Effective Network Vulnerability Assessment through Model Abstraction

Su Zhang[1], Xinming Ou[1], and John Homer[2]

[1] Kansas State University, {zhangs84,xou}@ksu.edu
[2] Abilene Christian University, jdh08a@acu.edu

Abstract. A significant challenge in evaluating network security stems from the scale of modern enterprise networks and the vast number of vulnerabilities regularly found in software applications. A common technique to deal with this complexity is attack graphs, where a tool automatically computes all possible ways a system can be broken into by analyzing the configuration of each host, the network, and the discovered vulnerabilities. Past work has proposed methodologies that post-process "raw" attack graphs so that the result can be abstracted and becomes easier for a human user to grasp. We notice that, while visualization is a major problem caused by the multitude of attack paths in an attack graph, a more severe problem is the distorted risk picture it renders to both human users and quantitative vulnerability assessment models. We propose that abstraction be done *before* attack graphs are computed, instead of after. This way we can prevent the distortion in quantitative vulnerability assessment metrics, at the same time improving visualization as well. We developed an abstract network model generator that, given reachability and configuration information of a network, provides an abstracted model with much more succinct information about the system than the raw model. The model is generated by grouping hosts based on their network reachability and vulnerability information, as well as grouping vulnerabilities with similar exploitability. We show that the attack graphs generated from this type of abstracted inputs are not only much smaller, but also provide more realistic quantitative vulnerability metrics for the whole system. We conducted experiments on both synthesized and production systems to demonstrate the effectiveness of our approach.

Keywords: enterprise network security, attack graph, quantitative vulnerability assessment, abstraction

1 Introduction

Network security control is an issue that increases in difficulty with growths in network size and the number of vulnerabilities. Automated approaches are needed to quickly and reliably evaluate the current security state of the network. Attack graphs are a common approach to security evaluation [1–3, 8, 10–13, 17–21, 23, 24, 26]. They show how an attacker can combine multiple vulnerabilities

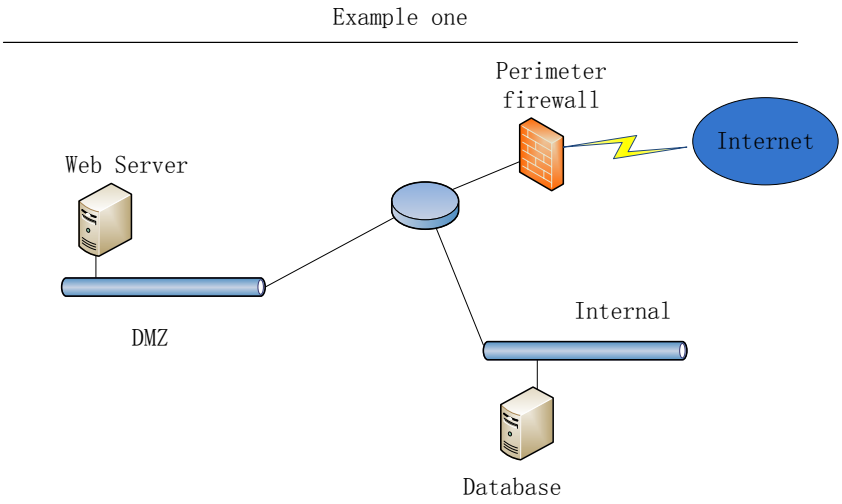
in a system to launch multi-stage attacks to gain privileges in the system. Attack graphs are often used in conjunction with risk assessment tools to provide recommendations to system administrators on how to mitigate the discovered problems [4, 6, 9]. There are two main utilities of attack graphs: visualization and risk assessment. A major obstacle in these utilities is the size and complexity of attack graphs from even moderate-size networks. The large number of attack paths towards the same target not only makes the graph too dense to read, but also distorts risk assessment results by ignoring the fact that many of the attack steps are similar and not independent.

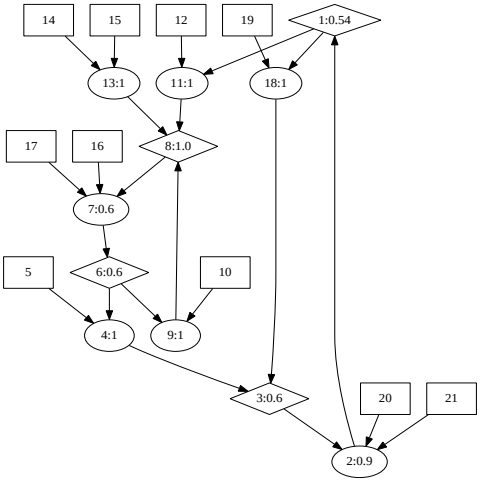
Figure 1 shows a simple network. An attacker could launch attacks from the Internet against the web server, which then provides him a stepping stone to exploit the database server in the internal network. The lower part of the figure shows a MulVAL attack graph [18, 19] generated from this network model. The labels of the graph nodes are shown at the right-hand side. Diamond-shaped nodes represent privileges an attacker could gain in the system; circle nodes represent attack steps that achieve the privileges; rectangular nodes represent network configuration settings.

