A UNIFIED FRAMEWORK FOR MEASURING A NETWORK'S MEAN TIME-TO-COMPROMISE

WILLIAM NZOUKOU TANKOU

A THESIS

IN

THE

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

JULY 2013

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **William Nzoukou Tankou**

Entitled: **A Unified Framework for Measuring a Network's Mean Time-to-Compromise**

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 commitee:

Dr. Zhigang Tian _____ Chair

Dr. Benjamin Fung _____ Examiner

Dr. Dongyu Qiu _____ External Examiner

Dr. Lingyu Wang _____ Supervisor

Approved Dr. Rachida Dssouli
Chair of Department

_____ 20 _____

Robin Drew, Dean

Faculty of Engineering and Computer Science

Abstract

A Unified Framework for Measuring a Network's Mean Time-to-Compromise

William Nzoukou Tankou

Measuring the mean time-to-compromise provides important insights for understanding a network's weaknesses and for guiding corresponding defense approaches. Most existing network security metrics only deal with the threats of known vulnerabilities and cannot handle zero day attacks with consistent semantics. In this thesis, we propose a unified framework for measuring a network's mean time-to-compromise by considering both known, and zero day attacks. Specifically, we first devise models of the mean time for discovering and exploiting individual vulnerabilities. Unlike existing approaches, we replace the generic state transition model with a more vulnerability-specific graphical model. We then employ Bayesian networks to derive the overall mean time-to-compromise by aggregating the results of individual vulnerabilities. Finally, we demonstrate the framework's practical application to network hardening through case studies.

Acknowledgments

I would like to thank my supervisor Dr. Lingyu Wang. It was a privilege for me to work with him. He has introduced me to rigorous world of academic and scientific research. Through the numerous meetings we had, he has always demonstrated high level of professionalism and mentorship. I am grateful to my family and friends for their support, love and encouragement. Finally, I also thank my current and former labmates.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer networks have long become the nerve system of enterprise information systems and critical infrastructures on which our societies are increasingly dependent. Potential consequences of a security attack have also become more and more serious as many high-profile attacks are reportedly targeting industrial control systems, implanted heart defibrillators, and military satellites. For example, the high profile worm *Stuxnet* exploits four different zero day attacks to specifically target the security of supervisory control and data acquisition (SCADA) systems used in power plants [24].

A major difficulty in securing computer networks in a mission critical system, such as SCADA, is the lack of means for directly estimating the effectiveness of a security configuration or solution, since *you cannot improve what you cannot measure*. Indirect measurements, such as the false positive and negative rates of a security device, are typically obtained through laboratory testing and may not reflect the actual effectiveness inside a real world network which could be very different from the testing environment. In practice, choosing and evaluating security configurations and solutions are still heavily based on human experts' experiences, which renders such tasks an art, instead of a science.

In such a context, a network security metric is desirable since it would enable a direct measurement of security provided by different solutions. Most existing approaches to network security metrics have focused on the threat of known vulnerabilities, and the metrics typically measure the relative difficulty for exploiting different vulnerabilities based

on existing knowledge about the vulnerabilities (Section 6 gives a more detailed review of related work). On the other hand, such approaches apparently do not work well for zero day attacks exploiting unknown vulnerabilities. To that end, a recent work estimates the threat of zero day attacks based on the least number of potential unknown vulnerabilities needed for compromising critical network assets [92].

A natural next step is to develop metrics that are capable of handling the threats of both known vulnerabilities and zero day attacks. At first glance, it may seem to be a viable approach to simply combine the two types of metrics through, for example, a weighted sum. Not surprisingly, such a straightforward approach may lead to misleading results, as demonstrated in our running example as follows.

1.1 The Running Example

The left side of Figure 1 shows a toy example of three hosts, on which the file transfer protocol (ftp) service on host 1 has a vulnerability (CVE-2001-0886) [15] and the remote shell service (rsh) another vulnerability (CVE-1999-1450); a buffer overflow vulnerability (CVE-2010-3814) is present on host 2. In addition, a secure shell service (ssh) free from any known vulnerability is running on both hosts. For simplicity, it is assumed that the firewall cannot be compromised.

Suppose the main security concern is to prevent unauthorized accesses to the root privilege on host 2. The right side of Figure 1 depicts what may potentially happen in this network, in which each predicate inside an oval indicates an exploit *vulnerability(source host, destination host)* (shaded ovals represent zero day exploits), each predicate in plaintext a security-related condition *condition(host)*, and each pair the connectivity *(source host, destination host)*. An exploit can be executed only if all of its pre-conditions are satisfied, and a condition may either be initially satisfied (e.g., $(0,1)$), or as the post-condition of an exploit (e.g., $user(1)$).

Applying a metric based on known vulnerabilities will find the network perfectly secure (since all zero day attacks, indicated by shaded ovals, will be ignored). The k-zero day

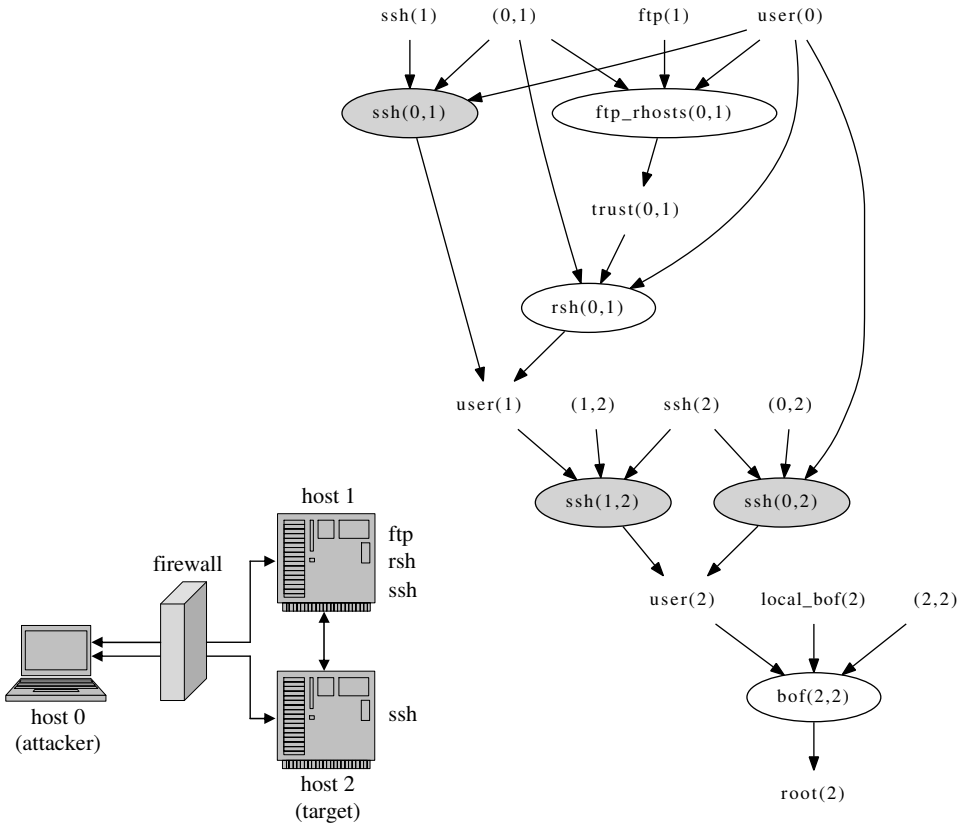


Figure 1: Running Example

safety metric [92] addresses this limitation by counting the minimum number of zero day vulnerabilities required to compromise key assets. Applying this metric in our example will yield a score of *one*, since one zero day vulnerability (in the ssh service) is necessary to reach the condition $root(2)$.

However, a key limitation of the *k*-zero day safety metric is that known vulnerabilities are essentially disregarded in measuring security (only regarded as shortcuts to bypass zero day exploits). A straightforward way to address this is to simply add a score of known vulnerabilities to the existing *k*-zero day safety metric result. However, this may produce inconsistent results, as shown below.

In Figure 1, consider only the leftmost sequence consisting of two zero day attacks on the ssh services on both hosts followed by a buffer overflow attack on host 2 (we will come back to this running example again later in the thesis).

1. Suppose we start with an initial state in which the ssh service on both hosts actually contained a known vulnerability whose metric score is s_{ssh}. Assume the buffer overflow attack has a score of s_{bof}. Clearly, the metric score of the leftmost sequence in this first case is equal to $s_{ssh} + s_{ssh} + s_{bof}$.

2. In this second case, we assume the ssh service on host 1 is patched to remove the known vulnerability, but ssh on host 2 is not patched. Assume a zero day exploit always has the fixed score of 1 and assume $s_{ssh} << 1$ (here a higher score indicates a less likely attack, and a zero day attack is typically considered much less likely than exploits of known vulnerabilities). Now, the metric score would become $s_{ssh} + 1 + s_{bof}$, which is larger than in the previous case.

3. Lastly, assume the ssh services on both hosts are patched. Now since there are two identical zero day exploits of the same service, the metric score becomes $1 + s_{bof}$, which is actually less than in case 2.

From the above three cases, we can observe inconsistent results yielded by this simple approach. That is, patching the ssh service has improved security from case 1 to 2, but it hurts security from case 2 to 3.

Furthermore, adding scores of different metrics not only may lead to inconsistent results, but may be simply meaningless when we consider the underlying semantics. Specifically, metrics based on known vulnerabilities typically indicate the *relative difficulty of exploiting* the vulnerabilities, whereas the *k*-zero day safety metric is more about the *likelihood of finding* unknown vulnerabilities. Therefore, the two metrics have incompatible semantics, and simply adding their results together indeed makes little sense.

In this thesis, we address this important issue by defining a novel metric based on a common property of both exploits of known vulnerabilities and zero day attacks, that is, the *Mean Time-to-Compromise* (MTTC). Generally speaking (we will present concrete models in later sections), for any vulnerability x, we define the MTTC to exploit x as:

$$t(x) = \begin{cases} f(x) & \text{if } x \text{ is a known vulnerability} \\ f'(x) & \text{if } x \text{ is known, and previously exploited} \\ \kappa & \text{if } x \text{ is an unknown vulnerability} \\ \kappa' & \text{if } x \text{ is unknown, and previously exploited} \end{cases}$$

To revisit the above example, we now have that

1. $t_1 = f(ssh) + f'(ssh) + f(bof)$ for case 1,
2. $t_2 = \kappa + min(f(ssh), \kappa') + f(bof)$ for case 2 (where $min()$ means the minimum value), and
3. $t_3 = \kappa + \kappa' + f(bof)$ for case 3.

Clearly, the comparison result between the three cases now depends on the specific definitions of the metric functions. For example, we may define them in a way such that $\kappa > \kappa' > f(ssh)$ (so case 3 is more secure) to reflect the case where finding or exploiting an unknown vulnerability takes more time than exploiting a known vulnerability. We may also define them such that $\kappa > f(ssh) > \kappa'$ (so case 2 and 3 are equally secure), meaning that although finding an unknown vulnerability is difficult, consequently exploiting it again on a different host takes very little time due to existing tools and experiences, which is also reasonable in some cases (the definition certainly depends on specific applications' requirements, and our goal is to provide administrators such a flexibility).

1.2 Contributions

The contributions of this thesis are as follows.

1. To the best of our knowledge, this is among the first efforts on network security metrics that can handle both known vulnerabilities and zero day attacks under the same metric model with coherent semantics.

2. The proposed metric based on time provides intuitive and easy to understand scores, which renders the metric more practical than abstract value-based metrics.

3. We take a top-down approach to defining our metric model, such that the high level framework and method do not necessarily depend on low level definitions or inputs, which may extend the scope of application.

The rest of this thesis is organized as follows. Background knowledge is reviewed in Section 2. Some intuitive but incorrect approaches are presented in Section 2.4. Section 3 presents our security metric approaches. A security metric based on the idea of the shortest path is presented in Section 3.4. A case study and discussions are provided in Section 4. Simulations and experiments are presented in Section 5. Section 6 discusses related work. Future research directions are presented in Section 7. Finally, concluding remarks are given in Section 8.

Chapter 2

Preliminaries

To be self-contained, we briefly review some background knowledge necessary for further discussions.

2.1 CVSS

The Common Vulnerability Scoring System (CVSS) is a widely adopted standard [61] for assigning numerical scores to vulnerabilities for their relative severity. CVSS scores are readily available through public vulnerability databases (e.g., the NVD [70]). Briefly speaking (more details can be found in [61]), each vulnerability is assigned a *Base Score* (BS) ranging from 0 to 10, which quantifies the intrinsic and fundamental characteristic of the vulnerability using the following equation.

$$BaseScore = round(((0.6 * Impact) + (0.4 * Expl) - 1.5) * f(Impact))$$

$$Impact = 10.41 * (1 - (1 - CI) * (1 - II) * (1 - AI))$$

$$f(Impact) = 0 \text{ if } Impact = 0,\ 1.176 \text{ otherwise}$$

$$Expl = 20 * AV * AC * AU$$

where AV, AC, AU, CI, II and AI are respectively:

- Access vector (AV): This indicates how the vulnerability is accessed before being exploited. The possible values for this metric are *Local or L* (numerical value 0.395),

Metrics		Values				
Base	AV	0.395 (L)	0.646 (AN)	1 (N)		
	AC	0.35 (H)	0.61 (M)	0.71 (L)		
	AU	0.45 (M)	0.56 (S)	0.704 (N)		
	CI / II / AI	0 (N)	0.275 (P)	0.66 (C)		
Temporal	E	0.85 (U)	0.9 (POC)	0.95 (F)	1.00 (H)	ND
	RL	0.87 (OF)	0.90 (TF)	0.95 (WA)	1.00 (U)	ND
	RC	0.90 (UC)	0.95 (UNC)	1.00 (C)	ND	
Environmental	CDP	0 (N)	0.1 (L)	0.3 (L-M)	0.4 (M-H)	0.5 (H)
	TD	0 (N)	0.25 (L)	0.75 (M)	1.00 (H)	
	CR / IR / AR	0.5 (L)	1.0 (M)	1.0 (H)		

Table 1: CVSS Metrics[62]

Adjacent Network or A (0.646), and *Network or N* (1.0).

- Access Complexity (AC): This metric quantitatively measures the attack complexity required to exploit the vulnerability after access to the system has been gained. Possible values are *High or H* (0.35), *Medium or M* (0.61), and *Low or L* (0.71).

- Authentication (AU): This metric measures the number of authentication required to a target in order to exploit a vulnerability. Range of values are *Multiple or M* (0.45), *Single or S* (0.56), and *None or N* (0.704).

- Confidentiality Impact (C): This metric measures the impact on confidentiality following a successful exploitation of the vulnerability. Possible values are *None or N* (0), *Partial or P* (0.275), and *Complete or C* (0.660).

- Integrity Impact (I): This metric measures the impact on integrity following a successful exploitation of the vulnerability. Possible values are *None or N* (0), *Partial or P* (0.275), and *Complete or C* (0.660).

- Availability Impact (A): This metric measures the impact on availability following a successful exploitation of the vulnerability. Possible values are *None or N* (0), *Partial or P* (0.275), and *Complete or C* (0.660).

Example 1. In Figure 1, the base score of the ftp vulnerability in host 1 is equal to:

$$\begin{aligned} Exploitability &= 20 * 0.395 * 0.71 * 0.704 = 3.9 \\ Impact &= 10.41 * (1 - (1 - 0.275)^3) = 6.4 \\ f(Impact) &= 1.176 \\ BaseScore &= round(((0.6 * 6.4) + (0.4 * 3.9) - 1.5) * 1.176) = 4.6 \end{aligned}$$

Optionally, the base score can be adjusted with a *Temporal Score* (TS) which quantifies characteristics that may change over time using the following equation:

$$TemporalScore = round\_to\_1\_decimal(BS * E * RL * RC)$$

where E, RL and RC are respectively:

- Exploitability (E): This metric indicates the current state of exploit techniques or code availability. Possibles values are *Unproven or U* (0.85), *Proof-Of-Concept or POC* (0.90), *Functional or F* (0.95), *High or H* (1.00), and *Not defined or ND* (1.00).
- Remediation Level (RL): This metric indicates the current situation related to the availability of patches, workarounds or fixes. Range of values are *Official Fix or OF* (0.87), *Temporary Fix or TF* (0.90), *Workaround or W* (0.95), *Unavailable or U* (1.00) and *Not defined or ND* (1.00).
- Report Confidence (RC): This metric indicates the current situation related to degree of confidence in the existence of the vulnerability. The possible values are *Unconfirmed or UC* (0.90), *Uncorroborated or UR* (0.95), *Confirmed or C* (1.00), and *Not defined or ND* (1.00).

Example 2. Assuming in Figure 1 that the ftp vulnerability in host 1 has the following temporal vector

$$E : H \ / \ RL : W \ / \ RC : C$$

The temporal score is then calculated as follows:

$$TemporalScore = round(4.6 * 1 * 0.95 * 1) = 4.4$$

Table 1 lists the CVSS metrics and their values.

