Interactive Analysis of Attack Graphs Using Relational Queries *

Lingyu Wang[1], Chao Yao[1], Anoop Singhal[2], and Sushil Jajodia[1]

[1] Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444, USA
{lwang3,cyao,jajodia}@gmu.edu
[2] Computer Security Division, NIST
Gaithersburg, MD 20899, USA
anoop.singhal@nist.gov

Abstract. Attack graph is important in defending against well-orchestrated network intrusions. However, the current analysis of attack graphs requires an algorithm to be developed and implemented, causing a delay in the availability of analysis. Such a delay is usually unacceptable because the needs for analyzing attack graphs may change rapidly in defending against network intrusions. An administrator may want to revise an analysis upon observing its outcome. Such an *interactive* analysis, similar to that in decision support systems, is difficult if at all possible with current approaches based on proprietary algorithms. This paper removes the above limitation and enables interactive analysis of attack graphs. We devise a relational model for representing necessary inputs including network configuration and domain knowledge. We generate the attack graph from those inputs as relational views. We then show that typical analyses of the attack graph can be realized as relational queries against the views. Our approach eliminates the needs for developing a proprietary algorithm for each different analysis, because an analysis is now simply a relational query. The interactive analysis of attack graphs is now possible, because relational queries can be dynamically constructed and revised at run time. Moreover, the mature optimization techniques in relational databases can also improve the performance of the analysis.

1 Introduction

As the result of topological vulnerability analysis, an *attack graph* describes all possible sequences of exploits an attacker can follow to advance an intru-

* This material is based upon work supported by National Institute of Standards and Technology Computer Security Division; by Homeland Security Advanced Research Projects Agency under the contract FA8750-05-C-0212 administered by the Air Force Research Laboratory/Rome; by Army Research Office under grants DAAD19-03-1-0257 and W911NF-05-1-0374, by Federal Aviation Administration under the contract DTFAWA-04-P-00278/0001, and by the National Science Foundation under grants IIS-0242237 and IIS-0430402. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

sion [16, 18, 1]. Attack graphs have been explored for different purposes in defending against network intrusions. First, an attack graph can more clearly reveal the weakness of a network than individual vulnerability does by providing the *context* of attacks. Second, attack graphs can indicate available options in removing identified weaknesses and help administrators to choose an optimal solution. Third, the knowledge encoded in attack graphs can also be used to correlate isolated alerts into probable attack scenarios.

However, many current approaches to the analysis of attack graphs share a common limitation. That is, a proprietary algorithm must be developed and implemented before the corresponding analysis becomes possible. Standard graph-related algorithms usually do not apply here due to unique characteristics of attack graphs. However, the delay in the analysis of attack graphs is usually unacceptable for defending against network intrusions. The needs for analyzing an attack graph usually change rapidly due to constantly changing threats and network configurations. An administrator may need to modify an analysis after the results of that analysis are observed. Such an *interactive analysis*, similar to that in decision support systems, is difficult if at all possible with current approaches based on proprietary algorithms.

In this paper, we provide a solution to the interactive analysis of attack graphs. First, we represent in the relational model the necessary inputs including network configuration and domain knowledge. We then generate the attack graph using relational queries, which can either be materialized as relations or simply left as the definition of relational views. The latter case is especially suitable for large networks where materializing the complete attack graph can be prohibitive. Second, we show how typical analyses of attack graphs can be realized as relational queries. The interactive analysis of attack graphs is now possible, because administrators can immediately pose new queries based on the outcome of previous analyses. Finally, as a side-benefit, the performance of an analysis can usually be transparently improved by the mature optimization techniques available in most relational databases.

The rest of this paper is organized as follows. The next section reviews related work. Section 3 proposes a relational model for representing the attack graph. Section 4 then discusses how typical analyses can be written as relational queries. Section 5 describes our implementation of the proposed methods. Finally, Section 6 concludes the paper and gives future direction.

2 Related Work

Attack graphs represent the knowledge about the inter-dependency between vulnerabilities [6, 21, 14, 4, 13, 16, 19, 17, 1, 18, 8]. Model checking was first used

to decide whether a goal state is reachable from the initial state [16, 15] and later used to enumerate all possible sequences of attacks connecting the two states [18, 9]. However, the number of attack sequences is potentially exponential, leading to high complexity. A more compact representation was thus proposed based on the *monotonicity assumption* (that is, an attacker never relinquishes an obtained capability) [1]. The new representation keeps exactly one vertex for each exploit or condition, leading to attack graphs of polynomial size.

