Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs

Steven Noel, Sushil Jajodia, Brian O'Berry, Michael Jacobs
Center for Secure Information Systems, George Mason University
{snoel, jajodia, boberry, mjacobs1}@gmu.edu

Abstract

In-depth analysis of network security vulnerability must consider attacker exploits not just in isolation, but also in combination. The general approach to this problem is to compute attack paths (combinations of exploits), from which one can decide whether a given set of network hardening measures guarantees the safety of given critical resources. We go beyond attack paths to compute actual sets of hardening measures (assignments of initial network conditions) that guarantee the safety of given critical resources. Moreover, for given costs associated with individual hardening measures, we compute assignments that minimize overall cost. By doing our minimization at the level of initial conditions rather than exploits, we resolve hardening irrelevancies and redundancies in a way that cannot be done through previously proposed exploit-level approaches. Also, we use an efficient exploit-dependency representation based on monotonic logic that has polynomial complexity, as opposed to many previous attack graph representations having exponential complexity.

1. Introduction

In the analysis of network vulnerability to attack, considering vulnerabilities in isolation is insufficient. This is because attackers often combine exploits against multiple vulnerabilities in order to reach their goals. While a single vulnerability may not pose a significant threat to a network, a combination of vulnerabilities may. Thus even well administered networks can be vulnerable to attacks, because of the security ramifications of offering a variety of combined services.

An approach to this problem is to build a model of global network security, e.g., as a state machine with security conditions as variables and attacker exploits as transitions. Various methods have been proposed for finding attack paths (sequences of exploit state transitions) in such models, including symbolic model checker (logic-based) approaches [1][2][3][4], and graph-based approaches [5][6][7][8][9]. However, such methods generally have serious scalability problems, since they must contend with the exponential complexity of the full security state search space.

More recently [10][11], it has been recognized that under an assumption of monotonic logic, it is not necessary to represent attack paths (usually organized as graphs) explicitly. Instead, the dependencies among exploits and security conditions encode the same information provided by attack graphs. Monotonic logic leads to an efficient (low-order polynomial) exploit dependency graph representation that scales well. Semantically, monotonic logic simply means that the attacker need not relinquish resources already gained in order to further advance the attack. This is a valid modeling decision, corresponding to the observation that the control that attackers exert over networks effectively increases monotonically over time.

Attack graphs (and even exploit dependency graphs) show sequences of exploits, which may be useful for applications that focus on the attacks themselves. But network administrators usually don't care about exploit sequences – they just want to know the best way to harden their network. What is needed is an explicit and manageable set of network hardening options that provide a guarantee for the safety of given network resources.

In this paper, we go beyond attack paths and exploit dependency graphs to compute actual network hardening options. In addition to guaranteeing safety, these hardening options incur minimal overall cost, assuming relative costs have been given for individual hardening measures. Previous approaches have addressed this problem by computing minimal critical sets of exploits [3][4][10]. But such approaches ignore the generally complex relationships among exploits and elements of the network configuration. Our approach works directly with configuration elements, resolving hardening irrelevancies and redundancies in a way that cannot be done through exploit-level approaches.

While the possible number of hardening options can be large, we greatly reduce the number of choices by selecting only those with minimal impact on the network. In particular, a "minimal" safe network configuration, in which a given set of components are hardened, allows us to ignore all other configurations in which supersets of these same components are hardened.

These minimal-impact configurations lead directly to minimum-cost hardening options. That is, the only costs incurred are those associated with hardened network components, and our minimal-impact configurations correspond to fewer numbers of such components. Given that the network administrator assigns relative costs for individual hardening measures, we select the configuration with the lowest total cost.

As we describe in subsequent sections, we use an efficient exploit dependency graph representation in computing network hardening measures. In building the exploit dependency graph, we resolve cycles and other redundancies via the graph distance from initial conditions, as consistent with monotonic logic.

From the exploit dependency graph, we compute an expression of the safety of given network resources, in terms of possible assignments of initial network conditions. We then compute minimal-impact network configurations via the minimal elements of the conjunctive-normal-form partial ordering. From these minimal-impact configurations, we then find the configuration with minimum total cost.

2. Problem

We consider the problem of computing minimum-cost hardening measures that guarantee network safety. In this problem, we model the presence of a network security condition as a Boolean variable. For example, if some condition represents a vulnerable version of a software component on a particular machine, the condition being true means the component is present and the condition being false means it is not present. Under the assumption of monotonicity, a condition may go from false to true, but may not go from true to false. That is, once a condition contributes to the success of an exploit (or overall attack), it will continue to do so.

Next, we model the success of an attacker exploit as a Boolean function of some set of conditions. While it is possible for such an exploit to take on a general Boolean form, for simplicity we constrain it to a conjunction (Boolean ANDs). There is no loss of generality here, since if an exploit requires disjunction (e.g. more than one version of a vulnerable program), we simply divide the disjunctive portions into separate (conjunctive) exploits and consider them separately.