2.2 Attack Graphs

As illustrated in the right side of Figure 1, attack graph is a model that graphically represents knowledge about vulnerabilities' inter-dependence and potential sequences of attacks. It is a directed graph with two types of nodes as vertices (exploits and security conditions) and their causal relationships as edges.

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

The require relation R_r is conjunctive meaning that all the *pre-conditions* indicated by $R_r(e)$ of an exploit e must be satisfied before the exploit can be executed. On the other hand, the imply relation R_i is disjunctive in the sense that, even if more than one exploit may imply the same *post-condition*, executing any one of those exploits is sufficient to satisfy that condition. A condition can be either initially satisfied (called *initial condition*) or satisfied as the post-condition of some exploits.

An Exploit node is typically represented inside oval with a label similar to $v_x(h_s, h_d)$. h_s and h_d denote two connected hosts. h_d is the source of the attack and h_d the target. v_x represents a vulnerability on the destination host. The subscript x represents the service, software, hardware or component affected by the vulnerability.

A Security condition is represented as a plain text with the form $c(h_s, h_d)$ to indicates that a security condition related to one more exploits between h_s and h_c is satisfied. The security condition is denoted as $c(h)$ when it involves a single host.

Attack graphs are generated using two types of inputs; *type graph* and *configuration graph*. Type graph model experts knowledge on vulnerabilities dependencies. Configuration graph represents the hosts, their connectivity and the vulnerabilities information.

For this research, we assume attack graphs are constructed using tools such as the Topological Vulnerability Analysis (TVA) system [45] which is built upon existing vulnerabilities and exploits databases (Xforce, Bugtraq, CVE, CERT, ...) and network discovery tools (Nessus).

2.3 Bayesian Networks

Bayesian Networks (BN) are probabilistic graphical models used to represent knowledge about an uncertain domain [9]. BN can be represented by a pair $\langle G, Q \rangle$ where G is a directed acyclic graph and Q the set of parameters of the network. G is defined by a set of nodes and a set of edges. The nodes represent random variables of the system, and the edges the direct dependence among the variables. An edge starting from a node x_i to a node x_j indicates that the value taken by x_j depends on the value of x_i. x_i is referred as the *parent* of x_j. Each variable of the graph is thus associated with a conditional distribution (often represented as conditional probability tables for discrete variables) which are included in Q. BN defines a unique joint distribution represented by:

$$P(x_1, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | parents(x_i))$$

2.4 Naive Approaches

In this section, two intuitive approaches of computing security by combining the times to exploit vulnerabilities are presented. Limitations of each approach are shown afterwards.

2.4.1 Forward Traversal of Attack Graphs

In this approach, to estimate the time to compromise, a forward traversal of the attack graph is performed, starting at the initial condition. At each node n of the attack graph, the following values are computed; the time to reach the node $t_r(n)$, the time to exploit the node $t_e(n)$ and the probability p_{nm} of reaching the node n coming from a node m. The examples below present the idea behind this approach.

Example 3. Figure 2 shows a simple attack graph which consist of one exploit. The mean time-to-compromise the network represented by this attack graph is equal to the time to reach the goal. However, before the goal can be reached, the attacker must reach and compromise exploit 1. Condition 0 must also be reached. The mean time-to-compromise is equal to the time to reach exploit 1 added to the time to exploit it.

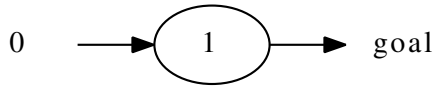


Figure 2: Sample Attack Graph with One Exploit

$$\begin{aligned} MTTC(goal) &= t_r(1) + t(1) \\ &= t(1) \end{aligned}$$

Example 4. Figure 3 shows a simple attack graph which consist of two exploits. The mean time-to-compromise is equal to the time to reach the goal. However, to reach the goal, an attacker must either exploit 1 or 2 (we assume he does not exploit both since it becomes redundant). Exploit 1 is chosen with probability p_{10} and exploit 2 with probability p_{20}.

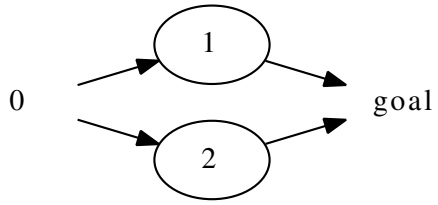


Figure 3: Sample Attack Graph with Two exploits

$$\begin{aligned} MTTC(goal) &= p_{10}(t_r(1) + t(1)) + p_{10}(t_r(2) + t(2)) \\ &= p_{10}t(1) + p_{20}t(2) \end{aligned}$$

with the condition that $p_{10} + p_{20} = 1$

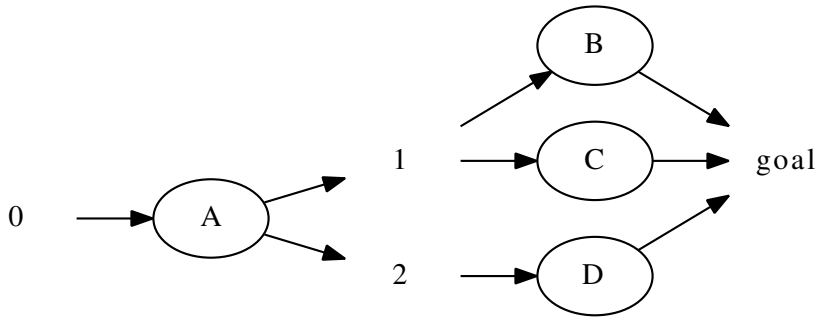


Figure 4: Sample Attack Graph with Four Exploits

Example 5. Figure 3 shows a slightly more complicated attack graph consisting of four exploits. Here, many attack sequences lead to the goal state. Initially, the attacker exploit A. This will costs him $t(A)$ amount of time. Then, he must either exploit D with a probability p_{D0} or choose between B and C with probability $1 - p_{D0}$ the vulnerability to exploit. If he chooses D, he will reach the goal with a total cost of $p_{D0}t(D)$. If he goes with the other path, he will have to choose B with a probability p_{1B} or C with a probability p_{C1} with $p_{B1} + p_{C1} = 1$. The cost of this path is $(1 - p_{D0})(p_{B1}t(B) + p_{C1}t(C))$

$$MTTC(goal) = t(A) + p_{D0}t(D) + (1 - p_{D0})(p_{B1}t(B) + p_{C1}t(C))$$

The main limitation of this method is that it does not give a sound approach to compute the probabilities used in the formulas. A tentative solution could be to use the difficulty associated with each vulnerability to compute theses probabilities. We may hypothesizes for example that given multiple choices, attackers will tend to go with vulnerabilities that are easier to exploit. Nevertheless, this cannot be applied to the proposed approach. This approach only considers nodes that are that are adjacent to the node where the attacker is present. No knowledge about vulnerabilities that are after the adjacent nodes are not considered. Although a particular node is chosen now with highest probability since it is the easiest to exploit, the exploits following it may be the hardest to exploit.

2.4.2 Using Attack Sequences

This method is similar to the one presented by [50]. However, it uses attack graphs instead of state transition models. The mean time-to-compromise is computed as the average of the

times needed to follow each attack path. First, the probability and the time to follow every attack sequence in the attack graph is computed. The time to follow an attack sequence is the sum of mean times to exploit every vulnerabilities in the sequence.

Second, a probability is assigned to each attack sequence. We also make the assumption that attack sequences requiring less time have more chances of being chosen by attackers. Using a formula representing the maximum of a normal probabilistic distribution[48], we assign to each path i of the attack graph the following probability:

$$p_i = \frac{1}{t_i} / \sum_{all\ paths} \frac{1}{t_k}$$

We can easily that:

$$\begin{aligned}
\sum_{all\ paths} p_k &= \sum_{all\ paths} \left(\frac{1}{t_k} / \sum_{all\ paths} \frac{1}{t_k} \right) \\
&= \sum_{all\ paths} \frac{1}{t_k} / \sum_{all\ paths} \frac{1}{t_k} \\
&= 1
\end{aligned}$$

Finally, the mean time-to-compromise is given by

$$\begin{aligned}
MTTC &= \sum p_i t_i \\
&= \sum t_i \frac{\frac{1}{t_i}}{\sum_{all\ paths} \frac{1}{t_k}} \\
&= \frac{\#\ paths}{\sum_{all\ paths} \frac{1}{t_k}}
\end{aligned}$$

Example 6. If we apply this to our running example, we have (using the values in table 4) that the probability of attack sequence $bof(2,2) \wedge ssh(1,2) \wedge ssh(0,1)$ is 0.27. The probability of the attack sequence $bof(2,2) \wedge ssh(1,2) \wedge rsh(0,1) \wedge ftp\_rhosts(0,1)$ is 0.34 and the probability of the attack sequence $bof(2,2) \wedge ssh(0,2)$ is 0.39. The time-to-compromise is equal to

$$MTTC = \frac{3}{1/155.89 + 1/126.55 + 1/109.89} = 128.11\ days$$

This approach has the following limitations. First, it considers all attack paths to be independent. This assumption may lead to inconsistent conclusion. The following example demonstrate this.

Example 7. Figure 5 represent two attack graphs. Both attack graphs have two attack sequences. We assume that the mean time-to-compromise (MTTC) to exploit B, D, E, F is equal to t. The MTTC to exploit A and C is respectively equal to $2t$ and $6t$. Using, these inputs, we find that the mean time-to-compromise both attack graphs is equal to

$$MTTC = \frac{2}{\frac{1}{4t} + \frac{1}{8t}} = \frac{8}{3}t$$

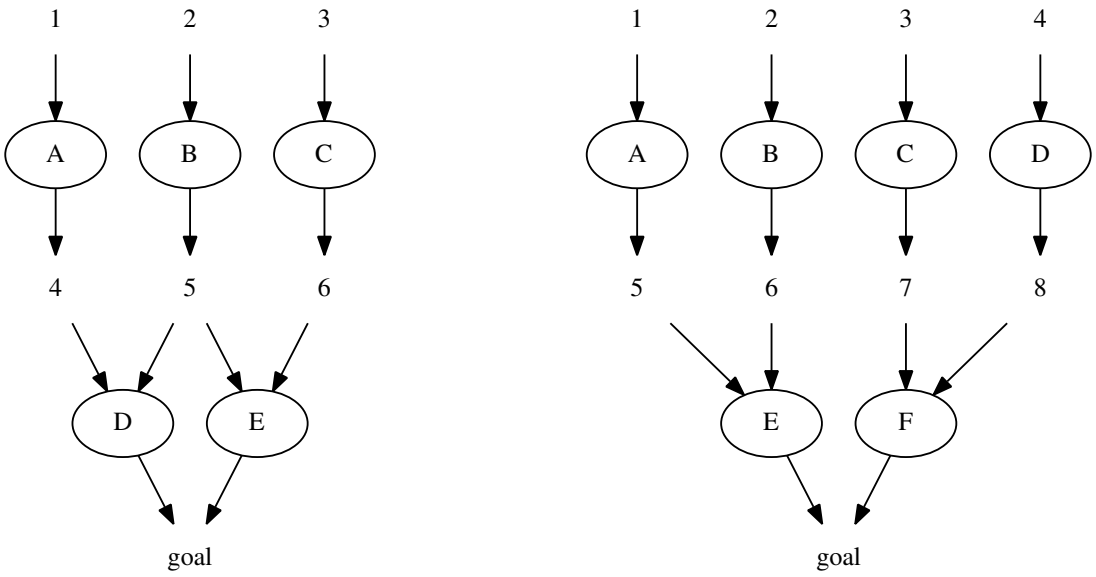


Figure 5: Inconsistent Results When Using Attacks Sequences

In the above example, the paths in the attack graph on the left side are not independent. By reaching condition 5, an attacker has more chances to arrive at the *goal* since 5 is part of both attack paths. The network on the left side is less secure than the one on the right side. However, as we found in example 7, this approach consider the two networks to have the same security level. This method has another limitation.

Let consider the shortest attack sequence leading to the *goal*. The shortest attack sequence is the attack sequence taking the least amount of time to exploit all vulnerabilities in it. In the network on the left side, the shortest attack sequence is $A \wedge B \rightarrow D$. The time to traverse it is equal to $4t$. The metric tells that on average, attackers spend less time to compromise the network that if there all following the shortest attack sequence. This does not quite make sense.

Chapter 3

The Models

We adopt a top-down approach to presenting our methods. We start by giving a general definition of the MTTC. We then discuss steps for calculating the exploit probabilities and the modeling of individual inputs. We emphasize that those components of our framework are relatively independent so each of them may be modified for specific needs without affecting the overall validity.

3.1 Mean Time-to-Compromise (MTTC)

In this section, we first give a high level description of the mean time-to-compromise (MTTC) concept. We will discuss concrete ways for instantiating this concept in following sections.

First, we need the concept of *minimal attack sequence*, as formalized in Definition 2. Intuitively, the concept assume an attacker to be efficient in the sense that s/he will not spend unnecessary effort (e.g., repetitively exploiting the same vulnerability or an irrelevant vulnerability). The main purpose here is to allow the metrics defined over minimal attack sequences to be unique for a given network and also to always yield a conservative result.

Definition 2. Given an attack graph $G(E \cup C, R_r \cup R_i)$,

- a sequence of exploits $Q \subseteq E$ is called an attack sequence, if for any exploit $e \in Q$,

all the pre-conditions of e are either initial conditions, or are post-conditions of some exploits that appear before e in Q.

- an attack sequence Q is minimal, if no sub-sequence of Q (not including Q) is also an attack sequence.

Given an attack graph, we define the MTTC for each condition (our discussions also apply to cases where a critical network asset is composed of multiple conditions). Intuitively, the MTTC of a condition is intended to reflect the average time required by an attacker in reaching that condition. A condition may be reached either as an initial condition, in which case no exploit will be executed (inside minimal attack sequences) so the MTTC is always zero, or as a post-condition of one or more exploits, in which case the MTTC is equal to the mean of the MTTCs of those exploits that are part of one or more minimal attack sequence (the MTTC of exploits will be defined later).

Definition 3. Given an attack graph $G(E \cup C, R_r \cup R_i)$ and any condition $c \in C$, the mean time-to-compromise (MTTC) of c, denoted as $MTTC(c)$, is defined as

$$MTTC(c) = \frac{\sum_{e \in E} MTTC(e) p_r(e \wedge c)}{p(c)} \tag{1}$$

where $MTTC(e)$ is the mean time-to-compromise of an exploit e, $p_r(e \wedge c)$ the probability that an attacker will execute exploit e inside some minimal attack sequences leading to condition c. This represents the ratio of attackers, among those who were able to successfully reach c, who choose to exploit e. The main difference between $p_r(e \wedge c)$ and $p(e \wedge c)$ is the fact the value of $p_r(e \wedge c)$ is dependent on the assumptions made (Section 3.2.3 presents two of the assumptions) while the value of $p(e \wedge c)$ is uniquely defined by the equation $p(e \wedge c) = p(e|\ c) * p(c) = p(e) * p(c)$. Finally, $p(c)$ the probability that an attacker will successfully reach c.

Clearly, to calculate the MTTC of a given goal condition, we will need to define both the probabilities for reaching the goal condition and for executing each exploit, and the MTTC of the exploit. In the remainder of this section, we will address those two issues.

Again, the general concept of MTTC defined in this section may still be applicable, even if the probabilities and MTTCs are defined differently from what we will describe.

3.2 Probabilities

In this section, we present a concrete approach to determining the probabilities that an attacker will successfully reach a given goal condition (called successful attacker, for short) and that a successful attacker will execute each exploit. First of all, we build intuitions by discussing our approach through a simple example in the following.

Example 8. Figure 6 depicts a simple attack sequence composed of two exploits and three conditions. We need to calculate the probability of an attacker to successfully reach the goal, and that of such an attacker to execute each exploit in doing so. We take following three steps in determining those probabilities.

1. First, we need the probability that an attacker can successfully execute each exploit independently (meaning given that all its pre-conditions are already satisfied). In Figure 6, the numbers below each oval indicate such probabilities (which will be defined later).

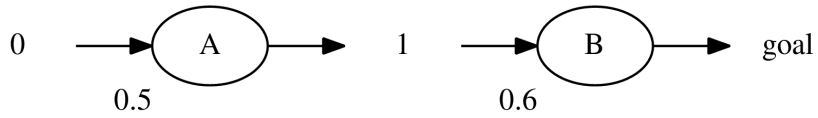
2. We next calculate the probability that an attacker can successfully execute each exploit when the pre-conditions are taken into consideration. The CPT tables on the right side of Figure 6 shows such calculations, with the results given to the right of the CPT tables.

3. From the above step, we know that each attacker will reach the goal with 0.24 likelihood (or equivalently, 24% attackers may reach the goal). For this simple case, each successful attacker will also has the same likelihood 0.24 to execute both exploits A and B (we will discuss more complicated cases where determining those probabilities is not so simple later on).