Figure 2 shows the topology and attack graph of a similar scenario, but with five identical servers in the DMZ zone. We can see that the attack graph gets very complicated. Human users, like a system administrator, may have difficulty tracing through the many identified attack paths. An abstracted view of the attack graph can highlight the real underlying issues in the network. We must also consider whether the multitude of attack paths shown in this attack graph reflects a realistic risk picture. The dotted lines in the network topology illustrate a subset of the attack paths identified in the graph. There are five ways to attack the database server, utilizing five different sources in the DMZ. However, the five servers in DMZ are identically configured. Thus if an attacker can exploit any one of them, he can exploit the others as well. In this case, having four more servers will not significantly increase the attacker's chance of success.

Prior research has proposed various techniques to address the visualization challenge [7, 15, 30]. However, we have not found substantial discussion in the literature addressing the distortion problem in risk assessment caused by the redundancy in attack graphs, especially in the context of quantitative security assessment. Traditional approaches [6, 27, 28] would assess all the attack paths to the attacker's target without taking the similarities of these paths into consideration. Consequently, the explosion in the attack-graph's size could yield high risk metrics, often misleading the system administrator's judgment. While one could post-process the graph and remove such redundancy, like in previous works [15, 30], we believe a better approach is to pre-process *the input* to attack-graph generation so that such redundancy is removed by abstracting the network model, instead of the attack graph. There are a number of benefits of abstracting the network model:

– From a human user's perspective, removing redundancy in the network description provides a better high-level view of both the system and the security vulnerabilities identified therein. The semantics of the abstract network





1: execCode(database,user)
2: remote exploit of CVE-2009-2446
3: netAccess(database,tcp, 3306)
4: multi-hop access
6: execCode(webServer,user)
7: remote exploit of CVE-2009-1918
8: netAccess(webServer,tcp,80)
9: multi-hop access
11: multi-hop access
13: direct network access
15: attackerLocated(Internet)
18: multi-hop access

Fig. 1. Scenario of example one and its attack graph

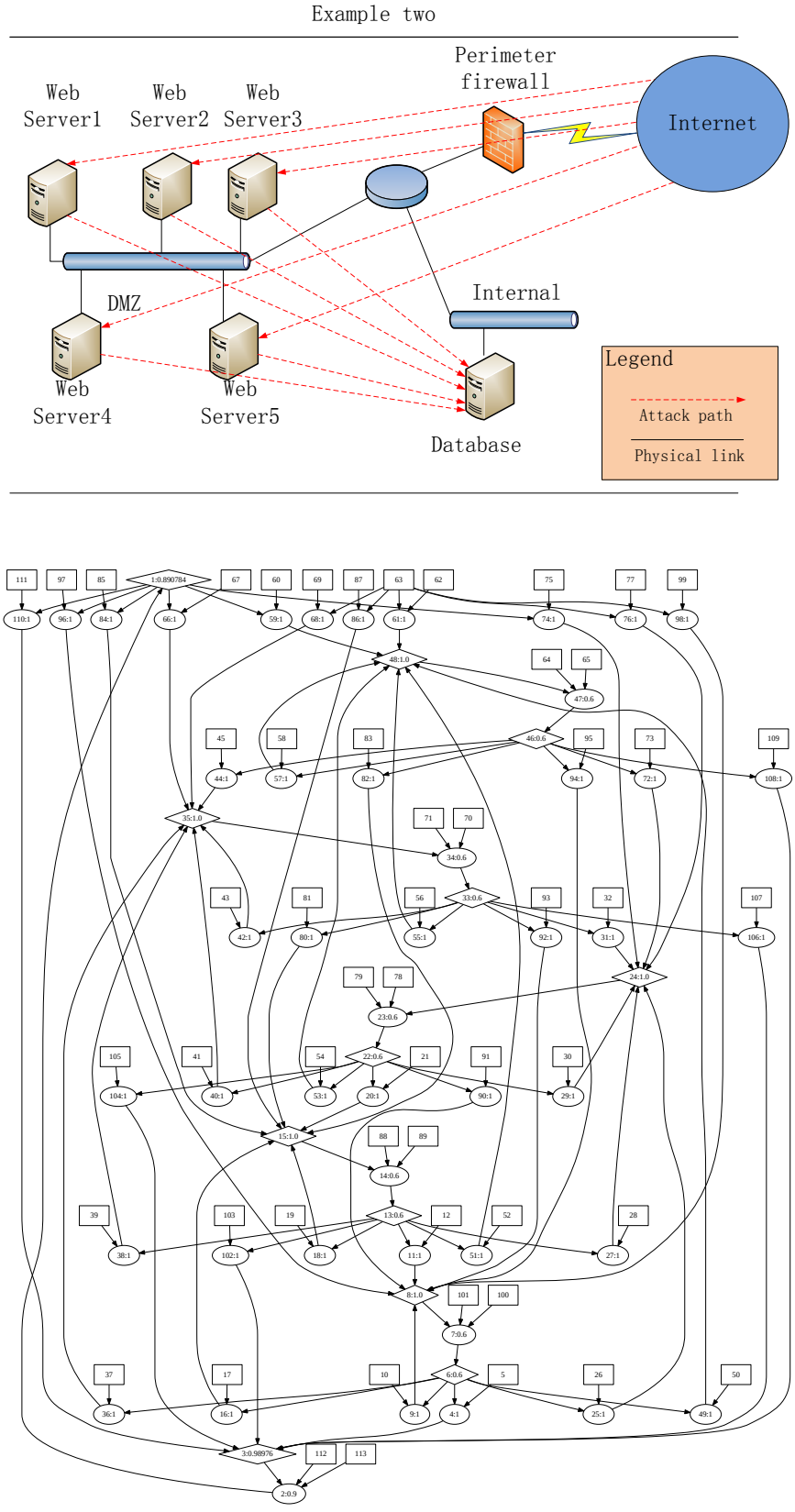


Fig. 2. Scenario of example two and its attack graph

model matches better with how a human would manage a large network system, and as a result the output of the attack-graph analysis is natural to communicate to human users.
– After abstracting the network model, the distortion in quantitative security assessment results due to repetitive similar attack paths will be rectified.