Analyses of attack graphs have been used for different purposes in defending against network intrusions [18, 9, 12, 11, 20]. The *minimal critical attack set* analysis finds a minimal subset of attacks whose removal prevents attackers from reaching the goal state [18, 9]. However, the attacks in a minimal critical attack set are not necessarily independent, and a consequence cannot be removed without removing its causes. This observation leads to the *minimum-cost hardening* solution, which is a minimal set of independent security conditions [12]. Finding the minimum set of attacks leading to given goals is computationally infeasible, whereas a minimal set can be found in polynomial time [18, 9, 1]. All attacks involved in at least one of such minimal sets of attacks can also be enumerated [1]. Finally, in *exploit-centric alert correlation* [11, 20], attack graphs assist the correlation of isolated intrusion alerts.

The afore-mentioned analysis of attack graphs is largely based on proprietary algorithms. However, as mentioned earlier, this may delay a new analysis and make interactive analysis impossible. To our best knowledge, our study is the first to remove this limitation and to enable interactive analysis of attack graphs. On the other hand, decision support systems, such as on-line analytical processing (OLAP) [7], have been used for interactive analysis of data for a long time. However, an analyst there is usually interested in generalized data and statistical patterns, which is different from the analysis of attack graphs.

3 A Relational Model For Representing Attack Graphs

Section 3.1 reviews the basic concept of attack graph. Section 3.2 then proposes a relational model for representing attack graphs as relational views.

3.1 Attack Graph

The attack graph is usually visualized as a directed graph having two type of vertices, *exploits* and *security conditions* (or simply *conditions*). An exploit is a triple (h_s, h_d, v), where h_s and h_d are two connected hosts and v is a vulnerability on the destination host h_d. A security condition is a pair (h, c) indicating the host h satisfies a condition c relevant to security (both exploits and conditions may involve more hosts, for which our model can be easily extended).

An attack graph has two types of edges denoting the inter-dependency between exploits and conditions. First, the *require* relation is a directed edge pointing from a condition to an exploit. The edge means the exploit cannot be executed unless the condition is satisfied. Second, the *imply* relation points from an exploit to a condition. This means executing the exploit will satisfy the condition. Notice that there is no edge between exploits (or conditions). Example 1 illustrates the concept of attack graph.

Example 1. The left-hand side of Figure 1 depicts our running example of attack graph. The right-hand side shows a simplified version where x denotes the existence of a vulnerability *SADMIND BUFFER OVERFLOW* (Nessus ID 11841), y the user privilege, and A the exploitation of that vulnerability. The attack graph shows an attacker having user privilege on host 3 can exploit the vulnerability on hosts 1 and 2 and obtain user privilege on the hosts.

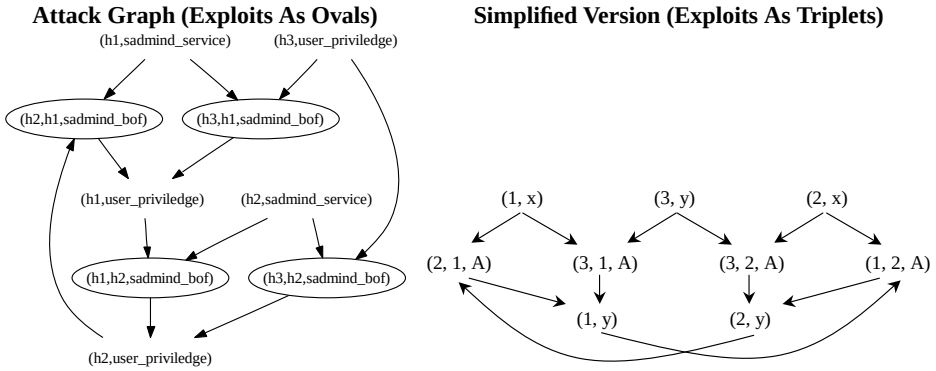

Fig. 1. An Example of Attack Graph

Two important aspects of attack graphs are as follows. First, the require relation is always *conjunctive* whereas the imply relation is always *disjunctive*. More specifically, an exploit cannot be realized until *all* of its required conditions have been satisfied, whereas a condition can be satisfied by any *one* of the realized exploits. Second, the conditions are further classified as *initial* conditions (the conditions not implied by any exploit) and *intermediate* conditions. An initial condition can be independently disabled to harden a network, whereas an intermediate condition usually cannot be [12].

