A MULTI-FACETED APPROACH TO NETWORK SECURITY METRIC THROUGH COMBINING CVSS BASE SCORES

PENGSU CHENG

A THESIS

IN

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2011

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Pengsu Cheng**

Entitled: **A Multi-Faceted Approach to Network Security Metric through Combining CVSS Base Scores**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Yong Zeng — Chair

Dr. Benjamin Fung — Examiner

Dr. Yuhong Yan — External Examiner

Dr. Lingyu Wang — Supervisor

Approved Dr. Mourad Debbabi

Chair of Department or Graduate Program Director

September 9, 2011

Dr. Robin Drew, Dean

Faculty of Engineering and Computer Science

ABSTRACT

A Multi-Faceted Approach to Network Security Metric through Combining CVSS Base Scores

Pengsu Cheng

A network security metric enables the direct measurement of the effectiveness of network security solutions. Combining CVSS scores of individual vulnerabilities provides a measurement of the overall security of networks with respect to potential attacks. However, most existing approaches to combining such scores, either based on Attack Graph or Bayesian Network, share two limitations. First, a dependency relationship between vulnerabilities will either be ignored, or modeled in an arbitrary way. Second, only one aspect of the scores, the probability of successful attacks, has been considered. In this thesis, we address those issues as follows. First, instead of taking each base score as input, our approach works at the underlying base metric level where dependency relationships have well-defined semantics. Second, our approach interprets and combines scores in three different aspects, namely, probability, effort, and skill which may broaden the scope of applications for CVSS and allow users to weigh different aspects of the score for their specific needs. Finally, we evaluate our approach through simulation.

Acknowledgments

I would like to thank all the people who provide help for this thesis and suggstion for my research.

First of all, I would like to express my sincere thanks to my supervisor, Dr. Lingyu Wang, for his help, guidance and advices to my study at Concordia University. I must say that his guidance allows me to learn how to do research which will benefit me for the rest of my life.

I cannot end without thanking my parents for their constant supports. Without their inspiration and encouragement, I would not have been able to tackle this challenging research. I would also thank my labmates Wen Ming Liu, Shuai Liu and William Nzoukou Tankou for their suggestions and help throughout my study at Concordia University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today's critical infrastructures and enterprises have become increasingly dependant on the proper functioning of interconnected information systems. Such systems must thus be secured against potential network intrusions. A network security metric is desirable in this context since *you cannot improve what you cannot measure*. By applying a network security metric immediately before, and after, deploying potential security solutions, these solutions' relative effectiveness can be judged in a more direct and precise manner. Such a capability will render securing computer networks a science rather than an art.

Standard techniques exist for measuring the relative severity of individual vulnerabilities, such as the Common Vulnerability Scoring System (CVSS) [48]. CVSS provides a standard way for security analysts and vendors to assign numerical scores to known vulnerabilities based on their exploitability (for example, whether it can be remotely exploited), potential impact (on confidentiality, integrity, and availability), and other factors. CVSS scores of many vulnerabilities are readily available in vulnerability databases (for example, the NVD [53]).

The CVSS standard provides a solid foundation for developing network security metrics. On the other hand, CVSS is mainly intended to rank different vulnerabilities in the same network, and it does not directly provide a way for measuring the overall security of

different network configurations. Naive ways for combining individual scores, such as taking the average or maximum value, may lead to misleading results, as we shall demonstrate shortly. The main reason is that such naive approaches do not take into consideration the causal relationships between vulnerabilities (that is, exploiting one vulnerability enables exploiting another). Several approaches address this issue based on attack graphs, which is a model of such causal relationships [22, 34, 73].

In this thesis, we first point out following two limitations shared by those existing approaches.

- First, a dependency relationship between vulnerabilities that is not captured in attack graphs may also affect the process of combining scores, which is either ignored, or handled in an arbitrary way, in existing approaches.

- Second, only one aspect of security metrics, namely, the probabilities of attacks, has been considered in most approaches, whereas other important aspects are being ignored.

To address the above issues, we propose a novel multi-faceted approach to separately combine CVSS base metrics. Specifically, instead of taking the base score as a black box input, our approach breaks it down to the underlying base metrics. At the base metric level, dependency relationships between vulnerabilities have well-defined semantics and can thus be easily handled. Our approach also interprets CVSS scores in three different aspects, namely, probability, effort, and skill. We show that the scores need to be combined in different ways for different aspects. We evaluate our approach through simulations. The results confirm the advantages of our approach.

The contribution of this thesis is summarized in the following.

- First, we identify and demonstrate important limitations of existing approaches in defining or interpreting security metrics.

- Second, working at the base metric level, our approach brings out more semantics in combining CVSS scores and consequently produces metric results that may be more meaningful and adoptable to security practitioners.

- Third, the multi-faceted approach to interpreting CVSS scores may broaden the scope of applications for the standard and allow users to weigh different aspects of the score based on their specific needs.

- Fourth, to the best of our knowledge, the simulation presented in this paper is among the first efforts on experimentally evaluating network security metrics.

The rest of this thesis is organized as follows. Chapter 2 reviews related work. Chapter 3 reviews background knowledge necessary for understanding this thesis and demonstrates limitations of existing approaches as the motivation of our work. Chapter 4 presents our approaches to combining base metrics for handling dependencies between vulnerabilities and then extends this approach to further consider three different aspects of the metric scores. Chapter 5 presents the algorithms for combining skill and effort aspects and presents simulation results. Chapter 6 briefly summarizes our other work related to this thesis. Finally, Chapter 7 concludes the thesis.

Chapter 2

Review of Literatures

In the field of vulnerability detection, many work including commercial efforts focus on designing tools for finding existing vulnerabilities in a network. However, these tools do not reveal how such vulnerabilities can be combined in a multi-step attack to penetrate networks. Security analysts are needed to take into account the combining impact of multiple vulnerabilities to qualitatively evaluate the security of a given network against multi-step attacks. Traditional methods for network vulnerability analysis usually involve heavy human intervention by the so-called red teams. Specifically, vulnerability scanners are used to detect vulnerabilities on each hosts with the given network. Then other information about about the network, such as assets, connectivities among hosts, and configuration of each host, are integrated with the detected vulnerabilities by the red team members in order to produce sequences of attacks. Each attack sequence, or attack path, will lead the attacker to an undesirable state against critical assets. For example, a state where the attacker obtains root accesses on an administrator's host. However, the red team approach largely relies on the skills of the team members; for a relatively large network, such manual processes will become error-prone, tedious, and complicated to an unmanageable extent. Therefore, we now review existing solutions that can address such limitations in the following.

2.1 Attack Graph and Applications

Attack graph models emerged as a promising approach towards automating the task of network security evaluation. Early efforts on the defence against multi-step network attacks exist [17, 19, 54, 81]. The concept of attack tree is proposed as trees with AND and OR relations for analyzing the security of systems in [65]. The authors proposed an attack graph model in [59].

Early efforts on building attack graph is also presented in [69]. The authors use forward search for building attack graph, and use following data as input: configuration files, attacker profiles, and a database of manually created attack templates. In their model of attack graph, each node represents a attack templates instantiated with particular hosts and users; each edge is assigned a weight as the probability of success or cost of the attack. Given a set of start nodes and end nodes, the security analysis is proposed as finding the shortest path leading to end nodes.

Another early effort on automatic attack graph generation is presented in [71]. The proposed *require and provide* approach is later widely adopted in defending against multi-step attacks. By linking attack steps via precondition requirements and postcondition capabilities, their approach is used to generate attack scenarios. Along the attack sequence, each attack step enables attackers with more capabilities.

In [62], model checking is applied to the analysis of multi-step network attacks. Each state is composed of known vulnerabilities on network hosts, connectivity among hosts, and the initial capabilities of attacker. Exploits form the transitions between states executed by attackers. Model checking is used to test the reachability of the goal states based on the given formal model. In case a sequence of exploits leading to the goal states exists, the model checker will find this counterexample as a potential attack path that should be blocked in order to secure the network.

Attack paths enumeration is also presented in [40, 66], where all possible attack paths

are generated as counterexamples to a query describing the secure state of the given network by modified model checker applied to the finite-state machine created from network configuration. Based on this model, the authors introduced the analysis of finding a *cut set* in the attack graph so that disabling nodes in the *cut set* can block all attack paths from reaching the goal conditions. The authors also show that it is intractable to find the minimum attack sequence leading to the given goal conditions.

In [66], the goal of an automated analysis is to enumerate potential multi-step intrusions based on prior knowledge about vulnerabilities and their relationships. To avoid the exponential explosion in the number of explicit attack sequences, a compact representation of attack graphs was proposed based on the *monotonicity assumption* saying an attacker never needs to relinquish any obtained capability [3]. The term of *topological vulnerability analysis* is introduced in [63], where the authors provided thorough discussion on how connectivity should be modeled at different layers of the network. Attack graphs have been applied to alert correlation and hypotheses of missed attacks [75].

The *monotonic assumption* is introduced in [3] to address the scalability problem of model checking-based approaches. Their approach relies on an assumption of monotonicity, which states that the preconditions of a given exploit are never relinquished by the successful application of another exploit. In other words, the attacker never needs to backtrack. This assumption reduce the complexity of attack graph from exponential in the number of hosts to the polynomial. The attack graph is built in a two-pass search. The first pass connects the exploits, by searching forward from the initial state, and then deletes those irrelevant states by searching backward in the second pass. Additional analysis is presented, such as finding the minimum attack sequence that leads to the given goal state. Their approach avoid the state explosion problem that may face a model checking-based approach [66].

Recently, a logic programming-based approach is introduced in [55], where Datalog

language is applied to encode knowledge of network attack scenarios. MulVAL is presented in [56] as a security analyzer using the off-the-shelf tools to retrieve information about software and vulnerabilities. Network configuration information is input into the engine, then the attack steps leading the compromise of the network is generated. The time complexity of the provided algorithm is reduced to polynomial in the size of the network.

In the aspect of scalability issue, an notable improvement is presented in the [51], where a hierarchical approach is used to build rules at every level of the aggregation and integrate every levels through common attribute values of attack graph elements or attack connectivity. The attack graph compression is performed by recursively collapse the subgraphs into single vertices, at same time, keeping the same semantics. In addition, the abstraction of protection domains is introduced to reduce the complexity where groups of machines have complete connectivity. The time complexity is reduced to quadratic complexity.

In [52], the authors apply a matrix clustering algorithm to the adjacency matrix of attack graphs, and the resulting adjacency matrix shows the feature of protection domain on the main diagonal. Further improvements to attack graph representations are presented in [35], where the authors use directed graph to model the subnets as nodes and possible inter-subnet attacks as edges. Based on the domination relationships, a dominator tree is used to find possible attacks of inter-subnet or intra-subnet. Then the groups of exploits are reduced to virtual nodes, and the reachability of the attack graph will increase after this abstraction. By applying these two approaches, the complexity of visualized attack graph is significantly reduced, which allow security analyst to quickly grasp imminent threats.

In [78], Wang et al. proposed a framework that uses combining functions to find the combined effect of vulnerabilities in a given network. They proposed the idea of using an analogy to the resistance of electrical circuits in [79] and address the issue of additional dependency between exploits although the solution is not entirely satisfactory since the cycles in attack graphs are mostly ignored.

2.2 Security Metrics

General reviews of security metrics are given in [7,39]. The NIST's efforts on standardizing security metrics are given in [48, 50, 68]. Intuitive properties for security metric are given in [54] to measure the difficulty of attacks in terms of time and efforts based on a Markov model. More recently, several security metrics are proposed by combining CVSS scores based on attack graphs [22, 73, 78, 79]. The minimum efforts required for executing each exploit is used as a metric in [9,58]. A mean time-to-compromise metric is proposed based on the predator state-space model (SSM) used in the biological sciences in [43].

Security metrics are developed for specific applications, such as IDSs [42] and distributed trust management [61]. *Attack surface* measures how likely a software is vulnerable to attacks [45, 57]. Few work exist on measuring zero day attacks. One exception is the effort on ordering different applications in a system by the seriousness of consequences of having a single zero day vulnerability [37]. Another work presents a novel model for measuring the number of zero day vulnerabilities that an network can resist, which is regarded as a metric [74]. Homer and Ou propose using MinCostSAT for automated network reconfiguration with numeric cost assigned to each configuration [33].

Metric has been employed for many different purposes in the broad area of network security. A novel approach to behavioral distance measurement using a new type of Hidden Markov Model is proposed for detecting the compromise of a process or mimicry attacks in [24–26]. Metrics were used for detecting common data types found within network data which can assist in measuring the similarity of network activities in the spatial and temporal aspects [16]. The measurement issues associated with deploying darknets and analyzing the data collected by darknets were studied in [8]. An approach to network risk assessment is proposed to determine the risk level of a network by using Hidden Markov Models [5]. The Mahalanobis distance is used as a metric to measure the similarity between the payload samples and pre-computed models [72].

Granger Causality Test is used as a measurement to determine whether two alerts have causal relationship [60]. A Bayesian network model is given for determining the probability of achieving attack goals in [49]. An average reachability measure is used to identify potentially malicious behavior [27]. Sawilla and Ou extend the PageRank algorithm to rank nodes in an attack graph [64]. Other than relying on universal metrics, a clinic trial approach is proposed to evaluate security products by deploying them with randomly selected targeted populations [67]. The cost of network hardening is quantified in [77].

Measurement of the threat of potential multi-step attacks is another research topic gaining increasing interest. Several different security measurement approaches are reviews in [7, 39, 47]. Several standard security metrics are introduced by NIST in [50]. Further contributions are presented in [68]. As a result, the Common Vulnerability Scoring System (CVSS) is proposed in [48]. In [36], the authors present an overview of many aspects network security metrics.

In [17, 18, 54], Dacier et al. propose more intuitive properties should be satisfied in all security metrics. An approach for the valuation of trustworthiness is proposed in [9], where the authors use this valuation method to determine whether accept or reject an entity as being suitable for sensitive tasks. They use the cost of time and efforts by attackers as metrics to measure the difficulty of attacks. Based on the observation that an attacker's success rate over time almost match an exponential distribution, they apply the Markov model and the Mean Time to Failure (MTTF) to measure the security of a given network. Simple cases of combining individual measurements are presented, while no discussion on general case are extensively given.

In [58], the authors propose to use *weakest attacker* model to the relative risk of different network configurations, where the least conditions leads to success attack are viewed as the key metric. McHugh proposes a metric called *attack surface* to measure the likelihood a software is vulnerable to attacks in [45], where partial order is established on different

network configurations based on their relative security. In [9], Balzarotti et al. present approach to compute the minimum efforts required for executing each exploit. However, there are improvement space in many aspects of security. For instance, the network resources are treated equally significant, and the resistance to attacks is considered as binary.

Relevant work also includes areas, such as the study of trust in distributed system. In the [12], the authors proposed a metric for measuring the trust in an identity that has been established through overlapping chains of certificates. They prove that the way they combine values of trust in certificates into an over-all value of trust is useful in the study of security metrics. In [61], Reiter et al. present a design of principles intended for developing metrics of trust and we find these principles applicable to our study. In [6], the authors use structures similar to attack graph for risk analysis in safety-critical systems, focusing on trust relationships. In [32], the authors present a technique for testing whether a finite execution of events generated by program violates a linear temporal logic (LTL) formula. In the aspect of attack generation, [62, 66] use topological vulnerability analysis to enumerates potential multi-step intrusions based on prior knowledge about vulnerabilities and their relationships. In the aspect of attack response, some studies [47, 76, 80] shows the use of attack graph for hypotheses of alerts missed by IDSs, attack correlation, and the prediction of possible future attacks.

In [21, 22, 73], Wang et al. proposed a probabilistic network security metric based on attack graphs. They presented the use of probability scores for each vulnerability to represent the likelihood that one attacker will exploit the vulnerability or the percentage of attackers that can successfully exploit the vulnerability. Their approach quantifies the cost of removing vulnerabilities in hardening a network, however, it does not consider other hardening options, for example, modifying connectivities. Their approach has another limitation about adopting a qualitative view of damages (when all the given assets are equally critical) and attack resistance when attacks on assets are either trivial or impossible.

In [44], Liu et al. present the idea of using Bayesian network to model network vulnerabilities and find a quantitative value to represent the security level of a network. Bayesian network is used to model each potential atomic attack step. Each vertex corresponds to a single security property violation state; each edge represents an exploitation of single or multiple detected vulnerabilities. Weight is assigned to each edge to represent the probability of successful exploits. In [4], An et al. apply Dynamic Bayesian Network (DBN) to privacy intrusion detection, where they use DBN to relate a database operator's intension to observable factors, such as the time spent on a certain operation. In [22], the authors deploy the similar ideal but use DBN model to combine the CVSS scores for measuring network security.