We design algorithms to create abstract network models for large-scale enterprise networks, based on network reachability and host configuration information. The algorithms make reasonable assumptions about available input describing the network structure and host configuration information. The abstracted models dramatically reduce the complexity of attack graphs, improving the visualization and also correcting skewed quantitative vulnerability assessment results. Moreover, by using the abstracted models, the quantitative vulnerability assessment process is hastened. We evaluate our methods on both synthesized and production systems, demonstrating the effectiveness of the approach.

The rest of the paper is organized as follows. Section 2 discusses the abstraction criteria and algorithms. Section 3 describes experimental evaluation of the abstraction method. Section 4 discusses related work and section 5 concludes.

2 Network model abstraction

2.1 Abstraction criteria

Similarity among hosts For large enterprise networks, it is not unusual to have thousands of machines in a subnet with same or similar reachability and configuration. If an attacker could compromise one of the machines, he is likely able to do the same for the others. This would result in a large number of similar attack paths in the attack graph. These attack paths should not be considered independent when assessing the system's security risk: if the attacker failed in compromising one of the hosts, he would probably fail on the others with the same properties (reachability and configuration) as well. Network reachability and host configuration determine to a large extent the exploitability of a host machine. For this reason, the machines with the same reachability and similar configurations can be grouped and treated as a single host.

Similarity among vulnerabilities A single host may contain dozens or even hundreds of vulnerabilities, each of which may appear in a distinct attack path to further compromise the system. However, not all these paths provide unique valuable information since many vulnerabilities are similar in nature. They may belong to the same application, require the same pre-requisites to be exploited, and provide the same privilege to the attacker. From a human user's perspective, it is more important to know, at a higher level, that *some* vulnerability in the application could result in a security breach, rather than enumerating all the distinct but similar attack paths. Since vulnerabilities in the same application are often exploited by the same or similar mechanisms, if the attacker fails in

exploiting one of them, it is reasonable to assume a low chance of successful attack by similar exploits. For this reason, these vulnerabilities can be grouped together as a single vulnerability and an aggregate metric can be assigned as the indicator on the success likelihood of exploiting any one of them, instead of combining them as if each exploit can be carried out with an independent probability. For example, when a host has 10 vulnerabilities in Firefox, we can say with X likelihood an attacker can successfully exploit any one of them, where X is computed based on each vulnerability's CVSS score [14], taking into consideration the similarity among the 10 vulnerabilities. One simple approach would be to use the highest risk probability value as representative of the whole set.

2.2 Abstraction steps

Our network model abstraction process is carried out in three steps.

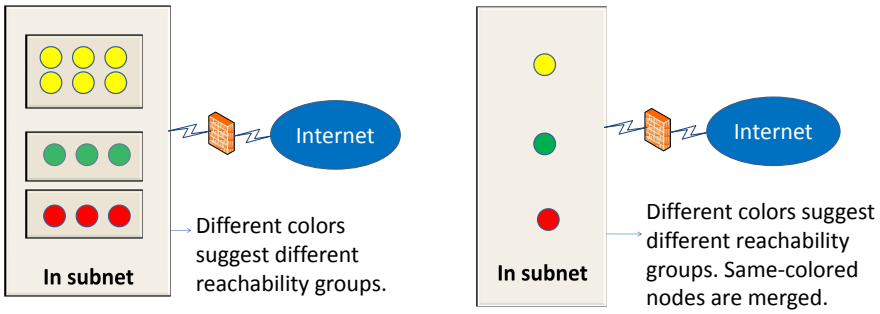
1. *Reachability-based grouping.* Hosts with the same network reachability (both to and from) are grouped together.
2. *Vulnerability grouping.* Vulnerabilities on each host are grouped based on their similarities.
3. *Configuration-based breakdown.* Hosts within each reachability group are further divided based on their configuration information, specifically the types of vulnerabilities they possess.

Reachability-based grouping We group all the hosts based on their reachability information. We first give two definitions.

Definition 1. *reachTo(H) is a set of triples (host, protocol, port) where H can reach host through protocol at port. Similarly, reachFrom(H) is a set of triples (host, protocol, port) where host can reach H through protocol and port.*

Definition 2. *Let H_1 and H_2 be two hosts. We say $H_1 \equiv_r H_2$ if reachTo(H_1) = reachTo(H_2) $\wedge$ reachFrom(H_1) = reachFrom(H_2)*

We put hosts into the same reachability group if they belong to the same equivalence class $\equiv_r$. Then all the hosts in the same reachability group can be abstracted as a single node. Figures 3(a) and 3(b) illustrate this idea, and Algorithm 1 explains the grouping process. The grouping is applied to all the machines in a subnet. We interpret a subnet as a collection of machines communication among which is unfiltered. We incrementally add reachability information into a set. If host H's reachability has been recorded, we find the existing group through a hash map and put H into the corresponding group. Otherwise we store the reachability information, create a new group label and map it to a singleton set with H in it. We do this for all the hosts in each subnet. The time complexity for this algorithm is $O(n^2)$ where n is the number of hosts in the network. We need to go over all the hosts within the subnet and for each host we need linear time to identify its reachability information.



