From Attack Graphs to Automated Configuration Management — An Iterative Approach

John Homer
Kansas State University
Manhattan, KS
jhomer@ksu.edu

Xinming Ou
Kansas State University
Manhattan, KS
xou@ksu.edu

Miles A. McQueen
Idaho National Laboratory
Idaho Falls, ID
miles.mcqueen@inl.gov

***Abstract*—Various tools exist to analyze enterprise network systems and to produce attack graphs detailing how attackers might penetrate into the system. These attack graphs, however, are often complex and difficult to comprehend fully, and a human user may find it problematic to reach appropriate configuration decisions. This paper presents methodologies that can 1) automatically identify portions of an attack graph that do not help a user to understand the core security problems and so can be trimmed, and 2) enable a user to use the information in an attack graph to reach appropriate configuration decisions, through a configuration generator that can be iteratively trained by the user to understand a wide range of constraints in configuring an enterprise system, such as usability requirements and trade-offs that need to be made between the cost of security hardening measures and the cost of potential damage. We believe both methods are important steps toward achieving automatic configuration management for large enterprise networks. We implemented our methods using one of the existing attack-graph toolkits. Initial experimentation shows that the proposed approaches can 1) significantly reduce the complexity of attack graphs by trimming a large portion of the graph that is not needed for a user to understand the security problem, and 2) automatically provide reasonable suggestions for resolving the security problem.**

I. Introduction

Enterprise networks continue to grow in both size and complexity. With this increase, concerns for security grow apace. Vulnerabilities are regularly discovered in a wide variety of software applications, and it is impossible to continually and manually verify that a network has no vulnerabilities to outside attacks. Tools have been developed to identify and report known vulnerabilities in running software versions, firewall policies, and other system settings. Attack graphs provide a visual representation of potential attack paths employing these vulnerabilities that could be followed to exploit system resources [22], [1], [6], [11].

Much work has already been done in the generation of attack graphs, steadily producing more efficient techniques for building them [19], [5]. Attack graphs, however, are difficult for a human to utilize effectively because of their complexity [17]. Even a network of moderate size can have dozens of possible attack paths, overwhelming a human user with the amount of information presented. In all of this complexity, it becomes very hard to determine causality in the attack paths and identify which vulnerabilities carry the highest potential for network exploitation. It is not easy for a human to determine from the information in the attack graph which configuration settings need to be changed to address the identified security problems.

To make things more complicated, the requirements for usability are often at odds with those for security. Configuration management would be a trivial problem if one only needed to consider security requirements — then all that would be needed is to shut down the whole enterprise system. When changing configurations to block discovered attack paths, one must balance the system's security and usability so that the security risks are appropriately controlled in a cost-effective manner.

We identify the following two research challenges for making an attack graph a useful tool for configuration management.

- C1: Presenting the security problems expressed by an attack graph in a manner that enables a human user to quickly grasp the core of the security problem.
- C2: Automatically provide suggestions for configuration changes that mitigate the threats presented in an attack graph, through an automatic decision procedure that can be trained by a human user.

C1 is important because the balance between security and usability inevitably involves subjective judgment and a human needs to be involved in reaching the final

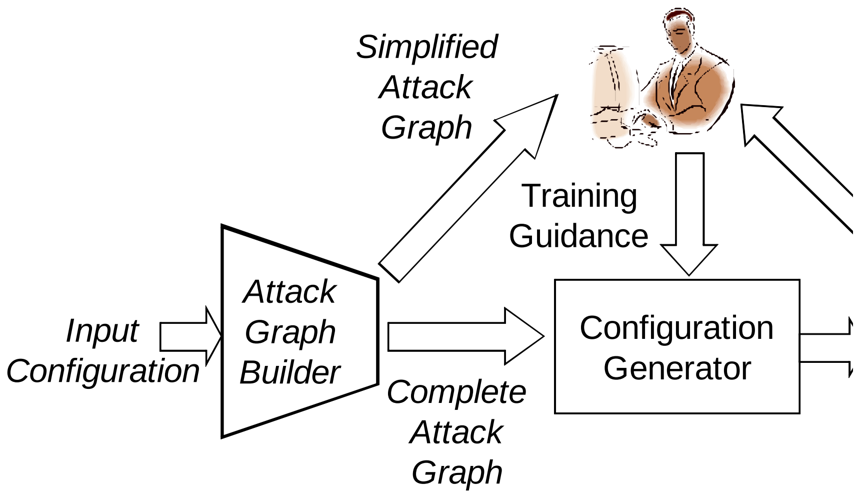

Fig. 1. Iterative configuration generation

decision. Without a clear understanding of the existing security problems, it is difficult for a human user to evaluate the proposed configuration changes and provide appropriate guidance.

C2 is important because it alleviates a human user's burden of searching for the best way to configure a system given a large number of complex security and usability constraints. Since configuration management is such an important and complex problem, it is unlikely that an automatic tool can take over a human system administrator's role overnight. Rather, we envision an incremental process whereby automated tools can assist a human user to manage an enterprise network, increasing the degree of automation as the tool is trained. The cost for changing a specific configuration parameter and the cost caused by a potential attacker privilege will vary from one organization to another. There is no one-size-fits-all configuration solution for all enterprise systems, and an automatic tool need to learn from a human user's guidance. This vision is depicted in Figure 1.

We developed a systematic approach that can realize such a vision. Our contributions are:

1) We developed an algorithm that can identify portions of an attack graph that are not helpful for a user to understand the core security problems, and simplify attack graphs by trimming those portions.
2) We developed an approach that can transform the configuration problem into a SAT solving problem, in polynomial time. This enables us to leverage a modern SAT solving technique, MinCostSAT [9], to find a mitigation solution that is optimal according to the various cost functions defined by the human user regarding human labor needed to carry out configuration changes, the possible loss in the system's usability, and the potential damage caused by a successful attack. Such cost functions serve as training guidance to the configuration generator and can evolve incrementally and iteratively.

