2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

Compressing Attack Graphs Through Reference Encoding

Pengsu Cheng, Lingyu Wang and Tao Long
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, QC H3G 1M8, Canada
Email:{pen_che,wang,l_tao}@ciise.concordia.ca

***Abstract*—As a widely accepted model of multi-step network intrusions, attack graph has been applied to topological vulnerability analysis, network hardening, alert correlation, security metrics, and so on. A major challenge faced by attack graphs is the scalability: Even the attack graph of a moderate-sized network is typically incomprehensible to the human eyes, whereas that of large enterprise networks usually has an unmanageable size. Such a complexity, however, is not entirely unavoidable. In this paper, we shall show that an attack graph may contain much redundancy due to the similarity between different hosts' configurations. We then present a novel representation of attack graphs based on reference encoding. Specifically, subnets of hosts with similar configurations are represented using reference hosts while textual rules are employed to describe minor differences. The compression process is lossless and the resultant attack graph can directly provide useful insights. The effectiveness of the proposed model is illustrated through a case study and simulation results.**

I. Introduction

By exploiting a series of related vulnerabilities, a cleverly crafted multi-step intrusion may infiltrate a well guarded network while evading detection efforts. Understanding such attacks is crucial to both forensic analysis and real-time defense of critical computer networks. As a widely accepted model, *attack graph* depicts the causal relationships between related vulnerabilities [12], [16]. Attack graph has been successfully applied to topological vulnerability analysis, network hardening, alert correlation, security metrics, and so on. In practice, attack graphs are readily available from existing tools [6], [8].

A major challenge faced by attack graphs is the scalability [9]. The attack graph of a small to moderate-sized network is typically large and incomprehensible to the human eyes. For large enterprise networks, even the generation of attack graphs can be computationally infeasible. On the other hand, this situation is not entirely inevitable. An attack graph usually carries much redundant information due to hosts with similar configurations and connectivity, such as those found in offices, computer labs, server farms, etc. Such redundant information is usually irrelevant in a useful analysis of attack graphs.

Earlier solutions typically focus on avoiding the state explosion problem in using a model checking approach [1], [6]. To build our work upon such results, we shall adopt the same graph-based model as in [1]. Other existing work attempt to temporarily hide the complexity of attack graphs through visualization techniques, such as the hierarchical approach [14], the matrix clustering approach [15], and the virtual node approach [5]. Such efforts are parallel to our work, since our compressed attack graphs can certainly benefit from a visualization technique while being presented to a user.

In this paper, we present a novel representation of attack graphs using a well known compression technique, namely, *reference encoding*. Specifically, we represent a large number of hosts with similar configurations and connectivity using a few reference hosts together with textual rules describing minor differences between hosts. Our representation has two main advantages. First, unlike a general-purpose data compression scheme (whose results are useless until decompressed), our compression process will produce compressed, but valid attack graphs, which can directly reveal similar threats as the original attack graph does. Second, the compression is lossless in the sense that any valid sequence of attack steps may be recovered from the compressed attack graph if necessary. These two facts together imply that our representation eliminates the need for ever generating the original attack graph, which is usually a prohibitive task for large networks.

The rest of the paper is organized as follows. Section II reviews the background. Section III devises our compression model. Section IV presents simulation results. Section V reviews related work. Section VI concludes the paper.

II. Attack Graph

An *attack graph* represents the causal relationship between *exploits* sharing common pre- and post-*conditions* [1]. An exploit (h_s, h_d, v) indicates the exploitation of a vulnerability v on the destination host h_d that is originated from the source host h_s. A condition (h, c) indicates a security-related condition c is satisfied on host h (both exploits and conditions may involve more hosts). Definition 1 formally states the attack graph model.

Definition 1: Denote by H a set of hosts, VT and CT a set of vulnerability types and condition types, respectively. An attack graph G is a directed graph $G(E \cup C, R_r \cup R_i)$ where $E \subseteq H \times H \times VT$ denotes a set of exploits, $C \subseteq H \times CT$ a set of conditions, and $R_r \subseteq C \times E$ and $R_i \subseteq E \times C$ two edge relations.