(a) Before reachability-based grouping (b) After reachability-based grouping

Fig. 3. Before and after reachability-based grouping

Algorithm 1 Pseudocode for reachability-based grouping

Input: A set of (reachTo(h), reachFrom(h)) for each host h in a subnet.
Output: A hash map L, which maps a group label α to a list of hosts having the same reachability (reachTo and reachFrom).

```
Lr ← {} {Lr is a set of triples (α, reachToSet, reachFromSet).}
Queue Q ← all the hosts of the given subnet
L ← empty map {initialize the return value}
while Q is not empty do
  n ← dequeue(Q)
  if Lr contains (α, reachTo(n), reachFrom(n)) then
    L[α] ← L[α]∪{n} {if the reachability of n is the same as some other host that
    has been processed, add n to its equivalent class.}
  else
    create a fresh α
    Lr ← Lr ∪(α, reachTo(n), reachFrom(n)) {Otherwise put its reachability infor-
    mation into Lr}
    L[α]← {n}
  end if
end while
return L
```

Vulnerability grouping We group vulnerabilities on each machine based on the application they belong to. Typically vulnerabilities in one application will be of the same type (local, remote client or remote service). For example, vulnerabilities of Adobe Reader are remote client since they are always triggered when a user opens the application on a malicious input, possibly sent by a remote attacker. Security holes in IIS, on the other hand, most likely belong to remote service vulnerabilities. After grouping based on applications, we can provide the system administrator a clearer view of the system's vulnerabilities — instead of showing a long list of CVE ID's, we show the vulnerable applications that affect the system's security. One issue that needs to be addressed is how to assign an aggregate vulnerability metric to the virtual vulnerability after grouping. Such vulnerability metrics, like CVSS scores, are important in quantitative assessment of a system's security. Intuitively, the more vulnerabilities in an application, the more exploitable the application is. But the degree of exploitability does not simply grow linearly since many of the vulnerabilities will be similar. Our current grouping algorithm (Algorithm 2) simply takes the highest value, but it will be straightforward to plug in a different aggregation method.

Algorithm 2 Pseudocode for vulnerability grouping

```
Input: A set of ungrouped vulnerabilities on a machine (S_u)
Output: A hash map L that maps an application to its vulnerability score
  Lr ←{} {Lr is a set of applications that have appeared so far}
  L ← empty hash map
  while S_u ≠ {} do
    take v from S_u
    if Lr contains (v.application) then
      if L[v.application] < v.score then
        L[v.application] = v.score
      end if
    else
      L[v.application] = v.score
      Lr.add(v.application)
    end if
  end while
  return L
```

Configuration-based breakdown For hosts in the same reachability group, their configurations could be different from one another. Thus, if an attacker is able to exploit one host within the group, it does not mean he could compromise the others as well. This means grouping based on reachability alone is too coarse. In order to reflect differences in attackability, we need to "break down" the merged node based on configuration settings. In our current implementation, we have only included software vulnerability as the configuration information. When deployed on production systems, one can rely upon package management

systems to decide whether two hosts have the same or similar software set up. Algorithm 3 shows the process of configuration-based grouping. The algorithm iterates over all the hosts in a reachability group and records its configuration information. If a host's configuration matches one previously recorded, meaning some other hosts have the same types of vulnerabilities, this host will not be recorded in the set. At the end of the algorithm, the returned set only contains one representative host for each group of hosts with the same reachability and configuration. The complexity of the algorithm is linear in the number of hosts.

Algorithm 3 Pseudocode for configuration-based break down

Input: A list L, each element of which is a set of machines belonging to the same reachability group, and with the vulnerabilities grouped.

Output: Further-refined group S_c based on vulnerability information. Each element in S_c is a representative for a group of hosts with the same reachability and configuration.

```
while L≠{} do
  remove h from L
  Lr ← empty map; {Lr is a set of pairs (hostname, configuration). It is used to
  store the distinct configurations that have appeared so far.}
  if Lr contains (_, h.configuration) then
    continue {if its configuration has appeared before, skip}
  else
    Lr.add((h, h.configuration)) {if its configuration has not appeared before, record
    it}
  end if
end while
S_c = ⋃_{(h,_)∈Lr} h {collect all representative hosts in Lr and put them into S_c}
return S_c
```

3 Experimentation Result

To evaluate the effect of model abstraction on quantitative security assessment of computer networks, we apply probabilistic metric models [6, 27] on the generated attack graphs. In such metric models, each attack step is associated with a (conditional) probability indicating the success likelihood of the exploit when its pre-conditions (predecessor nodes) are all satisfied. The model then computes the absolute probability that a privilege can be obtained by an attacker based on the graph structure. We use MulVAL [18, 19] attack-graph generator in the evaluation. Our security metric implementation follows Homer's algorithm [6].