Therefore, by Definition 3, we have the MTTC as

$$MTTC(goal) = (0.24MTTC(A) + 0.24MTTC(B)) \,/\, 0.24$$

□



A	
T	F
0.4	0.6

B		
A	T	F
T	0.6	0.4
F	0	1

P (A) = 0.4
P (B) = 0.24

Figure 6: Example of Calculating MTTC

As illustrated in the above example, our approach has three steps. In the following, we present an approach to determining those probabilities (note again that there may be many other possible approaches to defining those probabilities). Roughly speaking, we first find the probability of successfully executing each exploit independently (without considering pre-conditions) based on its CVSS scores. Then, based on the attack graph, we calculate the probability of executing each exploit, while taking into consideration its pre-conditions, using a Bayesian Network built upon the attack graph. Finally, we do a backward traversal of the graph from the goal condition, in order to determine the probability for the successful attackers to execute each exploit.

3.2.1 Step 1: Probability of Exploiting Vulnerabilities Independently

We consider two cases, exploits of known vulnerabilities and zero day exploits, respectively.

Exploits of Known Vulnerabilities

We derive the probability of successfully executing each exploit when their pre-conditions are satisfied. Such a probability reflects the intrinsic difficulty in exploiting a vulnerability, and hence the CVSS score is a natural source for deriving this probability. Specifically, for each exploit e of known vulnerability, we assign the following probabilistic value based on the CVSS score of the exploited vulnerability, denoted as $CVSS(e)$.

$$p(e = T | \forall c \in R_r(e)\ c = T) = \frac{CVSS(e)}{10}$$

Zero Day Exploits

Since zero day exploits are about unknown vulnerabilities, it is not always possible to distinguish between different zero day exploits. Instead, we assign a fixed nominal probability based on the following reasoning. A zero day vulnerability is commonly interpreted as a vulnerability that is not publicly known or announced (even though they may have been discovered by attackers). Rewriting such a definition using the CVSS metrics, we find that a zero day vulnerability can be modeled as a special vulnerability with a remediation level *unavailable* and a report confidence *unconfirmed*. Also, we assume zero day vulnerabilities do not have a *high* nor *functional* exploitability metric. Therefore, a suggested relationship between vulnerabilities' status and the CVSS temporal metrics is given in Table 2 (note this is intended to be a general guideline and a different interpretation of the relationships may be possible).

zero day	disclosed	public	scripted
E = U	E = POC	E = F	E = H
RC = UC	RC = UR	RC = C	RC = C
RL = U	RL = W	RL = TF or RL = OF	RL = OF

Table 2: Vulnerabilities Status and Corresponding CVSS Temporal Metrics

Since the CVSS base metrics of an unknown vulnerability are hard to predict, we choose to assuming base metrics such they correspond to a longer MTTC than exploits of known vulnerabilities. Specifically, we set the metrics as follows: local access vector

($AV = L$), high access complexity ($AC = H$), multiple authentication ($AU = M$). We set the impact metrics as $(II = P, CI = AI = N)$. We choose these values since they maximize the overall base score and a large group of vulnerabilities in the NVD share these characteristics [28]. Therefore, the base score is calculated as 0.8. A supporting argument for this value is the fact that the lowest base score (the most difficult known vulnerability) in the NVD is 1.7 (CVE-2012-0075 and CVE-2012-0174) [70]. Adding temporal metrics gives that the probability of successfully executing a zero day vulnerability (without considering its pre-conditions) is

$$p(e = T | \forall c \in R_r(e)\ c = T) = 0.06 \text{ if } e \text{ is zero day} \tag{2}$$

3.2.2 Step 2: Probability of Exploiting Vulnerabilities Considering Pre-Conditions

The assigned probabilities are used to build a Bayesian Network based on the attack graph (we do not consider cycles in the attack graph which may be dealt with as in [38]). For each node in the graph, we construct a CPT table to capture its relationship with respect to its parents (e.g., an exploit can only be executed if all of its pre-conditions are satisfied, and a condition is satisfied if any executed exploit implies it).

$ssh(0,1)$				
ssh(1)	(0,1)	user(0)	T	F
T	T	T	0.08	0.92
T	T	F	0	1
T	F	T	0	1
T	F	F	0	1
F	T	T	0	1
F	F	T	0	1
F	T	F	0	1
F	F	F	0	1

$user(1)$			
$ssh(0,1)$	$rsh(0,1)$	T	F
T	T	1	0
T	F	1	0
F	T	1	0
F	F	0	1

Table 3: CPT Tables For an Exploit and a Condition in the Running Example

Example 9. Table 3 shows the CPT tables for condition *user(1)* exploit *ssh(0,1)* in our running example shown in Figure 1. The key relationships captured here are that the exploit

node $ssh(0,1)$ is reached through a conjunction over condition nodes $ssh(1),(0,1)$ and $user(0)$, and the condition node $user(1)$ is reached through a disjunction over exploits $ssh(0,1)$ and $rsh(0,1)$. ⊡

Next, we use the Bayesian Network to find the probability that conditions are satisfied and exploits executed while taking into consideration all their relationships. We denote this probability by $p(node = T)$ or $p(node)$. This step is similar to the method we introduced in [26] except that we now include also zero day exploits as well.

3.2.3 Step 3: Calculating $P_r(e)$

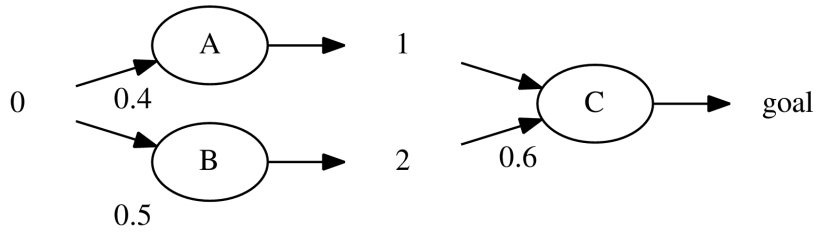
The last step is to decide the probability of a successful attacker executing each exploit inside minimal attack sequences leading to the goal, denoted by $p_r(e \wedge c)$ (or $p_r(e)$) for each exploit e. To find the value $p_r(e)$, we perform a backward traversal of the graph, starting from the goal condition. We estimate the ratio of successful attackers that have arrived at the current node from each of its parents. Different assumptions may be made for this purpose, as demonstrated in the following examples.

Example 10. Figure 7 shows an example with three exploits, in which exploits A and B respectively imply conditions 1 and 2 both required by exploit C. To reach the goal state, A and B must both be exploited first.

Like the previous example, we have assigned probabilities of executing exploits independently as the numbers shown below the ovals. On the right side, the BN for calculating the probabilities of executing those exploits while considering the pre-conditions is shown. From the results, we can see that $p_r(goal) = 0.12$. Among the successful attackers, all must exploit C and hence $p_r(C) = 0.12$. Since each of the successful attackers must have exploited both A and B, then $p_r(A) = p_r(B) = 0.12$. The time-to-compromise is equal to

$$t = (\ 0.12t(A)\ +\ 0.12t(B)/9\ +\ 0.12t(C)\ )\ /\ 0.12$$

⊡



A	
T	F
0.4	0.6

B	
T	F
0.5	0.5

C			
A	B	T	F
T	T	0.6	0.4
T	F	0	1
F	T	0	1
F	F	0	1

P (A) = 0.4
P (B) = 0.5
P (C) = 0.12
P(A, B) = 0.20

Figure 7: Example of Calculating MTTC

Example 11. Figure 8 shows another example with three exploits, in which exploits *A* and *B* both imply condition 1 required by exploit *C*. Therefore, to reach the goal state, either *A* or *B* (a minimal attack sequence will not include both) must be exploited first.

We have assigned probabilities of executing exploits independently as the numbers shown below the ovals. On the right hand side, we have shown the BN for calculating the probabilities of executing those exploits while considering the pre-conditions. From results we can see that $p_r(goal) = 0.42$. Among the successful attackers, all must exploit *C* and hence $p_r(C) = 0.42$. However, each of them could have either exploited *A* or *B* (not both). Based on the probabilities calculated in the second step, it can be calculated that, out of the 0.42 successful attackers, 0.12 can execute only *A*, 0.18 only *B*, and 0.12 can do both. Then, different assumptions may be made here about what attackers may chooses to do. For example (those cases are not intended to be exhaustive),

1. if we assume attackers always prefer to exploit the easiest vulnerability, then 0.12

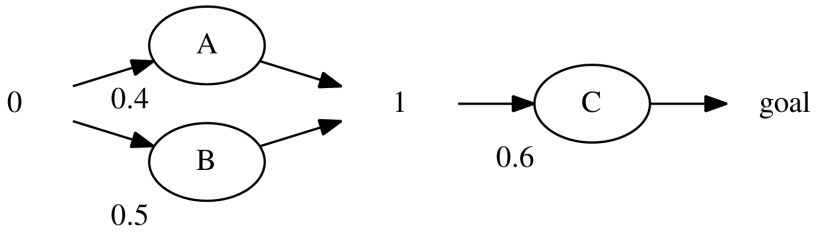
exploited A and 0.30 exploited B. The time-to-compromise is equal to (for simplicity, we will use $MTTC(.)$ and $t(.)$ interchangeably hereafter)

$$t(goal) \;=\; (0.12t(A) \;+\; 0.30t(B) \;+\; 0.42t(C)) \;/\; 0.42$$

2. If we assume attackers choose vulnerabilities based on their relative difficulty obtained from CVSS scores, then we have that $0.52/3$ exploited A and $2.22/9$ exploited B. The time-to-compromise is equal to

$$t = (\; 0.52t(A)/3 \;+\; 2.22t(B)/9 \;+\; 0.42t(C)\;) \;/\; 0.42$$

□



A	
T	F
0.4	0.6

B	
T	F
0.5	0.5

C			
A	B	T	F
T	T	0.6	0.4
T	F	0.6	0.4
F	T	0.6	0.4
F	F	0	1

P (A) = 0.4
P (B) = 0.5
P (C) = 0.42
P(A, B) = 0.2

Figure 8: Example of Calculating MTTC

Given a condition c, Algorithm 1 computes the value of $p_r(e_i)$ for each e_i. Roughly speaking, the algorithm first finds all possible combinations of exploits that lead to condition c. Second, the algorithm finds the probability of reaching each of those combinations.

Next, two assumptions can be made (as illustrated in the above example). That is, if attackers are assumed to choose the easiest exploits then we add the probability of each combination to that of the easiest exploit; if attackers are assumed to choose exploits based on relative difficulty, then we divide the probability accordingly. Due to the first line (enumerating all possible combinations of parents of a node), the algorithm has a worst case exponential complexity in the number of the maximum node in degree in the given attack graph. Nonetheless, this complexity is still acceptable since real world attack graphs usually have a constant in degree for most nodes (in size of the graph). Our simulation results will also confirm this.

Algorithm 1: Computing $p_r(e_i)$

Input : condition $c|\ c \in R_i(e_i)$
Input : set of exploits $\{e_j|\ c \in R_i(e_j)\}$
Input : set of exploits $\{r_i\}$ descendants of c
Input : an exploit e_i
Output : $p_r(e_i)$
Method:

1 $\mathscr{P}S = \{U : U \subseteq \{e_i | c \in R_i(e_i)\}\}$ **foreach** *set* $s \in \mathscr{P}S$ **do**
2 $\quad p(s) = p(s = T \mid \forall u \in \mathscr{P}S \text{ st } s \subset u,\ u = F)$
3 $p_r(e_i) = p(\{e_i\})$;
4 **foreach** *set* $s \in \mathscr{P}S$ *st* $(e_k \in s) \wedge (|s| > 1)$ **do**
5 $\quad$ **if** *attackers choose easiest vulnerability* **then**
6 $\quad\quad p_r(e_i) = p_r(e_i) + p_r(c)$ if $p(e_i) > p(e_j)\ \forall e_j \neq e_i \in \{e_j\}$
7 $\quad$ **else**
8 $\quad\quad p_r(e_i) = p_r(e_i) + \frac{p(e_i)}{\sum p(e_j)} p(s),\ e_j \in s$i
9 $p_r(e_i) = p_r(e_i) * \prod p(r_i)$;
10 **return** $p_r(e_i)$

3.3 MTTC of Exploits

We now discuss some possible approaches to estimating the MTTC of exploits. We note that such an estimation would critically depend on specific applications' settings and requirements, and what we will present here is only intended as some general guidelines

instead of the only choices. As we have stated before, our MTTC metric framework may work with many different ways for conducting such an estimation.

3.3.1 MTTC of Exploiting a Known Vulnerability

To estimate the MTTC for exploiting a known vulnerability, we distinguish between two cases, the first assumes an exploit code already exists for the vulnerability and the second assumes there is no corresponding exploit code. Given a vulnerability, we estimate the probability of each case, and the time to exploit the vulnerability in each case, then obtain the final averaged result.

Case 1: Exploit Code Existing

For a known vulnerability, the existence of exploit code can usually be directly determined based on various vulnerability databases (e.g., the NVD [70]) or exploit databases (e.g., the Metasploit DB [78]). The temporal scores (the exploitability E, the remediation level RL, and the report confidence RC) of a vulnerability, if available, also provides relevant information regarding the existence of an exploit code.

For the cases where such information is not available, we can still estimate the probability for an exploit code to exist based on general information about the availability of software, exploits, and the amount of software found in the given network. For this purpose, we can apply the search theory [54] as follows. If we denote by m the total number of softwares in existence (e.g., this can be estimated using the number of softwares included in the National Software Reference Library [69]), x the total number of software on the host being examined, and k the total number of available exploits (e.g., this may be estimated based on the number of exploits in the Metasploit DB [78]), then by applying search theory, the probability that an exploit code exists can be estimated as:

$$p_1 = 1 - e^{-xk/m}$$

As to the average time spent by an attacker when exploit code is available, we enhance McQueen's approach [58] by incorporating the CVSS scores. This will provide

more accurate estimation than McQueen's results, because the time required by an attacker would certainly depend upon the difficulty and severity of the vulnerability. Specifically, McQueen estimates the time taken to exploit a vulnerability, when an exploit is already available, to be equal to 1 day. We update his result with CVSS score as the following where $CVSS(e)$ denotes the CVSS score of the vulnerability being exploited (the adjusted estimation will range from 1 day to about 6 days since the currently smallest CVSS score is around 1.7 [70], which can certainly be further fine-tuned based on specific applications' needs).

$$t_1 = 1\ day * \frac{10}{CVSS(e)}$$

Case 2: Exploit Code Not Existing

The probability of this case can be similarly determined based on existing information about the vulnerability, or be estimated as the complement of the previous case as $1 - p_1$, that is,

$$p_2 = e^{-xk/m}$$

To estimate the average time an attacker will spend in this case, we again enhance McQueen's results with CVSS scores. McQueen hypothesizes that in this case the mean time will follow a gamma distribution with a mean of 5.8 days. Therefore, the time taken to exploit a vulnerability when assuming the exploit code is not available can be estimated as

$$t_2 = 5.8\ days * \frac{10}{CVSS(e)}$$

Combining Both Cases

Based on the two cases' results, the mean time to exploit a known vulnerability can be estimated as

$$t = p_1 t_1 + p_2 t_2$$
$$t = \frac{10}{CVSS(e)} \left(1 + 4.8 * e^{-xk/m}\right) \; days \tag{3}$$

Example 12. Using our method, the time to exploit the *ftp* vulnerability in Figure 1 can be estimated as:

$$t(ftp) = \frac{10}{4.6}(1 + 4.8 * e^{-450*3*1/7083}) = 10.8 \; days$$

⊡

3.3.2 MTTC of a Zero Day Exploit

To estimate the time to exploit an unknown vulnerability, we may take the following approach. First, we assume that given enough time, it is always possible to find an unknown vulnerability [86]. Second, we also assume that the availability of knowledge about the vulnerability greatly influences the mean time to exploit. More time will be spent if the attacker does not know about the vulnerability and has to find it. Thus, we divide the attacker into two processes based on knowledge on the vulnerability.

Case 3: Known Existence of Zero Day Vulnerability

This case assumes that the attacker knows a zero day vulnerability exists on the software or hardware he is attacking. This assumption is not unreasonable given the growing market for zero day vulnerabilities [32, 74]. An argument may be made against buying zero day vulnerabilities in that targeted systems do not always have a high value. However, the possibility of an attacker buying a vulnerability still exists since many attackers' main motivation may not always be financial in nature or the attacker may simply be willing to use every possible means [65]. Furthermore, a zero day vulnerability purchased by the attacker may be repetitively used on different targets.