An important semantics here is that an exploit cannot be executed until *all* of its pre-conditions are satisfied, whereas a condition can be satisfied by *any* exploit implying that condition. Figure 1 shows a toy example of attack graph in which exploiting a buffer overflow vulnerability in a vulnerable version of the Sadmind service (Nessus ID 11841) will lead to the user privilege on a remote machine.

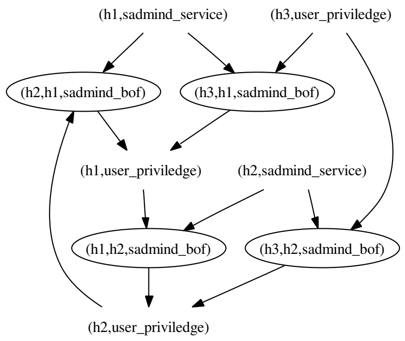


Fig. 1: An Example of Attack Graph

An attack graph can be generated using existing algorithms [1], which will not be repeated here. The generation requires two inputs. First, the expert knowledge about each vulnerability type's pre- and post-condition is readily available from existing databases [8]. Second, hosts' connectivity and vulnerability information can be obtained using network scanners, such as Nessus [3]. Definition 2 abstracts these two inputs as the *type graph* and *configuration graph*, respectively (the conditions can sometimes be omitted for simplicity).

Definition 2: A type graph is a directed graph $TG(VT \cup CT, E)$ where VT and CT denotes a set of vulnerability types and condition types, respectively, and $E \subseteq (VT \times CT) \cup (CT \times VT)$. A configuration graph $CG(N, E)$ is a directed graph where $N \subseteq H \times 2^V$, and $E \subseteq N \times N$.

III. The Compression Model

We describe our compression model in two steps, first in a special case where all the hosts have exactly the same connectivity and vulnerabilities, and then in the general case.

A. The Special Case

We first build intuitions through an example and then give the formal model at the end of this section. The left-hand side of Figure 2 shows a configuration graph. The three fully connected hosts, $h1$, $h2$, and $h3$, have the same three vulnerabilities $v1$, $v2$, and $v3$ (the symbol \$ indicates no vulnerability). We can interpret this as a small network of three identical machines connected to a switch and a router with no firewall, and an attacker may attack this network from his/her own machine. The right-hand side shows a simple type graph.

Clearly, even in this simple case, the number of possible sequences of attacks is significant (it can be shown that there will be 27 such sequences if the exploit can be executed locally). On the other hand, these attack sequences apparently include much redundancy. A seemingly viable solution is to use, say, $h1$ as the reference of the other two hosts. However, a more careful consideration will reveal the limitation of such

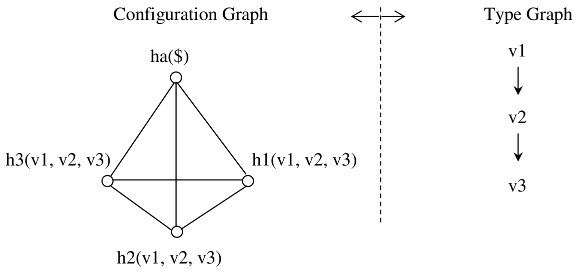


Fig. 2: An Example in the Special Case

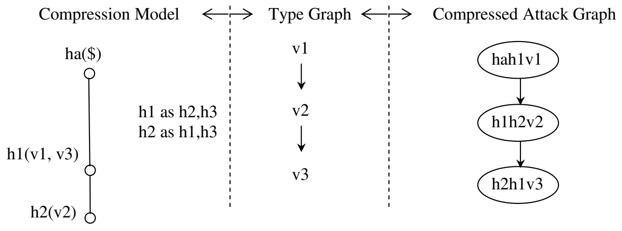


Fig. 3: A Compression Model in the Special Case

an one-node compression model. Specifically, in many cases we need to distinguish between *local exploits*, which indicate an exploit that originates from and ends at the same host, and *remote exploits*, which are launched between different hosts. With $h1$ as the reference to both $h2$ and $h3$, we would not be able to distinguish between the two.