2.3 General metrics and applications

In 1971, Akiyama [1], published the first attempt to use metrics for software quality prediction when he proposed regression-based model for module defect density. From the mid-1970, more interests in measurement of software complexity which is pioneered by Halstead [30,46], and measurement functional size such as function points in [2,70]. These approaches are independent of programming language. Later, research on extending and refining complexity metrics and functional metrics gain more academic interest. For example, in [15,31], software metrics that are relevant to object oriented language are introduced.

By borrowing ideas from the Total Quality Management field, Basili et al. proposed a simple scheme for making metrics activities always goal-driven [11]. They claimed that a metrics program established without clear and specific goals and objectives is doomed to fail [29]. Their measurement studies on quantifying the effectiveness of software engineering methods and technologies [10] point out the great challenge for academic software metrics to benchmark [41] and evaluate the effectiveness of different software engineering methods and technologies.

Causal models are also applied in the study of software engineering metrics. Because the underlying theory of Bayesian probability has been studied for a long time, together with the improvement in the modelling and algorithms of Bayesian belief net (BBN), Fenton et al. take the advantages of BBN to design risk management decision-support tools that build on existing relatively simple metrics [20]. Their approach combines different aspect of software development and testing and enable many kinds of predictions, assessments and trade-offs during the software life-cycle.

Chapter 3

Preliminaries

In this chapter, we first review two important concepts relevant to our further discussions, namely, attack graph and CVSS. We then motivate our discussions by demonstrating limitations of existing approaches through examples.

3.1 Attack Graph

Attack graph is a graphical representation of inter-dependent vulnerabilities found in networked hosts. An attack graph is a directed graph whose nodes are partitioned into two classes, namely, *exploits* and *security conditions* (or simply *conditions*). An exploit is typically represented as a predicate $v(h_s, h_d)$, where h_s and h_d represent two connected hosts and v a vulnerability on the destination host h_d. A security condition is a predicate $c(h)$, indicating the host h satisfies a condition c relevant to one or more exploits. Notice that h_s, h_d, and v are abstract notations that could in practice possess different semantics, for example, h_s and h_d can be host names, IP addresses, and so on, and v can be the name of a vulnerability or its ID in a vulnerability database. These are more formally stated as Definition 1.

Definition 1 *An attack graph G is a directed graph $G(E \cup C, R_r \cup R_i)$ where the set of nodes include E, a set of* exploits*, and C, a set of* conditions*, and the set of edges include the* require *relation $R_r \subseteq C \times E$ and the* imply *relation $R_i \subseteq E \times C$.*

Corresponding to the inter-dependency between exploits and conditions, the two types of edges in an attack graph have different semantics. First, the *require* relation R_r is a directed edge pointing from a condition to an exploit, which means the exploit cannot be executed unless the condition is satisfied. For example, an exploit $v(h_s, h_d)$ requires following two conditions, that is the existence of the vulnerability v on h_d and the connectivity between h_s and h_d. Second, the *imply* relation R_i pointing from an exploit to a condition means executing the exploit will satisfy the condition. Notice that there is no edge directly connecting two exploits (or two conditions).

Figure 1 shows a simple example of attack graphs which depicts a simple scenario where a file server (host 1) offers the File Transfer Protocol (ftp), secure shell (ssh), and remote shell (rsh) services; a database server (host 2) offers ftp and rsh services. The firewall only allows ftp, ssh, and rsh traffic from a user workstation (host 0) to both servers. In the attack graph, exploits of vulnerabilities are depicted as predicates in ovals and conditions as predicates in clear texts. The two numbers inside parentheses denote the source and destination host, respectively. The attack graph represents three self-explanatory sequences of attacks (attack paths).

Two important semantics 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.

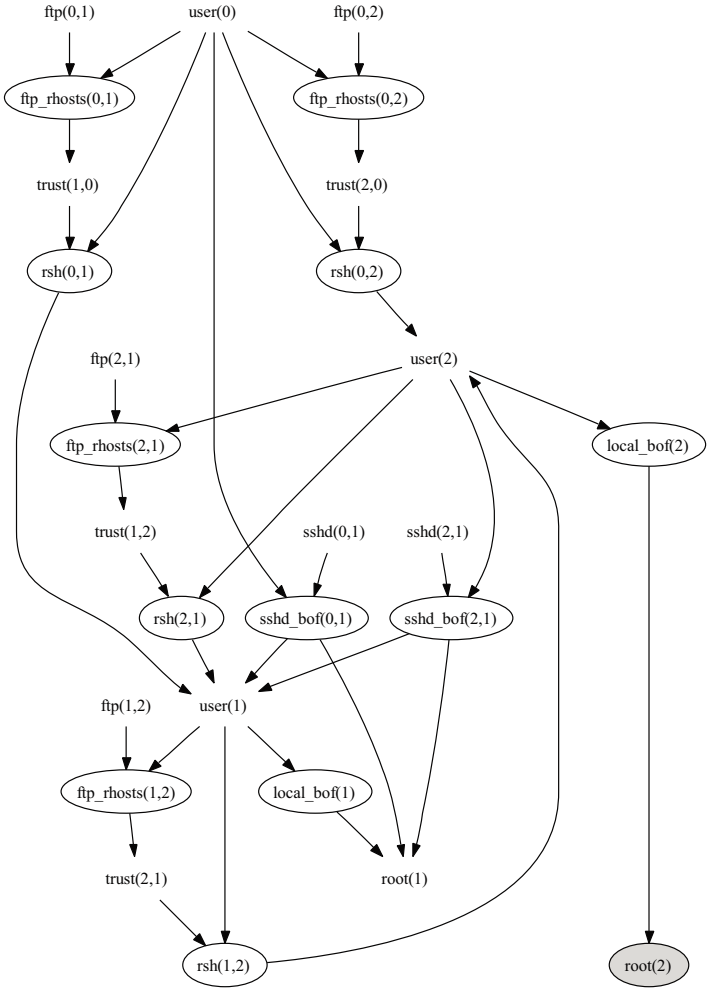


Figure 1: An Example of Attack Graph

To generate an attack graph, two types of inputs are necessary, namely, *type graph* and *configuration graph*. Type graph represents expert knowledge about the dependency relationship between vulnerabilities. On the other hand, configuration graph represents hosts and their connectivity and vulnerability information. We assume the domain knowledge required for type graph is available from 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 [38]. On the other

hand, we assume the configuration information including vulnerabilities and connectivity can be obtained using available network scanning tools, such as the Nessus scanner.

3.2 Common Vulnerability Scoring System (CVSS)

Our discussions in subsequent sections will need metric scores assigned to individual vulnerabilities according to the Common Vulnerability Scoring System (CVSS) [48]. The CVSS is an open and free framework that provides a means for assigning quantitative values to vulnerabilities based on well defined metrics. In CVSS, each vulnerability is to be assigned a *base score (BS)* ranging from 0 to 10, on the basis of two groups of totally six *base metrics* [48]. The base metrics are intended to stay constant over time and across different user environments. Optionally, the base score can be further adjusted with temporal and environmental scores. We briefly review the CVSS standard in the following to make this thesis more self-contained.

3.2.1 Base Scores

The *Base Score* (BS) for each vulnerability quantifies its intrinsic and fundamental properties that are supposed to be constant over time and independent of user environments. The base score ranges from 0 to 10. The Base Score is calculated based on the following six metrics:

- Access Vector - AV: This indicates the types of accesses required for exploiting the vulnerability. Possible values are Local (numerical value 0.395), Adjacent Network (0.646), and Network (1.0), which are all self-explanatory.

- Access Complexity - AC: A quantitative measure of the attack complexity required to exploit the vulnerability. The range of values are: High (0.35), Medium (0.61) and Low (0.71).

- Authentication - Au: A measure of the the number of times an attacker must authenticate to a target in order to exploit a vulnerability. The defined range of values are: Multiple (0.45), Single (0.56) and No (0.704).

- Confidentiality - C: A measure of the impact on confidentiality following a successful exploitation with the following defined range of values: None (0.0), Partial (0.275) and Complete (0.660).

- Integrity - I: A measure of the impact on integrity following a successful exploitation with the following defined range of values: None (0.0), Partial (0.275) and Complete (0.660).

- Availability - A: A measure of the impact on availability following a successful exploitation with the following defined range of values: None (0.0), Partial (0.275) and Complete (0.660).

The CVSS Framework imposes the use of a vector which encodes the metric score values used to compute the overall score for a vulnerability. The following is an example vector:

$$AV : N/AC : L/Au : N/C : N/I : C/A : C$$

from which we can derive the numerical scores as indicated above.

The Base Metric score (BS) is computed as follows,

$$\begin{aligned} BS &= round\_to\_1\_decimal((0.6 * Impact + 0.4 * Exploitability - \\ &\quad 1.5) * f(Impact)) \\ Impact &= 10.41 * (1 - (1 - ConfImpact) * (1 - IntegImpact) * (1 - AvailImpact)) \\ Exploitability &= 20 * AccessVector * AccessComplexity * Authentication \\ f(Impact) &= 0 \ \ if \ \ Impact = 0, \ 1.176 \ \ otherwise \end{aligned} \tag{1}$$

Using the example vector, the following demonstrates how to compute the BS:

- $Exploitability = 20 * 1 * 0.71 * 0.704 == 9.9968$
- $Impact = 10.41 * (1 - (1 - 0) * (1 - 0.660) * (1 - 0.660) == 9.2066$
- $f(impact) = 1.176$
- $BaseScore = round\_to\_1\_decimal((0.6*9.2066)+(0.4*9.9968)-1.5)*1.176 == 9.4$

3.2.2 Temporal Scores

The *Temporal Score* (TS) quantifies a vulnerability when considering properties of the vulnerability that may change over time. The three temporal metric values used in CVSS are:

- Exploitability - E: Indicates the current state regarding the availability of exploit codes or techniques, with the following defined range of values: Unproven (0.85), Proof-of-concept (0.90), Functional (0.95), High (1.00) and Not Defined (1.00).
- Remediation Level - RL: Indicates the current situation regarding the availability of remediation solutions. The defined range of values is: Official Fix (0.87), Temporary Fix (0.90), Workaround (0.95), Unavailable (1.00) and Not Defined (1.00)
- Report Confidence - RC: Indicates the degree of confidence regarding the existence of a vulnerability and the technical details. The range of defined values is: Unconfirmed (0.90), Uncorroborated (0.95), Confirmed (1.00) and Not Defined (1.00).

The Temporal Metric Score (TS) is computed as follows:

$$TS = round\_to\_1\_decimal(BS * E * RL * RC) \quad (2)$$

For convenience, the following product is defined:

$$TGS = (E * RL * RC) \tag{3}$$

$$TS = round\_to\_1\_decimal(BS * TGS) \tag{4}$$

Consider the example of a vulnerability with the above Base Score and the following Temporal vector:

$$E : POC/RL : W/RC : C$$

The Temporal Score (TS) is calculated as follows:

- $TGS = 0.90 * 0.95 * 1 == 0.855$
- $TS = 9.4 * 0.855 == 8.0$

3.3 Existing Approaches and Their Limitations

To illustrate limitations of existing approaches to combining CVSS scores, we consider a toy example. Figure 2 depicts a network consisted of two hosts (host 1 and 2), and an attacker on host 0 in the Internet. We shall consider two cases based on the same network. In Case 1, we assume host 1 to be a UNIX server running a telnet service and host 2 a Windows XP workstation running the Universal Plug and Play (UPnP) service. In Case 2, we assume host 1 and 2 swap their OS (and hence the corresponding services). In both cases, the firewalls disallow any traffic except accesses to those services.

We assume the telnet service contains the vulnerability CVE-2007-0956 [53], denoted by v_{telnet}, which allows remote attackers to bypass authentication and gain system accesses via providing special usernames to the service. We also assume the UPnP service contains

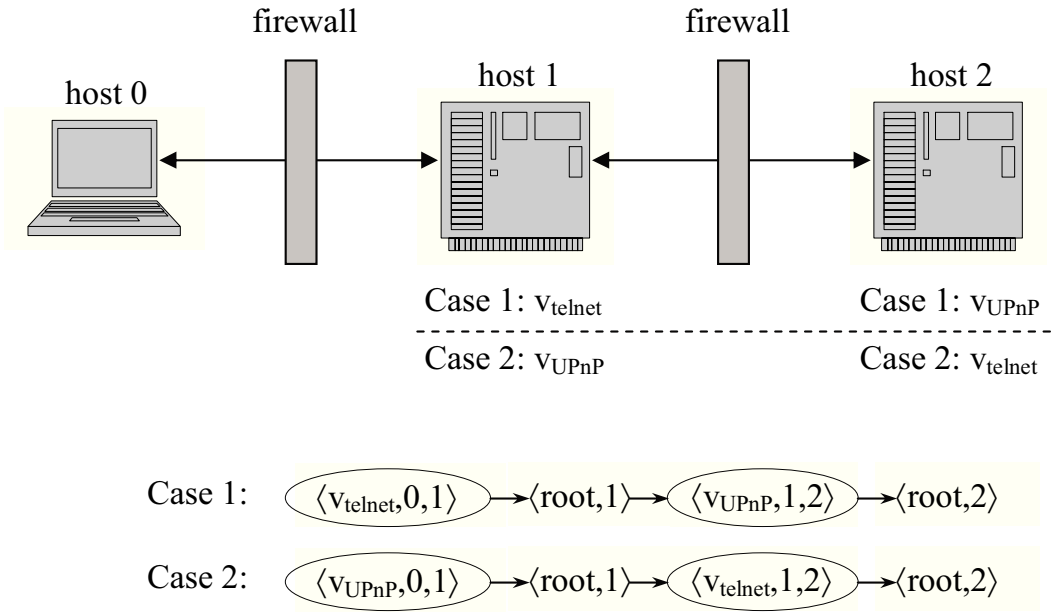


Figure 2: An Example Network

the vulnerability CVE-2007-1204 [53], denoted by v_{UPnP}, which is a stack overflow that allows attackers on the same subnet to execute arbitrary codes via sending specially crafted requests.

Table 1 shows the CVSS base metrics of those two vulnerabilities [53]. By applying Equation 1, we can calculate the base score of vulnerability v_{telnet} to be $BS = 7.6$. Similarly, we have $BS = 6.8$ for vulnerability v_{UPnP}. The difference in those base scores suggests that vulnerability v_{telnet} is relatively more severe then v_{UPnP}. This is sufficient for purposes like prioritizing vulnerabilities for removal. However, for other purposes, such as comparing the relative security of the two configurations of Case 1 and 2, we shall need to combine the base scores for judging the overall network security.

Metric Group	Metric	Metric Value of v_{telnet}	Metric Value of v_{UPnP}
Exploitability	Access Vector	Network (1.00)	Adjacent Network (0.646)
	Access Complexity	High (0.35)	High (0.35)
	Authentication	None (0.704)	None (0.704)
Impact	Confidentiality	Complete (0.660)	Complete (0.660)
	Integrity	Complete (0.660)	Complete (0.660)
	Availability	Complete (0.660)	Complete (0.660)
Base Score (BS)		7.6	6.8

Table 1: The CVSS Base Metrics and Scores of Two Vulnerabilities

3.3.1 Average and Maximum

First, we consider two naive approaches to combining the CVSS scores, namely, by taking the average value (7.2 in both Case 1 and 2) and maximum value (7.6 in both cases), respectively. Although those approaches provide a rough sense of overall security, their limitations are also obvious. Since the average and maximum values are both defined over a set, they do not depend on where those vulnerabilities are located in a network and how they are related to each other. For example, if we assume the UNIX server in Figure 2 is the only important network asset, then intuitively the overall network security is quite different between Case 1 (in which an attacker can directly attack the UNIX server on host 1) and Case 2 (in which the attacker must first compromise the Windows workstation on host 1 and use it as a stepping stone to attack host 2). Nonetheless, by taking the average or maximum base score, we cannot distinguish between the two cases.

3.3.2 Attack Graph-Based Approach

The above naive approaches lead to misleading results because they ignore causal relationships between vulnerabilities. Such causal relationships can be modeled in attack graphs, as illustrated in the lower portion of Figure 2. Each triple $\langle v, h_1, h_2 \rangle$ inside an oval represents an exploit of vulnerability v on host h_2 from host h_1; each pair $\langle c, h \rangle$ represents a security-related condition c on host h; each arrow either points from a pre-condition to an

exploit, or from an exploit to a post-condition.

The attack graph-based approach [73] first converts each CVSS base score into a probability by dividing with its domain size, and then assigns the probability to exploits with the corresponding vulnerability. Each condition is also assigned a probability 1. The probabilities are then combined based on following causal relationships: An exploit is reachable only if all of its pre-conditions are satisfied (that is, a conjunction); a condition is satisfied as long as one reachable exploit has that condition as its post-condition (that is, a disjunction).