The rest of the paper is organized as follows: Section II explains the attack graph semantics; Section III introduces a key example we will employ throughout the paper; Section IV presents our attack graph simplification algorithm and its application to the example; Section V defines our transformation of the problem into a SAT solving problem and demonstrates its effectiveness; Section VI discusses future work on this topic; Section VII addresses related work; and Section VIII concludes.

II. Attack Graph Semantics

We use the MulVAL attack graph tool suite [20], [19] for our work. MulVAL has reasonable performance and scalability for enterprise networks of a realistic size. The reasoning logic in MulVAL is specified declaratively in Datalog, which makes it easier for us to build our methodologies on top of the core MulVAL reasoning engine. But the techniques we present in this paper could be applied to other attack graphs that have similar semantics.

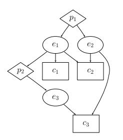

Fig. 2. A dependency attack graph

Figure 2 is a simple attack graph. c_1, c_2, c_3 represent configuration conditions. For example, c_1 could mean the existence of a software vulnerability on machine A, c_2 could mean the reachability from machine B to machine A, and so on. p_1, p_2 represent potential privileges an attacker can gain. For example, p_1 could mean an attacker can execute arbitrary code on machine A, and p_2 could mean an attacker can execute arbitrary code on machine B. e_1, e_2, e_3 are exploits that link the causality relations between various configuration conditions and attacker privileges. For example, e_1 is an exploit that requires all of p_2, c_1, c_2 to be true, with the consequence being p_1. The arcs in the attack graph point to the causes for potential attacker privileges. The arcs from an exploit vertex form the logical AND relations, meaning all of the preconditions must be true to enable the exploit. We symbolize those vertices as ellipses. The arcs from a privilege node (e.g. p_1) form the logical OR relation, since the out-neighbours of the vertex represent alternative ways to gain the privilege. We symbolize those vertices as diamonds. The vertices representing configuration conditions are sink nodes in the graph and we symbolize them as boxes. The dependency attack graph in Figure 2 shows that attackers can gain privilege p_1 in two different ways. They can launch exploit e_1 if all of the conditions c_1, c_2 and p_2 are true. Or they can launch exploit e_2 if conditions c_2 and c_3 are true. Privilege p_2 can be enabled only by exploit e_3, for which the only precondition is c_3.

This diagram is just to give the reader a basic understanding of the logic semantics in the attack graphs we use and does not correspond to any realistic scenarios. The attack graphs computed for real enterprise systems will be much larger.

III. An Example

Figure 3 shows an example enterprise network that is based on a real (and much bigger) system. We are not able to disclose the real system due to the sensitivity of the information. However, the specific problems we encountered when applying attack-graph techniques are the same for both the real and the adapted system.

Subnets: There are three subnets in the enterprise system: a DMZ (Demilitarized Zone), an internal subnet, and an EMS (Energy Management System) subnet, which is a control-system network for power grids.

Hosts: The functionalities of most of the hosts are self-explanatory. One functionality of the EMS network is to gather real-time statistics, such as voltage, load, etc.,

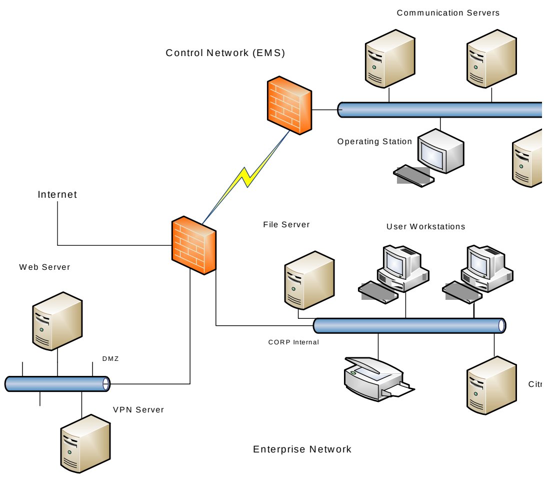

Fig. 3. An example enterprise network

from the physical power transmission and generation facilities. This information is the basis for generating various control signals to maintain appropriate operation of the power grid, and it is also useful for business transactions such as power brokerage between producers and consumers. The communication servers in the EMS subnet are responsible for communicating with the power grid physical infrastructures. The data historian is a database server that provides the power-grid statistics to be used for the various business/operation purposes.

Accessibility:

Src	Dst	Network access allowed
Internet	DMZ	all to web server all to VPN server
DMZ	internal	web server to file server VPN server to all
internal	EMS	Citrix server to data historian

The out-bound accessibility, such as from internal subnet to Internet, is ignored in the table. Note that the Internet can only directly access DMZ, and the Citrix server is the only host in the internal subnet that can access the EMS network, and it can only access the data historian.

Threats: For this kind of system, security of the control system network is of paramount concern, since privileges on those machines could enable an attacker to assume control of physical infrastructures and drive them to a failed state (such as causing the turbine of a power generator to spin at a high speed until self-destruction). A number of mechanisms in this example are in place to reduce such threats as shown in the accessibility table.

Vulnerabilities: There are plenty of vulnerabilities in the system. The data historian runs a proprietary communication protocol with the communication servers that can be very easily compromised. All the machines in the system may have software vulnerabilities in their various services and software applications. The users of the enterprise network may not be careful to protect their log-in credentials and could leak their user name and password to an attacker through various means, including social engineering. The VPN server in the DMZ can access every machine in the internal subnet. The web server in the DMZ can access the file server through the NFS file-sharing protocol. These vulnerabilities could affect the security of the enterprise network and the control system network.