We created one scenario to illustrate the visualization effect and rectification on the distortion in metric calculation generated by the large number of similar attack paths. The topology information of the example is shown in Fig. 5. There are three subnets: Internal Servers, DMZ, and Normal Users. Each subnet has

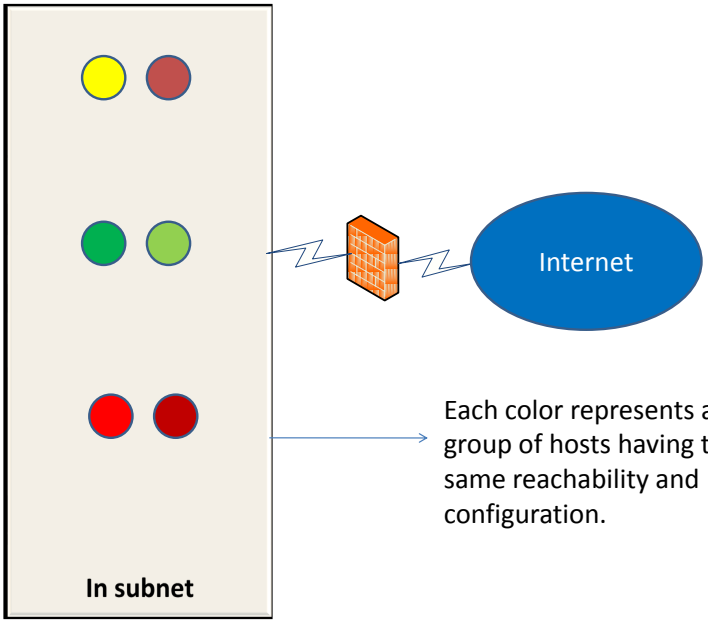


Fig. 4. After configuration-based breakdown.

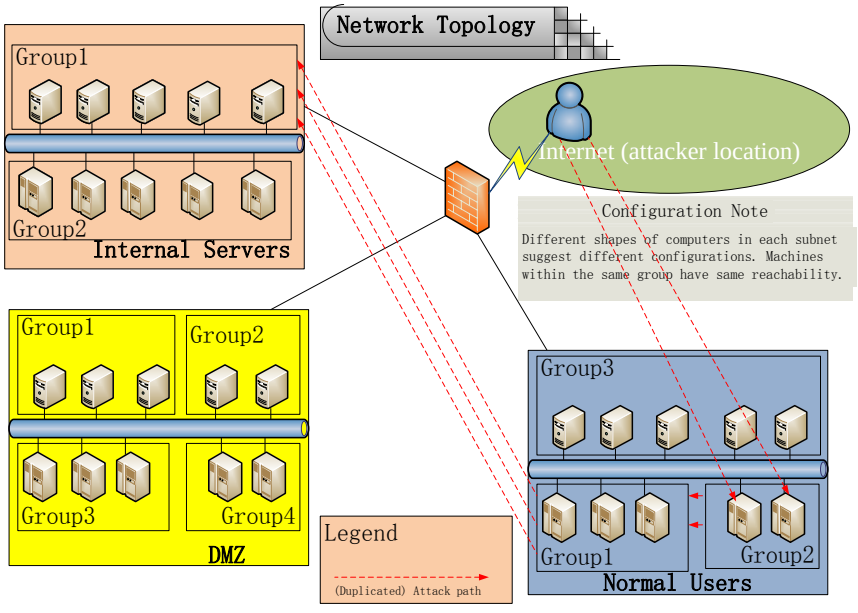


Fig. 5. Network topology.

Table 1. Reachability Table

source		destination		protocol	port
subnet	group	subnet	group		
Internet		DMZ	1	tcp	80
DMZ	1	Internet		tcp	25
Internet		DMZ	4	tcp	80
DMZ	4	Internal	2	tcp	1433
User	2	Internet		tcp	80
User	3	Internet		*	*
Internet		User	2	tcp	80
User	1	Internet		*	*
User	1	Internal	1	nfs	
User	1	Internal	1	Tcp	3306

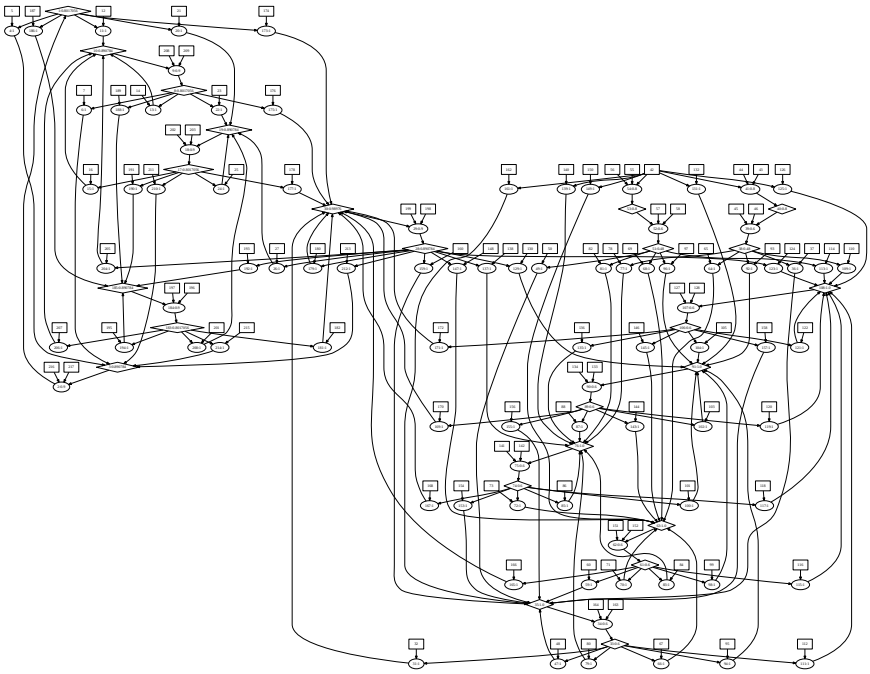
ten machines, evenly divided into two different types of configuration (one is Linux and the other Windows). Machines with different shapes represent different configurations. Machines in the same group have the same configuration and reachability. There are two types of vulnerabilities on each host, and the types of vulnerabilities could be either local, remote server or remote client. The reachability relations among those host groups can be found in Table 1. The table does not include reachability within a subnet, which is unfiltered. If a group does not have any inter-subnet reachability, it will not show up in the table.