In Case 1 of our example, we should assign $7.6/10 = 0.76$ to $\langle v_{telnet}, 0, 1\rangle$, and $6.8/10 = 0.68$ to $\langle v_{UPnP}, 1, 2\rangle$, and 1 to both conditions. We can then update $\langle root, 1\rangle$ as a post-condition of $\langle v_{telnet}, 0, 1\rangle$ to the new value 0.76; now by taking $\langle root, 1\rangle$ again as a pre-condition of $\langle v_{UPnP}, 1, 2\rangle$, we can then update $\langle v_{UPnP}, 1, 2\rangle$ and $\langle root, 2\rangle$ with the value $0.76 \times 0.68 = 0.52$. Similarly, we will obtain the same result for Case 2. At first glance, this is reasonable, since the attacker is exploiting the same two vulnerabilities in both cases.

Unfortunately, upon more careful observation, we shall see this is not the case. First, we recall that the vulnerability v_{UPnP} (CVE-2007-1204) requires the attacker to be within the same subnet as the victim host. In Case 1, exploiting v_{telnet} on host 1 helps the attacker to gain accesses to local network, and hence makes it easier to exploit host 2. In another word, exploiting v_{telnet} has the effect of increasing the probability of successfully exploiting v_{UPnP}. In contrast, in Case 2, there is no such affect due to the reversed order of exploits. This difference between the two cases is apparently not captured by the identical result 0.52 produced by this approach.

3.3.3 Bayesian Network (BN)-Based Approach

Next we consider the Bayesian network-based approach [22]. The lower left-hand side of Figure 3 shows the BN corresponding to Case 2 of our example. Similar to the previous approach, the CVSS base score is first converted into a conditional probability. For example, the conditional probability of successfully exploiting v_{UPnP}, given that all of its pre-conditions are satisfied, is assigned as 0.68. The lower right-hand side of Figure 3 depicts the corresponding Conditional Probability Table(CPT) for each exploit in Case 2. The probability of reaching the goal state, which is assumed as exploiting both vulnerabilities in this example, can be calculated as $P(v_{telnet} = T) = \sum_{v_{UPnP} \in \{T,F\}} P(v_{telnet} = T, v_{UPnP}) = 0.52$.

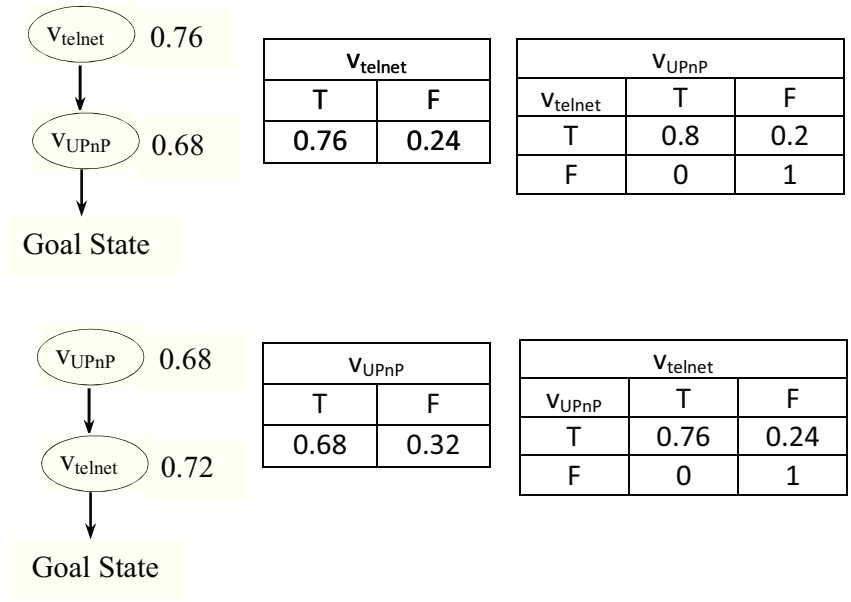


Figure 3: Bayesian Network-Based Approach [22]

The upper left-hand side of Figure 3 depicts the BN for Case 1. Since exploiting v_{telnet} on host 1 makes it easier to exploit v_{UPnP} on host 2, according to this approach, we should assign to $P(v_{UPnP} = T | v_{telnet} = T)$ a value higher than the one directly derived from the base score (that is, 0.68). If we assign, say, 0.8, then the possibility of achieving the goal

state is $P(v_{UPnP} = T) = \sum_{v_{telnet} \in \{T,F\}} P(v_{UPnP} = T, v_{telnet}) = 0.61$. This result is more accurate since it reflects the dependency relationship between the two exploits. However, note that we have chosen an arbitrary value 0.8 because this approach does not provide means for determining that value, which is clearly a limitation.

Chapter 4

Main Approach

In this chapter, we present our main approaches to dealing with dependencies between vulnerabilities by combining base metrics, and to combining metrics based on different aspects of the scores' semantics.

4.1 Combining Base Metrics

We first give an overview of our approach, which is followed by the formal framework and an example.

4.1.1 Overview

We first illustrate our approach by revisiting the example in Figure 2. The key observation is that the existing approaches discussed in the previous section all take the CVSS base scores as their inputs. The base score is regarded as a black box, and the underlying base metrics are not involved in the process of combining scores. However, we notice that the dependency relationships between vulnerabilities are usually only visible at the level of base metrics, which makes it difficult for those approaches to properly handle such relationships.

Approaches	Case 1	Case 2	Summary
Average	7.2	7.2	Ignoring causal relationships (exploiting one vulnerability enables the other)
Maximum	7.6	7.6	
Attack graph-based approach [73]	0.52	0.52	Ignoring dependency relationships (exploiting one vulnerability makes the other easier)
BN-Based approach [22]	0.61	0.52	Arbitrary adjustment for dependency relationships
Our approach	0.58	0.52	Adjustment with well-defined semantic

Table 2: Comparison of Different Approaches

Instead of working at the base score level, our approach handles potential dependency relationships between vulnerabilities at the base metric level. For the above example, the dependency relationship can be easily modeled at the base metric level as follows. When an attacker successfully exploits v_{telnet} on host 1, he/she gains accesses to the local network of host 2, which is required for exploiting v_{UPnP} on host 2. At the base metric level, this simply means the $AccessVector$ metric of v_{UPnP}, which has the value $AdjacentNetwork$, should be replaced with $Network$, since the attacker is effectively accessing v_{UPnP} remotely (using host 1 as a stepping stone).

With this adjustment to the base metric $AccessVector$, we can apply Equation 1 to recalculate a new *effective base score*, which is equal to 0.76 in this case. Clearly, the new result is also higher than the original value 0.68, but this result has well defined semantics, unlike the arbitrary value chosen by the previous approach [22]. The final score corresponding to Case 1 shown in Figure 2 can now be calculated as $P(v_{UPnP} = T) = \sum_{v_{telnet} \in \{T,F\}} P(v_{UPnP} = T, v_{telnet}) = 0.58$. In Table 2, we summarize our discussions about the above example and compare the results produced by different approaches.

4.1.2 Formal Framework

We are now ready to formalize our approach. We assume an *attack graph* is given as a directed graph $G = \langle E \cup C, \{\langle x, y\rangle : (y \in E \wedge x \in pre(y)) \vee (x \in E \wedge y \in post(x))\}\rangle$ where E, C, $pre()$, and $post()$ denote a set of exploits (each of which is a triple $\langle v, h_s, h_d\rangle$ denoting an exploit of vulnerability v on host h_d from host h_s), a set of security-related conditions, a function that maps an exploit to the set of its pre-conditions, and a function that maps an exploit to the set of its post-conditions, respectively [38].

We call a condition *initial condition* if it is not the post-condition of any exploit. A sequence of exploits is called an *attack sequence* if for every exploit e in the sequence, all its pre-conditions are either initial conditions, or post-conditions of some exploits that appear before e in that sequence. We say an exploit e' is an *ancestor* of another exploit e, if e' appears before e in at least one minimal attack sequence (that is, an attack sequence of which no subsequence is a valid attack sequence).

We also assume the CVSS base metrics can be obtained for each exploit e as a vector bm of six numeric values each of which corresponds to a base metric [48]. We shall use the notation $bm[AV], bm[AC], \ldots, bm[A]$ to denote each corresponding element of the vector bm. Finally, we assume the dependency relationships between exploits are given using a function $adj()$ formalized in Definition 2. When a base metric m ($m \in \{AV, AC, Au, C, I, A\}$) of an exploit e is affected by another exploits e' due to dependency relationships, we assume $adj(e, e', m)$ is given. And we use $< e', e >$ to denote that exploit e can be affected by exploit due to dependency relationship.

Definition 2 *Given an attack graph G with the set of exploits E, we define a function $adj() : E \times E \times \{AV, AC, Au, C, I, A\} \rightarrow [0, 1]$. We call $adj(e, e', m)$ the* adjusted value *for the metric m of exploit e due to e'.*

Next, we formalize the concept of *effective base metric* and *effective base score* in Definition 3. For each exploit e, the effective base metric simply takes the original base

metric if no adjusted value is given. Otherwise, the effective base metric will take the highest adjusted value defined over any ancestor of e (note that an exploit may be affected by many exploits in different ways, leading to more than one adjusted values), because a metric should always reflect the worst case scenario (that is, the highest value). The effective base score basically applies the same equation to effective base metrics instead of the original metrics. In the definition, both effective base metric and score can be defined with respect to a given subset of exploits, which will be necessary later in Chapter 4.

Definition 3 *Given an attack graph G with the set of exploits E, the adjusted values given by function $adj()$, the CVSS base metric vector bm for each $e \in E$, and any $E' \subseteq E$ (E' will be omitted if $E' = E$), we define*

- *the* effective base metric *vector ebm of e with respect to E' as*
  - *$ebm[m] = bm[m]$ for each $m \in \{AV, AC, Au, C, I, A\}$, if $adj(e, e', m)$ is not defined for any ancestor e' of e in E'.*
  - *$ebm[m] = adj(e, e', m)$, if $adj(e, e', m)$ is the highest value defined over any ancestor e' of e in E'.*
- *the* effective base score *ebs of e as the base score calculated using Equation 1 with the base metrics replaced with the corresponding effective base metrics.*

Finally, Definition 4 formalizes a Bayesian network (BN)-based model for combining the effective base scores. The directed graph is directly obtained from the attack graph. The conditional probabilities are assigned according to the causal relationships between an exploit and its pre- and post-conditions. Since the dependency relationships between exploits are already reflected in our definition of effective base scores, the BN needs not to explicitly model them. With the BN model, we can easily calculate the probability of satisfying any given goal conditions (or equivalently, the probability of important network assets being compromised).

Definition 4 *Given attack graph G with exploits E, and the effective base score ebs for each $e \in E$, we define a Bayesian network $B = \langle G, Q \rangle$ where*

- *G is the attack graph interpreted as a directed graph with each vertex representing a random variable taking either T (true) or F (false), and the edges representing the direct dependencies among those variables.*
- *Q is the collection of conditional probabilities assigned as the following.*
  - *$P(c = T|e = T) = 1$, for each $e \in E$ satisfying $c \in post(e)$.*
  - *$P(e = T| \bigwedge_{\forall c \in pre(e)}(c = T)) = ebs/10$.*

4.1.3 An Example

We now illustrate our approach by applying it to the example shown in Figure 4. The figure shows a fictitious attack graph in which the dotted lines indicate dependency relationships, which will be explained shortly. Table 3 gives the corresponding model obtained by applying our formal framework as introduced above.

Specifically, we assume exploit C will give an attacker the local shell accesses to target host, which is required for exploiting D (since its base metric AV is $Local$), as indicated by the dotted line from C to D. This dependency relationship is modeled using the function $adj()$, as shown on the right-hand side of Figure 4. Also, we assume that exploiting D requires the same authenticated account as B (both of their base score Au are $Single$). If attackers can exploit B, no additional accounts are required for exploiting D. This dependency relation is indicated by the dotted line from B to D, and modeled using the function $adj()$. Therefore, we can replace the base metric of exploit D with its effective base metrics for the two cases ($ebm_{D|C}$, $ebm_{D|B,C}$ as shown on the right-hand side of Figure. 4) in order to calculate its effective base scores (we assume the impact metrics of all exploits are $ConfImpact = Complete$, $IntegImpact = Complete$, and $AvailImpact =$

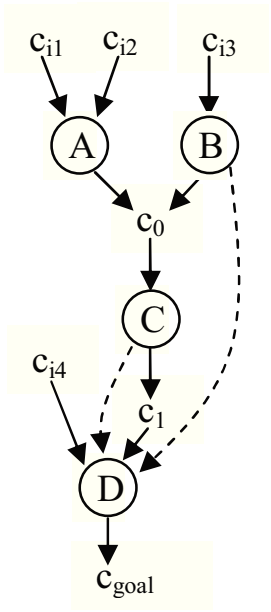


Figure 4: An Example Attack Graph

$None$). Here we need consider two cases: $ebs_{D|C}$ when the attacker has already exploited C; and $ebs_{D|B,C}$ when the attacker has exploited B and C (we don't consider the case that B is already exploited while C is not, because from the semantics of this attack graph we can see that D cannot be exploited without C being exploited at first). We then calculate $P(D = T)$ using the BN model shown in the upper right-hand side of Figure 4 as $P(D = T) = \sum_{A,B,C \in \{T,F\}} P(D = T|B, C)P(C|A, B)P(A)P(B) = 0.27$.

Adjusted Values:

$adj(D, C, AV) = 1.0$

$adj(D, B, Au) = 0.704$

Base Scores:

Exploits	AV	AC	Au	bs
A	Network	Low	None	9.43
B	Network	Medium	Single	7.95
C	Network	Medium	None	8.77
D	Local	Medium	Single	6

Effective base metric of D:

$ebm_{D|C} = \langle Network, Medium, Single \rangle$

$ebm_{D|B,C} = \langle Network, Medium, None \rangle$

Effective base score of D:

$ebs_{D|C} = 7.95$

$ebs_{D|B,C} = 8.77$

Table 3: The Corresponding Model

A	
T	F
0.943	0.057

B	
T	F
0.795	0.205

C			
A	B	T	F
F	F	0	1
T	F	0.877	0.123
F	T	0.877	0.123
T	T	0.877	0.123

D			
B	C	T	F
F	F	0	1
T	F	0	1
F	T	0.795	0.205
T	T	0.877	0.123

Table 4: The BN Model

4.2 Considering Different Aspects of Scores

In this section, we first demonstrate the need for interpreting and combining base scores from different aspects. We then extend our approach accordingly to combine metric scores based on different aspects, and provide an example of our approach.

4.2.1 The Need for Considering Different Aspects

CVSS metrics and scores can be interpreted in different ways. In this thesis, we shall consider three aspects of the scores.

- First, as discussed in the previous section, the difference in scores may indicate different probabilities of attacks. For example, an $AccessVector$ metric value of $AdjacentNetwork$ corresponds to a lower numerical score than the value $Network$, which can be interpreted as that a vulnerability that requires accesses to local networks is less likely being exploited than one that is remotely accessible.

- Second, we can also interpret the difference in scores as the minimum amount of time and effort spent by any attacker. For example, if a vulnerability requires multiple authentications at both OS and applications, then it certainly will require more time and effort than one that requires no authentication at all.

- Third, the difference in scores can also indicate the minimum skill level of an attacker who can exploits that vulnerability. For example, exploiting a vulnerability with its $AccessComplexity$ score as $High$ will likely require more skills than exploiting one that has the value Low (note that each exploitability metric may potentially be interpreted in all three aspects).

Considering different aspects of CVSS scores will require different methods for combining such scores. We demonstrate this fact through an example. Figure 5 shows a network

consisted of four hosts (host 1 through 4) and another host on the internet (host 0). We assume there are firewalls between the hosts that prevent any traffic except those indicated by lines shown in the figure. We also assume host 1 through 4 each has a vulnerability, denoted by a letter inside parentheses. Finally, we assume the base scores are partially ordered, that is, vulnerability B is more severe than all others, and A is more severe than C (for simplicity, we shall not consider effective base scores in this example). We now consider how the scores should be combined for each of the three aspects.