A. Problems in the attack graph

The MulVAL attack-graph toolkit identified a large number of potential attack paths on this example. The size of the attack graph (150 nodes, 173 arcs) prohibits a full illustration of the graph in this paper. However, in reading the attack graph, we found that many of the attack steps, while valid from a logical point of view, are not helpful for a human user to comprehend the core security problems in the configurations. We describe a few of them below.

1) The attacker, after compromising the VPN server, can further compromise the web server.
2) The attacker, after compromising the workstation, can further compromise the web server.
3) The attacker, after compromising the Citrix server, can further compromise the file server.

All these attack steps are valid, but they are not very enlightening. For example, for case 1, after an attacker compromises a VPN server, there is probably not much value for him to attack the web server. In case 2, the attacker moves "back" from the internal subnet to DMZ, which he must have visited already in order to reach the internal subnet. If we assume the attacker's goal is to obtain privileges on the control network, case 3 also does not show the top-priority security problem since the Citrix server is the only stepping stone into the control network. If the attacker has already compromised the Citrix server, he can attack the control network directly from there.

There are many such attack steps in the generated attack graph. They share a common characteristic which is they do not reveal the most important vulnerability in the system since the attacker does not penetrate "deeper" into the enterprise network along those steps. While these steps contain important information that would be useful if one wished to block every possible attack path, they can also be distracting to a human reader and often hides the root causes of the security problems. It is thus beneficial to remove these less useful attack steps from the attack graph so that the security problems become easier to grasp for a human reader. Note that this is not a simple problem of cycle detection and elimination. Not all cycles in attack graphs can be eliminated since some of them are useful. Detecting useless cycles in an attack graph is not a trivial problem [19].

We refer to attack steps that are not useful for a human reader to understand the underlying security problems as "useless attack steps." It is important to emphasize that those attack steps are valid and important to consider when determining upon appropriate countermeasures. For example, the attack step from VPN server to the web server (case 1) indicates that there is an alternative path to reach the web server. So even after blocking access from Internet to web server, and blocking access from VPN server to the internal subnet, an attacker can still potentially compromise the internal subnet if he can first compromise the VPN server, then compromise the web server, and from web server penetrate deeper into the internal subnet. However, when the user is first presented with the attack graph, this additional path is not crucial for understanding overall security threats. It would be beneficial if the user can quickly understand the core security problems from a simplified attack graph.

IV. Simplifying Attack Graphs

In order to trim the useless attack steps, we first need to identify them automatically. We say an attack step is associated with a *transition* $A \rightarrow B$ if it indicates that privileges on machine A enable an attacker to gain privileges on machine B. A useless attack step is one that is associated with a *useless transition*, which we will define shortly. We have observed a large number of useless attack steps in the attack graph produced for the example network, as well as attack graphs for other system scenarios studied before, and notice that there are two basic types of useless attack steps.

Useless intra-subnet transitions In one type of situations, the attack steps indicate an attacker's privilege on machine A can enable him to further compromise machine B in the same subnet, but privileges on machine B cannot help the attacker penetrate deeper into the enterprise system. Case (1) and (3) in the example are in this category. Since the VPN server can already access all the machines in the internal subnet, moving from the VPN server to the web server does not increase the attacker's ability to attack machines in the internal subnet. Similarly, since the file server cannot directly access the control network, attacking it from the Citrix server, which can access the control network, does not increase the attacker's ability to further compromise the control network.

Useless inter-subnet transitions In the second type of situations, the attacker's privileges on machine A in subnet S_1 enables him to attack a machine B in another subnet S_2, but S_1 is "deeper" into the enterprise network than S_2. Case (2) in the example is in this category. Intuitively, the internal subnet is "deeper" than the DMZ subnet in the enterprise system. Thus, attacking from the workstation in the internal subnet to the web server in the

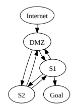

Fig. 4. A subnet graph

DMZ does not increase an attacker's ability to penetrate deeper into the system.

We need a few more definitions to formally characterize what transitions are useless.

Definition 1 *A subnet graph of an enterprise network is* (V, E)*, where* V *is the set of subnets*[1] *and* E *is the set of edges between subnets.* $(S_1, S_2) \in E$ *if there is a machine* H_1 *in* S_1 *such that* H_1 *can reach* H_2 *in* S_2 *using some protocol and port.*

Figure 4 shows an example subnet graph. It is slightly different from the subnet structure of the enterprise system in Fig 3, in that it has two internal subnets S1 and S2. We assume that the attacker is initially located in subnet Internet and he would like to gain some privileges in a "goal" subnet (e.g., Goal). This can be generalized to include cases where an attacker has multiple start locations and multiple goals in mind. In such cases we only need to introduce a dummy source node in the graph that has an edge to all the initial-location subnets, and a sink node that has an edge from each goal subnet. The information needed to build the subnet graph is most likely already available from an attack-graph toolkit, since it needs to analyze the topology of the enterprise network along with the firewalls, router configurations, etc., to compute reachability relations between hosts. We actually wrote very simple code to derive the subnet graph structure from the input tuples to the MulVAL reasoning engine.

From the subnet graph, it can be seen that there is some ordering on the subnets. For example, S1 and S2 are deeper than DMZ, since machines from the Internet can only access the DMZ, and machines in S1 and S2 can only be directly accessed from the DMZ. S1 is also deeper than S2 since S1 can directly access the Goal subnet. When producing attack graphs, such topological order is not taken into account, which is the reason why many useless and distracting attack steps are included.

We compute the dominator and post-dominator trees on the subnet graph to identify such topological order. Let d, n be vertices in a directed graph. d dominates n if every path from the source node to n must go through d. We write d *dom* n for this fact. p post-dominates n if every path from n to the sink node must go through p. We write p *postdom* n for this fact. In the subnet graph of Figure 4, assuming that Internet is the source and Goal is the sink, we have DMZ *dom* S1 and DMZ *dom* S2. We also have S1 *postdom* S2 and S1 *postdom* DMZ. Computing the dominators and post-dominators in a graph can be done in almost linear time [8].