Therefore, we use a two-node compression model shown in Figure 3. On the left-hand side is the compression model, which includes the *compressed configuration graph* and the *reference rules* stating that the host appearing before the word *as* can be replaced by any of the hosts appearing after. The right-hand side shows the *compressed attack graph* generated from the type graph and the compressed configuration graph. Notice that we shall not re-define the compressed configuration graph and compressed attack graph since these are simply a more compact version of their normal counterparts.

The compression model clearly satisfies the two requirements stated in Section I. First, the compressed attack graph can reveal similar threats as by a full attack graph (which is omitted due to space limitations). Second, the compression model encodes exactly the same information as the full attack graph would. Table I shows some example sequences of attacks and the corresponding reference rules where the first three only involve remote exploits and the last three also involve local exploits. Notice that each application of a reference rule is only effective to one occurrence of an exploit in a sequence. For example, the first three sequences all apply the same reference rule, $h1$ as $h3$, but in different ways. Also, reference rules are not transitive. For example, applying two rules, $h1$ as $h2$ and $h2$ as $h3$, does not imply the rule $h1$ as $h3$ can be applied.

TABLE I: The Reference Encoding of Attack Sequences

$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h3v3$	$h1$ as $h3$
$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h3v3$	$h1$ as $h3$
$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h1v3$	$h1$ as $h3$
$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h1v3$	$h2$ as $h1$
$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h1v3$	$h1$ as $h2$
$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h1v3$	$h1$ as $h3$, $h2$ as $h1$

The above model can be easily extended to the general case with any number of hosts or vulnerabilities. More hosts can simply be added to the reference rules. For vulnerabilities, we follow the procedure shown in Figure 4 to assign them to hosts. The procedure first generates the compressed attack graph using a temporary vulnerability assignment (we can employ any attack graph generation algorithm, such as the one shown in [1]). This step is to ensure that all possible sequences of attacks should appear in the compression model. Next, the procedure removes those assignments of vulnerabilities that do not appear in the compressed attack graph to avoid redundancy and any reference host with no vulnerability assigned.

```
Input: A configuration graph $CG$, a type graph $TG$, and an initial
       compressed configuration graph $CCG$
Output: An updated $CCG$ with vulnerability assignment function
        $f() : H \rightarrow V$ where $H$ and $V$ are the set of hosts and
        vulnerabilities in $CCG$, respectively
Method:
1.  For each $h \in H$
2.      Let $f(h) = V$
3.  Generate compressed attack graph $CAG$ from $TG$ and $CCG$
4.  For each $h \in H$
5.      For each $v \in V$
6.          If $\langle h, v \rangle \notin CAG$
7.              Let $f(h) = f(h) - \{v\}$
8.      If $f(h) = \phi$
9.          Let $H = H - \{h\}$
10. Return $CCG$
```

Fig. 4: A Procedure for Assigning Vulnerabilities

Definition 3 formally describes the two-node compression model for this special case, as illustrated above.

Definition 3: Given a type graph $TG(N_t, E_t)$ and configuration graph $CG(N_c, E_c)$ in the special case, we define

- the compressed configuration graph as a directed graph $CCG(N, E)$ where $N = \{(ha, \{\$\}), (hi, V_1), (hj, V_2)\}$ with V_1 and V_2 denoting two sets of vulnerabilities assigned using the procedure shown in Figure 4 and $E = \{(ha, hi), (hi, ha), (hi, hj), (hj, hi)\}$,
- the reference rule $RR = \{hi \text{ as } hx \mid hx \in H \wedge x \neq i\} \cup \{hj \text{ as } hy \mid hy \in H \wedge y \neq j\}$,
- the compression model as the pair $\langle CCG, RR \rangle$.

B. The General Case

Differences between hosts' configuration and connectivity can be handled in two ways. First, we may take a *negation approach* by starting with the special case and indicating missing connectivity or vulnerabilities using special notations in the reference rules. Second, when a host is significantly different from others, we may simply leave the host outside the compression model of the latter. However, such a decision may also depend on application-specific information, such as natural clusters of hosts formed by organizational boundaries. In the rest of this section, we provide methods for both situations while leaving the choice between the two as a future work.