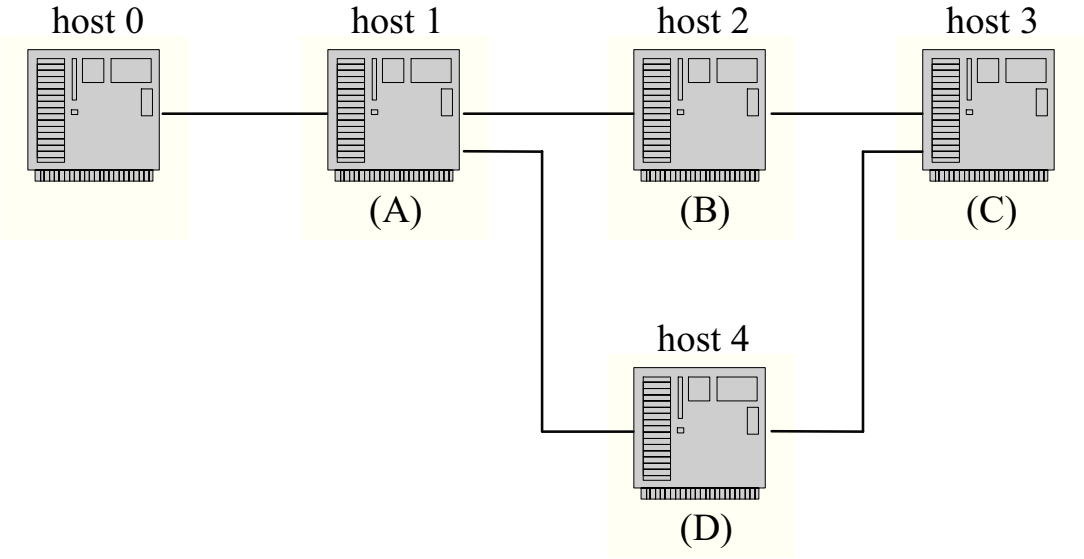


Figure 5: An Example of Different Aspects

- First, suppose we have assigned probabilities P_A, P_B, P_C, and P_D to those four vulnerabilities based on the base scores. Also suppose the security of host 3 is our main concern. Clearly, the probability of host 3 being compromised can be calculated as $P = P_A * (P_B + P_D - P_B * P_C) * P_C$. Next, suppose we remove host 4 from the network. The probability will change to $P = P_A * P_B * P_C$, which is now smaller. This is reasonable since, by removing host 4, an attacker now has only one choice left, that is, to reach host 3 through host 2. Finally, suppose we further remove host 2

from the network, the probability now becomes $P = P_A * P_C$, which is larger. This is also reasonable since now an attacker only need to compromise host 1 before he/she can reach host 3.

- We show a different story by considering the effort aspect. First, suppose we have assigned some effort scores F_A, F_B, F_C, and F_D to the four vulnerabilities based on their base scores (we shall discuss how the effort score should be defined later). Without considering dependency relationships, the effort spent on exploiting vulnerabilities will accumulate.

  Therefore, addition is the natural way to combine the effort scores. However, there is one more complication. In this example, an attacker may compromise host 3 in two ways, either through host 2 or host 4. Since a metric should measure the worst case scenario, it should yield the minimum effort required to compromise host 3. That is, $F = F_A + F_B + F_C$ (note that F_B is less than F_D due to our assumption).

  If we remove host 4 from the network, we can easily see that the effort score will remain the same, $F = F_A + F_B + F_C$, instead of becoming smaller like in the case of attack probability. This is reasonable since vulnerability D is not on the minimum-effort attack sequence so its removal will not affect the effort score. If we further remove host 2 from the network, we can see that the effort score now reduces to $F = F_A + F_C$.

- Finally, we show yet another different story by considering the skill aspect. First, suppose we have assigned some effort scores S_A, S_B, S_C, and S_D to the four vulnerabilities based on their base scores. Based on our assumption, we have that S_B is the smallest among the four and S_A is less than S_C. It is now easy to see that to compromise host 3, the minimum level of skills required for any attacker is S_C regardless of which sequence of attacks is being followed. Also, whether we remove host 4 or

host 2 (even host 1) from the network does not affect the skill score.

4.2.2 Combining Scores for Different Aspects

We now formalize our approach to combining scores for the effort and skill aspects. For both aspects, we shall only consider the exploitability metric group, that is, the first three elements of the effective base metric vector. The formula shown in Definition 5 basically converts the exploitability score (defined in CVSS as the multiplication of the three metrics) to the same domain as the CVSS base score. Note that the effective base metric vector of each exploit is now defined with respect to a given subsets of exploits. This is necessary since whether a base metric needs to be adjusted will depend on which attack sequence is involved.

Definition 5 *Given an attack graph G with the set of exploits E and the effective base metric vector ebm for each $e \in E$ with respect to some $E' \subseteq E$, we define for e both the* effort score $es(e)$ *and* skill score $ss(e)$ *with respect to E' as* $round\_to\_1\_decimal(frac0.6395ebm[AV] * eb$ $0.2793)$.

Although both scores are defined in the same way, they will need to be combined differently, as demonstrated in the previous section. Definition 6 formalizes the way we combine those scores. Roughly speaking, for combining effort score, we find an attack sequence whose summation of effort scores is the minimum among all attack sequences (although such an attack sequence is not necessarily unique, the minimum value is always unique); for combining skill scores, we find an attack sequence that whose maximum effort score is the minimum among all attack sequences. The range of the result of $es(e)$ and $ss(e)$ is within $[1, 10]$ so that we can follow the same range of CVSS Base score.

Definition 6 *Given an attack graph G with the set of exploits E, and the effort score $es(e)$ and skill score $ss(e)$ for each $e \in E$, we define*

- *the* accumulative effort score *of* e *as* $F(e) = min(\{\sum_{e' \in q} es(e') : q$ *is an attack sequence with* e *as its last element*$\})$ *(here* $es(e')$ *is defined with respect to* q*).*

- *the* accumulative skill score *of* e *as* $S(e) = min(\{max(\{ss(e') : e' \in q\}) : q$ *is an attack sequence with* e *as its last element*$\})$ *(here* $ss(e')$ *defined with respect to* q*).*

4.2.3 An Example

Now we demonstrate how our approach can be applied to calculate the accumulative effort and skill scores through a more elaborated example. Figure 6 shows an example attack graph in which two attack sequences can both lead to the assumed goal condition. In the upper table of Table 5 we show CVSS metrics of the vulnerabilities. The dashed lines in the attack graph indicate dependency relationships between the exploits. Specifically, the adjusted $AccessVector$ metric value of C should be $Network$ and the adjusted $Authentication$ metric value of F should be $None$.

The calculated cumulative effort scores and cumulative skill scores are shown on the lower part of Table 5. Note that in calculating the scores for each sequence, we need the effort and skill scores that are defined with respect to that sequence. In particular, exploit F has two different effort and skill scores, since its effective base metric $Authentication$ is adjusted in sequence q_2 (due to exploit E) but not in sequence q_1.

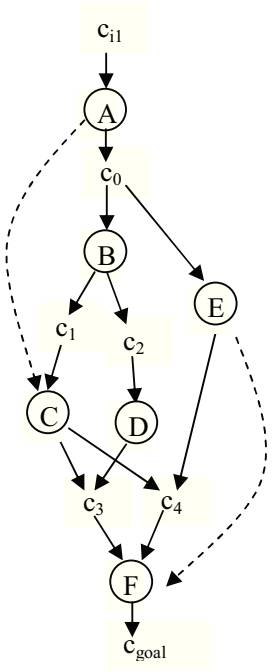


Figure 6: An Example Attack Graph

	AV	AC	Au	es, ss
v_A	Network	Low	None	1
v_B	Network	Medium	None	1.21
v_C	Local	Low	None	1 (w.r.t. q_1)
v_D	Local	Medium	None	3.49
v_E	Network	Medium	Single	1.59
v_F	Network	Medium	Single	1.59 (w.r.t. q_1) and 1.21 (w.r.t. q_2)

Attack Sequence	Effort $F(F)$	Skill $S(F)$
$q_1 : A \rightarrow B \rightarrow C \rightarrow F$	4.8	1.59
$q_2 : A \rightarrow B \rightarrow D \rightarrow E \rightarrow F$	8.5	3.49

Table 5: The Effort and Skill Scores

Chapter 5

Algorithm and Simulation

In this chapter, we first discuss algorithm design for implementing the proposed models, and then give simulation results that confirm the advantages of our approach.

5.1 Algorithms

In order to capture the dependency relation in an attack graph, we will need to introduce new nodes and edges into an existing attack graph so to represent the semantics of dependency relations. More specifically, given a set of exploit nodes $S_k = \{e_i | < e_i, e_k >\}$, which contains all exploit nodes that have dependency relation with exploit node e_k, we add additional *virtual linkage nodes* and corresponding edges to represent the equivalent dependency relation in the attack graph model.

For example, on the left-hand side of Figure 7 is an original attack graph with one dependency relation $< B, D >$. On the right-hand side, we add an virtual node D' and intermediate condition node $c_{d'}$ between node B and D to represent the scenario that, when B is exploited, the adjusted score for D is applied by D'. Note D and D' share the same set of pre-conditions and post-conditions so that D' keep the same relation in the attack graph, and meanwhile the intermediate condition node $c_{d'}$ is added to make sure that D' can be

exploited when B is already exploited.

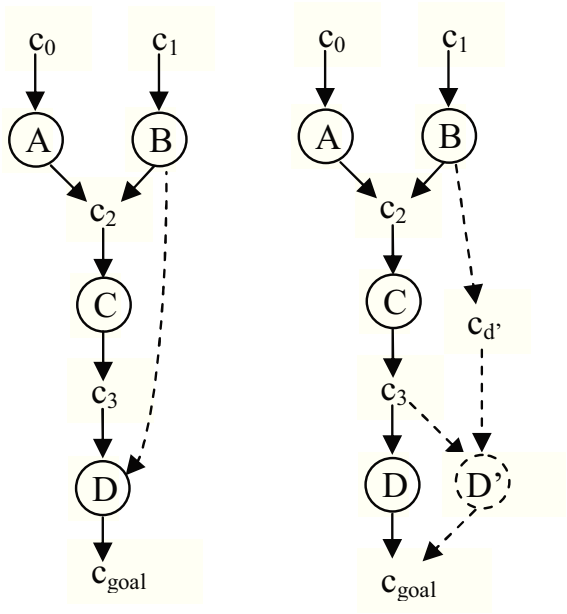


Figure 7: The Virtual Linkage Nodes

In Figure 7, we have shown a simple example for adding such virtual linkage nodes. More generally, the procedure includes three steps shown as follows.

1. Given the original attack graph $G =< V, E >$ and a set $S_k = \{e_i | < e_i, e_k >\}$, for each $e_i \in S_k$, add intermediate condition node c_i' onto V with $post(e_i) = post(e_i) \cup \{c_i'\}$.

2. For each subset $T_k \subseteq S_k (T_k \neq \emptyset)$, add one node e_{T_k} onto V with $pre(e_{T_k}) = pre(e_k)$ and $post(e_{T_k}) = post(e_k)$. For each $e_i \in T_k$, append the corresponding intermediate condition node c_i' to $pre(e_{T_k})$.

3. For each e_{T_k}, assign $es(e_{T_k})$ and $ss(e_{T_k})$ as $es(e_k)$ and $ss(e_k)$ respectively, with respect to the corresponding T_k.

Before applying the following two algorithms, we first extend the original attack graph G by appending virtual linkage nodes for all dependency relations.

Procedure *CombineSkill*
Input: An attack graph G, a set of goal conditions C_{goal}
Output: A non-negative real number as combine skill score
Method:
1. **Create** new exploit e_{goal} with $pre(e_{goal}) = C_{goal}$, $post(e_{goal}) = \emptyset$
2. **Create** a queue Q
3. **Create** a array $score$ to record the score of each condition
4. **Enqueue** initial conditions onto Q
5. **Create** a set $M = Q$, assign $s(c) = 0$ for each $c \in M$
6. **While** Q is not empty:
7. **Dequeue** an item from Q to v
8. **If** v is exploit node:
9. **Let** $s(v) = max(\{s(u) : u \in pre(v)\} \cup \{ss(v)\})$
10. **Let** $M = M \cup \{v\}$
11. **Enqueue** each node $c' \in post(v)$ such that $\{e : c' \in post(e)\} \subseteq M$
12. **Else** :
13. **Let** $s(v) = min(\{s(u) : v \in post(u)\})$
14. **Let** $M = M \cup \{v\}$
15. **Enqueue** each node $e \in post(v)$ such that $pre(e) \subseteq M$
16. **Return** $s(e_{goal})$

Figure 8: Combining the Skill Scores

5.1.1 Combining skill scores

In Figure 8, we show formally the method for combining the $skillscore$ of a given attack graph. Note that the input attack graph is the one after appending *virtual linkage nodes*.

In procedure *CombineSkill*, we first place all conditions to be achieved by attackers in a single set C_{goal}, and create a new exploit e_{goal} with pre-condition set $pre(e_{goal}) = C_{goal}$. Then, follow the aforementioned procedure, we add the corresponding *virtual linkage nodes* for each adjustable e_j with respect to the set S_j, which contains the exploits that can affect the skill score of node j due to dependency relationships. Then we conduct a *Breadth First Search (BFS)* to traverse the attack graph from initial conditions. At each step, we pick the minimal skill score of exploits that lead to condition v as the score of

the condition v; for each exploit v, we assign the $ss(v)$ as the maximal of the scores of the pre-conditions of exploit v. In this way, we update the skill score of each exploit e as the accumulative skill score $S(e)$, up to the goal exploit e_{goal}.

The time complexity of this procedure can be derived from that of a BFS as ($O(|V| + |E|)$). The difference between our procedure and standard BFS is that, before a node is about to be enqueued, all of its predecessors will be checked (Line 11 in Figure 8). The worst case is that each node has $|V| - 1$ predecessors. So the worst case time complexity of this procedure is $O(|E| \cdot |V| + |E|)$.

We now prove the correctness of procedure *CombineSkill* by induction. First, we extend the definition of $S(e)$ in Definition 6 to both exploit nodes and condition nodes. Specificlaly, if v is an exploit node, $S(v)$ still follow the definition in Definition 6. If v is a condition node, we let $S(v) = min(\{max(\{ss(e') : e' \in q\}) : q$ is an attack sequence with e' as its last element, and $v \in post(e')\})$. We need to show that $s(v) = S(v)$ is true for very node v at the end of the procedure, which shows that the procedure correctly computes the accumulative skill score of each exploit. We prove this by induction on $|M|$ with the induction hypothesis $\forall(v \in M)\ s(v) = S(v)$.

- Base case($M = \{c : c$ is initial condition$\}$): Since the initial conditions can be enabled without exploiting any exploits, $s(c) = 0 = S(c)$ for each $c \in M$.
- Inductive hypothesis: When a new node v to be added to M, $s(v) = S(v)$ for each $v \in M$.
- Inductive case: Here we only need to prove that the new node v satisfying $s(v) = S(v)$.
  - If v is an exploit: assume that $S(v) = s' < s(v)$. Let Q be the attack sequence ending with v such that $S(v) = max(\{ss(e) : e \in Q\}) = s'$. Since $v \in Q$, we have $ss(v) \leq s'$. By line 9 of the procedure *CombineSkill*, we have $s(v) =$

$max(\{s(u) : u \in pre(v)\} \cup \{ss(v)\}) > s'$, so $s' < max(\{s(u) : u \in pre(v)\})$. Let's assume $c \in pre(v)$ such that $s' < s(c)$. And since $c \in M$, $S(c) = s(c) > s'$. However, since the attack sequence Q can reach the condition c, $S(c) \leq max(\{ss(e) : e \in Q\}) = s'$. So we come to a contradiction. Therefore we have $s(v) = S(V)$.

- If v is a condition: assume that $S(v) = s' < s(v)$. Let Q be the attack sequence ending with e' such that $v \in post(e')$ and $S(e') = s' = S(v)$. By line 13 of the procedure *CombineSkill*, we have $s(v) = min(\{s(u) : v \in post(u)\}) \leq s(e')$, and $e' \in M$, because $v \in post(e')$. Then we have $s(e') = S(e') = S(v)$, so $s(v) \leq S(v)$, which contradicts with the assumption that $S(v) < s(v)$. Therefore $s(v) = S(v)$.

Since e_{goal} can be reached by at least one attack sequence, at last, $e_{goal} \in M$. According the previously proved claim, we have $S(e_{goal}) = s(e_{goal})$. Therefore the procedure *CombineSkill* is correct.

5.1.2 Combining effort scores

To calculate the combined effort score, we first create a goal exploit e_{goal} with respect to the set of goal conditions C_{goal}, and add *virtual linkage nodes* similar to the *CombineSkill* procedure. Then, starting from e_{goal}, we recursively call the *Min_Score* procedure to traverse though the attack graph G, and find the accumulative effort score of e_{goal}.