3.2 A Relational Model for Attack Graphs

The complete attack graph is not explicitly represented in our model, but left as the result of a relational query. The result to the query may be materialized, or the query can simply be left as a view. Such flexibility is important to large

networks where materializing the complete attack graph may be prohibitive. We model two inputs, the *network configuration* (vulnerabilities and connectivity of the network) and the *domain knowledge* (the interdependency between exploits and conditions), as illustrated in Example 2. The domain knowledge is available in tools like the Topological Vulnerability Analysis (TVA) system, which covers more than 37,000 vulnerabilities taken from 24 information sources including X-Force, Bugtraq, CVE, CERT, Nessus, and Snort [8]. On the other hand, the configuration information including vulnerabilities and connectivity can be easily obtained with tools such as the Nessus scanner [5].

Example 2. Figure 2 depicts the network configuration and domain knowledge required for generating the attack graph in Example 1. The left-hand side shows the connectivity between the three hosts, and initially hosts 1 and 2 satisfy the condition x and host 3 satisfies y. The right-hand side says that an attacker can exploit the vulnerability A on the destination (denoted by the symbol D) host, if it satisfies x and the source host satisfies y at the same time. This exploitation will then satisfy y on the destination host.

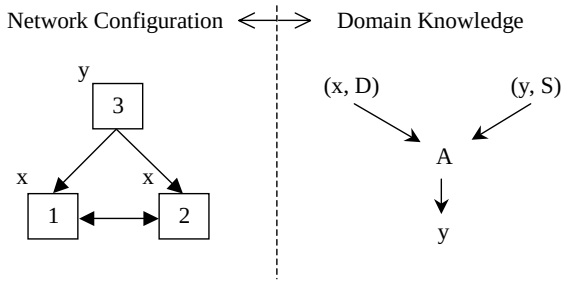

Fig. 2. An Example of Network Configuration and Domain Knowledge

Definition 1 defines the schema of our model. The connectivity relation represents the connectivity from each the source host H_s to the destination host H_d. The condition relation indicates a host H having an initial condition C. The condition-vulnerability dependency relation indicates a condition C is required for exploiting a vulnerability V on the destination host. The attribute F indicates whether the condition C belongs to the source (S) or the destination (D) host. The vulnerability-condition dependency relation indicates a condition C is satisfied by exploiting a vulnerability V.

The last three relations together with the condition relation are required for representing the complete attack graph (those relations may or may not need to be materialized). The vertices are conditions (the relation HC) and exploits (the relation EX), and the edges interconnect them are represented by relations CE and EC. Each relation has a composite key composed of all the attributes in that relation. Example 3 shows the relational model of Example 2.

Definition 1. *Define the following relational schemata:*

Connectivity $HH = (H_s, H_d)$
Condition $HC = (H, C)$
Condition-Vulnerability Dependency $CV = (C, F, V)$
Vulnerability-Condition Dependency $VC = (V, C)$
Exploit $EX = (H_s, H_d, V)$
Condition-Exploit $CE = (H, C, H_s, H_d, V)$
Exploit-Condition $EC = (H_s, H_d, V, H, C)$

Example 3. Table 1 describes a relational model composed of four relations, which precisely represents Example 2.

hh(HH)

H_s	H_d
1	2
2	1
3	1
3	2

hc(HC)

H	C
3	y
1	x
2	x

cv(CV)

C	F	V
x	D	A
y	S	A

vc(VC)

V	C
A	y

Table 1. Representing Network Configuration and Domain Knowledge in Relational Model

4 Analyzing Attack Graphs With Relational Queries

We first show how the complete attack graph can be generated using relational queries based on our model in Section 4.1. We then realize typical analyses of attack graphs as relational queries in Section 4.2.

4.1 Generating Attack Graphs Using Relational Queries

We regard the generation of the complete attack graph from given network configuration and domain knowledge as a special analysis, and we show how to conduct this analysis using relational queries. First, Example 4 illustrates a generation procedure similar to that in [1].

Example 4. Given the network configuration and domain knowledge in Example 2, the attack graph in Figure 1 can be generated by an iterative procedure as follows. Initially, the attack graph only includes the three initial conditions $(1, x)$, $(3, y)$, $(2, x)$ as vertices. First, domain knowledge implies that the conditions $(1, x)$ and $(3, y)$ jointly imply the exploit $(3, 1, A)$, and $(2, x)$ and $(3, y)$

jointly imply $(3, 2, A)$. Second, the two conditions $(1, y)$ and $(2, y)$ are satisfied. Next, we repeat the above two steps with the two new conditions and insert four more edges between $(1, y)$, $(2, y)$ and the two exploits. The process then terminates because no new conditions are inserted in the second iteration.