Figure 5 shows an example of the general case where the afore-mentioned negation approach is more applicable. The left-hand side shows a configuration graph in which $h1$ and $h3$ both have three vulnerabilities but $h2$ does not have $v2$; the three hosts are fully connected except that there is no connection from $h3$ to $h1$ (for example, due to a personal firewall software running on $h1$). The right-hand side of the figure shows the same type graph as before.

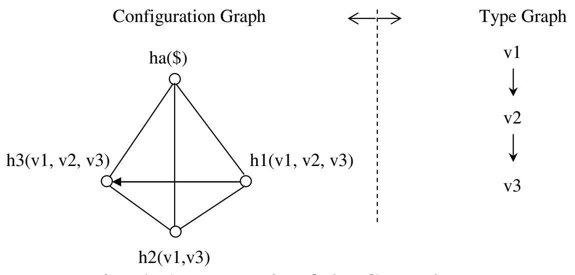


Fig. 5: An Example of the General case

Clearly, if we simply apply the previous compression model, the resultant compressed configuration graph would lead to invalid attack sequences. Intuitively, the negation approach is to start with such an incorrect configuration graph, and fix it using the reference rules. The modified compression model is shown in the left-hand side of Figure 6. The compressed configuration graph is identical to that in Figure 3 but the reference rules now include pairs of parentheses indicating non-existent vulnerabilities or connectivity.

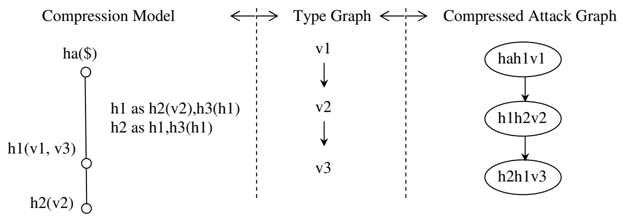


Fig. 6: The Compression Model

There are a few subtleties in this model. First, since both $h2$ and $h3$ lack some vulnerability or connectivity, no two hosts will exactly correspond to the compressed configuration graph. This reflects the conservative nature of the negation approach. That is, the compression model reveals the worst case threat, while details about non-existent vulnerabilities or connectivity are hidden in the reference rules. The advantage of this design choice is that a security administrator may immediately see the most pertinent threat while delaying the detailed analysis of exact attack sequences.

Second, in Figure 5, we have essentially regarded $h1$ as the *complete* host in terms of connectivity and vulnerabilities. However, sometimes none of the hosts will be complete, for example, when we remove $v3$ from $h1$ in the above case. In general, the reference host would be assigned the set of connectivity and vulnerabilities obtained as the union of such sets among all the hosts.

Next, we consider the situation where not all hosts should be included in the compression model. Figure 7 shows such an example. All the victim hosts have the same vulnerabilities but differ in connectivity. The two groups of hosts, $h1$, $h2$, $h3$ and $h4$, $h5$, $h6$, respectively correspond to the special case. However, the two groups are only connected through $h7$. Clearly, the difference between the two groups of hosts is so significant that we may need to create two separate compression models.

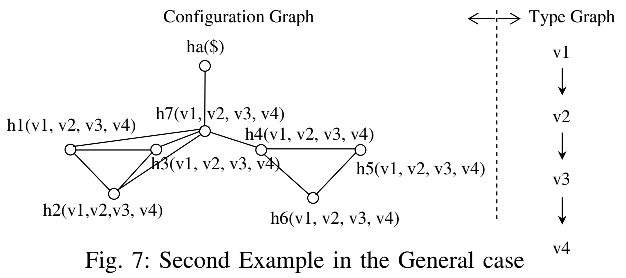


Fig. 7: Second Example in the General case

Since $h7$ is connected to all the three hosts in the first group, we can treat $h7$ in a similar way as with the attacker's host ha in previous cases. On the other hand, in the second group, only $h4$ is connected to the external node $h7$. If we create a compression model for the three nodes in this group, then the non-existent connectivity between $h5$, $h6$ and $h7$ will have to be explicitly represented in the reference rules, which introduces unnecessary redundancy. We thus do not include $h4$ in the compression model. The compression model for this case is shown in the upper left corner of Figure 8. The way of assigning vulnerabilities is different from previous cases due to the existence of hosts such as $h7$ and $h4$.