The success of an exploit then causes another set of conditions to become true. In other words, an exploit is a mapping from its *preconditions* to its *postconditions*, such that if all its preconditions are true then all its postconditions become true.

Given a network-attack model, the next step is to determine how the application of exploits impacts network vulnerability. As described in Section 1, previous work in this area has generally focused on generating attack paths that lead to compromise of a given critical resource. That is, some set of conditions is designated as the goal of the attack. Distinct sequences of exploits (attack paths) are then generated such that each sequence leads to the attack goal becoming true.

The general idea is that one could use attack paths (arranged as a graph) to determine network-hardening measures. However, such attack graph representations have high complexity, so that this approach does not scale well.

In this paper, we go beyond attack graphs, computing sets of network hardening measures via efficient exploit dependency graphs. That is, given a set of initial conditions, we wish to compute assignments of those conditions that guarantee the safety of a set of attack goal conditions. Moreover, we wish to compute hardening measure assignments that have minimum cost, corresponding to minimizing assignments of false to initial conditions, i.e., minimizing the number of hardening measures to be taken.

Here it is important to distinguish between two types of network security conditions. One type of condition appears as an exploit precondition only. The only way that such conditions can be true is if they are true initially, since they are postconditions of no exploit. These initial conditions are precisely the ones we must consider for network hardening measures.

The other type of condition appears as both exploit preconditions and postconditions. We cannot consider such conditions for network hardening, since they are not under our strict control, i.e. attacker exploits can potentially make them true despite our hardening measures.

In computing minimal-impact hardening sets, one set can be chosen over another if all its assignments of false also appear in the other set. This is true because the selected set represents a safe assignment with fewer hardened network components. By retaining only these minimal-impact hardening sets, we minimize hardening cost and greatly reduce the number of sets to consider.

If the network administrator has no *a priori* way to assign individual hardening costs, we could simply compute all possible minimal-impact sets. Armed with all possible sets, the administrator could then select the best one. If we further assume that individual hardening costs are equal, we could reduce the set of all possible minimal-impact assignments to a single best minimal-impact assignment. This assignment would thus be the one with the fewest hardening measures, i.e., the one with the fewest assignments of false initial conditions.

We can also consider the more realistic scenario that different hardening measures incur different costs (although it is not always trivial to assign such costs). Assuming that costs are independent and combine linearly, the overall cost of a particular hardening

assignment is just the sum of the costs of the individual hardening measures taken. In this way, from among all minimal-impact hardening sets, we choose the single set whose total cost is lowest.

3. Approach

We begin with a set of exploits in terms of security conditions. We then build a directed graph of the dependencies among exploits and conditions, via exploit preconditions and postconditions.

We build the dependency graph through a multi-step process, implemented through a custom Java application. We first build a dependency graph starting from an "initial conditions" exploit, i.e., an exploit with the initial network conditions as its postconditions (and null preconditions). The resulting graph represents forward dependencies from the initial conditions.

In building the forward-reachable dependency graph, we resolve cycles through the graph distances (number of vertices in shortest path) that given conditions are from the initial conditions. That is, preference is given to dependency edges that are closer to the initial conditions. Because such a resolved dependency graph is sufficiently well behaved, we can use it to construct a closed-form expression for the attack goal in terms of the initial conditions.

Actually, the resolution of cycles is part of a more general resolution of postcondition redundancies. That is, there is no reason to cycle among exploits if their postconditions remain true after an initial exploit execution, neither is there reason to execute exploits whose postconditions have already been met. Cycles and other redundancies are common in real networks, and are violations of monotonicity that must be resolved. Indeed, in the real world, attackers themselves would avoid such redundancies.

Next we do a backward traversal of the forward-reachable dependency graph, starting from the "attack goal" exploit, i.e., an exploit with the attack goal(s) as its preconditions (and null postconditions). The resulting dependency graph includes exploits that are not only reachable from the initial conditions, but are also relevant to (i.e. backward reachable from) the attack goal. In fact, this dependency comprises the necessary and sufficient set of exploits with respect to the initial and goal conditions, i.e., all exploits can actually be executed, and all exploits contribute to the attack goal. This represents the set of minimal attack paths, in which no exploit can be removed without impacting the overall attack.

Given an exploit dependency graph, we then construct an expression for the goal conditions in terms of the initial conditions. This involves the recursive algebraic substitution of dependency terms in a backward direction, starting from the goal-condition exploit. That is, we start from the goal exploit, and algebraically substitute it with the conjunction of its preconditions.

We then substitute each of the goal-condition preconditions with the exploit that yields it as a postcondition, since these are logically equivalent. In the event that more than one exploit yields this postcondition, we form the disjunction of all such exploits, since logically any one of them could provide the postcondition independent of the others.

We continue in a recursive fashion, substituting the newly generated exploit expressions in exactly the same way we treated the goal-condition exploit expression. In doing this recursive algebraic substitution, we make direct use of the exploit dependency graph by traversing it breadth-first. Recursion must end when the initial-condition exploit is encountered. It is precisely at this point that an initial condition has been added to the expression. Once the dependency graph has been fully traversed, the result is an expression for the safety of the attack goal in terms of the initial conditions.

