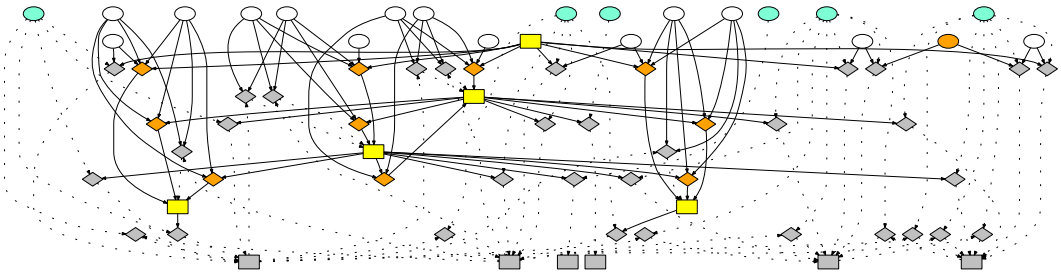
Another advantage to genetic algorithm is many solutions are evaluated in parallel. EVA keeps a record of the best solutions across all generations. Each of these solutions is unique. Currently the ten best solutions are saved, but this is a tunable parameter. When the maximum number of generations has been reached, EVA outputs all of these saved best solutions, ranked by their fitness. The administrator can then choose amongst the solutions. This is particularly useful when multiple solutions with identical fitness exist. The genetic algorithm cannot distinguish between them since their fitness is the same, but a human may have a preference for one solution over another. This is also useful to fine-tune the policy model, described below, to obtain better solutions if the first analysis was not satisfactory to the administrator. By reviewing the saved best solutions, the administrator can see if one hardening measure is being excessively preferred, which could indicate that its cost or benefit needs to be modified.

5.1 Policy Model

The policy model is designed to give the administrator great flexibility in overriding the default behavior of the analysis. The administrator can override the security provided by each class of hardening measures. This would affect how the risk metric is calculated for each goal node. The administrator can also override the cost



(a) Unpatched



(b) Patched

Figure 4: The attack graph and patched attack graph for the malicious student scenario. Since all students are allowed to log in as a user on the lab machines, the analysis cannot disable the user privilege nodes in the patched graph.

of hardening measures. This can be done for a specific hardening measure or a group of hardening measures. The cost can also be changed on different machines.

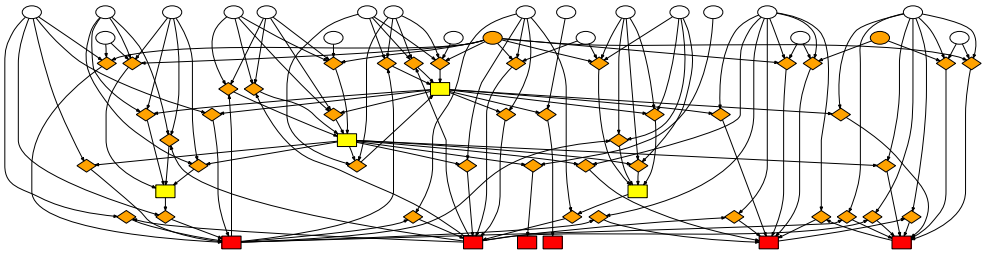
For patches, the policy model allows an administrator to specify an abstract vulnerability class from the abstract exploit templates, a machine name template and the new cost. The abstract vulnerability class corresponds to a class of patches. The machine name template can be an actual machine name, a cluster name or a partial name which will match all machine and cluster names containing that name. The administrator can specify just the vulnerability class or just the machine name template if desired. The most specific cost is used when there is overlap between multiple policies. For example, an administrator can set the cost of a "privilege escalation" class patch to 5 on all machines with one policy rule, but say that the cost of the "privilege escalation" class is only 3 on $host_4$ with another rule. The second rule would be used for $host_4$.

For firewall rules, the policy model allows the cost to be set based on the source of the packet, the destination of the packet and the abstract destination port from the abstract exploit templates. As with patches, the source and destination machine names can be an actual machine name, a cluster name or a partial name. The destination port can be one of the abstract port names or the keyword "all". Similar to the patch policy rules, not all fields need to be specified. If two rules overlap, again the most specific rule will be used. IDS sensor placement has all the fields that firewall rules have and adds a field for the abstract exploit class. The abstract exploit class field allows one to say it is cheaper or more expensive to monitor for certain types of exploits.