We assign a fixed nominal probability to represent the probability an attacker knows a zero day vulnerability. Again we apply the search theory [54] for an estimation. If we

denote by m the total number of softwares in existence, x the total number of softwares on the host being examined, and k' the total number of zero day exploits (which may be estimated from statistical information [59]), then by applying the search theory, the probability that an attacker may know about a zero day vulnerability in the network is

$$p_3 = 1 - e^{-xk'/m}$$

From [63], it is estimated that it takes about a month to sell a zero day vulnerability. Vulnerabilities are typically sold with proof-of-concept exploit codes instead of automated tools. Meaning that the attackers purchasing the exploit will spend certain amount of time to fine-tune the exploit code before s/he can apply it in reality. This is similar to the first case of the preceding section. We have that the mean time to exploit a zero day vulnerability, assuming the attacker is aware of its existence, can be estimated as (the $CVSS(e)$ value can be estimated similarly as in Section 3.2.1).

$$t_3 = 32 + \frac{10}{CVSS(e)}\ days$$

Case 4: Unknown Existence of Zero Day Vulnerability

In the majority of cases, if there is no known vulnerabilities, an attacker has to search for zero day vulnerabilities. This process is the complement of the previous, and therefore the probability that an attacker has to find a zero day vulnerability is

$$p_4 = e^{-xk'/m}$$

To estimate the time spent in this case, we reason as follows. If an attacker is unaware of a zero day vulnerability, then s/he must either find one, or wait for one to become available. The time spent for this purpose can be estimated based on the lifespan of unknown vulnerabilities in general. Based on the analysis of 491 zero day vulnerabilities [59], it is estimated that the average lifetime of a zero day vulnerability is about 130 days. We estimate that it takes about half the lifetime (65 days) before the vulnerabilities can be discovered by a number of attackers. A supporting argument for this is also found in [86] which states that

in some project types an eight-to-five-week is enough to find a zero day vulnerability with 95 percent probability.

After the vulnerability is found, an exploit code needs to be written for it. This is similar to the second case in the previous section. We have that the time to exploit a zero day vulnerability, assuming the attacker is unaware of its existence, can be estimated as (again the $CVSS(e)$ value may be estimated as before)

$$t_4 = 65 + 5.8 * \frac{10}{CVSS(e)} \; days$$

Combining Both Cases

The time to exploit a zero day vulnerability can thus be estimated as

$$\begin{aligned} t &= p_3 t_3 + p_4 t_4 \\ t &= (32 + \frac{10}{CVSS(e)}) + (33 + 4.8\frac{10}{CVSS(e)}) e^{-xk'/m} \; days \end{aligned} \tag{4}$$

Example 13. Using our method, the time to exploit the *ssh* vulnerability on host 2 in Figure 1 can be estimated as:

$$t(ssh) = (32 + \frac{10}{0.6}) + (33 + 4.8\frac{10}{0.6}) e^{-2*491/7083} = 147.03 \; days$$

□

3.3.3 MTTC of a Previously Exploited Vulnerability

If a vulnerability has already been exploited, then attackers already have a working exploit code. The next time when they exploit the same vulnerability, we have $p_1 = p_3 = 1$ and $p_2 = p_4 = 0$. The time to exploit a previously exploited known vulnerability can thus be estimated as

$$t = \frac{10}{CVSS(e)} \; days$$

The time to exploit a previously exploited zero day vulnerability is

$$t = 32 + \frac{10}{CVSS(e)}\ days$$

3.4 Minimum Time-To-Compromise (MinTTC)

In this section, we propose another security tool. We called it the minimum time to compromise (*MinTTC*). The MinTTC of a network is defined as the least amount of time it will take for an attacker to compromise the network. More specifically, given a network an its attack graph, the MinTTC is the time an attacker will spend to traverse the shortest attack sequence in the attack graph. It is the answer to the questions *How fast can this condition be reached?* or *How long will the most efficient attacker targeting this network will spend before reaching this condition?*

However, contrary to the mean time to compromise, the MinTTC is not a good at comparing system configurations, nor estimating the effect of hardening measure[43].

If we assume that attackers favor easier exploits, then finding the MinTTC is equivalent to finding the dominant (or critical) attack sequence. Knowing the dominant attack sequence can be useful to automate network hardening. In the following, we two approaches to compute the MinTTC. First, we present a Brute-force algorithm which produces the exact result but is not scalable. Then, we present an heuristic approach which approximate the exact result but is more scalable.

3.4.1 Brute-force Algorithm

The Brute-force algorithm is formally presented in algorithm 2. First, a logic proposition L representing all possible attack sequences leading to the goal state is computed (Lines 1 – 10). Second, a disjunctive normal form of the proposition is found (Line 12). For each term L_i in the DNF, a value T_i representing the time to traverse L_i is calculated (Lines 13–14). The shortest attack sequence is the L_i having the minimal T_i. The MinTTC is

equal to the smallest T_i (Line 15). An advantage to this method is that the MinTTC of any visited node can be computed.

Algorithm 2: Brute force Algorithm

Input : An attack graph AG
Output : A non-negative number MinTTC
Method:

1 $Q = TopologicalSort(AG)$;
2 **while** Q *not empty* **do**
3 $\quad q = Q.dequeue()$;
4 $\quad$ **if** $q \in C_i$ **then**
5 $\quad\quad L(q) = q$;
6 $\quad$ **else if** $q \in C \backslash C_i$ **then**
7 $\quad\quad L(q) = q \wedge (\vee_{e_i} L(e_i)) \mid q \in R_i(e_i)$;
8 $\quad$ **else if** $q \in E$ **then**
9 $\quad\quad L(q) = q \wedge (\wedge_{c_i} L(c_i)) \mid c_i \in R_r(q)$;
10 $\quad L = L(q)$;
11 L is a proposition logic representing *goal*;
12 Let $L_1 \vee L_2 \vee \cdots \vee L_n$ the DNF of L;
13 **foreach** L_i **do**
14 $\quad T_i = \sum_{e_j \in L_i} t(e_j)$;
15 **return** $minimum(T_i)$

We apply this brute-force approach to compute the MinTTC of the network in our running example (Figure 1).

Example 14. The DNF representation of the goal stae is given in the following equation.

$$\begin{aligned} L = {} & (root(2) \wedge bof(2,2) \wedge (2,2) \wedge user(2) \wedge ssh(1,2) \wedge user(1) \wedge rsh(0,1) \\ & \wedge (0,1) \wedge user(0) \wedge trust(0,1) \wedge ftp(1) \wedge ftp\_rhost(0,1)) \\ & \vee (root(2) \wedge bof(2,2) \wedge (2,2) \wedge user(2) \wedge ssh(1,2) \wedge user(1) \wedge ssh(0,1) \\ & \wedge ssh(1) \wedge user(0) \wedge (0,1)) \vee (root(2) \wedge bof(2,2) \\ & \wedge (2,2) \wedge user(2) \wedge ssh(0,2) \wedge ssh(2) \wedge (0,2) \wedge user(0))) \end{aligned}$$

The attack sequences (conditions have been removed for clarity) found using the DNF representation of the goal state along with the time required for an attacker to traverse them

Attack sequence	Time (in days)
$bof(2,2) \wedge ssh(1,2) \wedge ssh(0,1)$	196.19
$bof(2,2) \wedge ssh(1,2) \wedge rsh(0,1) \wedge ftp\_rhosts(0,1)$	170.68
$bof(2,2) \wedge ssh(0,2)$	154.02

Table 4: Attack sequences and Time to Traverse

are listed in table 4. The shortest or dominant attack sequence is $bof(2,2) \wedge ssh(0,2)$. The MinTTC of the network, which is the time to traverse to traverse the shortest attack sequence it is equal to 154.02 *days*.

This approach produces the correct value of the MinTTC. However, it is not scalable. The number of attack sequences in the graph has an exponential worst case complexity.

3.4.2 Heuristic Scalable Approach

In this section, we propose a heuristic algorithm to address the limitations of the previous algorithm. Algorithm 3 presents our heuristic approach. Our approach is based on the ideas proposed by [4].

On line 1, a topological sort of the attack graph is performed and the result pushed into a queue. Initial conditions are in the front of the queue. The goal condition is at the rear. While the queue is not empty, an element q is removed from it. If q is an initial condition, then the shortest attack sequence containing q is $L(q) = q$ (line 5). If q is an exploit, then all of its pre-conditions must be satisfied. The attack sequences containing q are equal to q added to the k shortest attack sequences needed to reach every condition in $R - r(q)$ (line 7). Finally, if q is an intermediate condition, then q can be reached by exploiting any exploit for which q is post-condition. The attack sequences containing q are equal to q added to the k shortest attack sequences to reach exploits in $R_i(q)$ (line 9). The algorithm is similar to the Brute-force algorithm. However, instead of keeping all paths leading to a node, only the k paths with the minimal time to be traversed are kept (Lines 10–11).

In the following example, we show how our model is affected by the parameter k.

Example 15. Figure 9 presents an attack graph. The MTTC of exploits is represented as

Algorithm 3: Heuristic Scalable Algorithm

Input : An attack graph AG
Input : A parameter k
Output : $MinTTC$
Method:

$Q = TopologicalSort(AG)$;
while Q *not empty* **do**
    $q = Q.dequeue()$;
    if $q \in C_i$ **then**
        $L(q) = q$;
    else if $q \in C \backslash C_i$ **then**
        $L(q) = q \wedge (\vee_{e_i} L(e_i)) \mid q \in R_i(e_i)$;
    else if $q \in E$ **then**
        $L(q) = q \wedge L(q_i)\ \forall q_i \in R_r(q)$;
    if $lenght(L(q)) > k$ **then**
        $L(q) = top(L(q), k)$
$L = L_1 \vee \cdots \vee L_k$;
foreach L_i **do**
    $T_i = \sum_{e_j \in L_i} t(e_j)$;
return $minimum(T_i)$

label outside of exploits nodes.

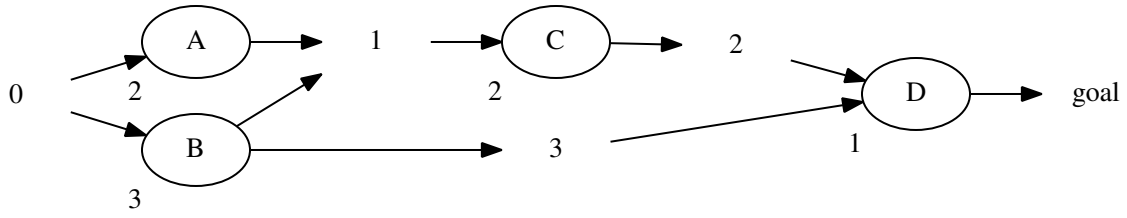


Figure 9: Effect of k on the MinTTC

- $k = 1$: To reach condition 1, the algorithm will prefers taking the attack sequence $0 \rightarrow A$, since it requires the lowest amount of time. To reach condition 2, the algorithm will select the attack sequence $0 \rightarrow A \rightarrow 1 \rightarrow C$. Finally, to reach the goal condition, the algorithm will select the attack sequence $0 \rightarrow A \rightarrow 1 \rightarrow C \rightarrow 2 \rightarrow B \rightarrow$

$3 \rightarrow D$. The time to follow this sequence is equal to 8. This is not however the exact value of the MinTTC. The MinTTC is obtained by following the path $0 \rightarrow B \rightarrow (1 \wedge 3) \rightarrow C \rightarrow 2 \rightarrow D$ and is equal to 6.

- $k = 2$: Now, to reach condition 1, the algorithm keeps track of the attack sequences $0 \rightarrow A$ and $0 \rightarrow B$. To reach the goal condition, it keeps the attack sequences $0 \rightarrow A \rightarrow 1 \rightarrow C \rightarrow 2 \rightarrow B \rightarrow 3 \rightarrow D$ and $0 \rightarrow B \rightarrow (1 \wedge 3) \rightarrow C \rightarrow 2 \rightarrow D$. The value return by the algorithm is the time to traverse the shortest of the two previous attack sequences. It is equal to the exact value of the MinTTC which is 6.

Increasing k increases the accuracy of the heuristic algorithm. However, there is a trade off between the accuracy and the execution speed. Increasing k also decreases the scalability of the heuristic algorithm.

Chapter 4

Case Study

4.1 Applying the Metric

We first revisit our running example shown in Figure 1 to apply the proposed metrics framework.

In previous sections, we have shown that the MTTC of a known vulnerability can be estimated as $t = p_1 t_1 + p_2 t_2$ where $t1 = 10/BaseScore$ (we will only consider the base score here since temporal scores are less available at this time) and $t_2 = 5.8 * 10/BaseScore$. We determine the base scores of vulnerabilities by referring to the public vulnerability database NVD [70]. The base scores are respectively equal to 4.6, 7.5 and 6.8 for the vulnerability in ftp, *rsh* services, and the local vulnerability on host 2.

We have presented two approaches to finding the values of p_1 and p_2 in previous discussion, that is, either the probabilities are known (e.g., exploit code listed in an exploit DB) or we can estimate them by applying search theory. In this case study, we take the first approach to assume $p_1 = p_3 = 1$.

For zero day vulnerabilities, the time to exploit can be estimated as $t = p_3 t_3 + p_4 t_4$. Here in this case, we have $t_3 = 32\ days * 10/BaseScore$ and $t_4 = 65\ days + 5.8 * 10/BaseScore$. Similarly as the case of known vulnerabilities, many approaches can be used to find p_3 and p_4. For example, if we know the status of the zero day vulnerability, then p_3 is a nominal value, typically 0.08 and $p_4 = 0.92$. Otherwise if we do not know the status then we may

use search theory to estimate them as $p_3 = 1 - e^{-xk/m}$ and $p_4 = 1 - p_3$. Here, we take the first approach.

Table 5 gives the results of our calculations. The second column lists the time to exploit each vulnerability. The third column is the value of p_r for each exploit. And finally, the last column gives the MTTC values for the post-conditions of each exploit.

Exploit	Time	p_r	Result
$ssh(0,1)$	140.5	0.00285056	140.5
$ftp\_rhosts(0,1)$	10.33	0.018768	10.33
$rsh(0,1)$	6.33	0.018768	16.66
$ssh(1,2)$	48.7	0.01988932	81.69
$ssh(0,2)$	147.03	0.0544	147.03
$bof(2,2)$	6.99	0.074289	136.53

Table 5: Results

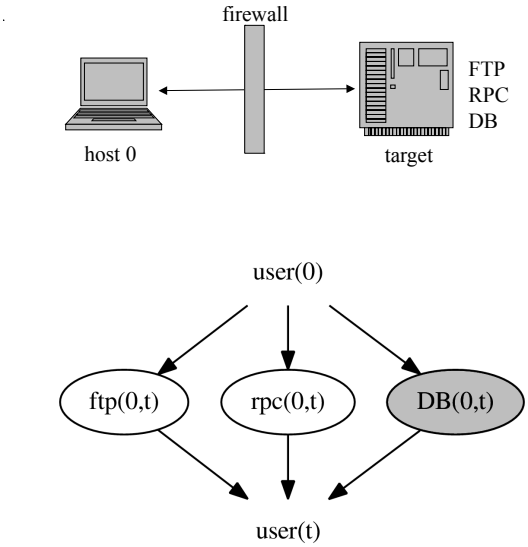
4.2 Comparison of Configurations

Next, we look at how our metric can be used to compare different network configurations (as shown in Figure 10). In all networks, the base score of ftp, rpc and *DB* vulnerabilities are respectively equal to 7.5, 6.4 and 0.8. We provide the results in the following for illustration purposes while omitting detailed calculations due to page limitations.

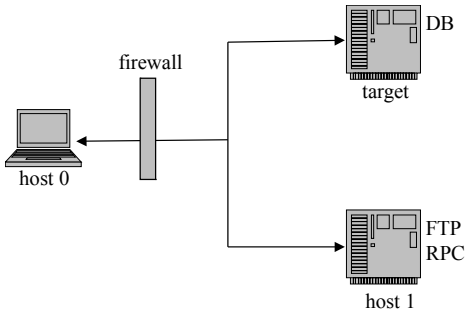
Configuration 1: Figure 10a shows a simple network consisting of a target behind a firewall running three services. Figure 10a also shows the corresponding attack graph. The MTTC in this case is calculated as 7.57.

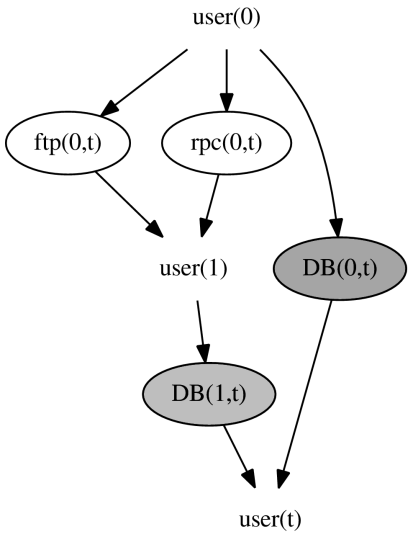
Configuration 2: In this case, the non critical services (FTP and RPC) are isolated and transferred to a new dedicated host (host 1). Figure 10b shows the network and the corresponding attack graph. The time to compromise is equal to 157.65. Isolating vulnerable service greatly improves the security of the network, even if the target is still reachable from the outside.