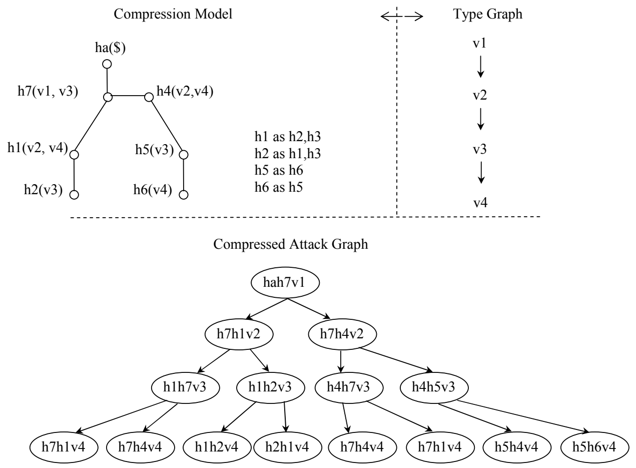


Fig. 8: The Compression Model

The above example can be easily generalized to any number of hosts or vulnerabilities. More groups will only lead to more two-node models. The vulnerabilities are assigned using the procedure shown in Figure 9. The main difference from the previous procedure is that instead of generating the compressed attack graph for each group of hosts, the procedure first generates a compressed attack graph for whole network using the type graph and temporary vulnerability assignment of each group. This ensures all attack sequences that may appear in the overall compressed attack graph should remain so in the final compression models.

```
Input: A configuration graph $CG$, a type graph $TG$, and compressed
       configuration graphs $CCG_i (i = 1, 2, \ldots, k)$ without vulnerability
       assignment
Output: Updated $CCG_i (i = 1, 2, \ldots, k)$ with vulnerability assignment
       function $f_i() : H_i \rightarrow V_i$ where $H_i$ and $V_i$ are the set of hosts
       and vulnerabilities in $CCG_i$, respectively
Method:
1.  For $i = 1$ to $k$
2.      For each $h \in H_i$
3.          Let $f_i(h) = V_i$
4.  Generate compressed attack graph $CAG$ from $TG$ and the $CCG_i$'s
5.  For $i = 1$ to $k$
6.      For each $h \in H_i$
7.          For each $v \in V_i$
8.              If $\langle h, v\rangle \notin CAG$
9.                  Let $f_i(h) = f_i(h) - \{v\}$
10.         If $f_i(h) = \phi$
11.             Let $H = H - \{h\}$
12. Return $CCG_i (i = 1, 2, \ldots, k)$
```

Fig. 9: A Procedure for Assigning Vulnerabilities

IV. Case Study and Simulation

In this section, we first present a case study to demonstrate the application of the proposed methods. We then show simulation results on the performance of our methods.

A. A Case Study

In Figure 10, there are four densely connected subnets, $Li(i = 1, 2, \ldots, 5)$, $Ri(i = 1, 2, \ldots, 8)$, $Si(i = 1, 2, \ldots, 9)$, and $Di(i = 1, 2, \ldots, 6)$, which are interconnected with four routers $P1$ through $P4$ (notice that hosts inside each subnet are connected to each other through a switch, represented by an unlabeled node in the figure, which is assumed to have no vulnerability and not considered as part of the model). All hosts inside each subnet have the same set of vulnerabilities unless it is explicitly indicated otherwise in the figure. The dash line between $S9$ and $S8$ indicates there is no connection between them due to personal firewalls. Suppose the two hosts in red color, $S9$ and D_6, are compromised, and we suspect the intrusion originates from one of the mobile hosts Li's that is connected to the network through a wireless access point.