3.1 Attack graph generation

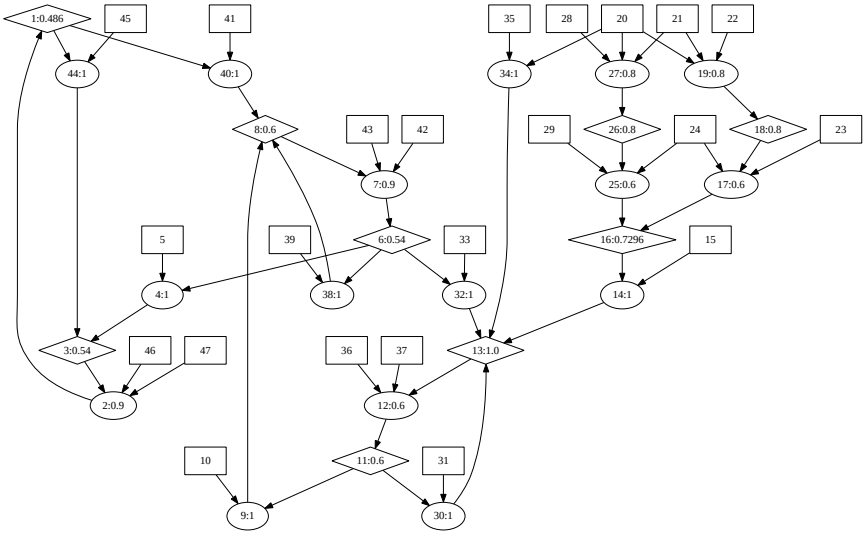
We created the input for MulVAL based on the configuration of the network, and we ran our abstraction model generator to generate an abstracted input. We ran MulVAL with both original and abstracted input and obtained two different attack graphs, shown in Figures 6(a) and 6(b). The size of the attack graph was reduced significantly after abstraction (281 arcs and 217 vertices, to 55 arcs and 47 vertices). We verified that all the "representative" attack paths leading to the attacker goal are retained in the abstracted model.

3.2 Quantitative security metrics

We compared the quantitative metrics results obtained from the original input and the abstracted input. There is a significant difference between the risk metrics on the original network (0.802) and the abstracted one (0.486) for a three-hop attack which is the deepest chain in this experiment (illustrated in the red dotted lines in Fig. 5). This attack chain includes three sets of attack steps: 1) from Internet to Group2 in the "Normal Users" subnet, via client-side vulnerabilities; 2) from Group2 to Group 1 in the "Normal Users" subnet, via service vulnerabilities; 3) from Group1 in the "Normal Users" subnet to Group1 in the "Internal Servers" subnet, via service vulnerabilities. Each group here



(a) Attack graph of the original model (281 arcs and 217 vertices).



(b) Attack graph of the abstracted model (55 arcs and 47 vertices).

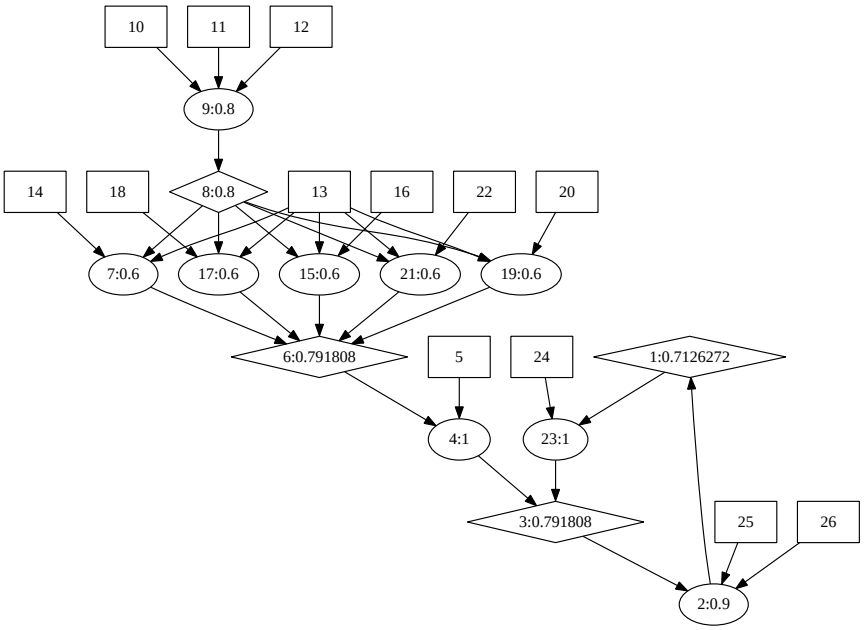
Fig. 6. Comparison of attack graphs from original and abstracted models

refers to a set of hosts with the same reachability and configuration (vulnerabilities). Usually there are multiple attack paths between two groups since there are multiple hosts within each group and they have similar configurations; thus the multiple attack paths have similar natures. From a pure probabilistic semantics, the more paths between two groups, the higher success likelihood the attacker will gain in moving on these paths. However, these paths are not independent and failure on one of them would likely indicate failures on the other; therefore the higher risk metrics are not justified. Moreover, the hosts in the two groups are equivalent in terms of the network access they provide the attackers. Due to the above reasons, the attack paths should be merged into one, before quantitative risk assessment. By removing redundancy in the attack graphs through model abstraction, we avoid distortion in the risk assessment result.