Definition 2 *A transition between subnets* $S_1 \rightarrow S_2$ *is useless if either* S_2 *dom* S_1 *or* S_1 *postdom* S_2*.*

Intuitively, if an attacker moves from subnet S_1 to one of its dominators S_2, this is not a useful step since he already must have reached S_2 in order to be in S_1. Likewise, if S_1 post-dominates S_2, the attacker will necessarily return to S_1 at some later point. We will eliminate such attack steps since they are distracting for a human reader trying to comprehend other, more enlightening attack paths. In the example of Section III, the transition from the internal subnet to the DMZ is useless since the DMZ dominates the internal subnet. The transition from the internal subnet to the control network subnet is useful.

Definition 3 *An outgoing set of a host* H *is the set of tuples* $(host, protocol, port)$ *such that host is not in the same subnet as* H *and can be reached directly from* H *through protocol and port. We use* $outgoing(H)$ *to denote this set.*

Definition 4 *A transition between two hosts* $H_1 \rightarrow H_2$ *in the same subnet* S *is useful if either* H_2 *is one of the target machines, or there exists* $s = (host, protocol, port), s \in outgoing(H_2), s \notin outgoing(H_1)$*, host is in subnet* S' *and* $S \rightarrow S'$ *is useful.*

Definition 5 *A transition between two hosts* $H_1 \rightarrow H_2$ *in two different subnets* S_1 *and* S_2 *is useful if the transition* $S_1 \rightarrow S_2$ *is useful.*

[1]We adopt the normal meaning of a subnet as a group of machines that are physically connected behind a switch with unrestricted access to each other (except for simple filtering to prevent source-IP spoofing or ARP spoofing).

The transitions between hosts that are not useful, based on the above definitions, are considered useless and will be trimmed from the attack graph. In Section III's example, the transition from the VPN server to the web server is useless since the two machines are in the same subnet (DMZ), and no machine outside the DMZ can be reached by the web server and not the VPN server. The transition from the workstation to the web server is also useless since the transition from the internal subnet to the DMZ subnet is useless. But the transition from the web server to the file server is useful since the transition from the DMZ to the internal subnet is useful. And the transition from the file server to the Citrix server is useful since the Citrix server can reach the data historian in the control network subnet and the file server cannot, and the transition from the internal subnet to control network subnet is useful.

A. Implementation

The MulVAL attack-graph toolkit uses Datalog to specify its reasoning logic. An attack step is just an instantiation of a Datalog rule. Thus instead of trimming useless attack steps from the produced attack graphs, we add one more constraint to each MulVAL rule such that the constraint will fail for useless attack steps. This is best illustrated through an example. The following is a MulVAL rule for multi-host network access.

```
netAccess(H2, Protocol, Port) :-
        execCode(H1, _Perm),
        reachable(H1, H2, Protocol, Port).
```

This rule states that if an attacker can execute arbitrary code on machine `H1` at some permission level, and machine `H1` can reach machine `H2` through `Protocol` and `Port`, then the attacker can access machine `H2` through `Protocol` and `Port`. In this way, privileges on `H1` can enable an attacker to access `H2`. The host transition associated with this rule is `H1` → `H2`. All the terms such as `H1` and `Protocol` are variables that can be instantiated with concrete terms for a specific attack step. MulVAL's reasoning engine can automatically find all the true instantiations derivable from the rules and input tuples.

We add one new subgoal, `advances(H1, H2)`, to this rule:

```
netAccess(H2, Protocol, Port) :-
        execCode(H1, _Perm),
        reachable(H1, H2, Protocol, Port),
        advances(H1, H2).
```

The `advances` predicate specifies that the transition `H1` → `H2` has to be valid for the rule to be successfully fired. We then write separate declarations for the `advances` predicate (also in Datalog) so that it returns true only when the transition `H1` → `H2` is useful. There is also a "non-trimming" option, in which the `advances` predicate will always return true and so a complete attack graph can be obtained.

The information needed to compute the `advances` predicate includes the machine reachability information, which is already computed by the MulVAL engine, and the information about what machine is in which subnet, which is included in the input to MulVAL. We wrote a simple program to compute dominator and post-dominator trees for the subnet graph in Python. The computation for determining useless transitions was implemented as Datalog programs. We employ the tabling techniques in XSB [21] (the underlying logic engine for MulVAL) with the `advances` tuple, to ensure that each tuple is computed only once. This tabling guarantees that very little is added to MulVAL's runtime.

B. Experimentation results

We applied the above trimming algorithm to the example shown in Section III and got satisfactory results. As previously noted, the untrimmed attack graph contained 150 nodes and 173 edges; after applying the trimming algorithm, the attack graph contained only 99 nodes and 106 edges.

Applying the algorithm to the example trims all three cases previously described in Section III, among many other useless edges. It is important to note that all useful nodes and edges are retained, so the attack graph now shows three key attack paths to reach the Citrix server, from which the control network subnet is accessible:

- Internet → web server → file server → Citrix server
- Internet → VPN server → workstation → Citrix server
- Internet → VPN server → Citrix server

This algorithm constrains the attack graph to include only useful edges both within each subnet and between different subnets. In this way, we eliminate the recognizably useless edges, reducing the number of edges and nodes appearing in the attack graph and thus simplifying the information produced.

C. Summary

Network configuration is a difficult task to manage because of the complexity. Human intervention is required to make appropriate decisions in consideration of both network security and usability. To aid in this decision, we have described how to present data that more clearly indicates the problematic aspects of the current configuration. A human user can utilize this smaller result set more quickly to the same level of effectiveness, which can help to realize the vision depicted in Figure 1.