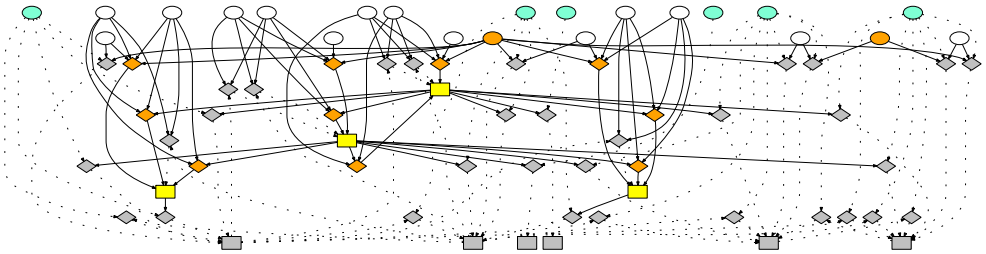
Policy rules can be set for each mode of analysis. Only the rules for the current mode of analysis will be considered. For any hardening measure not covered under a policy rule, the default cost is used. The administrator may alter these default costs for each hardening measure class as well. Default costs can also be altered based on not only the class, but also the mode of analysis.

5.2 Modes of Analysis

The genetic algorithm is adaptable to many modes of analysis. Besides finding a set of hardening measures, it can also be used for strategic planning, network design, forensic evaluation and IDS monitoring. This is done by changing the costs and priorities of each hardening measure (thus altering the fitness of a chromosome), by redefining the chromosome to only consider a subset of hardening measures or by altering the input to the attack graph generator.



(a) Unpatched



(b) Patched

Figure 5: These graphs are for the scenario where a user has a compromised laptop and plugs it in to the instructor's station in a lab. The attacker cannot be prevented from obtaining user privileges since an easily-guessed login is used for student access to the lab machines. The policy prevents this login from being disabled.

For strategic planning, the desired task is to evaluate how the network would respond to unknown risks by performing "what if" scenarios. Essentially, an administrator adds vulnerabilities to the vulnerabilities list file that have not actually been detected in the network and/or alters the connectivity of the network. For example, an administrator would ask "what if machine x has a remote to root vulnerability?" The "what if" scenarios are particularly useful to model vulnerabilities that a vulnerability scanner can not easily find. For example, Nessus cannot detect a client-side browser vulnerability, but this is becoming a common method used to compromise a machine. If the administrator does not have a client-based vulnerability analyzer, he can still model client-side attacks by performing a "what if" scenario. The tool computes the attack graph for the given scenario. The administrator can analyze the resulting attack graph in any of the other supported modes.

With network design, the administrator wants to create a network that is resistant to attack. There are two ways attack graphs can be used to support network design. The simplest method is to have the administrator design several potential networks as input to the strategic planning mode. The tool would then calculate an attack graph for each network and its associated risk metric. The results could then be displayed to the administrator so she can choose the design which has the lowest metric and which best suits the requirements of the installation.

A more interesting approach to network design analysis, and an approach unique to EVA, is to give a prototype network design to the tool and have the tool automatically reconfigure the network to minimize risk. The genetic algorithm in this mode does not consider patches as a possible hardening measure. Instead, it focuses on firewall rules, which could also be interpreted as routing rules, and IDS sensor placement. The fitness function still seeks to minimize the risk of the network. The costs are policy-driven, using the policy rules for network design. The set of firewall rules and IDS sensors that minimizes the risk and minimizes the cost is favored by the algorithm. It outputs several potential network designs that follow this desired outcome.

For forensic evaluation, the current evidence is given as input. This evidence can consist of known resources the attacker has achieved, which corresponds to nodes in the attack graph, or IDS alerts about attacks seen, which corresponds to edges in the attack graph. All evidence that corresponds to the attack graph of the network is highlighted and treated as the initial states of a subgraph of the attack graph. Any other nodes reachable by these states could be other resources the attacker could have compromised. The IDS monitoring mode works simi-

larly, but with current IDS alerts. While it has not been implemented yet, theoretically one could feed the output of the IDS monitoring mode to an intrusion response system. It could then use the knowledge of resources at risk to add further protection measures for those resources. This could prevent the attacker from compromising those resources.