Configuration 3: Host 1 which contains vulnerable services rcp and ftp is now transferred into a DMZ created by the addition of a new firewall. Outside connection to the target host



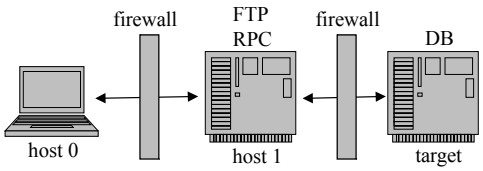
(a) Configuration 1

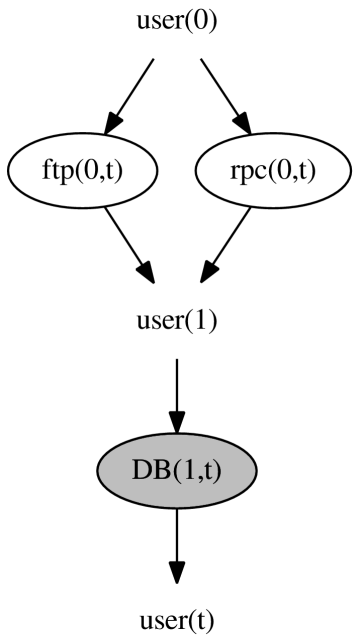




(b) Configuration 2

Figure 10: Comparison of Configurations





(c) Configuration 3

Figure 10: Comparison of Configurations

are now denied. Figure 10c shows the network and its attack graph. The MTTC is equal to 154.63. When the target is hidden, the security increases but not by a great factor.

4.3 Critical Asset

Next, we present an application of our metric to network hardening. Let's consider a network with two hosts. The first host is running a vulnerable *ssh* service. A vulnerable database software is running on host 2. The attack graph which consists of a single attack sequence is given in Figure 11. We assume resources to be limited such that only one host can be patched. Which one should be patched in priority?

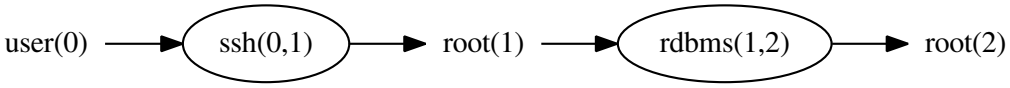


Figure 11: Initial State

Figures 12 and 13 show the network states after one host is patched. Both states will be considered equivalent by any metrics solely based on known vulnerabilities and by k zero day safety metric. Such metrics will randomly select one of the states or leave to human analyst the choice of the best state.

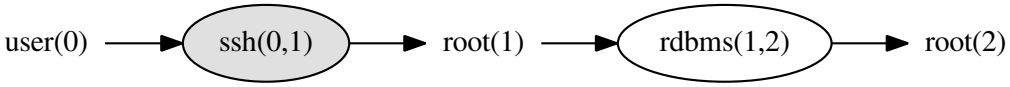


Figure 12: Network State: *ssh* Patched

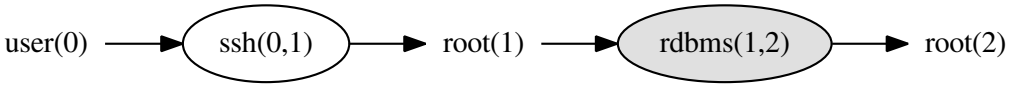


Figure 13: Network State: *rdbms* Patched

However, when using our approach, both network states can be differentiated. If we assume that the mean time to exploit ssh and rdbms are respectively equal to $t(ssh)$ and $t(rdbms)$. If we also assume that the mean time to exploit a zero day vulnerability is $t(zero\ day)$. Then, initially, when both services are vulnerable, the mean time-to-compromise the network is equal to $t(ssh) + t(rdbms)$ When the *ssh* service is fixed, the mean time-to-compromise the network is equal to $t(ssh) + t(zero\ day)$. When the *rdbms* service is patched instead, the mean time-to-compromise the network is equal to $t(rdbms) + t(zero\ day)$. The most critical asset is the asset which when patched produces the least increase in the mean time-to-compromise. When patching the most critical asset,
$\Delta t = t(after\ patching) - t(before\ patching)$ is the least.

In our example, the critical asset is found by comparing $t(ssh) + t(zero\ day)$ with $t(rdbms) + t(zero\ day)$ which equivalent to comparing $t(ssh)$ with $t(rdbms)$.
If $t(ssh) < t(rdbms)$, then the *ssh* service should be patched in priority. Otherwise, if $t(ssh) > t(rdbms)$, then the *rdbms* service should be patched first. If $t(ssh) = t(rdbms)$, then both outcomes provide the same security. In this case, other factors should be considered.

4.4 Some Applications of the MinTTC

4.4.1 Measuring Defense In Depth

One the recommended security best practice is defense in depth or layered defenses. Such security principle aims to increase the number of systems an attacker must exploit before compromising. This is similar to increasing the length of the attack sequences.

To validate defense in depth principles, we consider the networks in Figure 10. Figure 10c represents Figure 10a after a defense in depth procedure has been applied. Before, the network configuration is changed, the fastest way to reach $user(t)$ was to exploit the vulnerability in the ftp service. In Figure 10c however, the MinTTC is equal to time to exploit the ftp and the *DB* services. Separating the services and protecting the database with host 1 has improved the security of this system. The more efficient attacker will have

to spend more time before he can compromise this system.

4.4.2 Comparison with Others Shortest Path Metrics

An unintuitive conclusion we may reach when using shortest paths security metrics is that more vulnerabilities may means more security [43]. For example, if we consider the following shortest path security metrics:

1. Number of conditions; the shortest attack sequence is the attack sequence with the least amount of conditions.

2. Number of exploits; the shortest attack sequence is the attack sequence with the least amount of exploits.

3. Number of conditions and exploits; the shortest attack sequence is the attack sequence with the least amount of conditions and exploits.

The first metrics says that the more there is security condition, the more secure is the system since the shortest path will longer. Similarly, the second metrics says that the more vulnerabilities to exploits in the system, the more secure it becomes. Finally, the third metric states that the more conditions and vulnerabilities, the more secure is the system. Nonetheless, the conclusions derived from the metrics sometimes make sense. The following example (taken from [43]) demonstrate this.

A security engineer is tasked to choose between two web servers. The first web server WS_1 is vulnerable to an exploit leading an attacker to obtain control over the administrator panel. The second web server WS_2 however requires two exploits before its administrator panel can be accessed. A shortest path security metric such as the number of exploit will recommend the security engineer to choose the server WS_2 since it has two exploits in its attack sequence.

When applying the MinTTC, the importance of the number of exploits matters less. The MinTTC is more interested with the time to traverse each path. It is possible that the vulnerability in the server WS_1 is much harder than the two vulnerabilities in WS_2. The

shortest path security metrics presented above will not be able to capture this case. Their main limitation is that they do not discriminate conditions and exploits. Every conditions and exploits are equivalent. Our MinTTC provides better semantic and is capable of handling more cases.

Chapter 5

Simulations

The simulation is conducted using the Python Language and libraries including the Networkx[34], OpenBayes[27], Pygraphviz[33] and Matplotlib[42]. To render the graphs, we use GraphViz visualization package[23]. The experiments were performed inside an Intel Core I7 computer with 8Gb of RAM. The computer is running Ubuntu 12.04 LTS.

5.1 Simulation Environment

Figure 14: Simulation Process

Figure 14 presents the simulations process. Several python scripts were developed to analyze our model. We present them in the following.

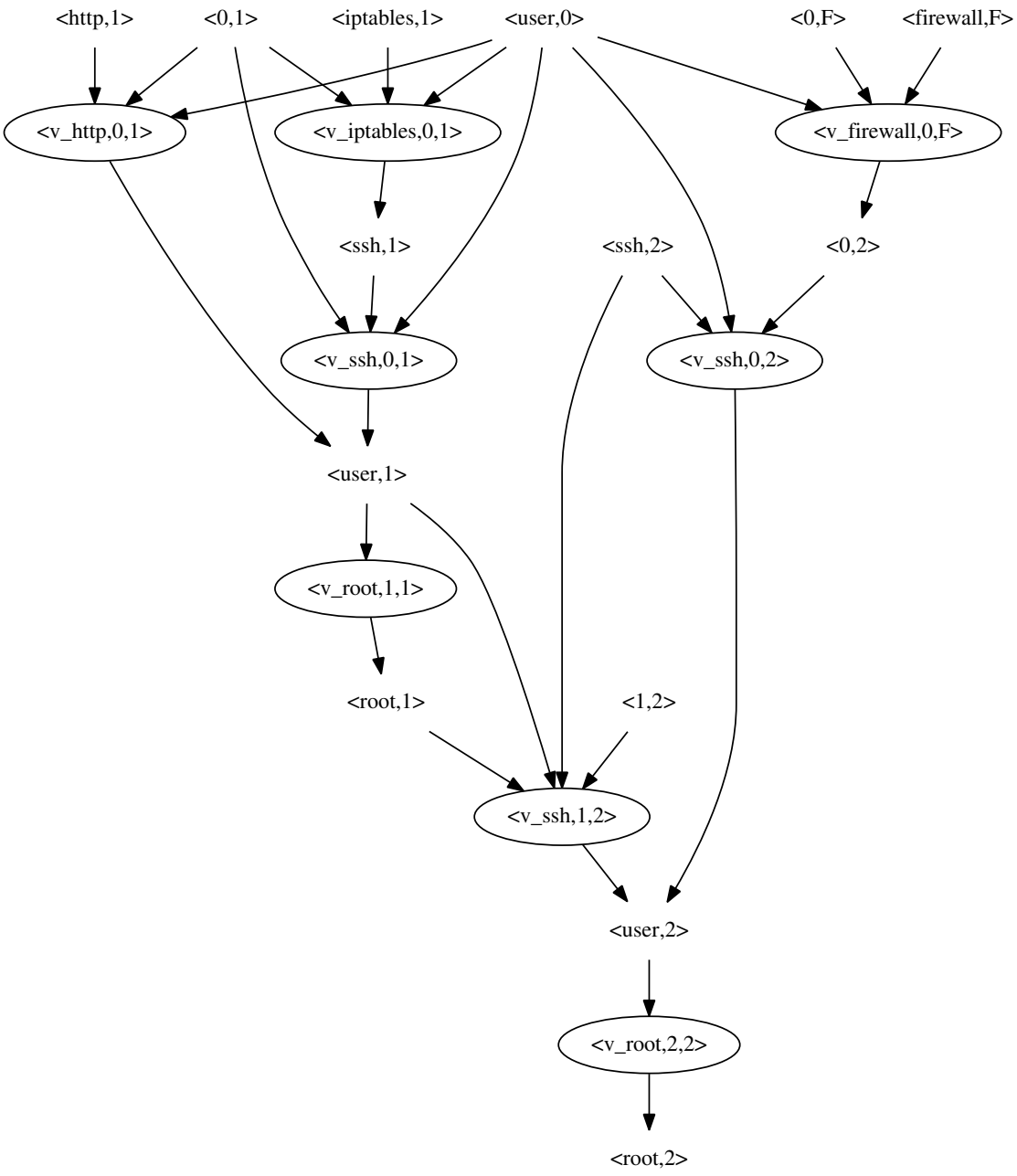


Figure 15: Initial Graph

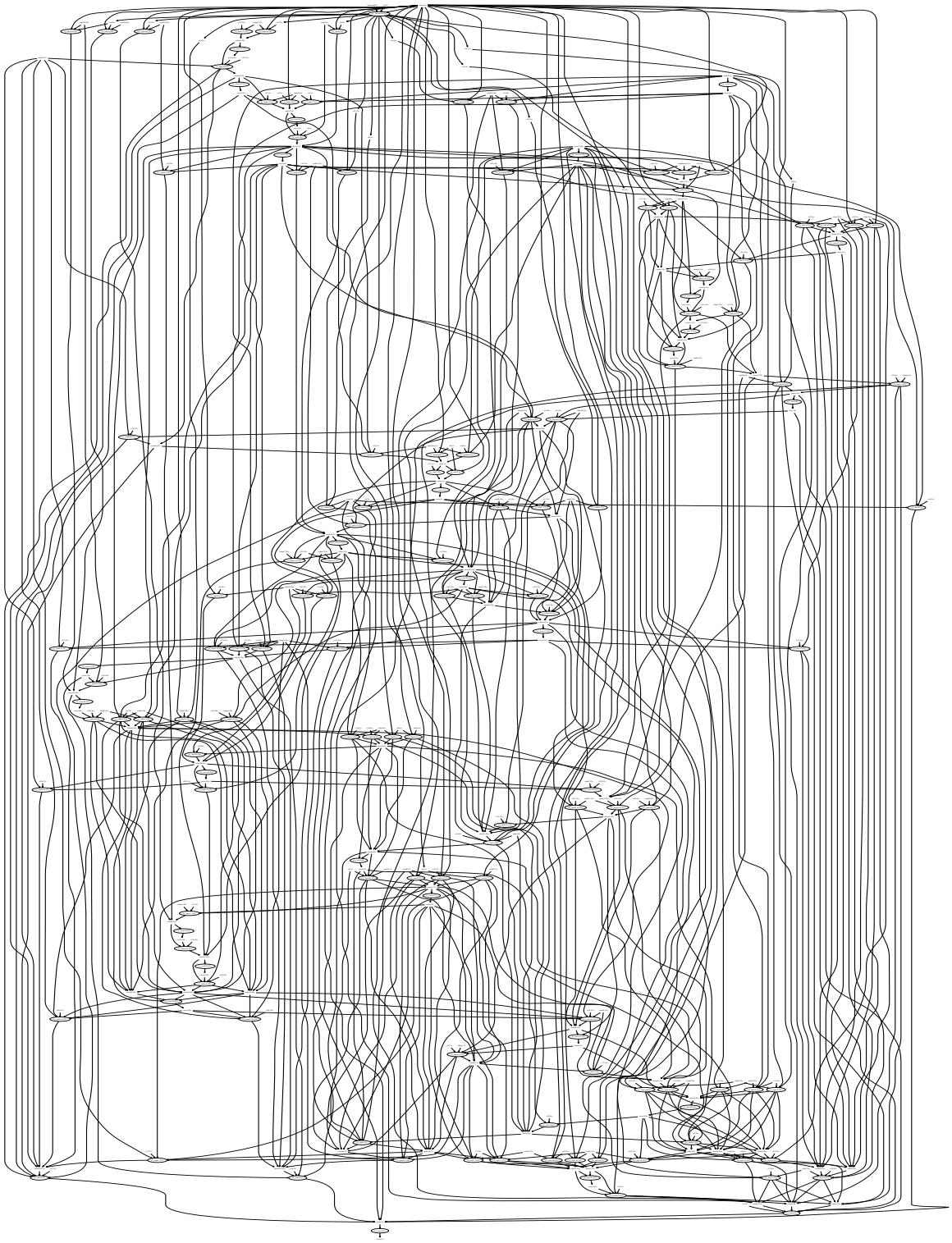


Figure 16: A Random Graph

Attack Graph Generation

The attack graphs were generated using python scripts. First, a seed graph with 20 nodes obtained from real world attack graphs is obtained (Figure 15). Then, conditions and exploits were randomly added to grow the seed graph to a desired size (Figure 16).

Figure 17 shows the components of the script which generates the attack graphs. First, security conditions (and connections) are randomly assigned to hosts. Second, exploits are generated from those conditions. Each exploit is associated with a CVSS base score and a metrics. We specify vulnerabilities to be known or 0day using a random number. In the case the vulnerability is known, we use the NVD to populate the values of the cvss base score and metrics. In the case, the vulnerability is zero day, w e use the values found in section 3.

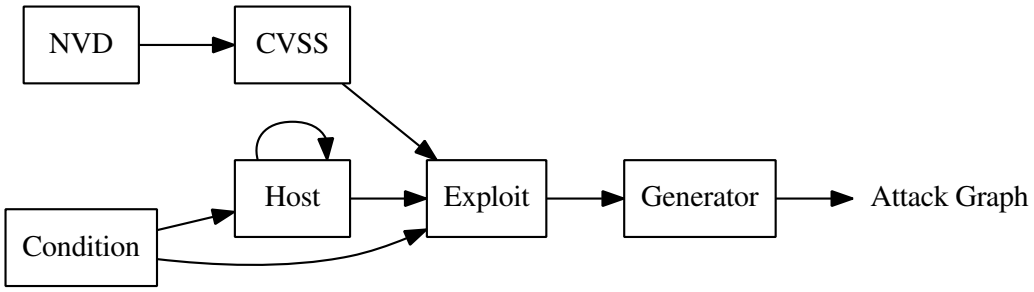


Figure 17: Attack Graph Generation

Attack graphs are generated in memory as dictionary of exploits and conditions and Networkx object. The Networkx object is used with the Pygraphviz library to export the graph to a more convenient format such as the *.dot* or *.png* file formats. The created graph are then used to build a corresponding Bayesian Network.