More specifically, in line 4 to 7, we find the set C of further conditions needed to be enabled by exploiting more exploits, excepted for initial conditions. Line 8 and 9 deal with the base case of this recursion, where no more conditions needed to be enabled, and E contains a set of exploits which are necessary for an attack sequence with e_{goal} as its last element. Line 10 to 17 deal with the recursive cases, where different attack sequences are explored. In line 14, we use some heuristic feature to avoid exploring attack sequences

that cannot result in minimal combined effort scores. In line 15 to 17, potential attack sequences with minimal combined effort scores are explored by recursively call the *Min_-Score* sub-procedure, and return the minimal combined effort scores among the different attack sequences.

The problem of combining effort score is NP-hard. This can be easily proved by considering a special case which is similar to the problem of computing k-zero day safety [74], which has been proven to be NP-hard. For example, when we assign $ss(e) = 1$ for each exploit e in the attack graph G, finding combined skill score is equivalent to finding k-zero day safety of G.

Procedure *CombineEffort*
Input: A attack graph G, a set of goal conditions C_{goal}
Output: A non-negative real number $Score$
Method:
1. **Create** new exploit e_{goal} so that $pre(e_{goal}) = C_{goal}$
2. **Let** $Score = Min\_Score(\{e_{goal}\}, \emptyset)$
3. **Return** $Score$

Sub-Procedure *Min_Score*
Input: A set of exploits to be exploited E, a set exploits already exploited E_{path}
Output: A non-negative real number $MinScore$
Method:
4. **Let** $C = \{c : c \in pre(e'), e' \in E\}$
5. **For** each c in C
6. **If** c is initial condition
7. **Delete** c from C
8. **If** $C = \emptyset$
9. **Return** $\sum_{e \in E} es(e)$
10. **Else**
11. **Let** $E' = \{e' : c \in post(e'), c \in C\}$
12. **Let** $MinScore = +\infty$
13. **Let** $E_{path} = E_{path} \cup E$
14. **For** each subset $E'' \subseteq E'$ satisfying $C \subseteq \bigcup_{e'' \in E''} post(e'')$ and for any set $E''' \subset E''$, $C \cap \bigcup_{e''' \in E'''} post(e''') \neq C$
15. **Let** $Score = Min\_Score(E'', E_{path})$
16. **If** $Score < MinScore$
17. **Let** $MinScore = Score$
18. **Return** $MinScore$

Figure 9: Combining the Effort Scores

5.2 Simulation Results

We evaluate the proposed approaches through Monte Carol simulations that takes random attack graphs and simulated attackers as inputs and compares the distribution of resultant metric scores. The motivation for simulation is that, to the best of our knowledge, there do not exist public datasets of real-world attack graphs that can be used for experiments.

We employ the Boston university Representative Internet Topology gEnerator(BRITE) [13], and the Georgia Tech Internetwork Topology Models topology generator(GT-ITM) [28] to generate simulated network topologies. We then inject vulnerability information into the generated network topologies to obtain network configurations, and finally generate attack graphs from the configurations using the standard two-pass procedure [3]. All simulations are conducted on a computer equipped with a 3.0GHz CPU and 8GB memory.

The objective of the first two simulations is to use Monte Carol simulation to evaluate our approach from the aspect of attack probability, as detailed in Chapter 4. For this purpose, we first randomly assign base metrics to each vulnerability and dependency relationships between pairs of vulnerabilities. We then apply both our approach and the existing approach by Marcel *et al.* [22] to calculate the probability of attacks with respect to a set of randomly chosen goal conditions. We also compare our result to the percentage of simulated attackers (each simulated attacker is modeled as a random subset of vulnerabilities that he/she can exploit) who can successfully reach the goal conditions.

In Figure 10, we use the BRITE topology generator to create random network topology. The X-axis is the average effective base score of all vulnerabilities in each network divided by 10, denoted by β. The Y-axis is either the combined score of attack probability (for both our approach and the approach by Marcel *et al.*) or the percentage of successful attackers. Each result is the average of 500 simulations on different network configurations. The curve $Simulation$ corresponds to the simulated attackers, which is used as a base line for comparison. The line β corresponds to the naive approach of taking the average base score

among all vulnerabilities in a network, which is clearly inaccurate.

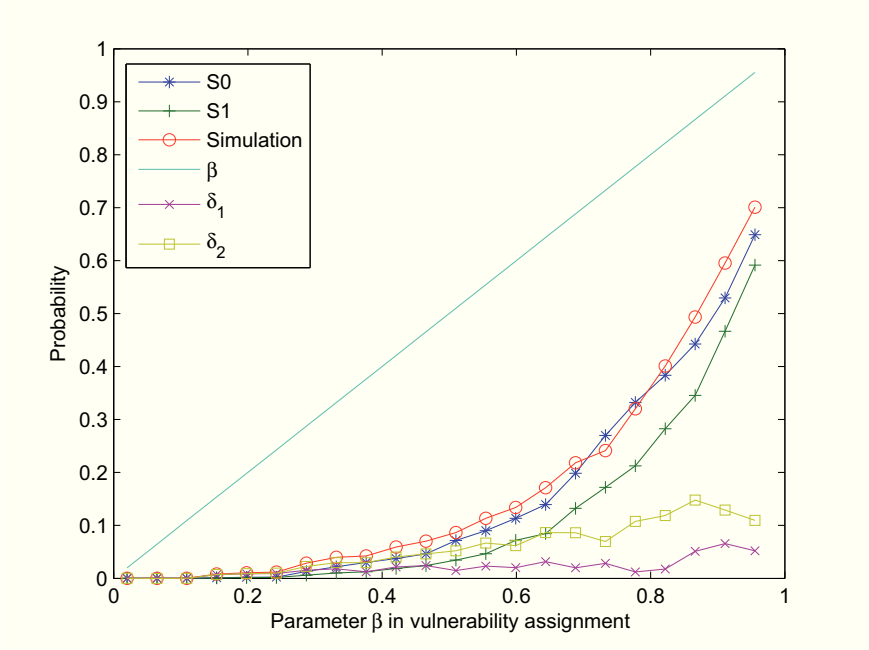


Figure 10: The Probability Aspect (BRITE)

In Figure 10, the curve S_0 corresponds to our approach and the curve S_1 the approach by Marcel *et al.*. The curve δ_1 represents the absolute value of the difference between the probability result from our approach and the simulated attackers, and the curve δ_2 represents the absolute value of the difference between the probability result from the approach by Marcel *et al.* and the simulated attackers. Clearly, our result is closer to the simulated attackers than theirs. Also, our probability is always higher than theirs due to the proper handling of dependency relationships. In Figure 10, we have assigned dependency relationships to n pairs of randomly chosen vulnerabilities where n is drawn from a uniform distribution on $[0, 3]$. Figure 11 shows a similar simulation, except that we increase the amount of dependency relationships to n pairs where n is now drawn from a (uniform distribution on $[0, 5]$. The results show that our approach is still very close to the simulated attackers, whereas Marcel's result further deviates from the baseline results. In Figure 12

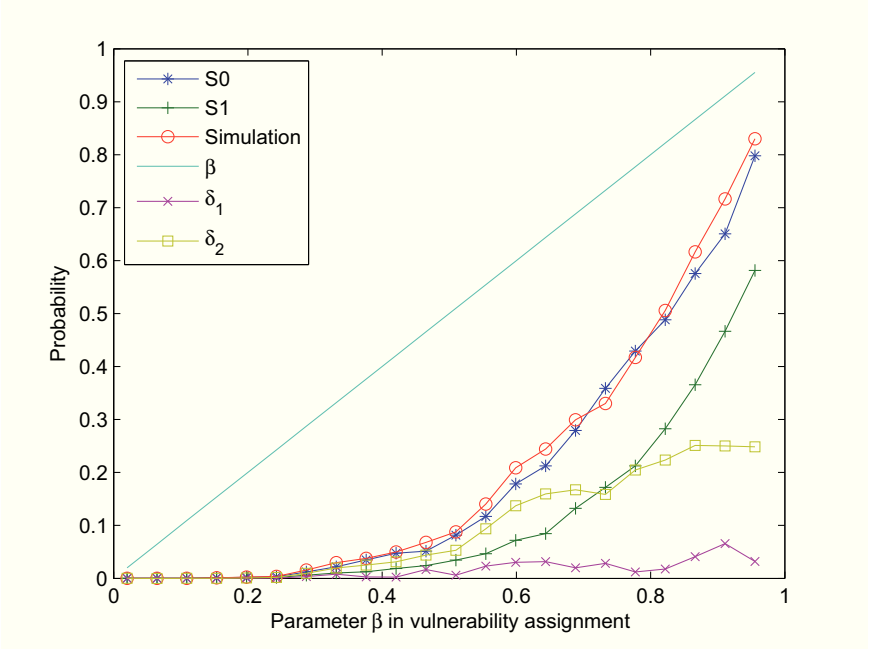


Figure 11: Increased Dependency Relationships (BRITE)

and Figure 13, we present similar experiment result as Figure 10 and Figure 11 respectively, by using another topology generator GT-ITM.

In Figure 14, we fix the distribution of CVSS score distribution based on [23]. We keep the uniform distribution of dependency number, but change the average number from 0 to 3 as shown on the X-axis. Similar to the aforementioned experiments, the Y-axis is either combined score of attack probability (by our approach and Marcel *et al.*), or the percentage of successful simulated attackers.

The objective of the next simulation is to study the deviation of combined scores from the baseline of simulated attackers. For this purpose, Figure 15 depicts the results computed on 800 different networks. The X-axis is the percentage of simulated attackers who can reach the goal conditions, and the Y-axis is the combined probability score. The dots S_0 and S_1 correspond to the results of our approach and Marcel's, respectively. The two solid lines labeled with S_0 and S_1 represent the average probability score within each 0.05

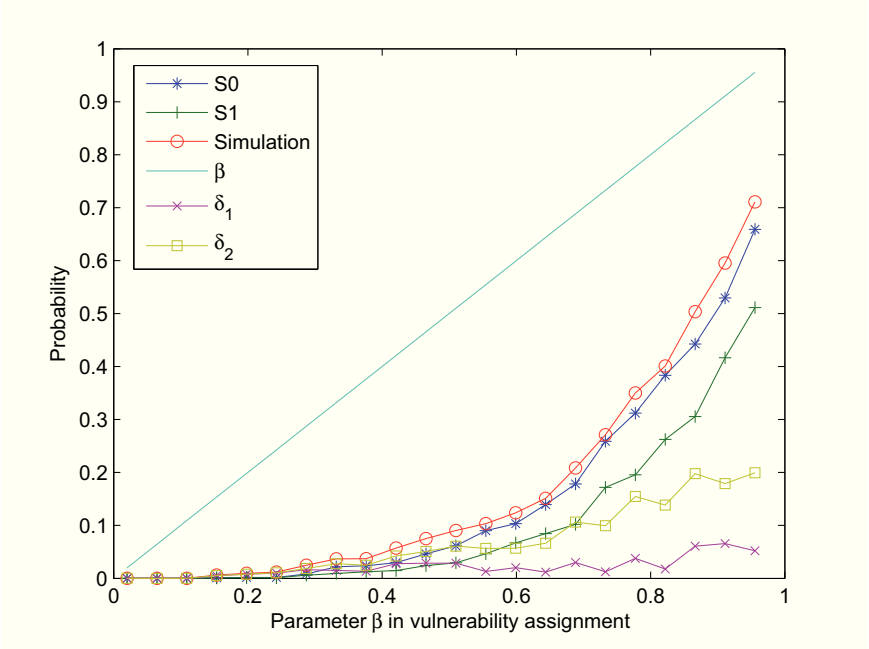


Figure 12: The Probability Aspect (GT-ITM)

interval of the X-axis. The two polygon areas depict the distribution of combined scores produced by the two approaches. As we can see from the figure, our results evenly spread around the simulated attackers' results, whereas Marcel's results are almost always lower than the baseline results.

The next simulation aims to evaluate our approach from the skill aspect. For this purpose, each simulated attacker is randomly assigned a skill level based on exponential distribution (significantly less attackers possess a higher level of skills). Each simulated attacker can only exploit those vulnerabilities whose skill scores (as defined in Chapter 4) are no greater than the attacker's assigned skill level. In Figure 16, the X-axis is the percentage of successful simulated attackers, and the Y-axis is either the skill score produced by our approach or the skill level of simulated attackers. Each result is the average of 100 simulations. The curve *Skill metric* is the cumulative skill score of our approach; the curve *Minimal skill* corresponds to the lowest skill level of simulated attackers among those who

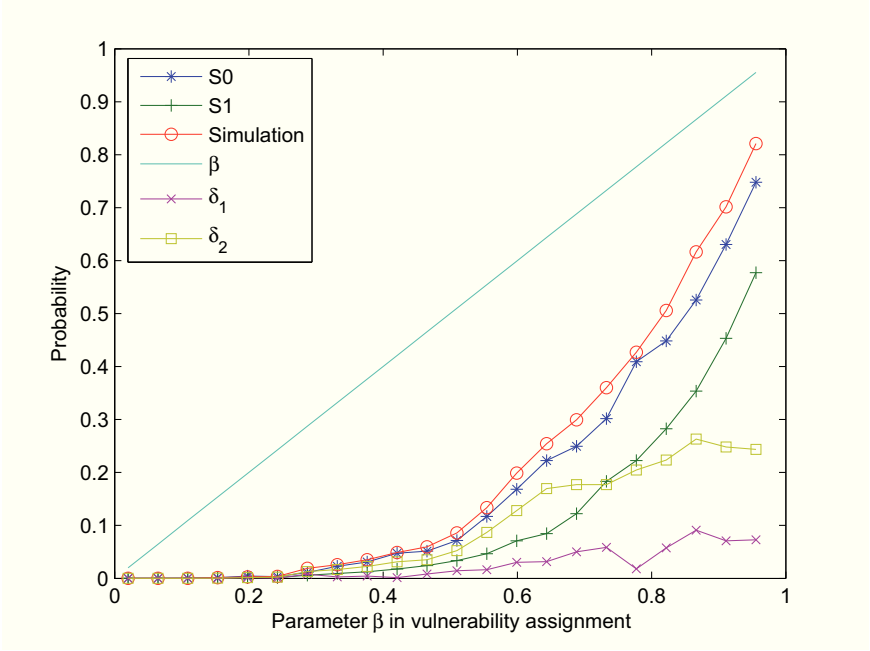


Figure 13: Increased Dependency Relationships (GT-ITM)

can reach the goal conditions. We can see that those two curves almost overlap each other, indicating the accuracy of our approach. The curve *Average skill* shows the average skill level among successful simulated attackers, which has the same trend, but is always higher than our result. The curve *Vulnerability average* shows the average skill score of all vulnerabilities in each network, which is clearly not a valid metric for skill level.

The next simulation evaluates our approach from the effort aspect. For this purpose, each simulated attacker is randomly assigned an*effort threshold* based on exponential distribution (less attackers are willing to spend more effort). We assume each simulated attacker will only exploit those vulnerabilities whose effort scores (as defined in Chapter 4) are no greater than the attacker's assigned effort threshold. In Figure 17, the X-axis is the percentage of successful simulated attackers, and the Y-axis is either the effort score or the effort threshold (of simulated attackers). The curve *Effort metric* is the cumulative effort score of our approach; the curve *Minimal effort* and *Average effort* respectively correspond

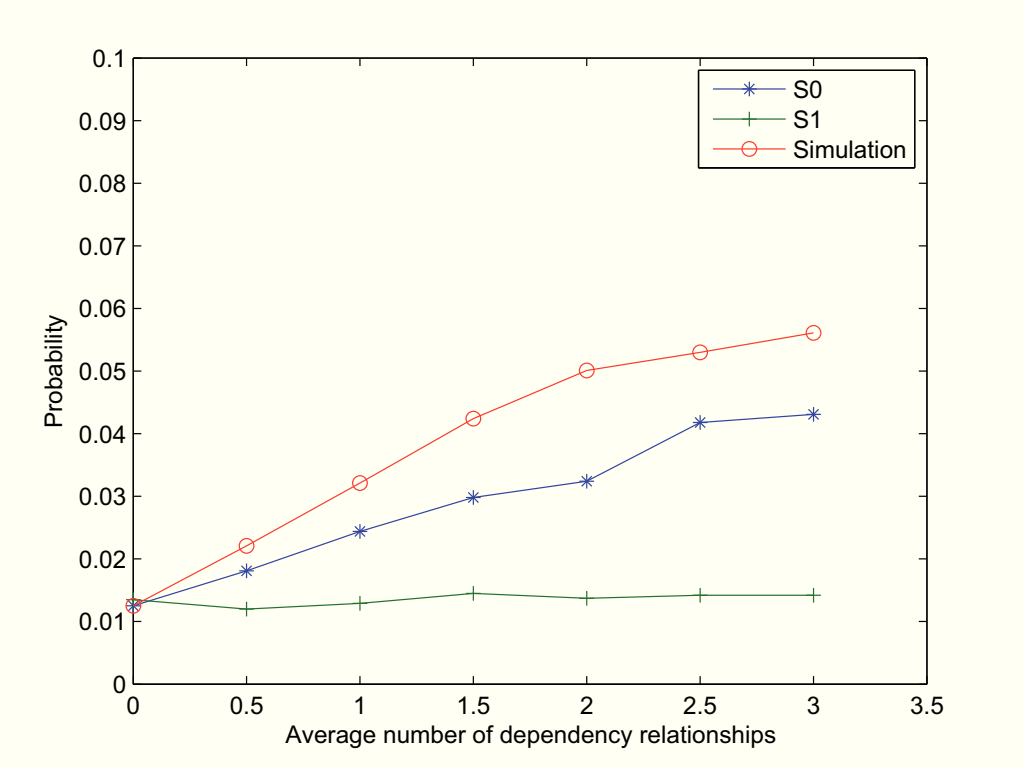


Figure 14: The Probability Aspect using real world CVSS distribution

to the lowest and average effort threshold of those simulated attackers who successfully reach the goal conditions. Again, we can see our effort scores closely match the minimum required effort and follow the same trend as the average effort. The *Vulnerability average* curve shows the average skill score of all vulnerabilities is not as good a metric for measuring effort.