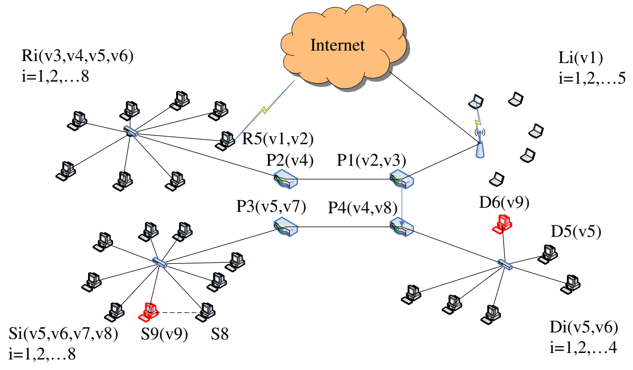


Fig. 10: An Example Network

Assuming a simple type graph of the total order $v1 \rightarrow v2 \rightarrow \ldots v9$. The traditional approach to attack graph generation would lead to a large attack graph with around 500 attack sequences. In our model, the four subnets naturally map to four compression models. In particular, hosts L_i's will reduce to a one-node model since we assume the access point prevents those hosts from directly communicating with each other. For each of the remaining subnets, a two-node model suffices except that host $R5$ will be left out of the model due to its direct connection to the internet, and hosts $S9$ and $D6$ are left out of the model due to their special role as victim hosts. Figure 11 shows the complete compression model.

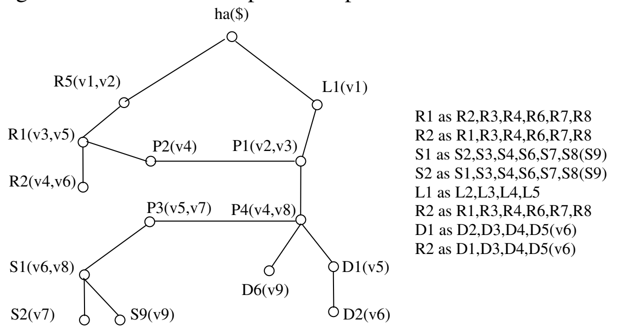


Fig. 11: The Compression Model

The compression model clearly reveals possible sequences of attacks. For example, an attacker may start from host $L1$ and move to $P1$, $P4$, $P3$, and $S1$ where he/she can either continue to compromise $S9$, or move back to $P3$, $P4$, and finally compromise $D6$. A simple analysis will yield the following two sequences leading to the victim hosts $S9$ and $D6$.

$ha\$ \rightarrow haL1v1 \rightarrow L1P1v2 \rightarrow P1P1v3 \rightarrow P1P4v4 \rightarrow P4P3v5 \rightarrow P3S1v6 \rightarrow S1S2v7 \rightarrow S2S1v8 \rightarrow S1S9v9$

$ha\$ \rightarrow haL1v1 \rightarrow L1P1v2 \rightarrow P1P1v3 \rightarrow P1P4v4 \rightarrow P4P3v5 \rightarrow P3S1v6 \rightarrow S1P3v7 \rightarrow P3P4v8 \rightarrow P4D6v9.$

Other sequences can be easily obtained using reference rules, if necessary. Since many existing attack graph analysis will depend on such sequences of attacks in one way or another, the compression model can facilitate such analysis without the need to first generate the full attack graph.

B. Simulation Results

We evaluate the proposed compression model through simulation (no publicly available datasets of real world attack graphs exist to the best of our knowlege) on machines equipped with an Intel 1.80GHz processor and 1024MB RAM. We employ two synthetic topology generators, the Boston university Representative Internet Topology gEnerator (BRITE) [2] and the Georgia Tech Internetwork Topology Models topology generator(GT-ITM) [4]. We inject vulnerability information into generated topologies to obtain random configuration graphs. We employ the Jaccard similarity coefficient [7] as a parameter for describing the degree of similarity between vulnerabilities on different hosts. To determine the grouping of hosts in the compression process, we use a metric defined as the weighted average of the total number of hosts and that of reference rules in the compressed configuration graph (there certainly exist other application-specific ways for defining the metric).