Bayesian Network Generation

Generation the Bayesian network is very simple (Figure 18) since the attack graphs generated are already acyclic. Using the OpenBayes library, each node and edge in the attack graph is converted into a node and edge in the Bayesian Network. The CPT tables

Figure 18: Bayesian Network Generation

corresponding to each node are also generated. This step allows us to compute all the probabilities of exploiting vulnerabilities considering pre-conditions.

Security Metric Calculation

This python script use the output of the two previous script to compute P_r for all exploits and compute the mean time to compromise. Figure 18 presents the different components of the program.

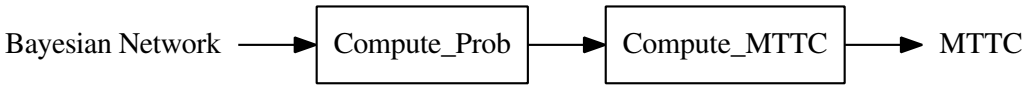


Figure 19: Metric Calculation

Simulator

This multi-threaded Python script uses the three previous script to compute the mean time-to-compromise networks of different sizes using a Monte-Carlo approach. To compute each point in the figures showing our results, we generated 1500 graphs of sensibly the same size, computed their security scores and took the average as our results. This program produced its output in a comma separated values *.csv* file.

5.2 Simulation Results

5.2.1 Size of Network

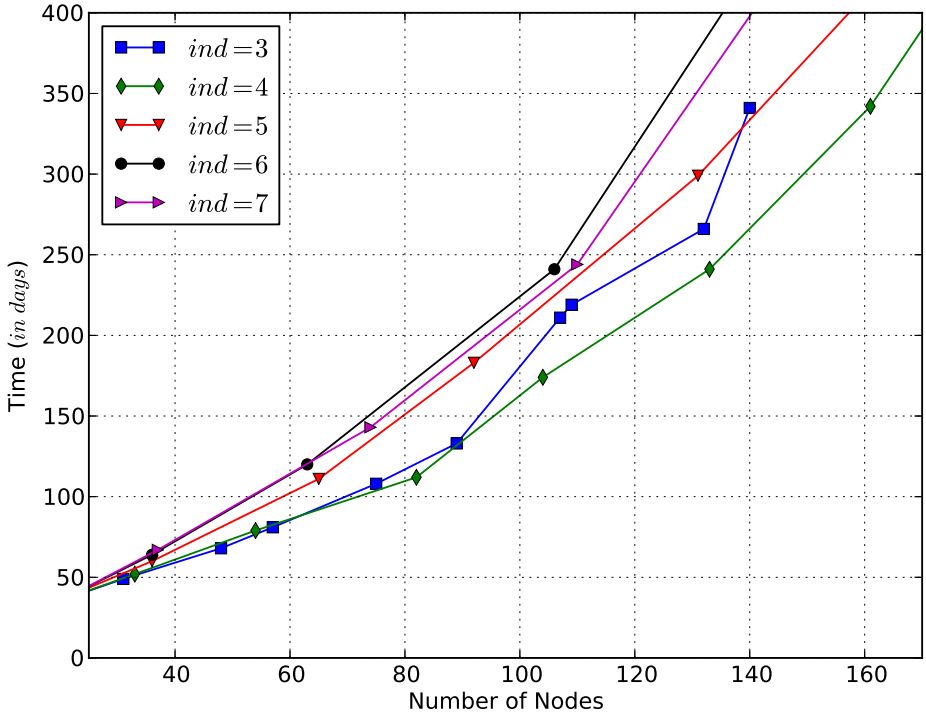


Figure 20: Time to Compromise vs number of nodes

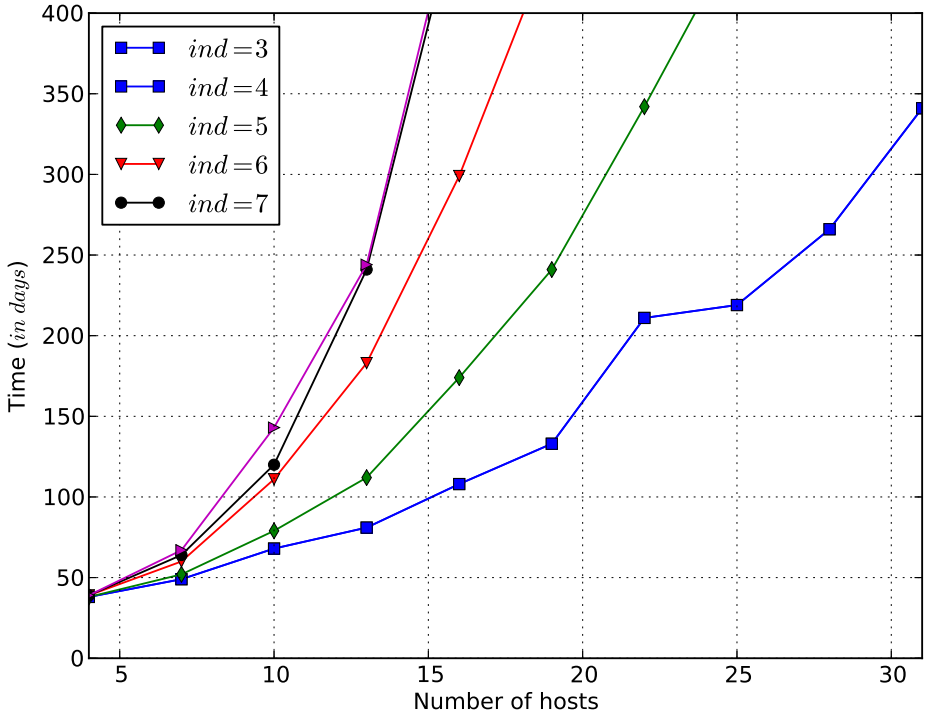


Figure 21: Time to Compromise vs number of hosts

The simulation is intended to first develop a tool to compute the MTTC of a given network and second to investigate the effect of the network size on the MTTC. We want to know if a larger network means a more secure network.

Figures 20 and 21 presents the results of our experiments. We can see that the MTTC grows quickly when the size of the attack graph increases. Increases that is greatly influenced by the number of hosts in the network or the number of nodes in the graph. The fact that the MTTC increases with the size of graph is due to fact that a bigger graph means potentially longer attack sequences since the *goal* condition is always at the bottom of the attack graph.

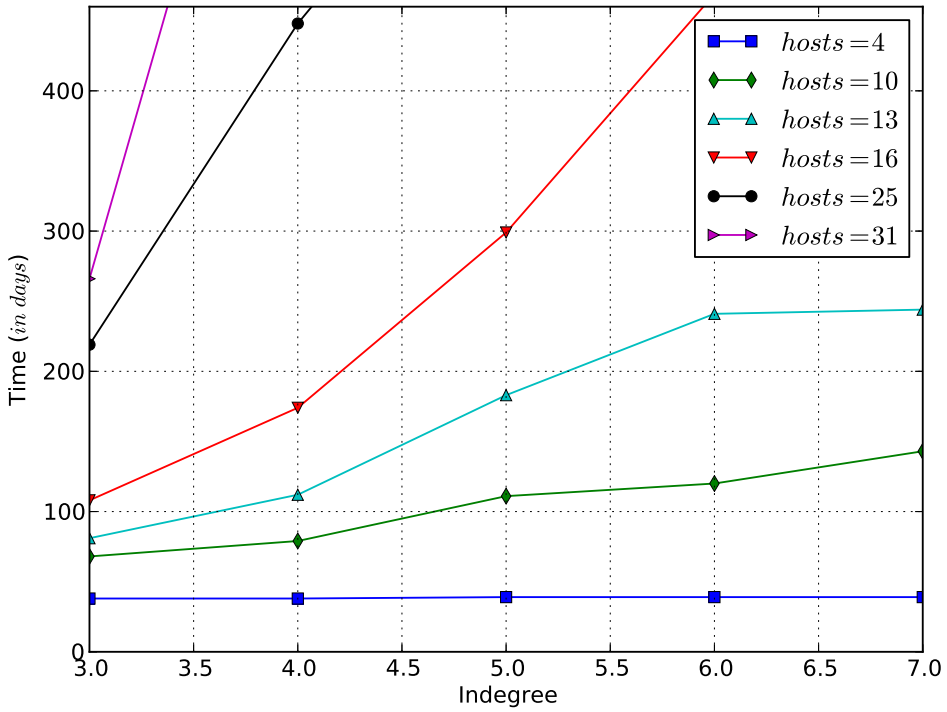


Figure 22: Time to Compromise vs Maximum Indegree

Moreover, the figures show that the MTTC increases with the maximum indegree allowed in the attack graph. The result is relevant since the indegree represents in case of an exploit node the number of preconditions required to launch the exploit. Figure 22 demonstrates more clearly this result. It shows the MTTC of network increases when we do not add more hosts but instead increase the indegree.

5.2.2 Scalability

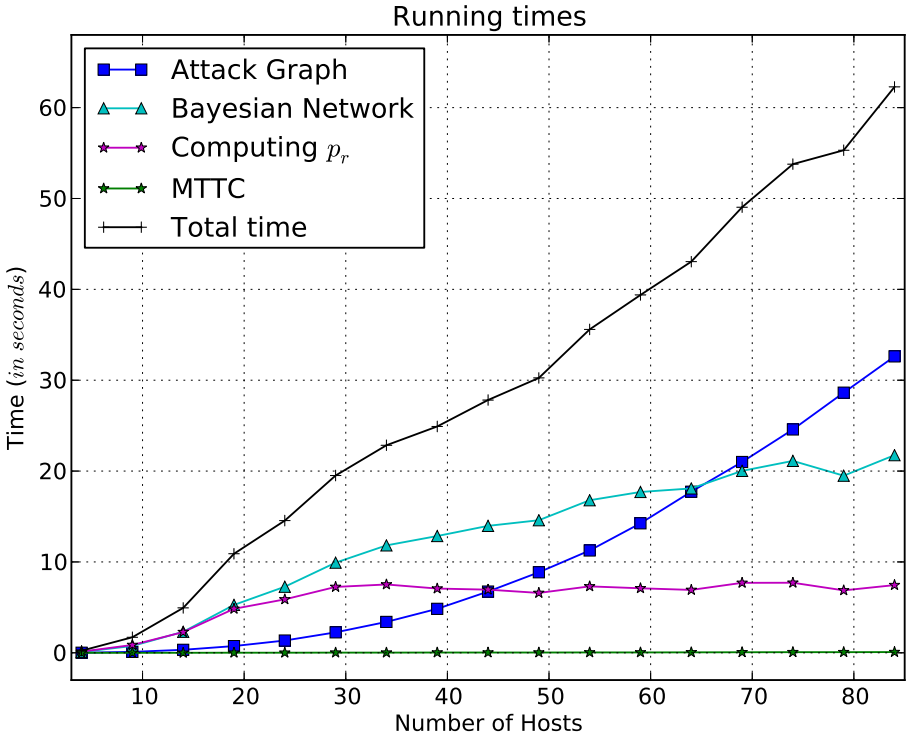


Figure 23: Running Times

The simulation is intended to investigate the scalability of our model. First, we were interested in finding how long it would take to compute the metric for a reasonably large network (approximately 300 nodes in this experiment). Second, we wanted to see what part of our proposed method is the performance bottleneck.

Figure 23 presents the results of our experiments. We can see that the running time is mostly due to the processing for building the attack graph (which includes generating, handling cycles, and removing unreachable nodes, etc.) and the time to construct the Bayesian Network, whereas the running time to actually compute our metrics is relatively scalable.