The key challenge in realizing the above procedure using relational queries lies in the conjunctive nature of the require relation. More specifically, an exploit cannot be realized unless *all* the required conditions are satisfied. In contrast, the imply relation can be easily realized using a join operation, since a condition can be satisfied by any *one* of the realized exploits. We deal with this issue with two set-difference operations as follows (similar to the division operation in relational algebra). Intuitively, we first subtract (that is, set difference) the satisfied conditions from the conditions required by all possible exploits. The result includes all the unsatisfied but required conditions, from which we can derive the exploits that cannot be realized. Then we subtract the unrealizable exploits from all possible exploits to derive those that can indeed be realized.

Definition 2 states the relational queries corresponding to each iteration of the procedure illustrated in Example 4. In the definition, Q_1 and Q_2 are intermediate results (we shall use subscripts in numbers to denote intermediate results) of satisfied and unsatisfied conditions up to this iteration, respectively. The vertices of the attack graph are Q_e and Q_c, which are realized exploits and satisfied conditions, respectively. The fourth and fifth relation jointly compose the edge set. The set union operations do not keep duplicates, and hence this process always terminates. Example 5 illustrates those queries.

Definition 2. *Given $hh(HH)$, $hc(HC)$, $cv(CV)$, and $vc(VC)$, let $Q_c = hc$, and let $Q_e(EX)$, $Q_{ce}(CE)$, $Q_{ec}(EC)$ be empty relations, define queries*

- $Q_1 = \sigma_{H_s=H \vee H_d=H}(hh \times \Pi_V(vc) \times hc)$
- $Q_2 = \Pi_{H_s,H_d,V,H_d,C}(hh \times \sigma_{F=D}(cv)) \cup \Pi_{H_s,H_d,V,H_s,C}(hh \times \sigma_{F=S}(cv)) - Q_1$
- $Q_e = (\Pi_{H_s,H_d,V}(hh \times cv) - \Pi_{H_s,H_d,V}(Q_2)) \cup Q_e$
- $Q_{ce} = \Pi_{H_d,C,H_s,H_d,V}(Q_e \times \sigma_{F=D}(cv)) \cup \Pi_{H_s,C,H_s,H_d,V}(Q_e \times \sigma_{F=S}(cv)) \cup Q_{ce}$
- $Q_{ec} = \Pi_{H_s,H_d,V,H_d,C}(\sigma_{Q_e.V=vc.V}(Q_e \times vc)) \cup Q_{ec}$
- $Q_c = \Pi_{H,C}(Q_{ec}) \cup Q_c$

Example 5. Table 2 shows the result to each query in the first iteration in generating the attack graph of Example 1. The relation Q_1 are the satisfied conditions and their related (but not necessarily realizable) vulnerabilities. Subtracting those from the conditions required by all possible exploits yields the two unsatisfied conditions and the unrealizable exploits in Q_2. Then, subtracting the unrealizable exploits from all possible exploits gives the two realizable exploits in Q_e. The exploits then imply the two conditions in Q_c. The edges in Q_{ce} and Q_{ec} interconnect the conditions and exploits.

$\mathbf{Q_1}$

H_s	H_d	V	H	C
1	2	A	1	x
1	2	A	2	x
2	1	A	1	x
2	1	A	2	x
3	1	A	1	x
3	1	A	3	y
3	2	A	2	x
3	2	A	3	y

$\mathbf{Q_2}$

H_s	H_d	V	H	C
1	2	A	1	y
2	1	A	2	y

$\mathbf{Q_e}$

H_s	H_d	V
3	1	A
3	2	A

$\mathbf{Q_{ce}}$

H	C	H_s	H_d	V
1	x	3	1	A
2	x	3	2	A
3	y	3	1	A
3	y	3	2	A

$\mathbf{Q_{ec}}$

H_s	H_d	V	H	C
3	1	A	1	y
3	2	A	2	y

$\mathbf{Q_c}$

H	C
1	y
2	y

Table 2. An Example of One Iteration in Deriving the Complete Attack Graph

4.2 Typical Analyses of Attack Graphs in Relational Queries

We now turn to typical analyses of attack graphs previously studied in the literature. We show how to rewrite those analyses as relational queries based on our model. In the following discussion, our queries are against the relations (or views) given by Definition 2.