In Figure 18, we demonstrate the comparison of computation time between *CombineEffort* and brute force algorithm for computing *Effort*. We tested four sets of cases. In each case, we generate random attack graphs of 90 instances. From this experiment shows that our heuristic algorithm *CombineEffort* reduces the computation time exponentially with respect to the size of attack graph.

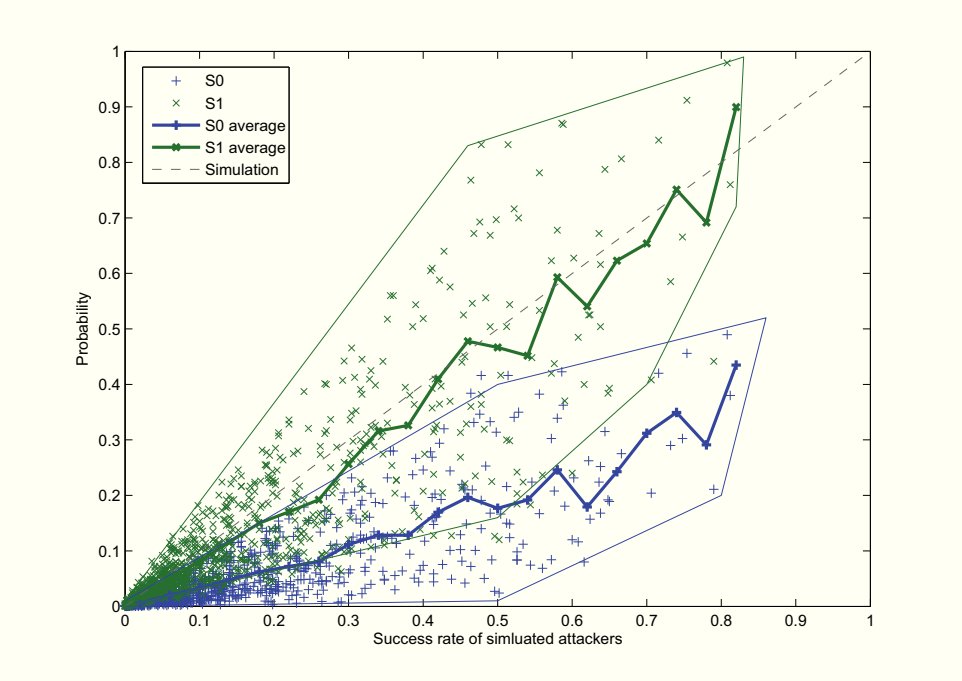


Figure 15: Distribution of Probability Scores

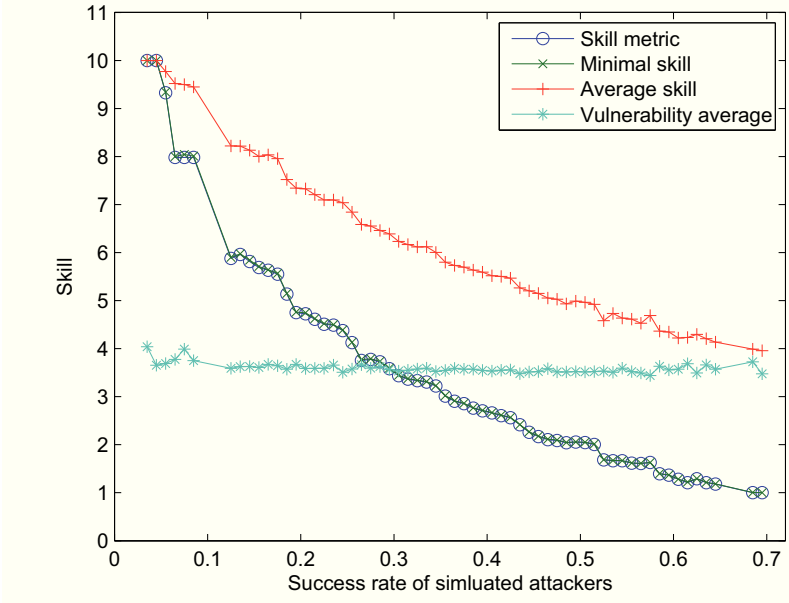


Figure 16: The Skill Aspect

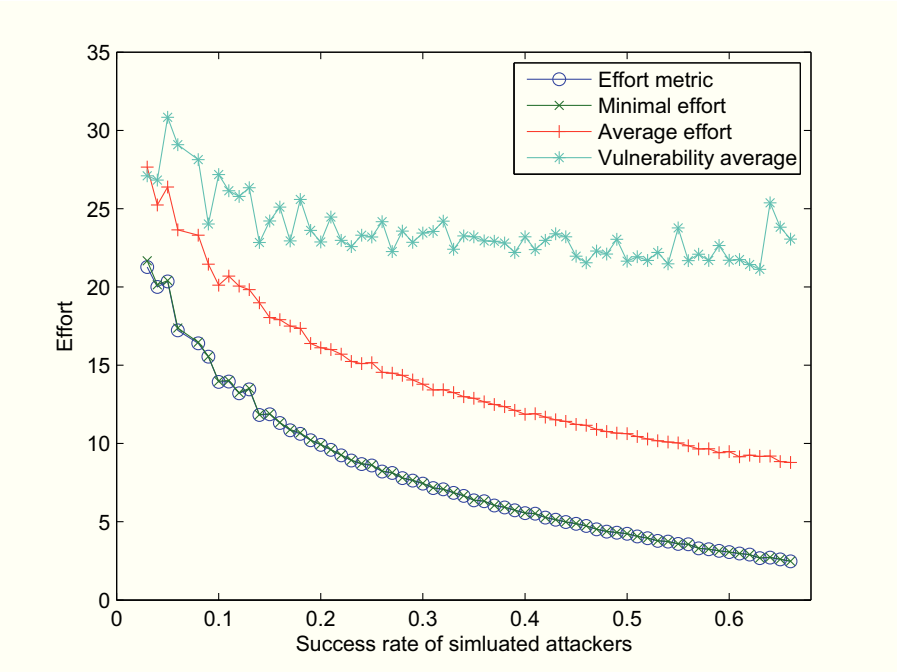


Figure 17: The Effort Aspect

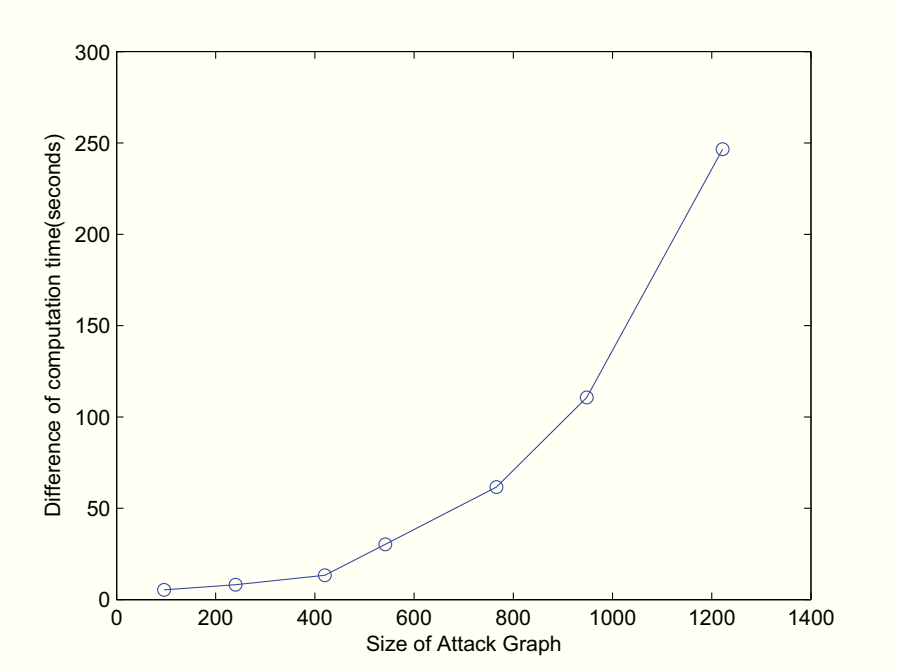


Figure 18: Performance Comparison for Combining Effort

Chapter 6

My Other Work

This chapter briefly introduces two other work on different but related topics, which have been completed during this thesis work.

6.1 Case Studies on Applying k-Zero Day Safety

In [74], a novel security metric, k-zero day safety, is proposed based on the number of unknown zero day vulnerabilities. That is, the metric simply counts how many unknown vulnerabilities would be required for compromising a network asset, regardless of what vulnerabilities those might be. The main motivation is that existing efforts on network security metrics typically assign numeric scores to vulnerabilities as their relative exploitability or likelihood. The assignment is usually based on known facts about each vulnerability (e.g., whether it requires an authenticated user account). However, such a methodology is no longer applicable when considering zero day vulnerabilities about which we have no prior knowledge or experience. In fact, a major criticism of existing efforts on security metrics is that unknown zero day vulnerabilities are unmeasurable.

The k-zero day safety model is proposed to address this issue. Instead of attempting to measure which zero day vulnerability is more likely, our metric counts how many distinct

zero day vulnerabilities are required to compromise a network asset (in our model, network asset is a general concept that may encompass one or more aspects of security, such as confidentiality, integrity, and availability). A larger number will indicate a relatively more secure network, since the likelihood of having more unknown vulnerabilities all available at the same time, applicable to the same network, and exploitable by the same attacker, will be lower. Based on an abstract model of networks and attacks, we formally define the metric and prove it to satisfy the three algebraic properties of a metric function. We then design algorithms for computing the metric. Finally, we show the metric can quantify many existing practices in network hardening and discuss practical issues in instantiating the model.

My main contribution to this work is a series of case studies that demonstrate how the k-zero day safety metric can be applied to evaluate different network hardening options. Those case studies will show that our metric may reveal interesting and sometimes surprising results, which are not always so obvious even in a small network. Therefore, for larger and more realistic networks, a systemetic approach to security evaluation using our metric and algorithms will become even more important.

6.1.1 Diversity

It is a commmon belief that a greater diversity in software and services may help to strengthen a network's resistence against potential security threats. However, we have shown that more diversity does not always mean more security, with respect to known vulnerabilities [78]. In this case study, we will reiterate this point with respect to zero day vulnerabilities by applying the proposed metric.

The left-hand side of Figure 19 shows a local network with four hosts represented by boxes, two firewalls represented by double lines, and an attacker's host in the Internet represented by a symbol of cloud. The active running services are marked within each box.

Host A, B and C provide an HTTP service ($http$) and host D provides a secure shell service (ssh). Firewalls enforce additional traffic restriction according to rules listed beside the firewalls. For example, in the direction from Internet to the local hosts, firewall FW1 only allows connections to host A, B or C, while in the reverse direction all connections are allowed. Firewall FW2 only allows connections from host A, B or C to its right-hand side, and allows all connections from its right-hand side to left-hand side.

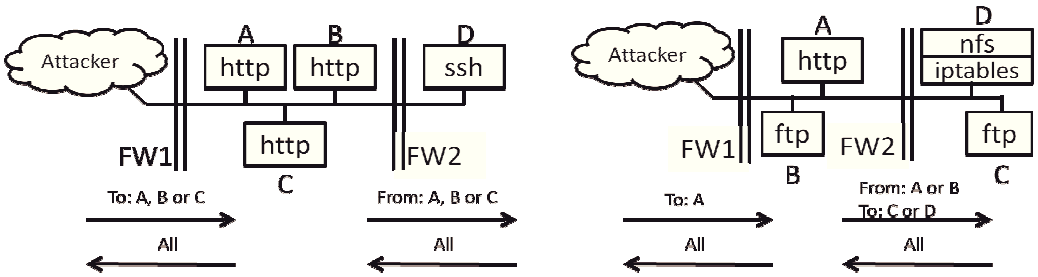


Figure 19: Case Study 1: Security by Diversity

We assume attacks originate from the attacker host on Internet and our main security concern is over the root privilege on host D. And we assume that all the services are free of known vulnerabilities unless explicitly stated. For the following examples, unless explicitly stated, we assume different services involve distinct zero day vulnerabilities; all the services except host-based access control service ($iptables$ and $tcpwrapper$) are not protected by isolation, that is, exploits of these services via zero day vulnerabilities directly yield root privilege on that host. And we assume that all the firewalls are already diversified due to security concern, that is, exploits of different firewalls involve distinct zero day vulnerabilities.(do not satisfy $\equiv_v$ relation)

Assume the three web servers are providing the $http$ service using exactly the same software. It may be tempting to think that this lack of diversity results in less security, since as long as the attacker can compromise any one of three web servers by exploiting an 0day vulnerability of $http$, he can exploiting the same zero day vulnerability to compromise the

others without further efforts. However, by applying the k-zero day metric, we can easily see that k remains to be 2 regardless whether the three web servers are running the same software or not, because each shortest attack sequence will only involve one of these three services (e.g., $[< v_{http}, attacker, A >, < v_{ssh}, A, D >]$). Therefore, increasing diversity in this case does not increase k.

Intuitively, the reason that diversity does not help security in the above case seems to be that the three web servers work in parallel so, informally speaking, increasing diversity among them does not help to move host D further away from attackers. Consequently, for the network shown on the right-hand side of Figure 19, it may seem obvious that increasing diversity in the ftp services of hosts B and C will certainly lead to better security. However, as we shall show, this is not the case, either.

First of all, in this network, we assume the $iptables$ services on host D only allow connection from host B and C. Given that host B and C are accessible from each other, once host B is compromised by zero day attack, the attacker can easily step further into the internal subnet by exploit the same zero day vulnerability on host B. It thus seems tempting to diversify ftp services on host B and C, so that it will be hard for the attacker compromise host C after gaining root privilege on host B.

However, by applying our approach on this network, we will find this hardening measurement does not help to increase the network's resistance against zero day attacks. Suppose we use $ftpX$ and $ftpY$ to indicate two different version of ftp service program on host B and C respectively. By applying our approach again, we will find that the shortest attack sequences of the original network are $[< v_{http}, attacker, A >, < v_{ftpX}, A, B >, < v_{nfs}, B, D >]$, $[< v_{http}, attacker, A >, < v_{ftpY}, A, C >, < v_{nfs}, C, D >]$ ($k = 2$). And the shortest attack sequences after diversifying ftp on host B and C are $[< v_{http}, attacker, A >, < v_{ftpX}, A, B >, < v_{nfs}, B, D >]$, $[< v_{http}, attacker, A >, < v_{ftpY}, A, C >, < v_{nfs}, C, D >]$ ($k = 2$). We can now find out that, although hosts B and C are not working in parallel,

the attacker still only needs one zero day vulnerability, of either v_{ftpX} or v_{ftpY}, in addition to v_{http} and v_{nfs}, to compromise host D.

This case study indicates that the way diversity affects security in networks is not always straightforward, even for small networks as described above. To manually conduct a similar analysis for larger and more complex networks will clearly be a daunting task to most security analysts. On the other hand, our proposed model and algorithms will automate such a task and yield meaningful results for networks of any reasonable size.

6.1.2 Known Vulnerability

In this case study, we show how the existence of known vulnerabilities may affect the k-zero day safety of a network by serving as shortcuts to help attackers in compromsing an asset with less zero day vulnerabilities.