The resulting expression for the attack goal allows us to decide if the goal is safe for a particular assignment of initial conditions. But we are concerned about the assignment version of this problem, i.e., actually finding sets of initial conditions that guarantee safety. This is much more useful from the standpoint of network vulnerability analysis, since it provides explicit sets of safe network configurations.

To solve the assignment problem, we rely on properties of the conjunctive normal form [12]. The canonical conjunctive normal form yields the attack goal written as a uniquely determined conjunction of maxterms, where each maxterm is a disjunction that contains all initial conditions.

Each maxterm corresponds to an assignment of initial conditions such that the attack goal is safe. In particular, if each non-negated initial condition in a maxterm is false, the attack goal is false (safe), independent of any other maxterm. The set of all maxterms thus represents all possible safe assignments.

To minimize hardening costs, we choose assignments with minimal hardening measures. More formally, we choose one maxterm over another if all of its non-negated conditions also appear non-negated in the other. In other words, we choose one maxterm over another if the non-negated conditions in the first are a (proper) subset of the non-negated conditions in the second.

Not all maxterms can be ordered according to this property, i.e., one maxterm's non-negated conditions may not be a subset of another's, so this is a partial order. In terms of this partial ordering of maxterms, we then choose *minimal* maxterms, i.e. those that are greater than no other maxterm. We choose these precisely because they correspond to minimal hardening measures.

While the total number of maxterms can potentially grow exponentially with the number of initial conditions, choosing only minimal-impact maxterms drastically reduces the number of sets to consider. Actually, the number of maxterms is critically dependent on the exact form of the attack goal expression. At one extreme, for a purely conjunctive expression, there is only one maxterm. At the other extreme, for a purely disjunctive expression with n initial conditions, there are $2^n - 1$ maxterms. But in general, there can be large numbers of maxterms, so it is crucial to reduce them in the way we describe.

If there are no known criteria for assigning individual hardening costs, we can simply stop after computing the full set of minimal-impact hardening options. The network administrator can then compare the various options and select the one that offers the best combination of offered services. We could go further by assuming that all hardening measures are equally costly. From this, we could simply choose the minimal-impact maxterm with the smallest number of unprimed (non-negated) terms.

However, we go even further in trying to capture the least costly set of hardening measures. In particular, the network administrator could assign costs to individual hardening measures. We then assume that the overall cost of a set of hardening measures can be accurately modeled as the sum of the individual costs. Thus we write the total cost $p(m_i)$ of maxterm m_i as

$$p(m_i) = \sum_j w_j^i a_j^i \ ,$$

with a_j^i being a primed (negated) initial condition within maxterm m_i with cost w_j^i. We then select the maxterm with the smallest total cost as the single best choice for network hardening.

4. Examples

In this section, we provide some example applications of our approach. We first work through a small example that has been described in previous work, which demonstrates each aspect of our approach. We then consider a slightly more elaborate example that underscores how our approach yields concise sets of hardening measures with minimal cost (e.g., minimal impact on network service availability), eliminating irrelevant security conditions in a way not possible through previous exploit-level approaches.

In the first example, the attacker's machine is denoted machine 0, and the two victim machines are denoted 1 and 2, respectively. The details of the attack scenario (such as network topology, available services, operating systems, etc.) are not needed here, although the interested reader can refer to [3] and [10] for such details. For the purpose of describing our approach, all that is really needed is the exploit dependency graph as described in the previous section.

Figure 1 shows the full attack dependency graph for this example, before applying the analysis described in the previous section. In the figure, exploits appear as ovals, and conditions appear as plain text (except the goal condition, which is marked with a triple octagon). Numbers in parenthesis identify associated machines. For example, *root*(2) denotes root privilege on machine 2, and *rsh*(2,1) denotes the execution of the *rsh* exploit from machine 2 to machine 1.

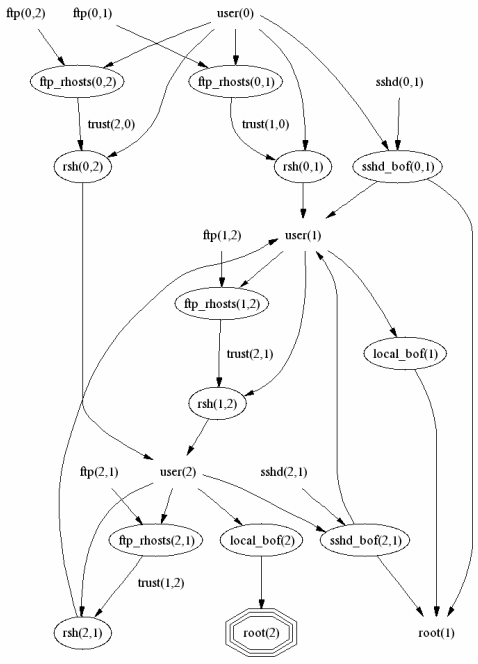

Figure 1: Exploit dependency graph for first example.