6 Experimental Results

The Computer Science instructional network, as shown in Figure 1, was profiled as the input network to this tool. The network consists of a server zone located outside the firewall and a NAT zone for all the instructional labs. The server zone contains five servers: two Debian Linux servers, one Solaris 8 server, one Solaris 7 server and one Digital Unix server. The instructional lab machines are all identical within a single lab room. There are several prototype lab machines that the administrator clones out to all the machines in a particular room. These prototypes are an Ubuntu Linux image for the general access labs (51 machines), an Ubuntu Linux image for the programming lab (36 machines), a Windows XP image for the hardware labs (24 machines) and an Ubuntu Linux image for the advanced computation lab (30 machines). In total, there are 141 lab machines in the NAT zone and 5 machines in the server zone.

The clustering Perl script derived four clusters based on the vulnerabilities present on the machines and the connectivity allowed by the machines. The first cluster consisted of the servers. The second cluster corresponded to the general access labs. The third cluster corresponded to the programming lab. The fourth cluster contained both the hardware and advanced computation labs since they had identical abstracted vulnerabilities. Even though the actual vulnerabilities differed, the abstracted vulnerabilities are what matters for purposes of clustering. The process of clustering the network took 0.25 seconds on a Xeon quad core 2.33GHz system.

Three "what if" scenarios were also generated for the network. The first scenario assumes that a student in the general access lab is using a version of Firefox with an exploitable vulnerability that would give a malicious website the same privileges on the machine as the student. It is then assumed the student visits such a website, giving the attacker user privileges on that machine. The attacker model states that the attacker's malicious website would place a bot on that machine which would then attempt to compromise other machines and would "call home" to the attacker, thereby allowing the attacker to communicate with the machine even though it is in a NAT.

The second scenario assumes a student has decided to compromise the network. This is a variation of the insider problem. Since the student already has user privileges on all lab machines and several servers, his goal is to escalate his privileges to root on one or more machines. The third scenario assumes an instructor has brought a compromised laptop on to campus. All lab rooms have an Ethernet jack at the instructor station where the instructor can plug in a laptop. There are no restrictions on the connectivity of these jacks. Therefore, once plugged in, they have full access to the LAN containing all the lab machines. Again, this scenario assumes the compromised laptop can "call home" to the attacker so the attacker can have direct access into the NAT zone via the laptop.

All three scenarios and the base configuration of the network were given as input to the attack graph generator. The attack graph for the base scenario showed that two of the servers could be compromised via "remote to root" vulnerabilities. These were two old servers approaching end-of-life which had not been maintained recently. The attack graph for the Firefox vulnerability scenario showed that once the attacker had a foothold into the NAT zone, he was able to get user on all lab machines via the "student" account, which is the account all students use to log in to the lab machines locally. The cluster containing the hardware and advanced lab had a "remote to root" vulnerability that the attacker was able to exploit to get root privileges on those machines. The programming lab had a "privilege escalation" vulnerability that allowed the attacker to elevate from user to root on those machines.

The attack graph for the malicious student showed a similar course of action. The student is able to escalate from user to root on the programming lab machines. The student is also able to exploit the "remote to root" vulnerability on the hardware and advanced lab machines. Likewise, the attack graph for the rogue laptop also showed these compromise routes once the laptop had been plugged into the NAT zone. The generation of each of these attack graphs took 0.5 seconds on the aforementioned quad core Xeon machine.

Each attack graph was then given to the hardening mode of EVA. The goal given to the analysis was to prevent the attacker from gaining root privileges on any machine. The analysis was further restricted to only considering patches that could be applied, instead of all possible hardening measures. The policy rules applied to the evaluation were that logins could not be turned off to any machine and on the lab machines the "student" account could not be disabled, even though it has a guessable password. A run was made without these policy rules and several of the highly fit solutions proposed by the genetic algorithm did indeed suggest these courses of action. When the policy rules were applied, none of the highly fit solutions contained these courses of action.