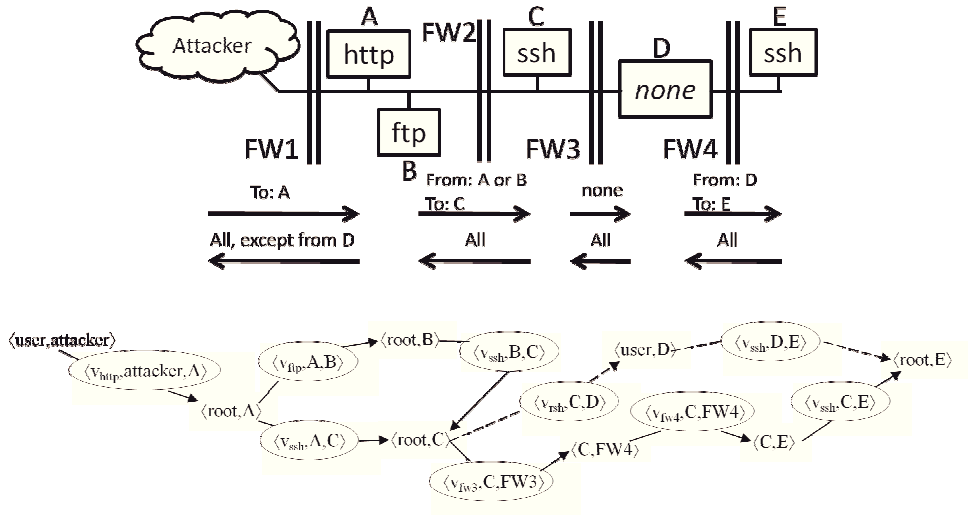


Figure 20: The Effect of Known Vulnerabilities

In Figure 20, host D is an administration client on which no services are running. And we assume no known vulnerabilities are found in this network. By applying our approach, we show two attack sequences in the lower part of Figure 20 (note that the path represented

by dashed line is the one after introducing a known vulnerability and we shall talk about it later). Note that here we omit some other attack sequences which are relatively longer. The lowest path in the figure is the shortest attack sequence, namely the one requires least zero day vulnerabilities.

Briefly speaking, the attacker first exploits a zero day vulnerability v_{http} to obtain root privilege on host A; then gains root privilege on host C through another zero day vulnerability v_{ssh}; continually bypasss firewall FW3 and FW4 through two distinct zero day vulnerabilities v_{FW3} and v_{FW4}; and lastly gains root privilege by exploiting v_{ssh} again. It can be observed that this shortest sequence requires four distinct zero day vulnerabilities, that is $k = 4$. Now we assume that a remote shell service(rsh) is opened on host D. And an additional rule allowing connection from host C to D is added to the access control rules of FW3. Once the attacker can access host D, she will obtain user privilege on host D which can be used as a steppingstone to gain connection to other hosts. We use v_{rsh} to represent this known vulnerability.

The dashed line in the lower part of Figure 20 shows the resultant attack sequence after introducing v_{rsh}: after obtaining root privilege on host C by exploiting $< v_{http}, attacker, A >$ and $< v_{ssh}, A, C >$, the attacker can access the rsh service on host D and gain user privilege; lastly, exploiting the ssh zero day vulnerability on host E, the attacker successfully completes the goal, while using only two distinct zero day vulnerabilities (v_{http} and v_{ssh}), instead of four.

Note that since the exploit of this introduced known vulnerability v_{rsh} is unreachable in generating the attack graph of known vulnerabilities [3], v_{rsh} is discarded in their approach. And from this example we can see that attack graph of zero day vulnerabilities can reveal much more potential risk of existing known vulnerabilities.

Since patching known vulnerabilities usually incur a cost (e.g., software upgrade, hardware replacement, or administrative effort), we need to prioritize the patching process based

on the effect of a vulnerability on security. In the following, we will show that patching known vulnerabilities does not necessarily help to increase k-zero day safety.

Instead of introducing rsh service on host D, let's consider a known vulnerability in ftp on host B. We assume that attacker can exploit this vulnerability as long as she has connection to the ftp service, and exploit of this vulnerability will provide root privilege to the attacker. However from the attack graph showed in the lower part of Figure 20, we can clearly see that this known vulnerability does not give the attacker any advantage. Note that in this case, v_{ftp} is considered as a known vulnerabilit. However, applying our approach we can find that k remains as 4. So if we patch v_{ftp}, it will be come the same case as the original network. But patching v_{ftp} does not help improve the security of the network.

6.1.3 Backup of Asset

In this case study, we will apply our metric to study the effectiveness of improving security of an asset by placing backups at different locations inside a network.

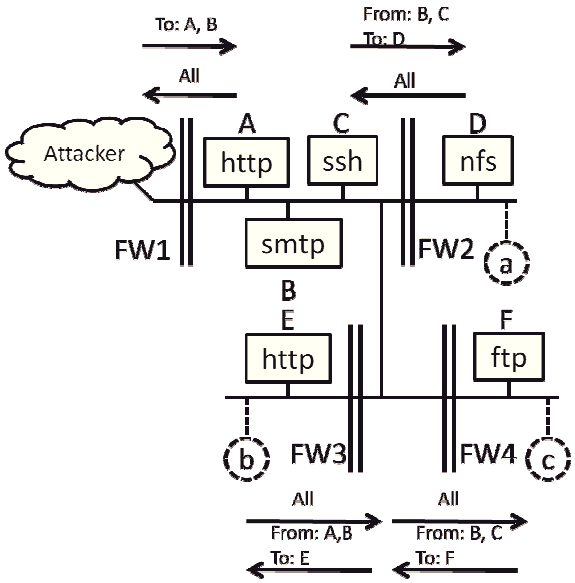


Figure 21: The Effectiveness of Asset Backups

In Figure 21, the attacker's goal is to gain root privilege on host D so that the availability of host D can be compromised (by shutting down host D). We assume that there exists a common known vulnerability in the $http$ service on host A and E, which will provide the attacker with root privilege after exploitation. Note that the three dashed line circles marked with 'a', 'b' and 'c' are three candidate positions to place a backup server synchronizing with host D. Before any security measures are enforced, we first apply our approach to find k of this network. Similar procedure as before, here we present the shortest attack sequences: $[< v_{http}, attacker, A >, < v_{ssh}, A, C >, < v_{nfs}, C, D >]$, $[< v_{http}, attacker, A >, < v_{smtp}, A, B >, < v_{nfs}, B, D >]$, $[< v_{smtp}, attacker, B >, < v_{nfs}, B, D >]$. Note that v_{http} is a known vulnerability. We can find that in this case $k = 2$.

Setting up backup server for host D helps to increase the difficulty for the attacker, because she has to gain root privilege on both hosts so that the availability of host D and its backup can be compromised. Now we consider effectiveness of the three different backup options. We assume that the new backup server is named host G. (1) If we place host G at the point a, we can find that k is not changed, because once host D is compromised, the attacker can directly exploit the same zero day vulnerability of nfs on host G. (2) If we place host G at the point b (note that an additional access rule from host G to host D should be allowed so that we can make the two hosts connected for synchronization), by applying our approach, we will find that the shortest attack sequence is $[< v_{http}, attacker, A >, < v_{http}, A, E >, < v_{nfs}, E, G >, < v_{nfs}, G, D >]$. In this case, the k decreases from 2 to 1. (3) If we place the host G at the point c, although we can follow the same attack sequence, which requires only two zero day vulnerabilities, as the original case, the shortest attack sequence to gain root privilege on both host D and host G is longer: $[< v_{smtp}, attacker, B >, < v_{ftp}, B, F >, < v_{nfs}, F, G >, < v_{nfs}, G, D >]$. We can see that in the scenario, k is increased from 2 to 3.

6.1.4 Intrusion Detection

In this case, we apply our metric to study the effectiveness of detection effort and isolation techniques when they are implemented differently inside a network.

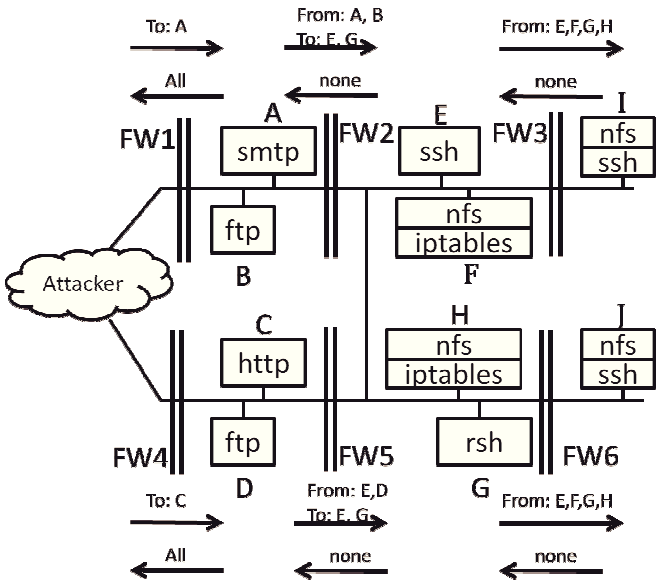


Figure 22: The Effectiveness of Intrusion Detection and Isolation

In Figure 22, a local network connects to Internet through two different physical connections and provides mail service ($smtp$) and web service ($http$) on host A and C respectively. On host F and H, to achieve more security, a host-based access control service ($iptables$) is enforced. More specifically, $iptalbes$ on host F only allows connection request from host E, G and H; on host H, only connection requests from host E, F and G are allowed. The attacker's goal is to gain root privilege on either host I or J ($< root, I > \vee < root, J >$).

Now consider two hardening scenarios by adding detection (IDS) to the network. First, since the attacker can penetrate into the network through either firewall FW1 or FW4, it seems necessary to place two IDSs behind FW1 and FW4 to assure that no intrusion attempts are missed. However, in reality, this is usually a costly, if not impractical, solution

due to high traffic throughput at the two entrances of the network. Secondly, since as long as either host I or J is compromised, we will fail to protect the network, it seems necessary to place two IDSs just behind the firewall FW3 and Fw6 so that any malicious traffic toward host I or J will be detected. This is a practical solution, however, not necessarily the most cost efficient one. By applying our approach, we will find that the two shortest attack sequence are $[< v_{smtp}, attacker, A >, < v_{ssh}, A, E >, < v_{ssh}, E, I >]$, $[< v_{smtp}, attacker, A >, < v_{ssh}, A, E >, < v_{ssh}, E, J >]$, $[< v_{http}, attacker, C >, < v_{ssh}, C, E >, < v_{ssh}, E, I >]$, $[< v_{http}, attacker, C >, < v_{ssh}, C, E >, < v_{ssh}, E, J >]$. All these four sequences exploit a common zero day vulnerability v_{ssh} on host E. So as long as an IDS (instead of two IDSs) placed before host E, all these four attack sequences can be blocked.

Similarly, although it will be helpful to protect the services on host I and J by isolation, our approach can help to identify more cost efficient hardening scenarios while still improving the network's resistance again zero day attacks. One example is to isolate the ssh service on host E, which will block the four shortest attack sequences.

6.1.5 Firewall

In this case, we evaluate the effectiveness of using firewall as temporary workaround for blocking exploitation of a known vulnerability.

In Figure 23, host-based access control services $iptables$ and $tcpwrapper$ are set up to further restrict connections among internal hosts. The $iptables$ service on host C allows inbound connection from A and outbound connection to D. The $tcpwrapper$ on host D allows inbound connection from host C and outbound connection to host E. The $tcpwrapper$ service on host F only allows inbound connection from E or G. The $iptables$ service on host C is detected vulnerable to a known vulnerability, exploiting which helps the attacker bypass the access control and gain connection to the ftp service on host C. The attacker's

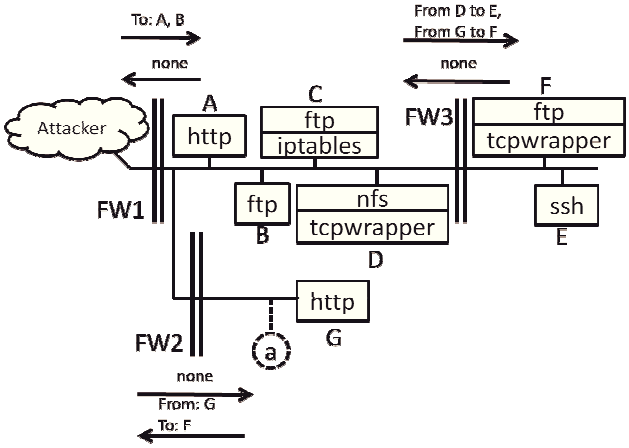


Figure 23: The Effectiveness of Firewalls

goal is to obtain root privilege on host F.

By applying our approach, we find the shortest attack sequences are $[< v_{ftp}, attacker, B >, < v_{iptables}, B, C >, < v_{ftp}, B, C >, < v_{nfs}, C, D >, < v_{ssh}, D, E >, < v_{ftp}, E, F >]$ and $[< v_{ftp}, attacker, B >, < v_{FW2}, B, FW2 >, < v_{http}, B, G >, < v_{ftp}, G, F >]$. Note that $v_{iptables}$ is a known vulnerability, so the resultant $k = 3$.

Suppose There is no patch available for the known vulnerability $v_{iptables}$. As a temporary workaround, the administrator moves host C behind firewall FW2 (at point a) and remove the $iptables$ service, but keep the same access control rules by adding extra rules to FW2: allow connections from A to C and C to D. However, our approach shows that this measurement makes the network less secure. After the moving host C behind FW2, the shortest attack sequence become $[< v_{http}, attacker, A >, < v_{ftp}, A, C >, < v_{http}, C, G >, < v_{ftp}, G, F >]$. In this scenario k decreases from 3 to 2. This is because, although the access control from/to host C is enforced by FW2, placing host C in the same subnet as host G will allow additional connection from C to G which does not exist in the original network. Without applying our approach to re-evaluate the security of 'improved' network against zero day vulnerability, some intuitive hardening measurement might not necessarily

enhance the security of the network, in contrary, enable more advantages for the attackers.

6.1.6 Other Cases

In following two cases, we show how insignificant hosts involve in zero day attacks and how a seemly insecure change of the network configuration helps to increase the resistance to zero day attacks.

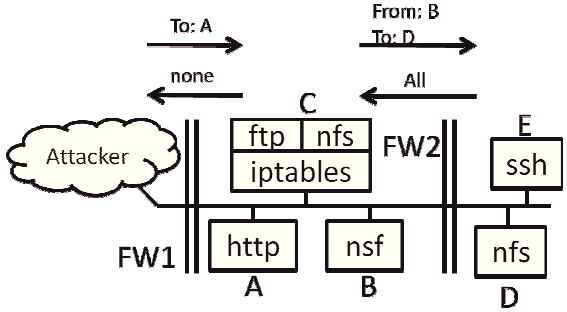


Figure 24: An Example of Seemingly Useless (Useful in Reality) Hardening

In Figure 24, the $iptables$ service running on host C only allows (1) connection requests of ftp service from host B and E, and (2) connection requests of nfs service from host E. The attacker's goal is to obtain root privilege on host C. And we assume that there is a known vulnerability of weak password in the ssh service of host E.

Since the attacker's goal is to gain root privilege on host C, it is intuitively to put higher priority on securing hosts A, B and C, because there is larger chance that the attacker can compromise these three hosts than host E and D which are placed in the inner part of the network and require more steps before the attacker can reach them.

However, our approach reveals that the shortest attack sequences leading to $< root, C >$ is $[< v_{http}, attacker, A >, < v_{nsf}, A, B >, < v_{nsf}, B, D >, < v_{weak_password}, D, E >, < v_{nfs}, E, C >]$ with $k = 2$, because the $v_{weak_password}$ is a known vulnerability and the exploits of v_{nfs} count as one unique zero day vulnerability. Note that, it seems more difficult

for the attacker to reach host D and E than host C, however, the hosts D and E involve as a stepping stone for the attacker gain access to the nsf service on host C.

Now consider hardening the network by moving the host D from the inner subnet to the point a(marked by the dashed line circle). This seems improper because placing host D in the middle subnet will enable more connection among host A, B, C and D. Applying our approach, we find that his seemly insecure change of the network increase k from 2 to 3(there we omit the details of attack graph), because the connection from host D to host E is now blocked which makes it harder to exploit the known vulnerability of ssh on host E. Such kind of hardening measurements may otherwise not be found by existing techniques.

6.2 Similation Results on Attack Graph Compression

In [14], 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.

My main contribution to this work is to evaluate the proposed compression model through simulation (since no publicly available datasets of real world attack graphs exist to the best of our knowlege). All simulations are conducted 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) [13] and the Georgia Tech Internetwork Topology Models topology generator(GT-ITM) [28]. We inject vulnerability information into generated topologies to obtain random configuration graphs. We employ the Jaccard similarity coefficient 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).