V. Reconfiguration using SAT Solving

Trimming attack graphs reduces the amount of data shown to a human user, simplifying the presentation of possible attack paths through an enterprise network, but the trimming algorithm cannot suggest possible solutions to the vulnerabilities thus exposed. Since any network misconfiguration is technically resolvable (if only by removing all inter-machine access), options for reconfiguration must be dependent on the cost of the changes needed, as measured in time required and impairment to current usability. We developed an approach based on advanced SAT solving techniques that can automatically suggest optimal configuration changes that can address the security problems presented in an attack graph. We use the complete attack graphs, not the trimmed ones, for automatic configuration generation, since we would like the suggested configuration changes to address all the attack possibilities discovered. Our approach allows a user to provide feedback to the SAT solver so that constraints on usability, cost of deployment, and potential damage due to successful attacks can all be optimized in a unified framework.

A. Transforming attack graphs to Boolean formulas

We first extract the causality relationships represented in a dependency attack graph and express them as a Boolean formula. This is best explained through an example. In the dependency attack graph of Fig 2, the AND node e_1 means that the exploit e_1 is successful since all of its children p_2, c_1, c_2 are enabled, and the result of the exploit is that the attacker can gain privilege p_1. This can be expressed by the following formula.

$$p_2 \wedge c_1 \wedge c_2 \Rightarrow p_1$$

Or, equivalently,

$$\neg p_2 \vee \neg c_1 \vee \neg c_2 \vee p_1$$

We do this for all the AND nodes in the attack graph, and get the following clauses:

$$\begin{aligned} e_1 &= \neg p_2 \vee \neg c_1 \vee \neg c_2 \vee p_1 \\ e_2 &= \neg c_2 \vee \neg c_3 \vee p_1 \\ e_3 &= \neg c_3 \vee p_2 \end{aligned}$$

Let $\phi = e_1 \wedge e_2 \wedge e_3$, then ϕ is a Boolean formula in CNF whose size is linear in the size of the attack graph[2]. Some of the Boolean variables, such as c_1, c_2, c_3, represent configuration conditions in the enterprise systems (they are sink nodes in the attack graph). Some of the variables, such as p_1 and p_2, represent an attacker's privilege. ϕ encodes all the causality relationships between configuration settings and the various attacker privileges in the attack graph. For example, if all of c_1, c_2, c_3 are assigned the truth value T (as in the current configuration), then both p_1 and p_2 have to be assigned T to make a satisfying assignment for the formula. Therefore, if one wishes both p_1 and p_2 to be false (meaning an attacker can gain neither privilege p_1 or p_2), some of c_1, c_2, c_3 have to be assigned F, meaning some of the current configuration settings need to be changed. Let $\psi = \phi \wedge \neg p_1 \wedge \neg p_2$; then seeking a satisfying assignment to ψ amounts to finding configuration settings that can prevent an attacker from gaining privileges p_1 and p_2.

When a sink-node variable is assigned F, we say that it is "disabled". For example, if a sink node represents an existence of a software vulnerability on a machine, the negation of that node means patching the vulnerability. If a sink node represents a reachability relationship between two machines, the negation of the node means blocking the access between the two machines (e.g., by changing firewall settings).

If we feed ψ to a SAT solver, we could get a satisfying assignment that disables all the sink nodes c_1, c_2, c_3. This is certainly not an optimal solution, and is analogous to the trivial solution to securing an enterprise system – blocking all access and shutting everything down. A more careful observation on the attack graph reveals that if one just disables c_3 but leaves c_1 and c_2 true, it can still prevent all the attack paths in the system and it is probably a cheaper solution than disabling all the sink nodes, since less changes are needed. Indeed, the truth-value assignment $(c_1 \rightarrow T, c_2 \rightarrow T, c_3 \rightarrow F, p_1 \rightarrow F, p_2 \rightarrow F)$ is also a satisfying assignment for ψ. Thus, we not only want to find a satisfying assignment for ψ, but one that is also, in some sense, optimal.

One might think that the optimal solution should be a satisfying assignment that disables the fewest sink nodes. But this may not be the case depending on the various meanings of those nodes. For example, if c_3 means HTTP accessibility to the enterprise's web server,

[2]In MulVAL's attack graphs an AND node has only one parent, so each AND node is transformed into exactly one clause. An MulVAL attack graph's size is quadratic in the size of the network (number of hosts) [19].

disabling it may have a high cost due to disrupted business operation. On the other hand, c_1 could mean the accessibility from the web server to the internal subnet, and c_2 could mean a vulnerability on a server for which a patch is already available. Blocking web server access to internal subnet and patching the vulnerability may have a lower cost than blocking HTTP access to the web server. Then one might choose to disable c_1 and c_2 to prevent an attacker from obtaining privilege p_1. However, the truth-value assignment $(c_1 \rightarrow F, c_2 \rightarrow F, c_3 \rightarrow T, p_1 \rightarrow F, p_2 \rightarrow F)$ won't be a satisfying assignment to ψ, since the constraints on ϕ will force p_2 to be true if c_3 is assigned the truth value T. This means that if we choose to disable only c_1 and c_2, the attacker would still be able to gain privilege p_2. But suppose p_2 represents a privilege that is more tolerable than disallowing the HTTP access to the web server. Do we still want to require that p_2 has to be false?