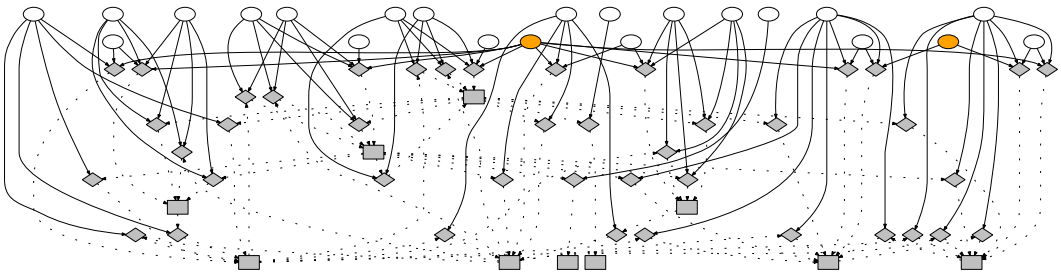


Figure 6: This is the attack graph for the rogue laptop scenario after redesigning the network to segment off the Ethernet port at the instructor's station. Since the laptop plugged into that port can no longer connect to the lab machines, it is unable to compromise them even though the easily guessed student login from Figure 5 remains.

The genetic algorithm was run several times using different population sizes and different maximum generation limits. Larger population sizes will more frequently generate optimal results, but require more CPU time to complete the analysis. A larger maximum generation limit likewise can increase the optimality of the result, but also takes more CPU time. Part of the testing was determining values for these two parameters that balanced good results against CPU time. In doing so, a "suggested parameters" matrix can be developed for other networks that are similarly sized.

When the base configuration was evaluated, the suggested course of action was to patch two servers which had "remote to root" vulnerabilities. No other courses of action were suggested because the remaining machines are inside the NAT and the attacker did not have a vector into the NAT zone in the base configuration. It took 0.01 seconds to evaluate the base scenario using a population of 50 chromosomes and 50 maximum generations for the population. The original and patched attack graph for the base scenario are shown in Figure 2.

For the Firefox scenario, the suggested course of action was to patch the two servers, as before, and to patch the Firefox vulnerability that gave the attacker a foothold into the NAT zone. Again, the genetic algorithm was run with a population of 50 and 50 maximum generations. It took 0.04 seconds for the genetic algorithm to derive this recommendation. The attack graph and analyzed attack graph for this scenario are shown in Figure 3.

For both the malicious student scenario and the rogue laptop scenario, the suggested course of action was to patch the two servers, patch the privilege escalation vulnerability in the programming lab and patch the remote to root vulnerabilities in the hardware and advanced labs. This limits the attacker to just getting user privileges on the machines via the "student" accounts, since it was not allowed to disable those accounts. Again, with a population of 50 and 50 maximum generations, it took 0.03 seconds for the genetic algorithm to derive these recommendations for each scenario. The attack graphs for the malicious student scenario are shown in Figure 4 and the graphs for the rogue laptop are shown in Figure 5.

6.1 Network Design

The three scenarios were also analyzed using the network design mode. For all scenarios, the most fit solutions only required new firewall or router rules. None of the recommendations included placing an IDS sensor for this data set.

For the Firefox vulnerability scenario, it was assumed that the vulnerability was just in the general access labs. The most fit recommendation stated to block Firefox in the general access labs, since there was no policy rule stating to avoid this action. Since blocking Firefox was considered the cheaper course of action, it was recommended over segmenting the NAT zone. With a population size of 250 and 250 maximum generations, the genetic algorithm was able to find this solution on the majority of its runs. It took on average 1.3 seconds to find this recommendation.

For the malicious student scenario, it was assumed the student just had class in the campus-wide general access lab. The student was assumed to not have physical access to the programming, hardware and advanced computation labs. The most fit recommendation was to segment the general access labs away from the remaining labs. Again, a population of 250 and 250 maximum generations were needed to consistently produce this result. It took an average of 1 second for the algorithm to run.

For the compromised laptop, the most fit design was to segment the laptop Ethernet jack at the instructor's station away from the rest of the labs in the NAT zone. As before, a population of 250 and 250 maximum generations were needed. It took an average of 1.05 seconds to calculate. Figure 6 shows the attack graph after the laptop port has been segmented into a different subnet.