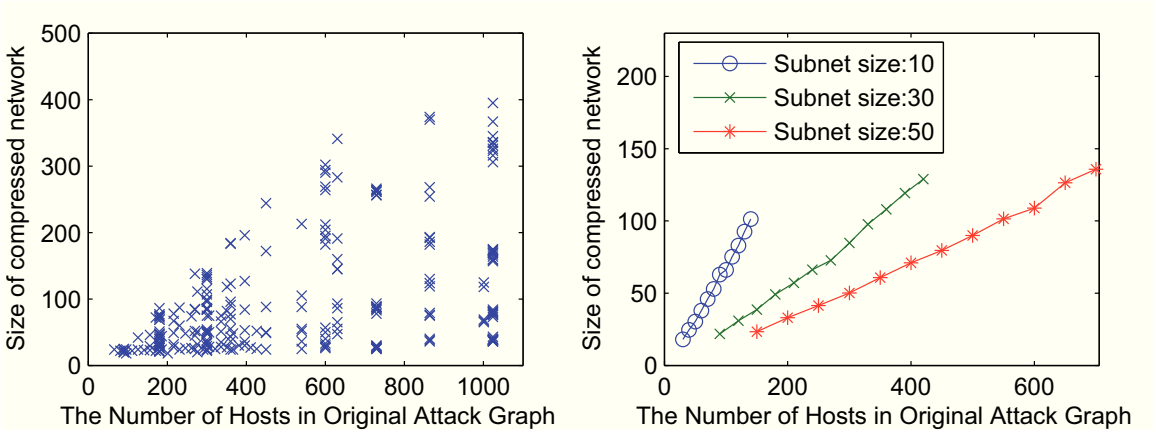
Figure 25a shows the compression results of 263 topologies with the numbers of hosts varying from 60 to 1020 and the number of subnets from 3 to 32. For each topology, the degree of similarity in vulnerabilities on hosts inside each subnet is within the range of $[0.930, 0.956]$. The weights on the number 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 25b 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 25d and 25e 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 25d and 25e 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.

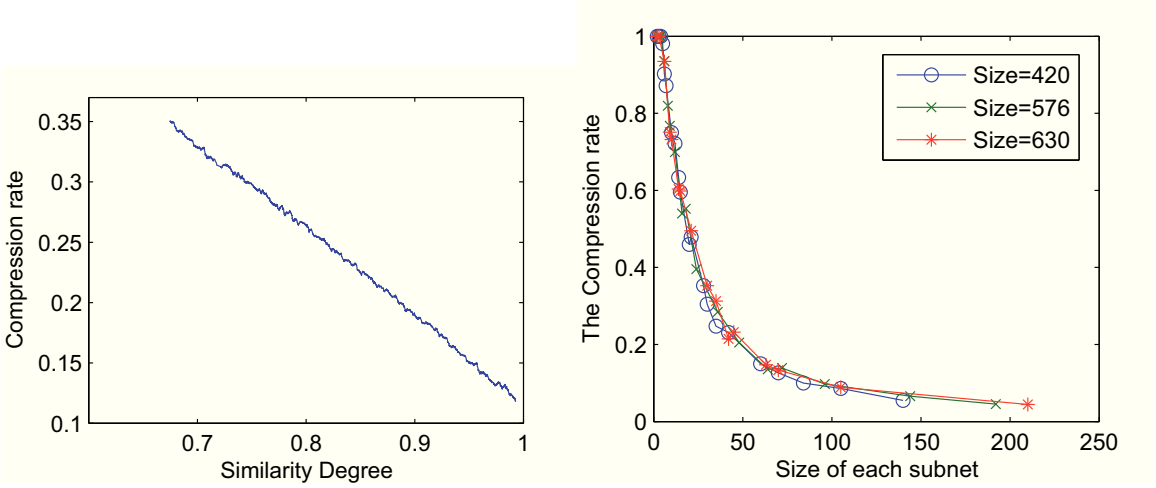
Figure 25f shows how the weight α assigned to the numbers of hosts (the weight assigned to the number of rules is $1 - \alpha$) will affect the result. Two sets of topologies are generated by BRITE and GT-ITM, respectively. The value of α in this experiment varies

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).

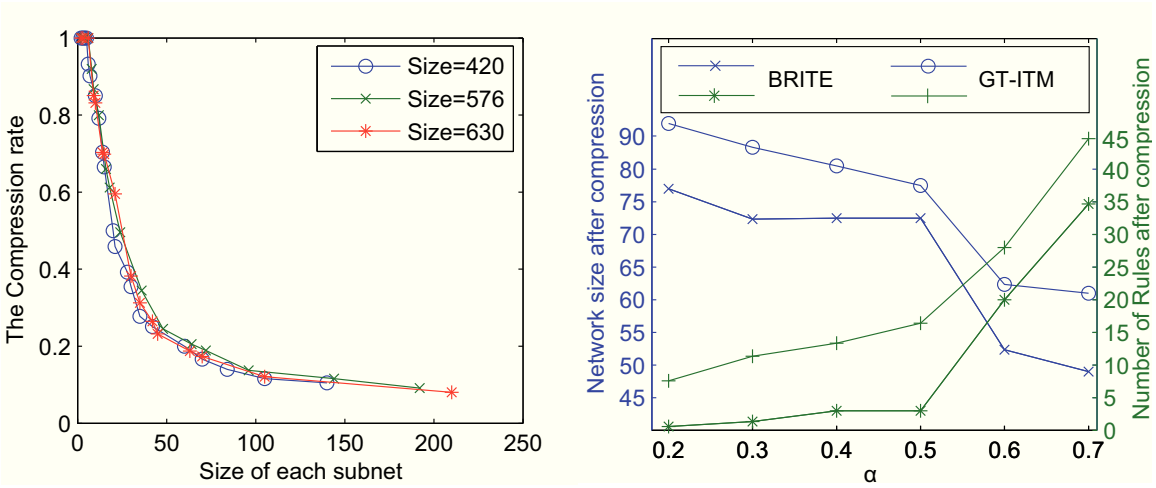
Figure 25c shows the compression rate in the degree of similarity of vulnerabilities on different hosts. We generate 2330 topologies with a fixed size of 420 nodes with 7 subnets, and we vary the average similarity degree. The metric weight assignment is $\alpha = 0.5$. We can see that the compression rate is almost linear in the degree of similarity of vulnerabilities.



(a) (b)



(c) (d)



(e) (f)

Figure 25: Simulation Results

Chapter 7

Conclusion

In this thesis, we have addressed two important limitations of existing approaches to combining CVSS scores, namely, the lack of support for dependency relationships between vulnerabilities, and the lack of consideration for aspects other than attack probability. We have formally presented our approaches to removing both limitations. Specifically, we handled potential dependency relationships at the base metric level so the resulted adjustment in base scores had well-defined semantics. We have also extended our approach to interpret and combine CVSS metrics and scores in the skill and effort aspects. The simulation results have confirmed the advantages of our approach. Future work will be directed to incorporating the temporal and environmental scores, the consideration of other aspects for interpreting the scores, and experiments with more realistic settings.

****

Bibliography

[1] Fumio Akiyama. An example of software system debugging. In *IFIP Congress (1)'71*, pages 353–359, 1971.

[2] A. Albrecht. Measuring application development productivity. 1979.

[3] Paul Ammann, Duminda Wijesekera, and Saket 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.

[4] Xiangdong An, Dawn Jutla, and Nick Cercone. Privacy intrusion detection using dynamic bayesian networks. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, ICEC '06, pages 208–215, 2006.

[5] A. Arnes, F. Valeur, G. Vigna, and R. Kemmerer. Using hidden markov models to evaluate the risks of intrusions. In Diego Zamboni and Christopher Kruegel, editors, *Recent Advances in Intrusion Detection (RAID'06)*, volume 4219 of *Lecture Notes in Computer Science*, pages 145–164. 2006.

[6] Yudistira Asnar and Paolo Giorgini. From trust to dependability through risk analysis. In *In Proceedings of the Second International Conference on Availability, Reliability and Security (AReS)*. IEEE Press, 2007.

[7] Applied Computer Security Associates. Workshop on. In *Information Security System Scoring and Ranking*, 2001.

[8] Michael B., Evan C., Farnam J., Andrew M., and Sushant S. Practical Darknet Measurement. In *Proceedings of the 40th Annual Conference on Information Sciences and Systems (CISS '06)*, pages 1496–1501, Princeton, New Jersey, USA, March 2006.

[9] D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st Workshop on Quality of Protection*, 2005.

[10] V.R. Basili and Jr. Reiter, R.W. A controlled experiment quantitatively comparing software development approaches. *Software Engineering, IEEE Transactions on*, SE-7(3):299 – 320, 1981.

[11] V.R. Basili and H.D. Rombach. The tame project: towards improvement-oriented software environments. *Software Engineering, IEEE Transactions on*, 14(6):758 – 773, 1988.

[12] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proceedings of the Third European Symposium on Research in Computer Security (ESORICS'94)*, pages 3–18, 1994.

[13] Boston university representative internet topology generator. Available at http://www.cs.bu.edu/brite/.

[14] Pengsu Cheng, Lingyu Wang, and Tao Long. Compressing attack graphs through reference encoding. In *Proceedings of 3rd IEEE International Symposium on Trust, Security and Privacy for Emerging Applications*, pages 1026–1031, 2010.

[15] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476 –493, 1994.

[16] S. Coull, F. Monrose, and M. Bailey. On Measuring the Similarity of Network Hosts: Pitfalls, New Metrics, and Empirical Analyses. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS'11)*, San Diego, California, USA, February 2011.

[17] M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.

[18] M. Dacier, Y. Deswarte, and M. Kaaniche. Quantitative assessment of operational security: Models and tools. Technical Report 96493, 1996.

[19] D. Farmer and E.H. Spafford. The COPS security checker system. In *USENIX Summer*, pages 165–170, 1990.

[20] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 357–370. ACM, 2000.

[21] M. Frigault and L. Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of the 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.

[22] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of ACM workshop on Quality of protection*, 2008.

[23] L. Gallon. Vulnerability discrimination using cvss framework. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–6, feb. 2011.

[24] D. Gao, M. Reiter, and D. Song. Behavioral distance for intrusion detection. In *Recent Advances in Intrusion Detection (RAID'05)*, Lecture Notes in Computer Science, pages 63–81. Springer Berlin / Heidelberg, 2005.

[25] D. Gao, M. Reiter, and D. Song. Behavioral distance measurement using hidden markov models. In Diego Zamboni and Christopher Kruegel, editors, *Recent Advances in Intrusion Detection (RAID'06)*, volume 4219 of *Lecture Notes in Computer Science*, pages 19–40. Springer Berlin / Heidelberg, 2006.

[26] D. Gao, M. Reiter, and D. Song. Beyond output voting: Detecting compromised replicas using hmm-based behavioral distance. *IEEE Trans. Dependable Secur. Comput.*, 6:96–110, April 2009.

[27] J. Giffin, D. Dagon, S. Jha, W. Lee, and B. Miller. Environment-sensitive intrusion detection. In Alfonso Valdes and Diego Zamboni, editors, *Recent Advances in Intrusion Detection (RAID'06)*, volume 3858 of *Lecture Notes in Computer Science*, pages 185–206. 2006.

[28] Georgia tech internetwork topology models topology generator. Available at http://www.cc.gatech.edu/projects/gtitm/.

[29] Tracy Hall and Norman Fenton. Implementing effective software metrics programs. *IEEE Softw.*, 14:55–65, March 1997.

[30] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[31] Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi. An evaluation of the mood set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, 24:491–496, 1998.

[32] Klaus Havelund and Grigore RoÅ§u. Efficient monitoring of safety properties. *International Journal on Software Tools for Technology Transfer (STTT)*, 6:158–173, 2004.

[33] J. Homer and X. Ou. Sat-solving approaches to context-aware enterprise network security management. *IEEE J.Sel. A. Commun.*, 27:315–322, April 2009.

[34] J. Homer, X. Ou, and D. Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. Technical report, Kansas State University, 2011. Available at.

[35] John Homer, Ashok Varikuti, Xinming Ou, and Miles McQueen. Improving attack graph visualization through data reduction and attack grouping. In *Visualization for Computer Security*, volume 5210 of *Lecture Notes in Computer Science*, pages 68–79. Springer Berlin / Heidelberg, 2008.

[36] K.S. Hoo. Metrics of network security. White Paper, 2004.

[37] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *Proceedings of ACSAC'09*, pages 117–126, Washington, DC, USA, 2009. IEEE Computer Society.

[38] 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.

[39] A. Jaquith. *Security Merics: Replacing Fear Uncertainity and Doubt*. Addison Wesley, 2007.

[40] 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.

[41] Capers Jones. *Applied software measurement: assuring productivity and quality*. McGraw-Hill, Inc., New York, NY, USA, 1991.

[42] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 130, Washington, DC, USA, 2001. IEEE Computer Society.

[43] D.J. Leversage and E.J. Byres. Estimating a system's mean time-to-compromise. *IEEE Security and Privacy*, 6(1):52–60, 2008.

[44] Y. Liu and H. Man. Network vulnerability assessment using bayesian networks. In *Proceedings of Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, pages 61–71, 2005.

[45] K. Manadhata, J.M. Wing, M.A. Flynn, and M.A. McQueen. Measuring the attack surfaces of two ftp daemons. In *ACM workshop on Quality of Protection*, 2006.

[46] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[47] J. McHugh. Quality of protection: Measuring the unmeasurable. In *Proceedings of the 2nd ACM workshop on Quality of protection (QoP'06)*, pages 1–2, 2006.

[48] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy Magazine*, 4(6):85–89, 2006.

[49] G. Modelo-Howard, S. Bagchi, and G. Lebanon. Determining placement of intrusion detectors for a distributed application through bayesian network modeling. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection (RAID'08)*, volume 5230 of *Lecture Notes in Computer Science*, pages 271–290. 2008.

[50] National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133, 1985.

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

[52] Steven Noel and Sushil Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *Proceedings of the 21st Annual Computer Security Applications Conference*, pages 160–169, 2005.

[53] National vulnerability database. available at: http://www.nvd.org, May 9, 2008.

[54] 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.

[55] X. Ou, W.F. Boyer, and M.A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, 2006.

[56] X. Ou, S. Govindavajhala, and A.W. Appel. Mulval: a logic-based network security analyzer. In *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, 2005.

[57] J. Wing P. Manadhata. Measuring a system's attack surface. Technical Report CMU-CS-04-102, 2004.

[58] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM*

workshop on Quality of protection, pages 31–38, New York, NY, USA, 2006. ACM Press.

[59] C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop (NSPW'98)*, 1998.

[60] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In Giovanni Vigna, Christopher Kruegel, and Erland Jonsson, editors, *Recent Advances in Intrusion Detection (RAID'03)*, volume 2820 of *Lecture Notes in Computer Science*, pages 73–93. 2003.

[61] M.K. Reiter and S.G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, 5 1999.

[62] R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 156–165, 2000.

[63] 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.

[64] R.E. Sawilla and X. Ou. Identifying critical attack assets in dependency attack graphs. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, pages 18–34, 2008.

[65] B. Schneier. Attack trees. In *Dr. Dobbs Journal*, pages 21–29, 1999.

[66] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[67] A. Somayaji, Y. Li, H. Inoue, J.M. Fernandez, and R. Ford. Evaluating security products with clinical trials. In *Proceedings of the 2nd conference on Cyber security experimentation and test*, CSET'09, pages 3–3. USENIX Association, 2009.

[68] M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo. Security metrics guide for information technology systems. NIST Special Publication 800-55, 2003.

[69] 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.

[70] CR. Symons. Software sizing and estimating. 1991.

[71] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 New Security Paradigms Workshop (NSPW'00)*, pages 31–38, 2000.

[72] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *Recent Advances in Intrusion Detection (RAID'04)*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. 2004.

[73] L. Wang, T. Islam, T. Long, A. Singhal, and S. 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.

[74] L. Wang, S. Jajodia, A. Singhal, and S. Noel. k-zero day safety: Measuring the security risk of networks against unknown attacks. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10)*, 2010.

[75] 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'05)*, pages 247–266, 2005.

[76] L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, 2006.

[77] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 11 2006.

[78] L. Wang, A. Singhal, and S. Jajodia. Measuring network security using attack graphs. In *Proceedings of the 3rd ACM workshop on Quality of protection (QoP'07)*, New York, NY, USA, 2007. ACM Press.

[79] L. Wang, A. Singhal, and S. 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.

[80] L. Wang, C. Yao, A. Singhal, and S. Jajodia. Interactive analysis of attack graphs using relational queries. In *Proceedings of 20th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2006)*, pages 119–132, 2006.

[81] D. Zerkle and K. Levitt. Netkuang - a multi-host configuration vulnerability checker. In *Proceedings of the 6th USENIX Unix Security Symposium (USENIX'96)*, 1996.