Vulnerability-Centric Alert Correlation and Prediction The alert correlation method first maps a currently received intrusion alert to the corresponding exploit. Then, it reasons about previous exploits (alerts) that prepare for the current one and possible exploits in the future [20]. The key difference between this analysis and the one used to generate the attack graph is that the conjunctive nature of the require relation should be ignored here. The relationship between alerts is usually regarded as *casual* instead of logical [10, 3]. Such a conservative approach is more appropriate in this context because alerts may have been missed by intrusion detection systems.

Example 6. In Figure 1, suppose the current alert maps to the exploit $(2, 1, A)$. The backward search will first reach conditions $(1, x)$ and $(2, y)$ and then follows $(2, y)$ to $(3, 2, A)$ and $(1, 2, A)$ to find a previous correlated alert if there is any, or to make a hypothesis for a missing alert, otherwise. The search continues from $(1, 2, A)$ to $(1, y)$ and $(2, x)$, then from $(1, y)$ to $(3, 1, A)$ (the branch to $(2, 1, A)$ is a loop and hence ignored) and consequently to $(1, x)$ and $(3, y)$. The search stops when it reaches only initial conditions or if a loop is encountered.

Definition 3 states the relational queries corresponding to the backward search in Example 6. The forward search can be realized in a similar way and

hence is omitted. First, the relation Q_3 includes the conditions reachable from the current exploits while ignoring the conjunctive relationship between those conditions. Second, subtracting from Q_3 the initial conditions in hc and the previously visited conditions in Q_5 (to avoid loops) yields the reachable conditions and consequently the exploits in Q_4. The above two steps are repeated until no more conditions are left (that is, all the conditions are in hc or in Q_5). The exploits encountered in this process are collected in Q_A as the final result. Loops are avoided in this process because the set union operation does not keep duplicates and the relation Q_5 ensures each condition to be visited at most once.

Definition 3. *Given $hh(HH)$, $hc(HC)$, $cv(CV)$, $vc(VC)$, and (h_s, h_d, V), let $Q_3(HC)$, Q_5, and Q_A be empty relations and $Q_4(EX) = \{(h_s, h_d, V)\}$. Define*

- $Q_3 = \Pi_{h_d,C}(Q_4 \bowtie \sigma_{F=D}(cv)) \cup \Pi_{h_s,C}(Q_4 \bowtie \sigma_{F=S}(cv))$
- $Q_4 = \Pi_{H_s,H_d,V}(\sigma_{H_d=H \wedge Q_3.C=vc.C}(hh \times (Q_3 - hc - Q_5) \times vc))$
- $Q_5 = Q_5 \cup Q_3$
- $Q_A = Q_A \cup Q_4$

Example 7. Table 3 shows the three iterations corresponding to the backward search in Example 6. The first iteration starts from the given exploit $(2, 1, A)$ and reaches two exploits $(1, 2, A)$ and $(3, 2, A)$ through the condition $(2, y)$. The second iteration reaches $(3, 1, A)$ and $(2, 1, A)$ through $(1, y)$. The exploit $(2, 1, A)$ leads to two previously visited conditions (that is, a loop) and the other exploit $(3, 1, A)$ reaches only initial conditions. Consequently, no new exploit appears in Q_4 in this iteration and the search terminates.

Enumerating Relevant Attacks and Network Hardening Enumerating the *relevant* exploits (those appears in at least one sequence of attacks leading to the goal conditions [1]) and finding a *network hardening* solution (given goal conditions represented as a logic formula of initial conditions [12]) share a similar backward search in the attack graph, as illustrated in Example 8 and Example 9, respectively.

Example 8. In Figure 1, we start from a given goal condition $(1, y)$ and search backwards in the attack graph. First, the two exploits $(3, 1, A)$ and $(2, 1, A)$ are reached. The former branch ends at initial conditions, and the latter leads to one initial condition $(1, x)$ and an intermediate condition $(2, y)$. The condition $(2, y)$ then leads to $(3, 2, A)$ and $(1, 2, A)$. The former ends at initial conditions, and the latter leads to a loop back to $(1, y)$. The relevant exploits with respect to the goal condition $(1, y)$ are thus $(2, 1, A)$, $(3, 1, A)$, and $(3, 2, A)$ (the exploit $(1, 2, A)$ is not relevant because it can never be realized before satisfying the goal $(1, y)$ itself).