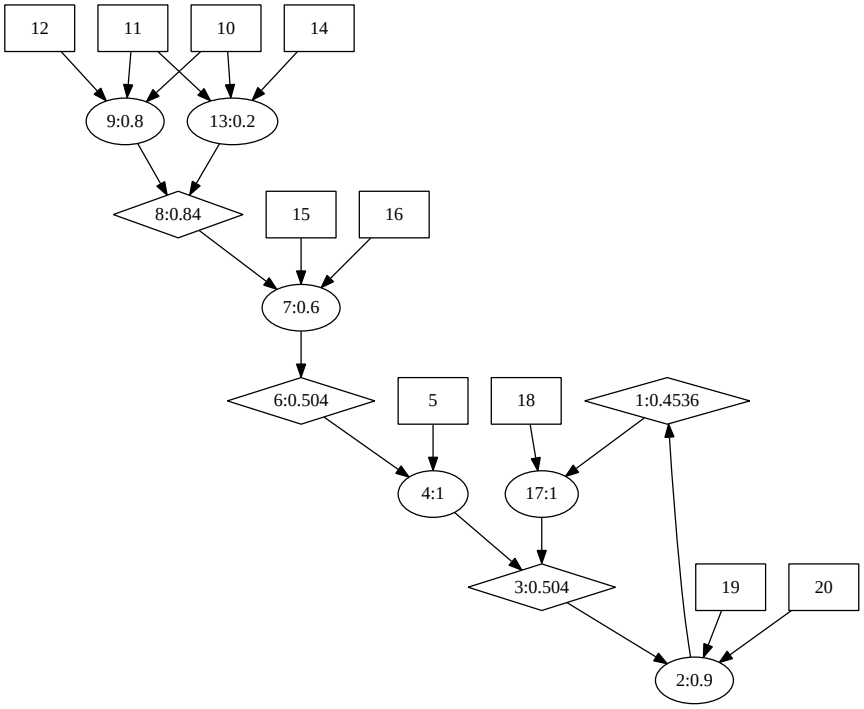
To demonstrate the effect of vulnerability grouping on the quantitative security assessment result, we used the network topology shown in Fig. 1, assuming there are five client-side vulnerabilities (from the same application) on the web server and the remote service vulnerability has been patched. We then computed the likelihood that the web server could be compromised through any of the client-side vulnerabilities, assuming the client program may occasionally be used on the server. The nature of client-side vulnerabilities from the same application are similar from both attacker and the victim's perspective, because the victim would open the same application to trigger the exploits, and due to the similar functionalities (and therefore program components) of the same application, the security holes are also similar. If an attacker knows the structure of the application very well, he should be able to utilize the vulnerability easily; if he does not understand the mechanism of the software, he probably will not be able to utilize any of the security holes with ease. Therefore viewing the same type (client-side or service) of security holes on an application as one is more realistic than treating them independently. We compared the results before and after grouping vulnerabilities. It is obvious that the complexity of the attack graph is reduced significantly from Figure 7(a) to Figure 7(b). More importantly, the quantitative metrics indicating the likelihood that the server can be compromised through one of the client-side vulnerabilities drops from 0.71 to 0.45. This is a more realistic assessment, since the five client-side vulnerabilities are similar and should not significantly increase the attacker's success likelihood.

4 Related Work

Attack graphs have been developed for the purpose of automatically identifying multi-stage attack paths in an enterprise network [1–4, 8–13, 17–21, 23, 24, 26]. It has been observed that attack graphs are often too large to be easily understood by human observers, such as system administrators. In order to reduce the complexity of attack graphs to make them more accessible to use by system administrators, various approaches have been proposed to improve the visualization through abstraction, data reduction, and user interaction [7, 12, 13, 15, 30]. However, not much work has been done to study the effect of attack



(a) Attack graph of a single machine before vulnerability grouping.



(b) Attack graph of a single machine after vulnerability grouping.

Fig. 7. Effect of vulnerability grouping on a single host

graph complexity on quantitative security assessment approaches based on attack graphs. Our study found that complexity caused by repetitive information commonly found in attack graphs not only increases the difficulty for the system administrator in digesting the information provided by the graph, but also distorts the risk picture by unrealistically casting the attack success likelihood for some privileges under probability-based security assessment. We show that such distortion can be avoided by abstracting the input to the attack-graph generator, *i.e.*, the network model, so that such redundancy is removed a priori. By performing abstraction directly on the network model, the attack graph result can also be rendered on a higher level of system description which is easier to grasp by a human user.