Figure 12a shows the compression results of 263 topologies with the numbers of hosts varying from 60 to 1020 and the number of subnets from 3 to 32. For each topology, the degree of similarity in vulnerabilities on hosts inside each subnet is within the range of $[0.930, 0.956]$. The weights on the number of host and rules are both set as 0.5. From the results, we can see that the compression rate varies even if the size of topologies is fixed. Figure 12b shows similar compression results by fixing the size of subnets to 10, 30, and 50, respectively, while increasing the number of subnets. We can see that the compression method generally works better for networks with larger subnets.

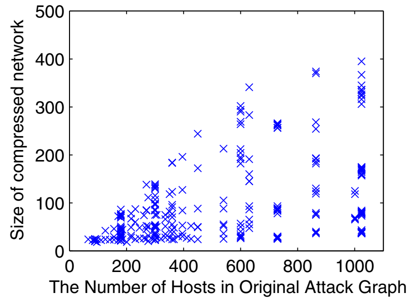
Figure 12d and 12e show the compression results with the total number of hosts fixed at 420, 576, and 630, respectively, while the size of each subnet increases and the number of subnets decreases. Figure 12d and 12e are compression results of topologies generated by BRITE and GT-ITM, respectively. From the results we can see that while different generators do not make a significant difference to the compression rate, the size of subnets is clearly a major factor.

Figure 12f shows how the weight α assigned to the numbers of hosts (the weight assigned to the number of rules is $1 - \alpha$) will affect the result. Two sets of topologies are generated by BRITE and GT-ITM, respectively. The value of α in this experiment varies between 0.2 and 0.7. The size of original networks is fixed to 240 hosts with 40 hosts in each subnets. From the results, we can see that an optimal weight assignment to α is between 0.5 and 0.6 (which helps to achieve a balance between the reduction of the number of hosts and that of the number of rules).

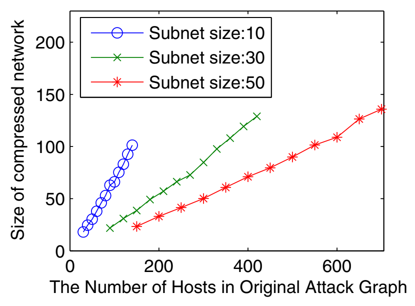
Figure 12c shows the compression rate in the degree of similarity of vulnerabilities on different hosts. We generate 2330 topologies with a fixed size of 420 nodes with 7 subnets, and we vary the average similarity degree. The metric weight assignment is $\alpha = 0.5$. We can see that the compression rate is almost linear in the degree of similarity of vulnerabilities.

V. Related Work

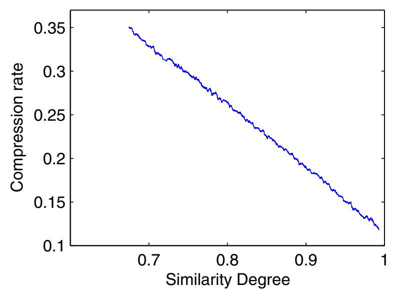
Various vulnerability scanners, such as Nessus [3], can find known vulnerabilities in a network. However, they cannot reveal how such vulnerabilities can be combined in a multi-step attack to infiltrate a network. A detailed attack graph model is described in [11] and a tool is proposed to build an attack graph using forward search in [17]. A *require and provide* approach to automatic attack graph generation is mentioned in [18], which has later been widely adopted in defending against multi-step attacks. Model checking is applied to the analysis of multi-step network attacks in [12]. A later work [13] provides more details on how connectivity should be modeled at different layers. The term *topological vulnerability analysis* is also introduced. Model checking is used for a different purpose in [16], that is, to enumerate all attack paths. To address the scalability of model checking-based approaches, a *monotonic assumption* is proposed in [1], which states that the further exploits will never cause the attacker to relinquish any obtained privileges. More recently, a logic programming-based approach to the representation of attack graphs is given in [10]. A hierarchical approach to efficient attack graph visualization is given in [14]. Another effort applies a matrix clustering