In configuring an enterprise network, we not only want to compare the cost of various hardening measures, but also want to compare those costs against the potential damage a successful attack may incur. If the cost of the hardening measures is too much higher than the cost of the potential damage, it could be a better solution simply to acknowledge and tolerate the possibility that an attacker can obtain some privilege on the enterprise system. Thus, we need to relax the requirement that all attack paths in the attack graph need to be prevented. In the above example, we may want to tolerate the possibility that an attacker can gain privilege p_2. Thus the formula will become $\psi' = \phi \wedge \neg p_1$. The reconfiguration problem can be cast into the problem of finding a satisfying assignment for ψ' such that the combined cost of the hardening measures and the potential damage is minimized. The optimal results may contain some trade-off between applying hardening measures and tolerating possible attacks. Some of the attacker privileges will remain possible in the solution, in the form of T assignment to the corresponding Boolean variables.

B. MinCostSAT

MinCostSAT is a SAT problem which minimizes the cost of the satisfying assignment [9]. Mathematically, given a Boolean formula ψ with n variables $x_1, x_2, \ldots, x_n$ with cost $c_i \geq 0$, find a variable assignment $X \in \{0,1\}^n$ such that X satisfies ψ and minimizes

$$C = \sum_{i=1}^{n} c_i x_i$$

where $x_i \in \{0,1\}$ and $1 \leq i \leq n$.

MinCostSAT has been thoroughly studied by the SAT solving community [2], [14], [9], [4]. Although the problem is NP-hard, modern SAT solvers have been very successful in practice, being able to handle Boolean formulas with millions of variables and clauses in seconds. We use the MinCostChaff solver [4] which is a MinCostSAT solver based on the zChaff SAT solver [13].

The MinCostSAT problem minimizes the cost for variables that are assigned T. This matches the semantics for internal-node variables, whose T assignment means an attacker can gain some privilege and thus will incur some cost. But for sink-node variables, the cost is incurred when it is disabled, or assigned F. Thus we first transform our formula so that we use the negation of a Boolean variable to represent a sink node. This way when the variable is assigned T, it means the corresponding sink node is disabled which will incur some cost. For the example we used, the new formula derived from the attack graph will be $\tilde{\phi} = \widetilde{e_1} \wedge \widetilde{e_2} \wedge \widetilde{e_3}$, where

$$\begin{array}{rcl} \widetilde{e_1} & = & \neg p_2 \vee \widetilde{c_1} \vee \widetilde{c_2} \vee p_1 \\ \widetilde{e_2} & = & \widetilde{c_2} \vee \widetilde{c_3} \vee p_1 \\ \widetilde{e_3} & = & \widetilde{c_3} \vee p_2 \end{array}$$

Here $\widetilde{c_1}, \widetilde{c_2}, \widetilde{c_2}$ are the new Boolean variables for the sink nodes, whose T assignment will mean the sink node is disabled. Let $\tilde{\psi} = \tilde{\phi} \wedge \neg p_1$, the MinCostSAT solution to $\tilde{\psi}$ would be the desired solution for the enterprise system's reconfiguration, if we have defined the costs correctly for all variables.

C. Cost function declaration and iterative refinement

For the MinCostSAT algorithm to produce desired configuration suggestions, numeric costs for each configuration change and each possible attacker privilege need to be specified consistently. How to assign those costs is not obvious since many of the factors are subjective judgments. However, those subjective judgments are important in reaching configuration decisions and thus must be quantified and incorporated into an automatic configuration generator. We propose an incremental approach for a user to specify and refine those costs as logical declarations. This way a user does not need to have all the costs ready before he can use the tool. He can start with a very general and coarse-grained policy.

For example, one may decide that blocking access using firewall rules has a lower cost than patching a vulnerable service application, and that patching a service application has a lower cost than disabling it. One may also decide that allowing an attacker privileges on the internal subnet has a higher cost than privileges on the DMZ. The following rules can describe such a policy.

```
cost(reachable(H1, H2, Protocol, Port), 10).
cost(vulExists(H, VulID, Program), 100).
cost(networkService(H, Program,
                Protocol, Port, User), 200).
cost(execCode(H, Privilege), 400) :-
           inSubnet(H, internal).
cost(execCode(H, Privilege), 250) :-
           inSubnet(H, dmz).
```

Such Prolog declarations for cost functions are easier to understand than the raw mapping from Boolean variables to numbers. One can leverage the expressive power of Prolog to assign cost values to various configuration conditions and attacker privileges, to whatever degree of detail desired by the user. Using only a single clause, identical costs can be assigned, for example, to all machines in the internal subnet reachable from the VPN server. It is straightforward to compute the cost for each variable from such declarations in the XSB Prolog system. After the MinCostSAT solver returns a satisfying assignment to the boolean variables, the assignment is mapped back to the corresponding configuration changes and potential attacker privileges still remaining in the system. This assignment will have a provably minimum cost according to the cost function specified by the system administrator.

When viewing the results, it is likely that the system administrator will find the need for changes in some cost values to enforce previously unspecified policy assumptions. For example, even though changing firewall rules to block access is generally cheaper than patching, if the blocked access is the HTTP traffic to the company's web server, it may have a much higher cost in terms of loss in business revenues. When the system administrator sees a suggestion that access to the web server be blocked, he will realize that the cost function he specified does not encompass all the local requirements. He can then modify the cost declaration so that the Boolean variable for `reachable(internet, webServer, tcp, 80)` is assigned a higher cost.

```
cost(reachable(internet, webServer,
                         tcp, 80), 300).
```

The system administrator may need to do this for a number of rounds to refine the cost assignments until viable changes to the configuration are suggested. Then he can save the cost declaration as his configuration policy and use it again next time he needs to make changes in the enterprise system to address new security problems, at which time he will likely be able to obtain a satisfactory result from the MinCostSAT solver with little modification to the iteratively-developed policy.