In comparison to the original examples in [3] and [10], one may notice that some exploit preconditions are missing. This is because we make some modeling changes to eliminate redundancy, which in turn reduces model complexity. For example, without loss of generality, we model the combination of transport-layer ftp connectivity, physical-layer connectivity, and the existence of the ftp daemon as simply transport-layer ftp connectivity (see [13] for details). Similarly, we model the combination of application-layer trust and physical-layer connectivity as simply application-layer trust. Also, we represent the privilege level of "none" implicitly, i.e. as the absence of a defined level.

The dependency graph in Figure 1 shows all exploits that can possibly be executed[1]. In the analysis of

[1] Note that Figure 1 omits exploits in which the attacker's own machine is the victim. That is, no exploits need be launched against the attacker's

hardening measures, we first remove initial condition *user*(0) from the graph, since this is the attacker's initial privilege on his own machine, which the network administrator cannot control. The forward analysis pass (described in the previous section) then detects 2 cycles:

1. $user(1) \rightarrow rsh(1,2) \rightarrow user(2) \rightarrow ftp\_rhosts(2,1) \rightarrow trust(1,2) \rightarrow rsh(2,1) \rightarrow user(1)$

2. $user(1) \rightarrow rsh(1,2) \rightarrow user(2) \rightarrow sshd\_bof(2,1) \rightarrow user(1)$

The postconditions *user*(1) of *rsh*(2,1) and *user*(1) of *sshd_bof*(2,1) are removed from the graph, since they have the highest graph distances within their respective cycles.

In the subsequent backward phase of analysis, the condition *root*(1) and the exploit *local_bof*(1) are removed from the graph, since they are not backward-reachable from the goal (i.e., they are a "dead end" in terms of the attack). Also, exploit *sshd_bof*(2,1) is removed in the backward pass since its remaining postcondition *user*(1) was removed in the forward pass because of a cycle. Similarly, exploits *ftp_rhosts*(2,1) and *rsh*(2,1) are removed in the backward pass because of the pruning of postcondition *user*(1) of exploit *rsh*(2,1) in the forward pass.

Figure 2 shows the exploit dependency graph resulting from the forward and backward analysis phases. As a comparison, note that the attack graph in [3] for this same example is considerably more complex (25 vertices and 43 edges), even though it uses an ordered binary decision diagram as a way to make the graph more compact. This is evidence in support of our position that exploit dependency graphs scale much better than traditional attack graphs. The reason is that the space of exploit dependencies (low-order polynomial) is much less complex than the space of security states (exponential).

For this example, we now construct an expression for the goal condition *g* in terms of the initial conditions, via traversal of the exploit dependency graph:

$$\begin{aligned} g &= root(2) = rsh(0,2) + rsh(1,2) \\ &= ftp\_rhosts(0,2) + ftp\_rhosts(1,2) \cdot user(1) \\ &= c\_ftp(0,2) + c\_ftp(1,2) \cdot user(1) \cdot user(1) \\ &= c\_ftp(0,2) + c\_ftp(1,2) \cdot [rsh(0,1) + sshd\_bof(0,1)] \\ &= c\_ftp(0,2) + c\_ftp(1,2) \cdot c\_ftp(0,1) + \\ &\quad c\_ftp(1,2) \cdot c\_sshd(0,1) \\ &\equiv p + qr + qs \end{aligned}$$

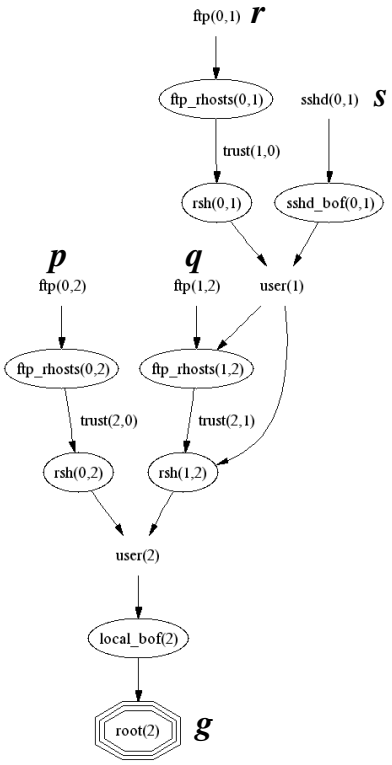

Figure 2: Resolved exploit dependency graph for first example.

Here, the plus symbol denotes disjunction (Boolean OR) and the dot symbol denotes conjunction (Boolean AND). Also, for notational simplicity we rename the initial conditions as $p \equiv ftp(0,2)$, $q \equiv ftp(1,2)$, $r \equiv ftp(0,1)$, $s \equiv sshd(0,1)$, and the goal condition as $g \equiv root(2)$.