Quantitative security assessment methods based on attack graphs have been proposed to indicate the severity levels of various vulnerabilities [5, 6, 16, 22, 25, 27–29]. Such methods typically utilize the dependency relations represented in an attack graph to aggregate individual vulnerability metrics to reflect their cumulative effects on an enterprise network. However, not all dependency relations are explicitly presented in an attack graph, particularly the similarities among large numbers of attack paths leading to the same privilege. Not accounting for the existence of this dependency on a large scale will significantly skew the analysis results. One method of dealing with such hidden dependency is to introduce additional nodes and arcs in the graph to model them, but this will make the visualization problem even more severe. We proposed a method based on model abstraction to remove the redundancy, and thus the hidden dependency resulted from it, so that it is no longer a problem for realistic risk assessment.

The size of enterprise networks could make vulnerability scanning prohibitively expensive [31]. Our abstraction technique provides a possible angle to address this problem. Prioritization can be applied based on the abstract model for identifying scanning which host can potentially provide critical information on the system's security. For example, if a host in the same abstract group has already been scanned, scanning one more host in the group may not provide the most useful information about the system's security vulnerabilities.

5 Conclusion and Future Work

We have presented an abstraction technique to aid in network security assessment based on attack graphs. We show that the large amount of repetitive information commonly found in attack graphs not only makes it hard to digest the security problems, but also distorts the risk picture by disproportionately amplifying the attack likelihood against privileges that have a large number of similar attack paths leading to them. We proposed an approach to abstract the network model so that such repetitive information is removed before an attack graph is generated. The abstraction happens at both the network and the host level, so that machines that have the same reachability relation and similar configurations with respect to vulnerability types are grouped together and represented as a single node in the abstracted model. Our experiments show

that such abstraction not only effectively reduces the size and complexity of the attack graphs, but also makes the quantitative security assessment results more conforming to reality. This shows that appropriate abstraction on the input is a useful technique for attack graph-based analysis.

The abstraction techniques we have proposed are mostly suitable for risk assessment on the macroscopic level of an enterprise network. Abstraction unavoidably loses information and in reality no two hosts are completely identical. The abstracted network model can help in identifying security risks caused by the overall design and structure of the network, but may lose subtle security breaches that may occur due to, *e.g.* misconfiguration of a single host that is mistakenly deemed identical to a group of other hosts since the details of the differences may have been abstracted away. In general the more homogeneous the system is, the more pronounced the effect of abstraction will be. However, since no two hosts are really completely identical, the process is a balancing act. Being overly detailed about a host's configuration may lead to no possibility of abstraction and result in a huge attack graph where important security problems are buried. On the other hand, overly abstract models may lose the important information for subsequent analysis. More research is needed in identifying the most effective abstraction granularity for attack graph-based analysis.

Acknowledgment

This material is based upon work supported by U.S. National Science Foundation under grant no. 1038366 and 1018703, AFOSR under Award No. FA9550-09-1-0138, and HP Labs Innovation Research Program. 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 National Science Foundation, AFOSR, or Hewlett-Packard Development Company, L.P.

References

1. Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.
2. Marc Dacier, Yves Deswarte, and Mohamed Kaâniche. Models and tools for quantitative assessment of operational security. In *IFIP SEC*, 1996.
3. J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of Second IEEE International Information Assurance Workshop*, pages 48 – 56, April 2004.
4. Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.
5. Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic Bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, 2008.

6. John Homer, Xinming Ou, and David Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. Technical report, Kansas State University, 2009.
7. John Homer, Ashok Varikuti, Xinming Ou, and Miles A. McQueen. Improving attack graph visualization through data reduction and attack grouping. In *The 5th International Workshop on Visualization for Cyber Security (VizSEC)*, 2008.
8. Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.
9. Sushil Jajodia and Steven Noel. Advanced cyber attack modeling analysis and visualization. Technical Report AFRL-RI-RS-TR-2010-078, Air Force Research Laboratory, March 2010.
10. Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.
11. Wei Li, Rayford B. Vaughn, and Yoginder S. Dandass. An approach to model network exploitations using exploitation graphs. *SIMULATION*, 82(8):523–541, 2006.
12. Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory, March 2005.
13. Richard P. Lippmann, Kyle W. Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Artz, and Robert Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, October 2005.
14. Peter Mell, Karen Scarfone, and Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Forum of Incident Response and Security Teams (FIRST), June 2007.
15. Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004. ACM Press.
16. Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1), July 2010.
17. Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaâniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25(5), 1999.
18. Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.
19. Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.
20. Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM Press, 1998.
21. Diptikalyan Saha. Extending logical attack graphs for efficient vulnerability analysis. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

22. Reginald Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain, October 2008.
23. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.
24. Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, volume 2, June 2001.
25. Mathias Ekstedt Teodor Sommestad* and Pontus Johnson. A probabilistic relational model for security risk analysis. *Computer & Security*, 29:659–679, 2010.
26. T. Tidwell, R. Larson, K. Fitch, and J. Hale. Modeling Internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, West Point, NY, June 2001.
27. Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*, 2008.
28. Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using attack graphs. In *Third Workshop on Quality of Protection (QoP)*, 2007.
29. Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring the overall security of network configurations using attack graphs. In *Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'07)*, 2007.
30. Leevar Williams, Richard Lippmann, and Kyle Ingols. An interactive attack graph cascade and reachability display. In *IEEE Workshop on Visualization for Computer Security (VizSEC 2007)*, 2007.
31. Yunjing Xu, Michael Bailey, Eric Vander Weele, and Farnam Jahanian. Canvus: Context-aware network vulnerability scanning. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID))*, November 2010.