D. User queries

With the expressiveness of Boolean formulas and the power of a SAT solver, a system administrator can ask questions like "what is the best way to reconfigure my system if I want to guarantee that the file server will not be compromised?" This can be done by forcing the Boolean variable x that corresponds to the privilege `execCode(fileServer, someUser)` to be false, i.e. conjoining $\neg x$ to the original formula. He can also ask questions like "Can I make the file server secure while allowing web server to be accessed from the Internet?" We have implemented mechanisms that allow an system administrator to specify those additional constraints for the various queries he would like to conduct. Those constraints can be straightforwardly specified in Datalog and automatically transformed into additional clauses in the Boolean formula to be solved by the MinCostSAT solver. This kind of constraint can also become a part of the configuration policy. For example, a user might decide that the web server must be accessible from the Internet. If this setting is forced to be true, MinCostSAT will never return a suggested reconfiguration that requires this access to be removed. Similarly, potential attacker privileges can be forced to be always false; for example, a user might decide that an attacker should never access the data historian, and so this access could be forced to be false, meaning that MinCostSAT will never allow it to be true. This effect could also be simulated by assigning unrealistically high costs for those variables, but by forcing them to be true or false, there will never be a suggestion that reverses this specification.

E. Experiment on the example of Section III

Looking at the example in Section III, we began with a simple configuration policy that assigned an equal cost to all changes in basic settings and to all potential attacker privileges. Based on this policy, running MinCostSAT produces recommendations to remove access from the internet to the webServer and vpnServer as well as removing network services from citrixServer, vpnServer, and workStation, essentially cutting off all outside connections to the network. We then started refining the cost functions to make it aware of realistic requirements on usability. In our testing, it took only three to four iterative steps to produce a realistic suggestion. We began by assigning high costs to changes in desired network access permissions and network services, as well as varying costs for allowing an attacker access to machines in various subnets, with highest costs in the control network subnet, lower in the internal subnet, and still lower in the DMZ. In this way, the policy asserted that there were higher potential costs in allowing an attacker access further into the system. After several more iterations, re-assessing and reassigning costs, the suggested changes are to patch the existing vulnerability in webServer and either remove one employee's account on the VPN server or ensure that the employee's log-in information will not be compromised. Acting on these suggestions, an administrator could patch the vulnerability and spend some time ensuring that the employee understands good security procedures, such as using strong passwords. Based on the costs we used in our policy, MinCostSAT has determined that an attacker might be allowed minimal access to the webServer or vpnServer but will be prevented from accessing any machine in the internal and control network subnets.

The policy thus produced can be saved for re-use in the future. We added to our example a second web server and file server, with a remote exploitation vulnerability in the newly-added web server. Running MulVAL and MinCostSAT again, using the same cost policy, immediately showed that this vulnerability must be patched. In this way, it took much less effort to produce useful results once a baseline policy was well established.

It is important to note that assigning different costs can easily produce different suggestions. For example, if the cost assigned to patching the vulnerability in the web server was sufficiently high, the suggested solution might be to change the accessibility from the web server into the internal subnet and/or to remove the file sharing relationship between the Citrix server and the file server. The cost for changing a specific configuration parameter and the cost caused by a potential attacker privilege will vary from one organization to another. There is no one-size-fits-all cost function suitable for all enterprise systems, and the user of the tool will have to define costs based on local requirements and policies.

F. Scalability

While SAT solving is theoretically NP-hard, modern SAT solvers typically perform well for Boolean formulas arising from practical problems. To test the scalability of our approach, we constructed simulated enterprise networks with two different sizes:

I: 100 host machines, evenly divided in 10 subnets
II: 250 host machines, evenly divided in 25 subnets

We also tested using two different cost functions:

A: All clauses were assigned an equal cost. The effect of this cost policy would be simply to minimize the number of configuration changes made plus the number of compromised machines.
B: Privilege to execute code on a machine has a cost that rises deeper into the network, blocking network access to hosts or disabling network services has significant cost, and all other changes have low cost.

The test was conducted on a Linux machine with Opteron Dual-Core 2214 2.2 GHz CPU, with 16GB memory, and running gentoo Linux with kernel version 2.6.18-hardened-r6.

Sz	Cfn	# variables	# clauses	time (sec)
I	A	11,853	12,053	0.11
I	B	11,853	12,053	0.21
II	A	70,803	72,553	3.03
II	B	70,803	72,553	6.49

The simulated networks on which we performed the above tests were certainly not representative of realistic enterprise network settings. But the performance indicates that modern SAT solvers are likely to be powerful enough to handle the configuration management problem we described in this paper.

VI. Discussions and Future Work

The two problems we addressed in this paper – making attack graphs more human-readable and generating suggestions for configuration changes – are both important to achieve automation in enterprise network security management. Since configuration management is such an important and complex problem, it is unlikely that an automatic tool can take over a human system administrator's role overnight. Rather, we envision an incremental process whereby automated tools can assist a human user to manage an enterprise network, increasing the degree of automation as the technology matures. An easy-to-understand attack graph that presents straightforward information about a system's security vulnerability can provide immediate benefit to a human user, and is the first step in the long journey towards complete automation of enterprise security management.

Since security management has to factor in constraints from usability requirements and the cost of hardening measures, as well as cost of potential damage from successful attacks, there is a great deal of vagueness in a human's decision-making process. An automatic tool needs to be "trained" so that the implicit policy assumptions become explicit in the form of formally defined policy declarations that can be used to automatically search for a desirable solution to address discovered security threats. We adopted an iterative approach for configuration policy declaration and refinement, allowing a human user to train the configuration generator by providing logic statements on the various costs for security hardening measures and losses from security compromises. We intend to further study how to better specify the cost functions so that a human can more easily and more accurately express the various requirements and constraints one may encounter in configuring realistic enterprise systems.