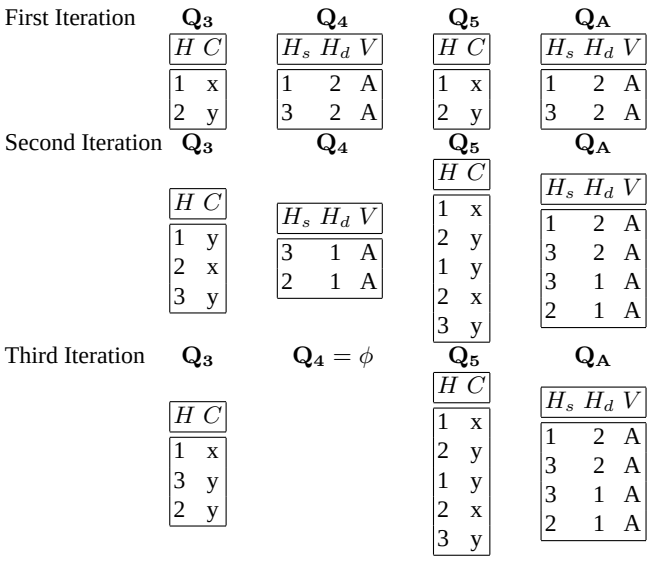

Table 3. An Example of Analyzing Attack Graphs for Alert Correlation and Prediction

Example 9. With a similar search, we can transform the goal condition $(1, y)$ into a logic formula of initial conditions as follows (by regarding the exploits and conditions as Boolean variables). In the fourth line, the value $FALSE$ replaces the second appearance of the goal condition $(1, y)$, because it is a predecessor of $(1, 2, A)$, indicating a loop. The final result says that if any of the two conditions $(1, x)$ and $(3, y)$ is disabled, then the goal can no longer be satisfied.

$$
\begin{aligned}
(1,y) &\equiv (3,1,A) \vee (2,1,A) \\
&\equiv (1,x) \wedge (3,y) \vee (1,x) \wedge (2,y) \\
&\equiv (1,x) \wedge (3,y) \vee (1,x) \wedge ((3,2,A) \vee (1,2,A)) \\
&\equiv (1,x) \wedge (3,y) \vee (1,x) \wedge ((3,y) \wedge (2,x) \vee (2,x) \wedge FALSE) \\
&\equiv (1,x) \wedge (3,y)
\end{aligned}
$$

The key differences between the above backward search and that used for correlating alerts are as follows. First, the conjunctive nature of the require relation must be considered. In Example 8, the exploit $(1, 2, A)$ is not relevant, because one of its required conditions $(1, y)$ is not satisfiable, even though the other required condition (that is, $(2, x)$) is already satisfied. Second, duplicate appearances of exploits and conditions must be kept. This is required for obtaining sequences of relevant exploits leading to the goal, as well as for generating the logic formula in network hardening. In the former case, different sequences may share common exploits or conditions, whereas the logic formula in the second case clearly contains duplicates. In order for the search to traverse an exploit or condition for multiple times, the set union operation needs to keep duplicates. Hence, loops must be avoided by maintaining a predecessor list for

each vertex as in standard breadth-first search (BFS) [2] (although the search discussed above is different from a BFS).

Definition 4 states the relational queries used to enumerate relevant exploits or to generate the logic formula in network hardening. The two queries simply traverse the attack graph given by Definition 2. The two relations in the definition keep duplicates in set union operations. Notice that the actual construction of the logic formula (adding the *and* and *or* connectives) is external to the relational queries and can easily be incorporated.

Definition 4. *Given relations $hh(HH)$, $hc(HC)$, $cv(CV)$, $vc(VC)$ and a non-empty relation $Q_7(HC)$, let $Q_6(EX)$ be an empty relation. Define*

- $Q_6 = \Pi_{H_s,H_d,V}((Q_7 - hc) \bowtie Q_{ec})$
- $Q_7 = \Pi_{H,C}(Q_6 \bowtie Q_{ce})$

Example 10. Table 4 shows the iterations corresponding to the procedure in Example 8 and Example 9. Originally, $Q_7 = \{(1, y)\}$.

First Iteration

$\mathbf{Q_6}$		
H_s	H_d	V
3	1	A
2	1	A

$\mathbf{Q_7}$	
H	C
1	x
2	y
1	x
3	y

Second Iteration

$\mathbf{Q_6}$		
H_s	H_d	V
3	2	A

$\mathbf{Q_7}$	
H	C
3	y
2	x

Table 4. An Example of Enumerating Relevant Exploits and Network Hardenning

Reachability From Subsets of Initial Conditions and Incremental Updates of Attack Graphs Many analyses ask a similar question, that is whether the goal condition is still satisfiable, if a given subset of initial conditions are disabled. The question may arise when we need to determine the potential effect of enforcing a security measure (so some initial conditions will be disabled), or when we want to decide whether the goal condition is reachable with only stealthy attacks [18]. The question may also be asked simply because the network configuration has changed and some initial conditions are no longer satisfied (on the other hand, new initial conditions can be easily handled with more iterations of the queries in Definition 2.) In each case, we can certainly recompute the attack graph from

scratches, with the given conditions removed from the relation hc. However, this is not desired especially when the attack graph is much larger than the set of conditions to be disabled. Instead, we should incrementally update the attack graph by computing the effect of disabling the given conditions. The conjunctive nature of the require relation must be taken into accounts, but in a different way, as illustrated in Example 11.