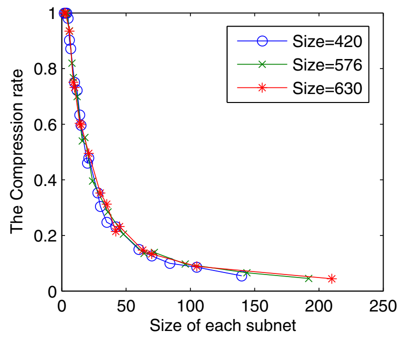
(a)



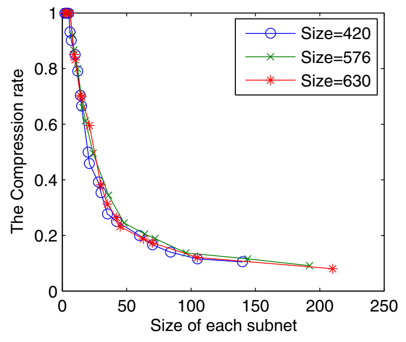
(b)



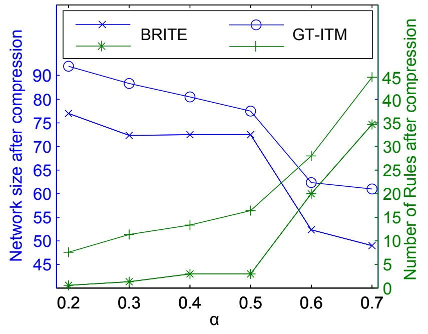
(c)



(d)



(e)



(f)

Fig. 12: Simulation Results

algorithm to the adjacency matrix of attack graphs in order to indicate the feature of protection domain on the main diagonal [15]. Finally, a directed graph is used to model subnets and potential inter-subnet attacks and a dominator tree is used to determine whether inter-subnet and intra-subnet attacks are useful [5].

VI. Conclusion

We have proposed a novel representation for removing redundant information caused by hosts with similar configurations from attack graphs. Our compression model may allow useful analysis to be conducted directly on the compressed attack graph without ever generating the full attack graph, which is usually a prohibitive task especially for large networks. Our simulation results confirm a significant reduction in the size of compressed attack graphs when the degree of similarity of vulnerabilities is high. Our future work will address the optimal partitioning of hosts into clusters based on different definitions of a metric, and the integration of the compression model into existing attack graph generation tools.

Acknowledgement

The authors thank the anonymous reviewers for their valuable comments. This material is based upon work partially supported by Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, and by Fonds de recherche sur la nature et les technologies.

References

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS'02*, 2002.
[2] Boston university representative internet topology generator. Available at http://www.cs.bu.edu/brite/.
[3] R. Deraison. Nessus scanner, 1999. Available at http://www.nessus.org.
[4] Georgia tech internetwork topology models topology generator. Available at http://www.cc.gatech.edu/projects/gtitm/.
[5] J. Homer, A. Varikuti, X. Ou, and M.A. Mcqueen. Improving attack graph visualization through data reduction and attack grouping. In *VizSec'08*, pages 68–79, 2008.
[6] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC'06*, pages 121–130, Washington, DC, USA, 2006. IEEE Computer Society.
[7] Paul Jaccard. Paul jaccard. etude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Socit Vaudoise des Sciences Naturelles*, 37:547–579.
[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] R. Lippmann and K. Ingols. An annotated review of past papers on attack graphs. Technical Report PR-IA-1, 2005.
[10] X. Ou, W. F. Govindavajhala, and M.A. McQueen. A scalable approach to attack graph generation. In *CCS'06*, 2006.
[11] C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *NSPW'98*, 1998.
[12] R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *S&P'00*, pages 156–165, 2000.
[13] R. Ritchey, B. O'Berry, and S. Noel. Representing TCP/IP connectivity for topological analysis of network security. In *ACSAC'02*, page 25, 2002.
[14] S. Jajodia. S. Noel. Managing attack graph complexity through visual hierarchical aggregation. In *VizSec'04*, 2004.
[15] S. Jajodia. S. Noel. Understanding complex network attack graphs through clustered adjacency matrices. In *ACSAC'05*, 2005.
[16] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *S&P'02*, 2002.
[17] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer attack graph generation tool. In *DISCEX'01*, 2001.
[18] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *NSPW'00*, pages 31–38, 2000.