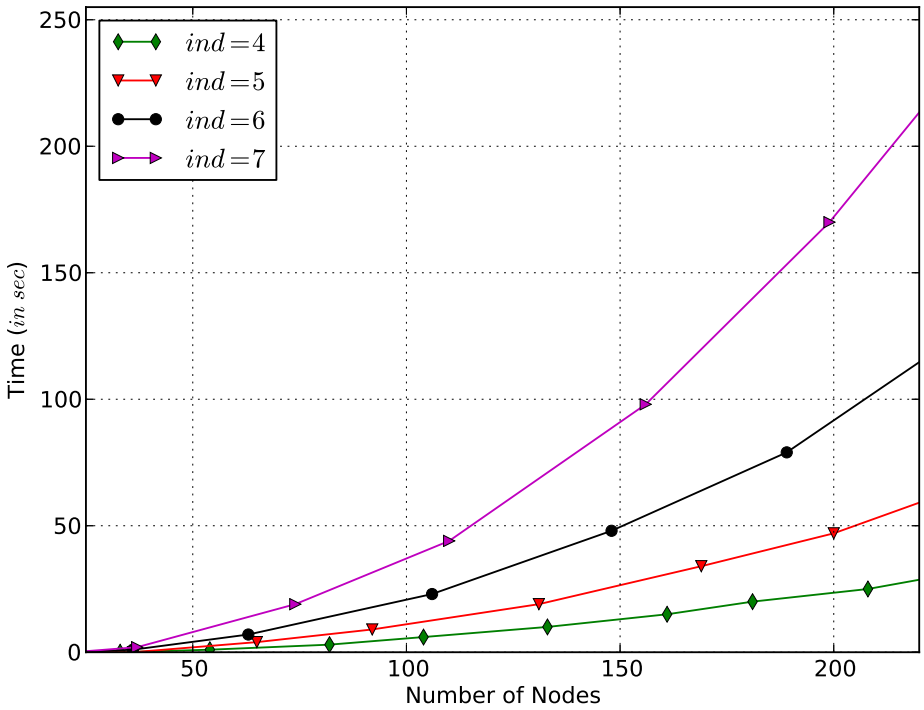


Figure 24: Running Time vs Number of Nodes

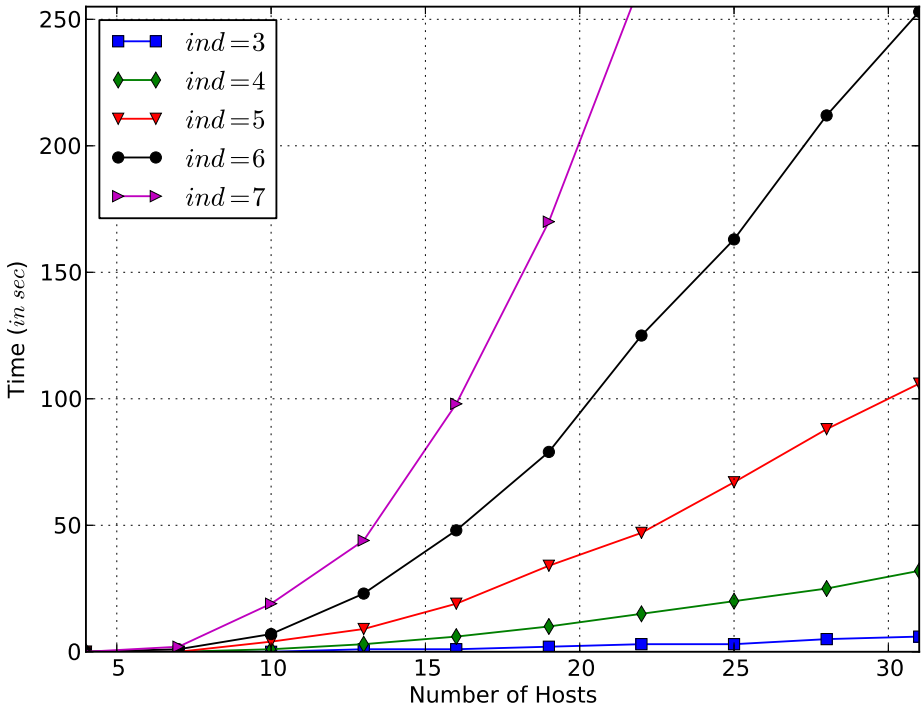


Figure 25: Running Time vs Number of Hosts

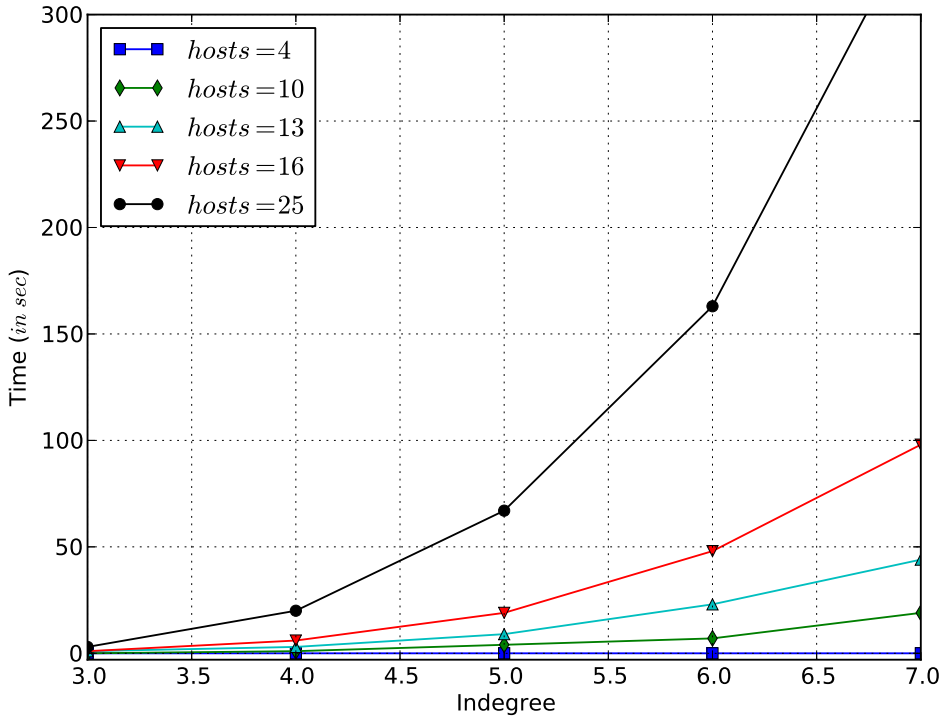
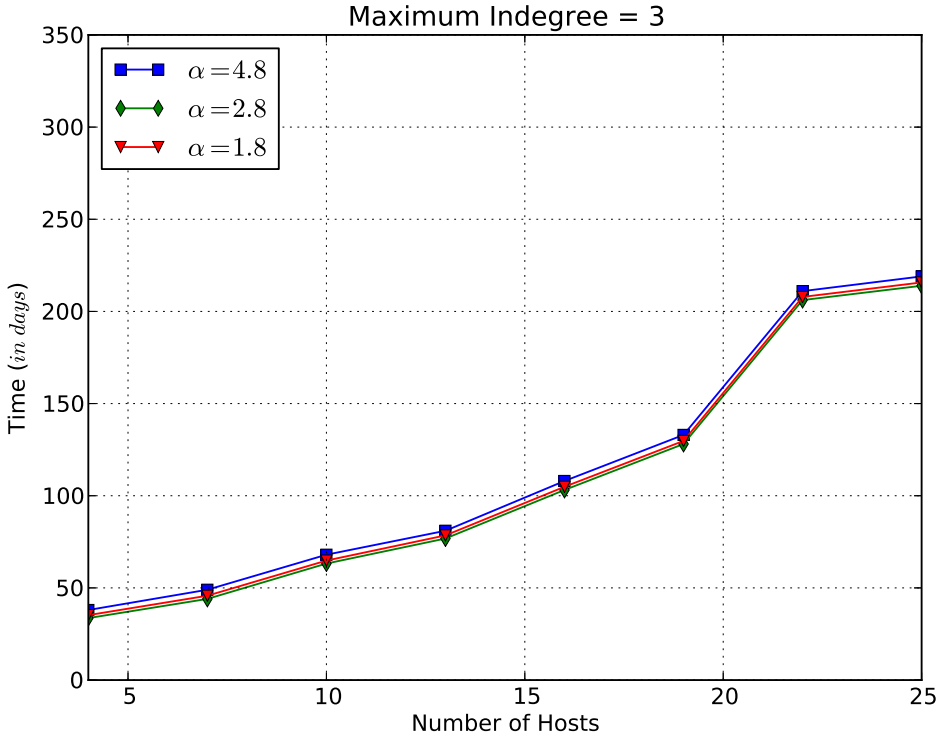
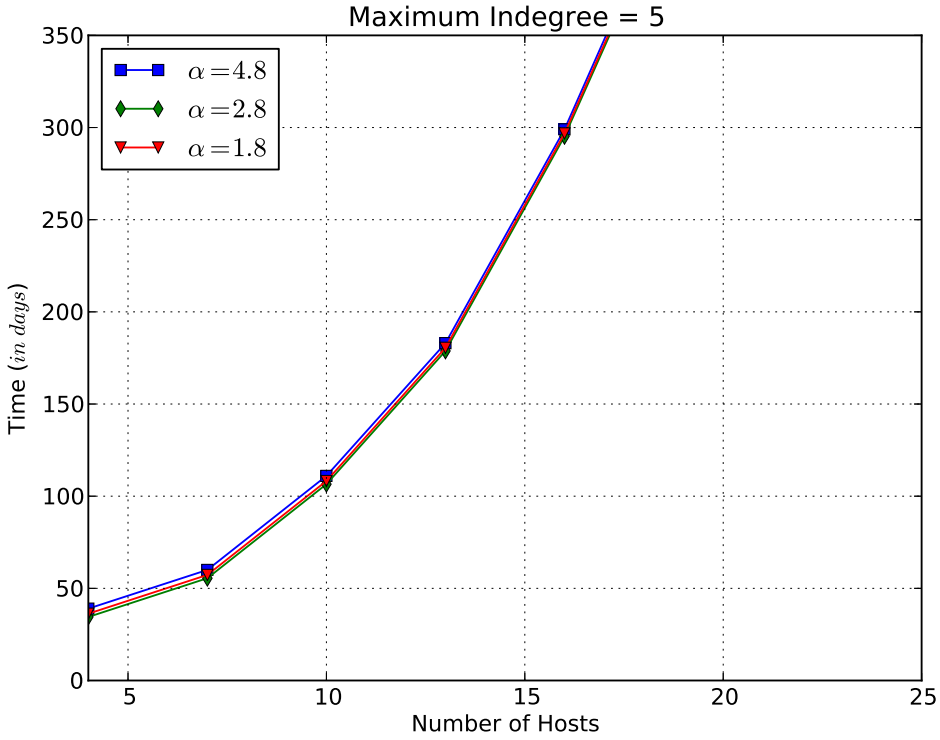


Figure 26: Running Time vs Maximum Indegree

Similarly to the previous experiment, we were interested in the time to run the algorithm for random networks of different sizes. Figures 24, 25 and 26 show that the time to run our algorithm increases with the size of the network. Result which is intuitively correct. The figures also show that the running time increases almost exponentially with the maximum allowed indegree.

5.2.3 Susceptibility





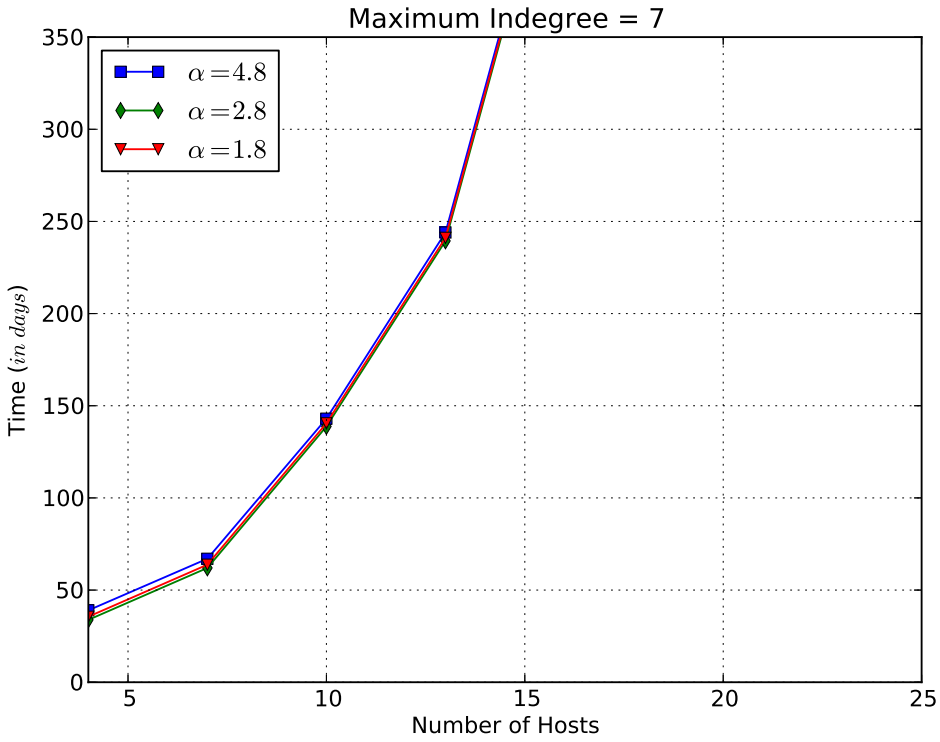


Figure 27: Parameter in Equation 3

To recall, we found the mean time-to-compromise a known vulnerability to be equal to (equation 3):

$$\frac{10}{CVSS(e)}\left(1+4.8*e^{-xk/m}\right)$$

In the above equation, the parameter 4.8 which somehow looks like a magic number was derived using a formula proposed by McQueen [58].

The simulation is intended to investigate its effect on the result of the equation. We want to see how the MTTC changes according to its. For notation purposes, we call it α. For different allowed indegree, we compute the MTTC for α varying between 1.8 and 4.8.

Figure 27 present the result of our experiments. We can see that the shapes in all graphs have the same slope. Meaning that the parameter 4.8 in equation 3 has little incidence on the interpretations made using the result of our metrics. We only choose the value 4.8 for consistency with previously published works.

Chapter 6

Related Works

6.1 Security Metrics

Recently research on security metrics has attracted increasing attention [46]. Standardizing efforts on security metrics has been proposed by NIST [71, 68]. The CVSS metrics measure the severity of individual vulnerabilities [62]. Works such as [80] provide guidelines on implementing security metrics on enterprise network. Intuitive properties that should be satisfied by security metrics are given in [19, 18]. In [93], Wang et al. explore the idea of using attack graphs to compute a generic attack resistance metric.

The arithmetic mean of all attack paths' lengths is regarded as a security metric of average attackers' expected efforts in compromising given critical assets in [51]. In a more recent work [60], the authors rank states in an attack graph based on probabilities of attackers reaching these states during a random simulation; the PageRank algorithm is adapted for such a ranking. In [7], an attack tree is parsed to find sequences of attacks that correspond to the easiest paths followed by potential attackers, and the amount of minimum effort needed along such paths is used as a metric. A similar work replaces attack trees with more advanced attack graphs and replace attack paths with attack scenarios [75]. More recently, the authors in [43] observe that different security metrics will provide only a partial view of security, and the authors then propose a framework for grouping such metrics based on their relative importance. A recent work proposes a risk management framework using

Bayesian networks to quantify the chances of attacks and to develop a security mitigation and management plan [77]. Another recent study of several CVSS-based vulnerability metrics shows the correlation between those metrics and the time-to-compromise of a system [37].

[13] presents properties based on seven security principles that should be satisfied by a security metric to be used on a control system. In [57], a methodology to construct security metrics for supervisory control and data acquisition (SCADA) systems is proposed. The proposed methodology requires the security to determine the dominant attack sequences. In Arce et al. [6], weakest links in legacy systems —such as mainframe and personal computer – are enumerated. A security metric which quantify the most significant security risk factors is presented by [12, 1]. The proposed metric is divided into three parts: the *Existing Vulnerability Measure (EVM)* which measures this risk toward the services within the network, the *Historical Vulnerability Measure (HVM)*, which measures likely is a service to have vulnerabilities given its security history and *The Probabilistic Vulnerability Measure (PVM)*, which gives an indication of the risk faced by the network in the near future. Built upon attack graphs, Ge et al. [29] model security in terms of asset loss, threat value of attack and coefficient of asset importance. Their model has the advantage of being able to asset the security situation of an area of the overall network. Salim Ahmed et al [2] propose a metric, which identifies and quantifies the most significant security risk factors. Their proposed algorithm computes the severity of existing, past and of future vulnerabilities in order to calculate security policies such as attack immunity of a service or the propagation of an attack in the network.

The weakest link idea is applied in the metric by [75]. In his paper, Pamula et al. express the security of a network in terms of the strength of the weakest adversary that can compromise it. The algorithms he proposes find the minimal set of initial attributes necessary for an adversary to fully compromise a target network. Similarly, Schudel et al. [83] measure the workload done by adversaries in order to compromise the security of a system. In his approach, Adversaries are modeled as red teams which are constrained by the tools, techniques they are allowed to use. Red teams are required to only use publicly

available method with broad publications within the information security community.

In [81], interaction between human and computer are used to determine and mitigate the causes of undesirable user's behavior. An early approach to model quantify security in terms of attackers behavior was proposed by [47].

In [21, 20], Dantu et al. used behavior based attack graphs and Bayesian methodology to estimate the security of a given system.

Other security metrics based on user's behavior are presented in [36, 52]. [10, 30] propose game theoretic models to model physical security. In their works, they suggest to view security modeling as a game between attackers and defenders. [35] propose an application this idea to information warfare. In [53], a stochastic game theoretic model to model network security is proposed along with a method to Nash equilibrium or best strategies for the attacker and the network security administrator.

Econometric models measure security in term of the costs and the gain associated to attacking or protecting a system. In [3, 11, 49, 82, 39], metrics based on the concept of return on security investment (ROSI) or return on investment (ROI) are proposed. ROI models try to estimate if the value given by an investment, in this case a security investment, is greater or less than the loss that may occur by not making the investment. A Security metric based on the lowest expected cost for anyone to discover and exploit a vulnerability or cost of break is proposed by [87]. In [90], the compromise of a network is modeled as a functional breakdown of a house system. The interaction and relationship among network components are captured and prioritized using tools such as weakest link or weighted weakest link. 'Finally, a security score is given in term of confidentiality, integrity and availability.

In [91], Wang et al. assign an individual score based on expert knowledge to each exploit. Using this and the score of 1 assigned to each condition, a cumulative score is then calculated. The cumulative score takes in account the relationship between exploit and condition in the graph. The score is computed similarly to the probability of the intersection and union of random events; only disjunctive and conjunctive relations are present in the graph.

Tupper et al. [89] propose a Vea-bility (a CVSS based metric). The metric has the

advantage of working even when connectivity limitations between hosts are in place. The metric uses three inputs: *Network vulnerability dimension*, which is the degree an exploit can impact the network, *Network Exploitability dimension*, which is the overall exploitability of each host and *Network Attackability dimension*, which represents the attackability of each host.

A security model for software systems is proposed by [94]. [40, 56, 7] use attack surface measurement as an indicator of security. The attack surface is the set of ways in which a system can be attacked and compromised. It is the subset of the system's resources that an attacker can use to attack the system. Attack surface of different popular operating systems and ftp daemons are measured in [55, 41].

Most existing work focus on developing security metrics for known vulnerabilities in a network. A few exceptions include an empirical study on the total number of zero day vulnerabilities available on a single day based on existing facts about vulnerabilities [59], a report on the popularity of zero day vulnerabilities among attackers [32], an empirical study on software vulnerabilities' life cycles [84], and more recently an effort on estimating the effort required for developing new exploits [86]. Another recent effort ranks different applications in the same system by how serious the consequence would be if there exists a single zero day vulnerability in those applications [44].

This thesis takes its inspiration from the concept of mean time to compromise (MTTC), which was initially proposed in [58] as a metric for measuring security. In this thesis, attack actions are divided into different statistical processes based on attackers' capabilities, and a probability and time are calculated afterward for each process and then averaged to yield the final result. Leversage et al. [50] extend McQueen's work by breaking the evaluated network into multiple zones (defined as a group of components separated by boundary devices such as a firewall) and a space state predator model is used to represent the attacker's moves toward its target. The main limitation of those works lies in their lack of distinction between different vulnerabilities and an overly simplified attack model. In this thesis, we employ our experiences with attack graphs and vulnerability modeling to improve the MTTC models over those by McQueen and Leversage. In our model, we link

MTTC to specific vulnerabilities' well known CVSS metric values [61], which helps us to utilize readily available inputs and produce more concrete and meaningful results. Also, instead of modeling at the components (hosts) level, we model at the exploit level, which leads to more precise and finer grained results. Finally, instead of computing the MTTC using attack paths like in those, which essentially assume independent attacking steps [38], we use Bayesian network to avoid this limitation.

Another works on which this thesis is based are [25, 26]. Considering an attack graph as a directed acrylic graph, Frigault et al. build a Bayesian Network to estimate network security as the probability of an attacker reaching a specified goal condition. First, he assign individual score to each vulnerability using their CVSS base (or temporal) score. Second, conditional probability tables (CPT) are constructed to encode the relationship among conditions and exploits. Then, the Bayesian Network is used to propagate probabilities in the graph and infer the security score. We include this in as a step in our work. We improve it by providing a model by which zero day vulnerabilities can be handled.

6.2 Attack Graphs

Attack graphs are models used for automating security evaluation. One of the first work to propose attack graphs for modeling network security was done by Philips et al. [76]. In their work, an attack graph is modeled as a set of nodes and edges. Nodes represent possibles attack states and edges changes of state caused by a single action. One of the nodes was chosen as the goal node to represent the target of attacks. An automated attack graph generator is proposed. The generator is based on attack templates, a configuration file and an attacker profile. The graph is then built backwardly starting from the goal node.

Another early effort on attack graph was proposed by Templeton et al. [88]. Here, JIGSAW, a model and a specification language to describe components of attacks in terms of capabilities and concepts was proposed. Capabilities were defined as informations or situations required before an attack can occur. Concepts are subtasks in attack scenario. They specified a set of required capabilities and their value assignments, mapping requirements.