Security hardening measures also include detection mechanisms such as Intrusion Detection Systems (IDS). An IDS cannot completely prevent an attack from happening, but, if used appropriately, it can help contain the damage from a true attack. Currently the effect of an IDS is not modeled in the MulVAL system, and as a result, our optimization process cannot automatically factor in the cost/benefit of deploying IDS systems at various locations in an enterprise system. We will study how to model the effect of an IDS in the reasoning framework of attack graphs and the optimization framework we have developed.

VII. Related Work

Wang *et al.* [23] developed a graph search-based method for network hardening, with previous work by Noel *et al.* [18]. Their approach is different from ours. An attacker's goal is expressed as a propositional formula on the initial conditions through recursive substitution during graph traversal. The formula is then converted to DNF which will give all possible hardening options, from which an optimal one is chosen. The converted DNF could be exponential in the size of the attack graph, as admitted in the paper. In our approach, we formulate the optimization problem as a MinCostSAT problem on a Boolean formula whose size is *linear* in the size of the attack graph, and applies a well developed modern SAT solver to find the solution. This is likely to yield better performance since SAT solvers have been very successful in efficiently finding solutions to large Boolean formulas arising from practical problems. Moreover, our approach also allows for specifying a variety of optimization problems that not only trade off between various hardening measures, but also trade off between the hardening cost and potential loss due to a successful attack.

Dewri, *et al.* [3] formulates the security hardening problem as a *multi-objective optimization* problem, in which the cost for security hardening and the cost for potential damage caused by successful attacks are two objectives in the multi-objective optimization problem, whose solution is searched for by a genetic algorithm (GA). We adopted a different approach, MinCostSAT, to find a *provably* minimum-cost solution to the configuration problem. GA, on the other hand, cannot always guarantee to converge to the global optimum. MinCostSAT is a specific optimization problem that has been studied extensively and thus is likely to outperform a general optimization algorithm such as GA for the specific problem. Adopting the SAT solving approach also allows the user to make various queries we discussed in the paper, in addition to finding the optimal solution for reconfiguration. The benefit of multi-objective optimization is that a user can be presented multiple optimal trade-offs between the two objectives. It will be interesting to study whether the MinCostSAT techniques can be extended to provide multiple trade-off solutions for the optimization problem.

A number of other previous works addressed the problem of how to use attack graphs to better manage the security of enterprise networks [7], [12], [10], [15], [24]. The observations and insights from these previous works helped us develop the approach in this paper, and our work either complements or improves upon them. Our contribution is the development of formal, logic-based approaches to simplifying an attack graph for a human to better understand, and to using the unsimplified complete attack graph to compute reconfiguration suggestions automatically, taking into account not only security requirements, but also requirements on usability and trade-offs between costs of security hardening and costs of possible loss due to successful attacks.

SAT solving has been used to address general configuration management problem in Narain's work [16]. The general configuration problem concerns various requirements on service availability, performance, fault-tolerance, and security. The full first-order logic is used to model the various constraints arising from those requirements. A solution (model) to the first-order formula is then searched for by the Alloy[3] model-finder, which subsequently converts the problem to Boolean formulas and calls a SAT solver. Our work addresses configuration problem that concerns another important requirement (i.e., robustness against potential attacks) and we use a Boolean formula converted from a dependency attack graph to express the various constraints arising from

[3]http://alloy.mit.edu/

this requirement. It is likely that the two technologies can be integrated into a unified framework to address a more complete set of requirements regarding enterprise network configuration management.

VIII. Conclusion

The system vulnerabilities identified by an attack graph are useful only inasmuch as they are actionable. If a user cannot easily prioritize the system problems and act upon them, the system remains vulnerable to attack. The contribution of this paper is to utilize attack graph data to suggest changes in network configuration to mitigate threats and avoid attacks. We have introduced methods by which the data presented in the attack graph can be made more succinct and usable and also methods by which network configuration changes can be automatically identified and suggested. We believe that the joint effect of these propositions is to create an attack graph better suited for review by a human user and to provide quick and verifiable suggestions as to how the network configurations can be altered to make the network more secure. Benefits are gained both by automating computation-intense aspects of this approach and by retaining human intervention to provide subjective decisions according to the data presented.

References

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[2] Olivier Coudert. On solving covering problems. In *33rd Design Automation Conference (DAC'96)*, pages 197–202, 1996.

[3] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[4] Zhaohui Fu and Sharad Malik. Solving the minimum-cost satisfiability problem using sat based branch and bound search. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'06)*, San Jose, CA, USA, 2006.

[5] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[6] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.

[7] Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 49–63, Nova Scotia, Canada, June 2002.

[8] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

[9] Xiao Yu Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem.* PhD thesis, North Carolina State University, Raleigh, North Carolina, 2004.

[10] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and restoring defense in depth using attack graphs. In *Military Communications Conference (MILCOM)*, Washington, DC, U.S.A., October 2006.

[11] Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory, March 2005.

[12] Richard P. Lippmann, Kyle W. Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Artz, and Robert Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, October 2005.

[13] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004: An efficient SAT solver. In *Lecture Notes in Computer Science SAT 2004 Special Volume*, pages 360–375. LNCS 3542, 2004.

[14] Vasco M. Manquinho and Jo ao P. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithmsfor the binate covering problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21:505–516, 2002.

[15] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking attack graphs. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, September 2006.

[16] Sanjai Narain. Network configuration management via model finding. In *Proceedings of the 19th conference on Large Installation System Administration Conference (LISA)*, 2005.

[17] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004. ACM Press.

[18] Steven Noel, Sushil Jajodia, Brian O'Berry, and Michael Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC)*, December 2003.

[19] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.

[20] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[21] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David S. Warren, and Juliana Freire. XSB: A system for efficiently computing well-founded semantics. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

[22] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.

[23] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost

network hardening using attack graphs. *Computer Communications*, 29:3812–3824, November 2006.

[24] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using attack graphs. In *Third Workshop on Quality of Protection (QoP)*, 2007.