Given the expression $g = p + qr + qs$ for the attack goal in terms of initial conditions, we then convert g to canonical conjunctive normal form. This yields the attack goal as a conjunction of maxterms, each maxterm representing a safe assignment of initial conditions. That is, the set of maxterms represents all possible safe assignments. For network hardening, each non-negated initial condition in a maxterm must be assigned false to provide safety.

Thus for the first example we have

$$\begin{aligned} g &= p + qr + qs \\ &= (p+q+r+s) \cdot (p+q+r+s') \cdot (p+q+r'+s) \cdot \\ &\quad (p+q+r'+s') \cdot (p+q'+r+s) \end{aligned}$$

Here the prime symbol denotes negation (Boolean NOT). In principle, we could then harden the network by selecting one of the maxterms, and take hardening

---

own machine, since all attacker capabilities can simply be modeled as initial conditions. The figure also omits the initial-condition and goal-condition exploits described in the previous section, which are algorithmically convenient but have little value for attack graph visualization.

measures that eliminate the vulnerability associated with each unprimed initial condition.

However, it is apparent that among the maxterms, not all are equally desirable. Generally speaking, there is some cost associated with each network hardening measure. We therefore seek maxterms that have minimal impact on the network (involve fewer hardening measures), which corresponds to minimizing unprimed initial conditions.

Thus, between a pair of maxterms, we choose one over another if all of the unprimed conditions in one also appear unprimed in the other. We can partially order maxterms in this fashion. Figure 3 shows the partial order of maxterms for this example.

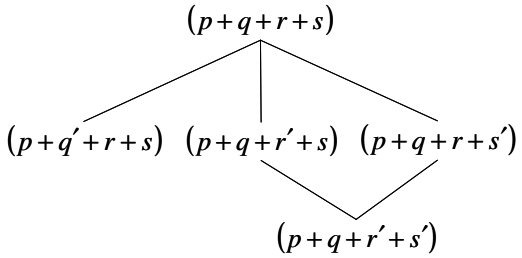

Figure 3: Partial order of maxterms for minimal-impact hardening measures (for first example).

For minimal-impact network hardening, we select the minimal maxterms in the partial order, i.e. those that have no maxterm below them. For this example, we select the 2 maxterms $(p+q+r'+s')$ and $(p+q'+r+s)$. These 2 maxterms correspond to 2 minimal-impact hardening options:

1. Harden *ftp*(0,2) and *ftp*(1,2), or
2. Harden *ftp*(0,2), *ftp*(0,2), and sshd(0,1)

The remaining maxterms, while still providing safety, have a greater impact on the network in terms of hardening measures.

In choosing among hardening options, our approach is for the network administrator to assign a cost to each individual hardening measure (initial condition). We then select the maxterm with the lowest total cost, where we sum only the costs for non-negated (unprimed) initial conditions.

In this example, since *ftp*(0,2) occurs unprimed in both maxterms, we can disregard it in comparing total costs. We must therefore compare the cost of hardening *ftp*(1,2) (Option 1) to the combined cost of hardening *ftp*(0,1) and *sshd*(0,1) (Option 2).

Since *sshd* represents a version of secure shell vulnerable to a buffer overflow attack, we might assume that its hardening cost is relatively low, e.g., installing a vendor patch. On the other hand, *ftp* represents the existence of a properly functioning ftp service, which is simply used by the attacker in a clever way. Thus the only way to harden this "vulnerability" is to block the service between the respective machines. In this case, the hardening cost is relatively high, i.e., the lack of service availability. So overall, the choice is dominated by the relative importance of offering the ftp service from machine 1 to machine 2 (Option 1) versus offering it from machine 0 to machine 1 (Option 2), perhaps with a slight bias toward Option 1, since Option 2 includes the low-cost patch to *sshd*.

We now consider a slightly more elaborate example of our approach, e.g., with attacks at multiple layers of the TCP/IP stack. While the purpose of the first example was to show each step of the process, this example focuses on the resulting hardening measures and how a complicated set of exploits can be resolved to a simple set of hardening measures. Through this example we also explain how previously proposed exploit set minimization approaches [3][4][10] are insufficient for network hardening, providing additional motivation for our approach.

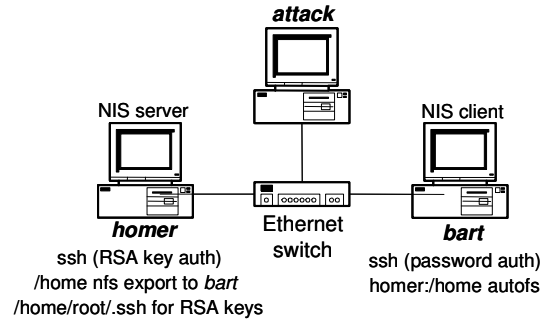

Figure 4: Network for second example.

Figure 4 shows the network for the second example. An Ethernet switch provides connectivity at the link layer. At the transport layer, unused services have been removed, secure shell replaces FTP, telnet and other cleartext password-based services, and there is tcpwrapper protection on RPC services. Application-layer trust relationships further restrict NFS and NIS domain access. The exploits and security conditions for this example are described in Table 1 and Table 2.