Example 11. In Figure 1, suppose the condition $(2, x)$ is disabled. Then the exploits $(1, 2, A)$ and $(3, 2, A)$ can no longer be realized. Then the condition $(2, y)$ becomes unsatisfiable, because it can only be implied by the above two exploits. Finally, the exploit $(2, 1, A)$ cannot not longer be realized. However, the condition $(1, y)$ is still satisfiable, due to another exploit $(3, 1, A)$.

Example 11 shows that such a *negative* analysis is quite different from the previous ones. The previous searches are all unidirectional in the sense that the edges are only followed in one direction (either forwards or backwards). However, the above analysis follows edges in both directions. For example, after the forward search reaches the condition $(1, y)$ from the exploit $(2, 1, A)$, it must go back to see whether other exploits also imply the condition $(1, y)$ (in this case, the exploit $(3, 1, A)$ does so). Definition 5 states the relational queries for this purpose. The first query simply derives unrealizable exploits from unsatisfied conditions. The next three queries use two set difference operations to derive the unsatisfied conditions while taking into accounts the conjunctive nature of the require relation. Finally, the results are collected.

Definition 5. *Given relations $hh(HH)$, $hc(HC)$, $cv(CV)$, $vc(VC)$ and a non-empty relation $Q_{11}(HC)$ as a subset of hc, let $Q_8(EX)$, $Q_9(EC)$, $Q_{10}(EC)$, Q_e, and Q_c be empty relations. Define*

- $Q_8 = \Pi_{H_s,H_d,V}(Q_{11} \bowtie Q_{ce}))$
- $Q_9 = Q_8 \bowtie Q_{ec}$
- $Q_{10} = Q_{ec} \bowtie \Pi_{H,C}(Q_9) - Q_9$
- $Q_{11} = \Pi_{H,C}(Q_9) - \Pi_{H,C}(Q_{10})$
- $Q_e = Q_e \cup Q_8$
- $Q_c = Q_c \cup Q_{11}$

Example 12. Table 5 shows the iterations corresponding to the procedure in Example 11. Originally, $Q_{11} = \{(2, x)\}$.

5 Empirical Results

As proof of concept, we have implemented the analyses discussed in the previous section. The queries are written in PL/SQL. The queries are tested in Oracle

First Iteration

$\mathbf{Q_8}$

H_s	H_d	V
3	2	A
1	2	A

$\mathbf{Q_9}$

H_s	H_d	V	H	C
3	2	A	2	y
1	2	A	2	y

$\mathbf{Q_{10}} = \phi$

$\mathbf{Q_{11}}$

H	C
2	y

Second Iteration

$\mathbf{Q_8}$

H_s	H_d	V
2	1	A

$\mathbf{Q_9}$

H_s	H_d	V	H	C
2	1	A	1	y

$\mathbf{Q_{10}}$

H_s	H_d	V	H	C
3	1	A	1	y

$\mathbf{Q_{11}} = \phi$

Table 5. An Example of Incremental Updates

9i in its default settings on a Pentium IV 2GHz PC with 512MB RAM. In our preliminary experiments, we test the queries against the attack scenario originally studied in [18, 1] [3]. The results of the analyses match those in the previous work, which justifies the correctness of our techniques. Next we test the performance of our techniques. We have two main objectives. First, we want to determine whether the running time of the queries is practical for interactive analysis. For most decision support systems, the typical delay to a query that is considered as tolerable in interactive analyses is usually in a matter of seconds. Such a short delay is also critical to the analysis of attack graphs, especially when the analysis is used for real-time detection and prevention of intrusions.

Second, we want to determine whether the techniques scale well in the size of attack graphs. Although the attack graph may be very large for a large network, an analysis and its result usually only involves a small part of the attack graph. The running time of an analysis thus depend on how efficiently an analysis searches the attack graph. We expect the mature optimization techniques available in most databases can transparently improve the performance and make the analyses more scalable. To test the queries against large attack graphs in a manageable way, we increase the number of vertices in the original attack graph by randomly inserting new hosts with random connectivity and vulnerabilities. We then execute the same set of analyses in the new network and measure the running time of each analysis. The main results are shown in Figure 3. All the results have 95% confidence intervals within about 5% of the reported values.