This mode needed a larger population size and a higher maximum generation limit than finding a patch set because there were more possible solutions. The number of

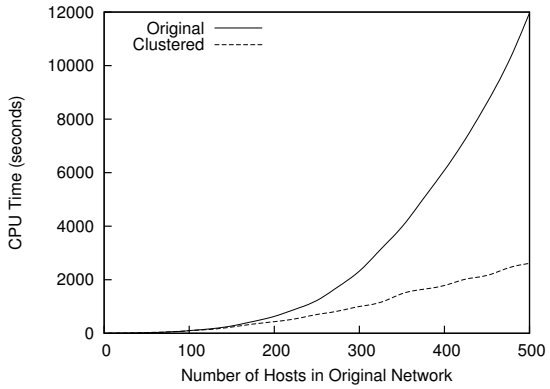


Figure 7: The CPU time for running the attack graph generator and analysis in hardening mode for generated networks with 5 to 500 machines. The original time is for the unclustered machines. The clustered time includes the time it takes to cluster the machines before generating and analyzing the graph.

edges in an attack graph are far greater than the number of nodes in an attack graph since most nodes are highly connected. Determining how to segment the network involves finding the minimal set of edges to cut to disconnect the attacker from the goal nodes, while finding a patch set involves finding a minimum set of nodes to disable. Since there are more edges than nodes, the network design mode has more possible solutions than deriving a set of patches.

6.2 Scalability Testing

The simulated network described in [5] was run through the clustering script, had the attack graphs generated and then was evaluated using the hardening mode in order to test the scalability of this approach. Previously in [5], the tool was tested to a network with 500 unclustered machines. Those same networks were clustered and run again.

For both the unclustered and clustered networks, the proposed hardening measures completely prevented the attacker from getting root privileges on any machine in the network. Figure 7 shows the CPU time of the two methods when the genetic algorithm had a population of 250 and 500 maximum generations. The CPU time is used for this figure since the results in [5] did not use a multi-threaded form of the genetic algorithm. Comparing the CPU time allows the clustering results, which do use the multi-threaded algorithm, to be meaningfully compared to the single-threaded algorithm. It is clear that with clustering, it took far less time to derive the hardening set.

Again, the tool was run with multiple values for the population size and maximum generations. This allowed the "suggested parameters" matrix to be filled with information from larger networks than the Computer Science instructional lab network. As expected, the smaller networks needed only small values for these two parameters. The largest network tested, which contained 2500 unclustered nodes and 337 clustered machines, needed a population size of 500 and a maximum generations of 500 to determine a set of patches. It took an hour and a half on the Xeon quad core 2.33GHz system to analyze this graph due to the complexity of the graph and the large genetic algorithm parameters needed to produce optimal results.

7 Future Work

There are still several areas of improvement for this tool. The first area of improvement is the gathering of input data for the tool. Currently, the firewall and routing rules have to be imported by hand. The next improvement will be to automatically import firewall rules using tools that can extract firewall rules from the network. Another area of input automation is the Nessus plugin ID to abstract vulnerability mapping. A student is currently working on a evolutionary technique to scan the plugin description and classify the plugin based on the keywords in the description. If this works, it should greatly reduce the maintenance needed for the abstraction mappings. Of course, another area for input improvement is to support other vulnerability scanners besides Nessus. This is also planned for the tool.

The second area of improvement is the attacker model. Currently, only one attacker is assumed. If one wishes to model multiple attackers, one needs to run several scenarios, similar to what was described in the results section. A future improvement is to allow multiple attacker models for a single attack graph. This will require marking the nodes to identify which attackers have gained that node and altering the genetic algorithm to pay mind to this node marking.

Another area of improvement is the visualization of the attack graphs. While DOT [3] is nice for small networks, it does not visualize large networks well. A better visualization technique would allow an administrator to "drill down" into the graph to see more specific details or "zoom out" to see more general details.

On the analysis side, one desired area of improvement is to integrate the IDS correlation mode with an intrusion response system to see if it would be feasible to run the analysis in real-time and also if doing so would stop an attacker before they compromised resources. This would be a very powerful extension to the tool.