Table 1: Exploits for second example

Exploit	Description
arp_spoof	Spoof (impersonate) machine identity via ARP poison attack
ypcat_passwd	Dump encrypted NIS password file
crack_passwd	Crack encrypted user password(s)
scp_upload_pw	Secure shell copy, upload direction, using password authentication

Exploit	Description
scp_download_pw	Secure shell copy, download direction, using password authentication
ssh_login_pw	Secure shell login using password authentication
rh62_glibc_bof	Red Hat 6.2 buffer overflow in glibc library
create_nfs_home_ssh_pk_su	Exploit NFS home share to create secure shell key pair used for superuser authentication
ssh_login_pk_su	Secure shell login using public key authentication

Table 2: Security conditions for second example

Exploit	Description
link_arp	Attacker shares link-level connectivity with victim (both on same LAN)
trans_yp	Transport layer connectivity to NIS server
trans_ssh_pw	Transport layer connectivity to secure shell server that supports password authentication
trans_ssh_pk	Transport layer connectivity to secure shell server that supports public key authentication
trans_nfs	Transport layer connectivity to NFS server
app_nfs_home_su	Application "connection" representing sharing superuser's home directory
app_yp_domain	Application "connection" representing NIS domain membership
app_yp_passwd	Application "connection" representing acquisition of encrypted NIS password database
app_pwauth	Application "connection" representing acquisition of unencrypted user password
app_ssh_pk_su	Application "connection" representing acquisition/creation of key pair used for superuser authentication
pgm_glibc_bof	Program used to exploit glibc library buffer overflow vulnerability
execute	Execute access obtained
superuser	Superuser privilege obtained

Figure 5 shows the resulting exploit dependency graph. In this graph, we have removed initial conditions that the network administrator cannot control, and applied the forward and backward graph analysis passes, as described in the previous section.

As before, we traverse the exploit dependency graph to construct an expression for the attack goal g (*execute* access and *superuser* privilege on machine *homer*) in terms of the initial conditions:

$$g = (\alpha\beta\chi + \alpha\beta\chi\delta\varepsilon)\cdot(\phi\gamma)\cdot(\alpha\beta\chi)\cdot\eta$$
$$= \alpha\beta\chi\phi\gamma\eta$$

The exploit dependency graph has been reduced to an expression that leads to simple choices for network hardening. Note here that 2 initial conditions in the dependency graph do not appear in the expression for goal g:

1. $\delta \equiv trans\_ssh\_pw(bart, attack)$, and
2. $\varepsilon \equiv app\_pwauth(bart, attack)$.

These drop out in this fashion:

$$rh62\_glibc\_bof(bart, bart) = \alpha\beta\chi + \alpha\beta\chi\delta\varepsilon$$
$$= \alpha\beta\chi(1 + \delta\varepsilon)$$
$$\alpha\beta\chi$$

Through our approach, such irrelevant conditions as δ and ε do not get considered for network hardening. That is, the goal expression contains the necessary and sufficient set of initial conditions needed for network hardening decisions.

This kind of sufficiency is not present in previously proposed approaches to network hardening via proposed exploit set minimization [3][4][10]. These approaches search for minimal sets of (generic) exploits, where a minimal set is one in which every exploit is needed in reaching the goal.

In this example, there are 2 such minimal sets of exploits:

1. The set of all exploits except *scp_upload_pw*(*attack*,*bart*), and
2. The set of all exploits except *scp_download_pw*(*bart*, *attack*)

For network hardening using these minimal exploit sets, given no other information, one must assume that all exploits in the union of the minimal exploit sets must be stopped. In this example, one would conclude that the exploit *scp_download_pw*(*bart*,*attack*) must be stopped (via either δ or ε false in the initial conditions), even though stopping it has no effect on the attacker reaching the goal. Indeed, one could imagine that this exploit was the final one in a long chain of exploits, so that the entire chain would erroneously be considered critical to the attack.