The left-hand side shows the running time of generating the attack graph in the size of that attack graph. The attack graph with about 20,000 vertices can be generated in less than seven minutes. The result also shows that our methods scale well in the size of attack graphs. The right-hand side shows the running time of each analysis in the size of the attack graph. The result shows that all the analyses require less than a second, which clearly meets the requirement of

[3] Our ongoing work will compare the performance of our approach with that of the proprietary algorithms proposed before.

an interactive analysis. The analyses all scale well with the size of the attack graph. This proves our conjecture that the optimization techniques in databases such as indexing can transparently help to keep analyses efficient. A closer look at the result reveals that the increase in running time is mainly caused by larger results. This also explains the fact that the incremental update analysis scales differently from the other two (the effect of disabled initial conditions does not change much when the size of the attack graph increases).

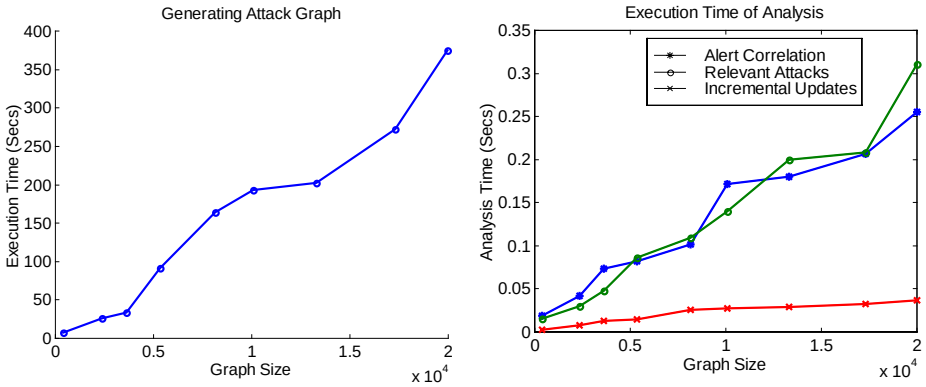

Fig. 3. The Performance of Analyzing Attack Graphs Using Relational Queries

6 Conclusion

We have proposed a relational model to enable interactive analysis of attack graphs for intrusion detection and prevention. We have shown that the complete attack graph can be generated as relational views. Any analysis of the attack graph are thus relational queries against such views. We have shown how to write relational queries for typical analyses previously studied in the literature. This approach made the analysis of attack graphs an interactive process similar to that in the decision support systems. As a side effect, the mature optimization techniques existing in most relational databases also improved the performance of the analysis.

Acknowledgements The authors are grateful to the anonymous reviewers for their valuable comments.

References

1. P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 217–224, 2002.

2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
3. F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02)*, pages 187–200, 2002.
4. M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.
5. R. Deraison. Nessus scanner, 1999. Available at http://www.nessus.org.
6. D. Farmer and E.H. Spafford. The COPS security checker system. In *USENIX Summer*, pages 165–170, 1990.
7. J. Gray, A. Bosworth, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
8. S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publisher, 2003.
9. S. Jha, O. Sheyner, and J.M. Wing. Two formal analysis of attack graph. In *Proceedings of the 15th Computer Security Foundation Workshop (CSFW'02)*, 2002.
10. P. Ning, Y. Cui, and D.S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 245–254, 2002.
11. S. Noel and S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distance. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, 2004.
12. S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency grpahs. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, 2003.
13. R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Trans. Software Eng.*, 25(5):633–650, 1999.
14. C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop (NSPW'98)*, 1998.
15. C.R. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1/2):189–209, 2002.
16. R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Research on Security and Privacy (S&P'00)*, pages 156–165, 2000.
17. R. Ritchey, B. O'Berry, and S. Noel. Representing TCP/IP connectivity for topological analysis of network security. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02)*, page 25, 2002.
18. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02)*, pages 273–284, 2002.
19. L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX'01)*, 2001.
20. L. Wang, A. Liu, and S. Jajodia. An efficient and unified approach to correlating, hypothesizing, and predicting intrusion alerts. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS 2005)*, pages 247–266, 2005.
21. D. Zerkle and K. Levitt. Netkuang - a multi-host configuration vulnerability checker. In *Proceedings of the 6th USENIX Unix Security Symposium (USENIX'96)*, 1996.