8 Acknowledgements

I would like to thank the undergraduate students who have worked on this project for their hard work. Jonathan Berling was instrumental in translating the Nessus reports into the appropriate format for attack graph generation and in assisting with the creation of the scenarios that were presented in this paper. Fred McHale and John Millikin played a key role in setting up the isolated network that was used to test the scalability of EVA. I'd also like to thank the Computer Science network administrator, Steve Garcia, and his student assistant, Nick Toothman, for their help in scanning and modeling the Computer Science instructional network.

References

[1] CLIPS: A Tool for Building Expert Systems. [Online] http://clipsrules.sourceforge.net/.

[2] AMMANN, P., WIJESEKARA, D., AND KAUSHIK, S. Scalable, Graph-Based Network Vulnerability Analysis. In *CCS02: 9th ACM Conference on Computer and Communication Security* (Washington, DC, November 2002), ACM, pp. 217 – 224.

[3] AT&T RESEARCH. Graphviz - Open Source Graph Drawing Software. [Online] http://www.graphviz.org/, April 2006. Version 2.8.

[4] DANFORTH, M. *Models for Threat Assessment in Networks*. PhD thesis, University of Califonia, Davis, Davis, CA, USA, June 2006.

[5] DANFORTH, M. Scalable Patch Management using Evolutionary Analysis of Attack Graphs. In *Proceedings of the 7th International Conference on Machine Learning and Applications* (San Diego, CA, USA, December 2008), pp. 300–307.

[6] DAWKINS, J., CAMPBELL, C., AND HALE, J. Modeling Network Attacks: Extending the Attack Tree Paradigm. In *Proceedings of the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection* (June 2002).

[7] DEWRI, R., POOLSAPPASIT, N., RAY, I., AND WHITLEY, D. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *CCS '07: Proceedings of the 14th ACM conference on Computer and Communications Security* (New York, NY, USA, 2007), ACM, pp. 204–213.

[8] FRIEDMAN-HILL, E. JESS: Java Expert System Shell. [Online] http://www.jessrules.com. Version 6.1p6.

[9] INGOLS, K., LIPPMANN, R., AND PIWOWARSKI, K. Practical Attack Graph Generation for Network Defense. In *Proceedings of the 22nd Annual Computer Security Applications Conference* (Miami, FL, USA, December 2006), pp. 121–130.

[10] JAJODIA, S., NOEL, S., AND O'BERRY, B. *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publisher, 2003, ch. Topological Analysis of Network Attack Vulnerability.

[11] JHA, S., SHEYNER, O., AND WING, J. Two Formal Analyses of Attack Graphs. In *IEEE Computer Security Foundations Workshop* (Cape Brenton, Nova Scotia, Canada, June 2002), pp. 49–63.

[12] J.TEMPLETON, S., AND LEVITT, K. A Require/Provides Model for Computer Attacks. In *Proceedings of the New Security Paradigms Workshop* (Cork Island, September 2000).

[13] MIT PRESS RELEASE. MIT Lincoln Laboratory software aims to thwart cyber hackers. [Online] http://web.mit.edu/newsoffice/2008/security-0827.html, August 2008.

[14] NOEL, S., JAJODIA, S., O'BERRY, B., AND JACOBS, M. Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference* (Las Vegas, NV, USA, December 2003).

[15] PHILLIPS, C., AND SWILER, L. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the New Security Paradigms Workshop* (Charlottesville, VA, 1998).

[16] RITCHEY, R. W., AND AMMANN, P. Using Model Checking to Analyze Network Vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (Oakland, CA, May 2000), pp. 156 – 165.

[17] SHEYNER, O. *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, April 2004.

[18] SHEYNER, O., HAINES, J., JHA, S., LIPPMANN, R., AND WING, J. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2002), pp. 254 – 265.

[19] SWILER, L., PHILLIPS, C., ELLIS, D., AND CHAKERIAN, S. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II* (June 2001).

[20] TENABLE NETWORK SECURITY. Nessus. [Online] http://www.nessus.org/.

[21] WILLIAMS, L., LIPPMANN, R., AND INGOLS, K. GARNET: A Graphical Attack Graph and Reachability Network Evaluation Tool. In *Proceedings of the 5th International Workshop on Visualization for Computer Security* (Cambridge, MA, USA, September 2008), pp. 44–59.