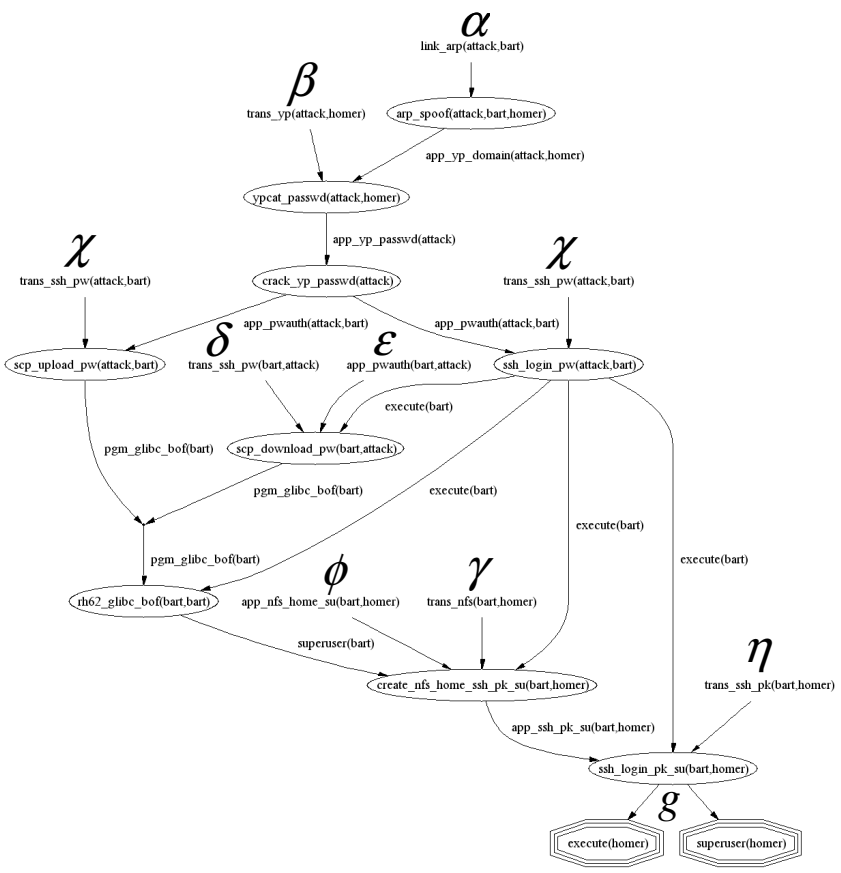

Figure 5: Exploit dependency graph for second example.

Another observation is that assigning false to initial condition *trans_ssh_pw*(*attack*,*bart*) simultaneously stops 2 exploits, i.e.,

1. *scp_upload*(*attack*,*bart*), and
2. *ssh_login_pw*(*attack*, *bart*)

This would not be apparent by looking at minimal exploit sets only. In other words, a single initial condition could control many exploits. General relationships among initial conditions and exploits can be many-to-many and arbitrarily complex. To solve the network-hardening problem, analysis must be at the level of initial conditions, as in our approach, and not at the level of exploits, as previously proposed.

Proceeding with our approach for this example, the attack goal expression $g = \alpha\beta\chi\phi\gamma\eta$ leads to the following 6 minimal-impact hardening options:

1. Harden *link_arp*(*attack*,*bart*), or
2. Harden *trans_yp*(*attack*,*homer*), or
3. Harden *trans_ssh_pw*(*attack*,*bart*), or
4. Harden *app_nfs_home_su*(*bart*,*homer*), or
5. Harden *trans_nfs*(*bart*,*homer*), or
6. Harden *trans_ssh_pk*(*bart*,*homer*)

Among these options, we can make some assumptions about the relative costs of individual hardening measures. For Option 1, *link_arp* involves the application of ARP (address resolution protocol) to map IP addresses to hardware (MAC) addresses. To harden *link_arp*, the network administrator could hard-code IP/MAC address and switch port relationships throughout the network. However, this has associated management overhead costs.

Option 4 *app_nfs_home_su*(*bart*,*homer*) is an application layer "connection" representing the sharing of the superuser home directory. This is a poor practice from a security standpoint (e.g., in this example it allows the overwriting of a secure shell authentication key pair), though the share was presumably created to make administration easier. It is easy to harden this by simply removing the file share.

For Option 3, *trans_ssh_pw*(*attack*,*bart*) could be hardened by modifying *bart*'s sshd configuration to use public key authentication

only (disable password authentication). This is a relatively low-cost option. In fact, this is already being done for *homer*.

The hardening measures for the remaining options (Options 2, 5, and 6) would make critical network services unavailable, i.e., their cost is too high. So overall, the choice comes down to hard-coding IP/MAC address relationships (Option 1), removing the superuser home directory file share (Option 4), and relying on secure shell public key authentication on *bart* (Option 3). Here Option 3 is the best (lowest cost) choice. This example illustrates how our approach can determine the best combination of lower-level vulnerabilities to harden for overall security.

5. Related Work

There are a number of tools available for vulnerability scanning, e.g. Nessus, Computer Oracle and Password System (COPS), McAfee CyberCop ASaP, and SAINT™ Scanning Engine. But no commercially available tools consider how attackers could combine low-level vulnerabilities to carry out an overall attack against specified network targets.

On the research front, model checking has been applied to the analysis of single hosts [1] as well as network wide [2]. In related work [14], model checking was used to discover attacks that could thwart intrusion detection systems. More recently [3][4], the NuSMV model checker was modified to compute all attack paths (organized as a graph), rather than a single path.

However, model checkers have known scalability problems, a consequence of the exponential complexity of the general state space they consider. Indeed, model checkers performed poorly for the network attack models we experimented with in our early work, leading us toward the approach we describe in this paper.

For example, the work in [3][4] attempts to reduce attack graph size using ordered binary decision diagrams. But the resulting graphs are still complex, even for the small examples described. Moreover, since the complexity of the algorithm for computing minimal critical attack sets is linear with the size of the attack graph, it again suffers from scalability problems. The work in [3][4] also computes minimal critical exploit sets via model checking. However, as we have argued, such exploit sets are insufficient for network hardening. At any rate, we are able to compute the same minimal critical sets with our low-order polynomial exploit dependency representation, rather than with the exponential attack graph representation.

Graph-based approaches for analyzing attack combinations [5][6][7][8] have generally suffered the same exponential state space problem. More recently, a scalable graph-based approach based on monotonic logic has been proposed [10][11]. Our approach can be considered an extension of that work. That is, we apply the representation for the purpose of computing safe network configurations. To some extent, [10] does address network hardening by describing an algorithm for computing a minimal critical set of exploits. However, as we have argued, such exploit-level approaches are insufficient for network hardening.

In related work [15], graph-based attack models have been proposed for computing the likelihood of attacks. Also, there has been work that emphasizes the modeling rather than the analysis of attacks [16][17]. The notion of modeling exploit sequences has been proposed for intrusion detection alert correlation [18]. Also, in [11], we describe a method for generating elements of network attack models automatically via the Nessus vulnerability scanner.

6. Summary and Conclusions

In this paper, we go beyond traditional attack paths to compute network configuration hardening options that guarantee the safety of given network resources. Assuming that relative costs have been assigned for individual hardening measures, the hardening options we compute minimize overall cost.

The network hardening solutions we provide are in terms of network configuration elements rather than exploits. We can therefore take into account the often-complex relationships among exploits and configuration elements. In this way, we can resolve hardening irrelevancies and redundancies that cannot be resolved through exploit-only approaches.

We greatly reduce the number of hardening options to consider by selecting only those with minimal impact on the network. That is, we are able to ignore options that contain supersets of the same hardened components, since they incur additional cost but offer no additional safety.

Our approach extends a recently proposed graph-based representation of exploit dependencies, based on monotonic logic. This representation has low-order polynomial

complexity, as opposed to the exponential complexity generally found in earlier work. The assumption of monotonic logic also allows us to resolve cycles and other redundancies in the dependency graph via the graph distance from initial conditions.

From the resolved exploit dependency graph, we compute an expression for the safety of given network resources, in terms of initial network conditions. We then assign minimal-impact safe values of the initial conditions, via the minimal elements of the partial ordering of conjunctive normal form terms. From these minimal-impact assignments, we find the safe network configuration with minimum total cost.

Because attackers often reach their goals through multiple exploits, network vulnerability analysis must consider the effects of combined vulnerabilities. The analysis of exploit sequences is a good first start. But what is really needed are explicit and manageable network hardening options that provide guarantees of safety, as we describe here.

7. References

[1] C. Ramakrishnan, R. Sekar, "Model-Based Analysis of Configuration Vulnerabilities," in *Proceedings of 7th ACM Conference on Computer and Communication Security*, November 2000.

[2] R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, 2000.

[3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, 2002.

[4] S. Jha, O. Sheyner, J. Wing, "Two Formal Analyses of Attack Graphs," in *Proceedings of 15th IEEE Computer Security Foundations Workshop*, Nova Scotia, Canada, June 2002.

[5] R. Baldwin, *Kuang: Rule based security checking*, technical report, MIT Lab for Computer Science, May 1994.

[6] D. Zerkle, K. Levitt, "Netkuang – A Multi-Host Configuration Vulnerability Checker," in *Proceedings of the 6th USENIX Unix Security Symposium*, San Jose, CA, 1996.

[7] C. Phillips, L. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," in *Proceedings of the New Security Paradigms Workshop*, Charlottesville, VA, 1998.

[8] L. Swiler, C. Phillips, D. Ellis, S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, June 2001.

[9] J. Dawkins, C. Campbell, J. Hale, "Modeling Network Attacks: Extending the Attack Tree Paradigm," in *Proceedings of Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Johns Hopkins University, June 2002.

[10] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[11] S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher, 2003.

[12] E. Mendelson, *Introduction to Mathematical Logic*, 4th ed., Chapman & Hall, 1997.

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

[14] G. Rohrmair, G. Lowe, "Using CSP to detect Insertion and Evasion Possibilities within the Intrusion Detection Area," in *Proceedings of BCS Workshop on Formal Aspects of Security*, 2002.

[15] R. Ortalo, Y. Deswarte, M. Kaâniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *IEEE Transactions on Software Engineering*, 25(5):633-650, September/October 1999.

[16] F. Cuppens, R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," in *Proceedings of Third International Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, October 2000.

[17] S. Templeton, K. Levitt, "A Requires/Provides Model for Computer Attacks," in *Proceedings of New Security Paradigms Workshop*, Cork Ireland, 2000.

[18] P. Ning, Y. Cui, D. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," in *Proceedings of the 9th ACM Conference on Computer & Communications Security*, Washington D.C., November 2002.