Exploits are combined using this specification to find new attack scenarios.

Ning et al. [67] presented a model to generate attack scenarios based on intrusion alerts. The basic constructs of his model was prerequisites and consequences of attacks. Cuppens et al. proposed a similar concept in [16]. In their paper, a module named CRIM *cooperative module for intrusion detection systems* is described. This module, built upon their previous work on LAMBDA (*A Language to Model a Database for Detection of Attacks*) [17], implements functions to manage, cluster, merge and correlate alerts. Similarly, Cheung et al [14] described a language called *Correlated Attack Modeling Language* to model attack scenarios and recognize scenarios from intrusion alerts.

Gorodetski et al. [31] proposed a model based on stochastic context free grammar to create attack paths. The outcomes of model are a malefactor intention-centric attack modeling, a multi-level attack specification, an ontology-based distributed attack model structuring, an attributed stochastic LL(2) context-free grammar for formal specification of attack scenarios and its components, a formal grammar substitution for specification of multi-level structure of attacks, a state machine-based formal grammar framework implementation and an on-line generation of the malefactor activity. A similar model to build attack graphs from network components, attacker's privileges on hosts, reachability of hosts and vulnerabilities is also proposed by [22].

In [79], a model checking approach is used to determine if a given network is secure with respect to a security condition. Model checking verifies the reachability of a given security condition. The idea is to postulate using a model checker that the network is secure. If the network was indeed secure, the model checker will confirm it. If it was not, the model checker will produce a counterexample to show how the network could be compromised (attack sequence). [85] extended this idea of using model checking. The network state is modeled as a collection of Boolean variables, attacker's actions as transitions between states. The security of the network is specified as a formula, which is tested by a model checker. This approach has the advantage of being able to produce all counterexamples instead of just one. Model checking approaches however have some scalability issues due to the potential states explosion.

Amman et al [5] proposed a *monotonicity assumption* to reduce the running complexity of generating attack graphs. The *monotonicity assumption* states that an attacker never relinquish what it gained. He never backtrack. This assumption reduces the complexity from exponential in the number of hosts to polynomial. The attack graph is constructed in two phases. In the first (forward) phase, exploits and conditions are connected. In the second (backward) phase, irrelevant (those which cannot be reached from the goal state) states are removed.

Another scalable approach for generating attack graph is proposed by [72] using logic programming. The output of the model is a logical attack graphs in which nodes are logical statements. The edges specify the causality relations between network configurations and an attacker's potential privileges. The graph generation and logic programming is done through MulVAL [73], a framework for modeling the interaction of software bugs with system and network configurations. The idea is that configuration information are represented as Datalog (a subset of the programming language Prolog) tuples and most attack techniques and OS security semantics can be specified using Datalog rules [72].

6.3 Time-To-Compromise and Time to / before Failures

Dependability is a large concept that encompass reliability, availability, safety and performability. Reliability is the probability that a device will perform its function over a certain amount of time, subjected to certain conditions. It is quantified as the Mean time to failure (MTTF) if the object is not repairable and as the mean time before failure (MTBF) if the object is repairable. MTTF and MTBF account for any type of failure; whether it is human induced or not.

The time-to-compromise on the other hand only accounts for failure caused by intentional attacks. In this work, we have defined the time to compromise as the time it takes for an attacker, from the moment he starts his attack until he reaches his goal. However, from the viewpoint of a security officer monitoring a system, the time-to-compromise may represent the time between compromises or breaches.

Since MTTF/MTBF represents more general cases, then the time-to-compromise is likely greater or equal to its value. However, it is difficult to apply MTTF/MTBF in security analysis. A comprehensive analysis on the issues and challenges when transferring dependability analysis models to security is proposed by [66]. The authors basically state that although probabilistic structure can be assumed when modeling cyber attacks, developing and validating good stochastic models is still an open issue.

Chapter 7

Future Work

7.1 Limitations

In this section, we present some of the limitations of our model. The first and most important limitation is the difficulty to apply our MTTC model in practice. Some of the assumptions we make in this work could hard to apply for real networks. Second, there is no data to validate our results. A future work may be to aggregate those data through empirical works such as security exercises or logs from real networks. Third, a limitation that affect most of the current security is the fact it is difficult to translate real networks to a model that will be used by our metric.

We also acknowledge that the MTTC of known exploits (computed in section 3.3.1) does not take into account the time the vulnerability was discovered. It is evident the longer a vulnerability is known, the easier exploiting may become. Adding a parameter $\beta = t_c - t_d$ (where t_c is the current day and t_d the time of disclosure) to the MTTC of exploit may provide better results.

7.2 Extending The Models

As future works, we propose to extend our work to perform network hardening. As shown in section 4, the mean time-to-compromise can be utilized to prioritize hardening effort. A

logical follow up would be to develop algorithm so that the MTTC could produce a ordered list of assets that should be patched so that a given network security becomes lower or equal to a specified accepted risk.

A similar idea to the one previously proposed is to use the mean time-to-compromise for cyber situational awareness, in particular to "Be aware of the impact of the attack" and to "Identification of better response plans and actions" [8].

Another extension that can be made to work is to use the MTTC to compute the return on security investment (ROSI). ROSI is defined by the European Network and Information Security Agency [64] as:

$$ROSI = \frac{ALE * \text{mitigation ratio} - \text{Cost of the solution}}{\text{Cost of the solution}}$$

where *ALE* is the Annual Loss Expectancy.
It can easily be seen that there is strong positive correlation between the mean time to compromise and the mitigation ratio in the above equation. We believe that the greater is the mean time-to-compromise a network, the fewer is the number of attackers attempting to break-in. The increase of the MTTC induces an increase of the ROSI. A future could be to compute the value of the increase; rewrite the ROSI formula using the MTTC.

Another work that can be done is to measure how realistic is our model of the mean time-to-compromise exploits. An approach could be to experiments such as the one described in [37] to build a comprehensive database of the mean time-to-compromise exploits. Such database could then be used by researchers in the field to measure the accuracy of their proposed models.

Another application of our metric is to create a best configuration for a given set of network components. Given a set of components and requirements (an example of requirement could be the aggregation of certain hosts into a specific subnet), the model will construct the safest network topology. A non scalable approach could be to create all possible configurations, which respect the requirements and select the one with the higher mean time-to-compromise. Another approach could be to add one host at the time in the network. Hosts are added such that the mean time to compromise remain maximal. Although

the second approach is more scalable, more work needs to be done to prove it correctness. The same goes with the feasibility of both approach.

Chapter 8

Conclusion

In this thesis, we have proposed a MTTC framework for addressing an important limitation of existing approaches, namely, the lack of support for both known and unknown vulnerabilities. We have defined the generic MTTC concept, and then provided concrete methods for instantiating the concept into actionable metrics. Although our methods for estimating exploits' likelihood and mean time may not fit the needs of every application, the general framework will still work with a different realization of the input values.

Chapter 9

Publications

Publication related to this thesis are the following:

- William Nzoukou, Lingyu Wang, Sushil Jajodia, Anoop Singhal, "A Unified Framework for Measuring a Network's Mean Time-to-Compromise", Proc. *32st IEEE International Symposium on Reliable Distributed Systems (SRDS 2013)*, Braga, Portugal, September 30 - October 3, 2013

Chapter 10

Bibliography

[1] M. Abedin, S. Nessa, E. Al-Shaer, and L. Khan. Vulnerability analysis for evaluating quality of protection of security policies. In *Proceedings of the 2nd ACM workshop on Quality of protection*, QoP '06, pages 49–52, New York, NY, USA, 2006. ACM.

[2] M.S. Ahmed, E. Al-Shaer, and L. Khan. A novel quantitative approach for measuring network security. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1957 –1965, april 2008.

[3] M. Al-Humaigani and D.B. Dunn. A model of return on investment for information systems security. In *Proceedings of the 2003 IEEE 46th Midwest Symposium on Circuits and Systems,*, volume 1, pages 483 –485 Vol. 1, dec. 2003.

[4] M. Albanese, S. Jajodia, and S. Noel. Time-efficient and cost-effective network hardening using attack graphs. In *Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '12, pages 1–12, Washington, DC, USA, 2012. IEEE Computer Society.

[5] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 217–224, New York, NY, USA, 2002. ACM.

[6] I. Arce. The weakest link revisited [information security]. *IEEE Security & Privacy*, 1(2):72 – 76, mar-apr 2003.

[7] D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st ACM QoP*, 2005.

[8] P. Barford, M. Dacier, T. Dietterich, M. Fredrikson, J. Giffin, S. Jajodia, S. Jha, J. Li, P. Liu, P. Ning, X. Ou, D. Song, L. Strater, V. Swarup, G. Tadda, C. Wang, and J. Yen. Cyber sa: Situational awareness for cyber defense. In *Cyber Situational Awareness*, volume 46 of *Advances in Information Security*, pages 3–13. Springer US, 2010.

[9] I.E. Ben-Gal. Bayesian networks. `http://www.eng.tau.ac.il/~bengal/BN.pdf`.

[10] V. Bier. Game-theoretic and reliability methods in counterterrorism and security. *Statistical Methods in Counterterrorism*, pages 23–40, 2006.

[11] S. Bistarelli, F. Fioravanti, and P. Peretti. Defense trees for economic evaluation of security investments. In *ARES 2006. The First International Conference on Availability, Reliability and Security, 2006.*, pages 8–pp. IEEE, 2006.

[12] R. Bohme and T. Moore. The iterated weakest link. *IEEE Security & Privacy*, 8(1):53 –55, jan.-feb. 2010.

[13] W. Boyer and M. McQueen. Ideal based cyber security technical metrics for control systems. *Critical Information Infrastructures Security*, pages 246–260, 2008.

[14] S. Cheung, U. Lindqvist, and M.W. Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 284 – 292 vol.1, april 2003.

[15] The Mitre Corporation. Common vulnerabilities and exposures (cve). `http://cve.mitre.org/`.

[16] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security & Privacy*, SP '02, pages 202–, Washington, DC, USA, 2002. IEEE Computer Society.

[17] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, RAID '00, pages 197–216, London, UK, UK, 2000. Springer-Verlag.

[18] M. Dacier. *Towards quantitative evaluation of computer security*. PhD thesis, Institut National Polytechnique de Toulouse,, 1994.

[19] M. Dacier, Y. Deswarte, and M. Kaâniche. Quantitative assessment of operational security: Models and tools. *Information Systems Security, ed. by SK Katsikas and D. Gritzalis, London, Chapman & Hall*, pages 179–86, 1996.

[20] R. Dantu and P. Kolan. Risk management using behavior based bayesian networks. *Intelligence and Security Informatics*, pages 165–184, 2005.

[21] R. Dantu, K. Loper, and P. Kolan. Risk management using behavior based attack graphs. In *Proceedings of ITCC 2004. International Conference on Information Technology: Coding and Computing, 2004.*, volume 1, pages 445–449. IEEE, 2004.

[22] J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of the Second IEEE International Information Assurance Workshop, 2004.*, pages 48 – 56, april 2004.

[23] J. Ellson, E. Gansner, L. Koutsofios, S. North, G. Woodhull, Short Description, and Lucent Technologies. Graphviz — open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[24] N. Falliere and O. Liam. W32.Stuxnet Dossier, 2011.

[25] M. Frigault and L. Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of the 2008 32nd Annual IEEE International Computer*

Software and Applications Conference, COMPSAC '08, pages 698–703, Washington, DC, USA, 2008. IEEE Computer Society.

[26] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, QoP '08, pages 23–30, New York, NY, USA, 2008. ACM.

[27] K. Gaitanis and E. Cohen. Openbayes 0.1.0. `http://pypi.python.org/pypi/OpenBayes/0.1.0`.

[28] 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.

[29] H. Ge, L. Gu, Y. Yang, and K. Liu. An attack graph based network security evaluation model for hierarchical network. In *Proceedings of the 2010 IEEE International Conference on Information Theory and Information Security (ICITIS)*, pages 208 –211, dec. 2010.

[30] B. Golany, E.H. Kaplan, A. Marmur, and U.G. Rothblum. Nature plays with dice–terrorists do not: Allocating resources to counter strategic versus probabilistic risks. *European Journal of Operational Research*, 192(1):198–208, 2009.

[31] V. Gorodetski and I. Kotenko. Attacks against computer network: formal grammar-based framework and simulation tool. In *Proceedings of the 5th international conference on Recent advances in intrusion detection*, RAID'02, pages 219–238, Berlin, Heidelberg, 2002. Springer-Verlag.

[32] A. Greenberg. Shopping for zero-days: A price list for hackers' secret software exploits. *Forbes*, 23 March 2012.

[33] A.A. Hagberg, D.A. Schult, and P.J. Swart. pygraphviz. `http://networkx.lanl.gov/pygraphviz/`.

[34] A.A Hagberg, D.A. Schult, and P.J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug 2008.

[35] S.N. Hamilton, W.L. Miller, A. Ott, and O.S. Saydjari. The role of game theory in information warfare. In *4th Information survivability workshop (ISW-2001/2002), Vancouver, Canada*, 2002.

[36] K. Hausken. Protecting complex infrastructures against multiple strategic attackers. *International Journal of Systems Science*, 42(1):11–29, 2011.

[37] H. Holm, M. Ekstedt, and D. Andersson. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable Secure Computing*, 9(6):825–837, November 2012.

[38] J. Homer, X. Ou, and D. Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Technical report, Kansas State University, Computing and Information Sciences Department*, 2009.

[39] S. Hoo and K. John. *How much is enough: a risk management approach to computer security*. PhD thesis, Stanford, CA, USA, 2000. AAI9986202.

[40] M. Howard. Fending off future attacks by reducing the attack surface, february 2003. *URL http://msdn. microsoft. com/library/default. asp*, 2004.

[41] M. Howard, J. Pincus, and J. Wing. Measuring relative attack surfaces. In D. T. Lee, S. P. Shieh, and J. D. Tygar, editors, *Computer Security in the 21st Century*, pages 109–137. Springer US, 2005. 10.1007/0-387-24006-38.

[42] J.D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[43] N. Idika and B. Bhargava. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on Dependable and Secure Computing*, 9:75–85, 2012.

[44] 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, 2009.

[45] S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In Vipin Kumar, Jaideep Srivastava, and Aleksandar Lazarevic, editors, *Managing Cyber Threats*, volume 5 of *Massive Computing*, pages 247–266. Springer US, 2005. 10.1007/0-387-24230-9.

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

[47] E. Jonsson and T. Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *Software Engineering, IEEE Transactions on*, 23(4):235 –245, apr 1997.

[48] J. Labelle. *Recueil de problèmes de probabilités avec solutions*. LOZE-DION, 2011.

[49] V.C. S. Lee and L. Shao. Estimating potential it security losses: An alternative quantitative approach. *IEEE Security & Privacy*, 4(6):44 –52, nov.-dec. 2006.

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

[51] W. Li and R.B. Vaughn. Cluster security research involving the modeling of network exploitations using exploitation graphs. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 26–, Washington, DC, USA, 2006. IEEE Computer Society.

[52] P. Liu, W. Zang, and M. Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Trans. Inf. Syst. Secur.*, 8(1):78–118, February 2005.

[53] K. Lye and J.M. Wing. Game strategies in network security. *International Journal of Information Security*, 4(1):71–86, 2005.

[54] J.A. Major. Advanced techniques for modeling terrorism risk. *The Journal of Risk Finance*, 4(1):15–24, 2002.

[55] P. Manadhata, J. Wing, M. Flynn, and M. McQueen. Measuring the attack surfaces of two ftp daemons. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 3–10. ACM, 2006.

[56] P.K. Manadhata and J.M. Wing. A formal model for a system's attack surface. *Moving Target Defense*, pages 1–28, 2011.

[57] M.A. McQueen, W.F. Boyer, M.A. Flynn, and G.A. Beitel. Quantitative cyber risk reduction estimation methodology for a small scada control system. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences, 2006. HICSS '06.*, volume 9, page 226, jan. 2006.

[58] M.A. McQueen, W.F. Boyer, M.A. Flynn, and G.A. Beitel. Time-to-compromise model for cyber risk reduction estimation. In Dieter Gollmann, Fabio Massacci, and Artsiom Yautsiukhin, editors, *Quality of Protection*, volume 23 of *Advances in Information Security*, pages 49–64. Springer US, 2006. 10.1007/978-0-387-36584-8_5.

[59] M.A. McQueen, T.A. McQueen, W.F. Boyer, and M.R. Chaffin. Empirical estimates and observations of 0day vulnerabilities. *Hawaii International Conference on System Sciences*, 0:1–12, 2009.

[60] V. Mehta, C. Bartzis, H. Zhu, E.M. Clarke, and J.M. Wing. Ranking attack graphs. In *Recent Advances in Intrusion Detection 2006*, 2006.

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

[62] P. Mell, K. Scarfone, and S. Romanosky. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-Forum of Incident Response and Security Teams*, pages 1–23, 2007.

[63] C. Miller. The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales. In *Proceedings of the Sixth Workshop on the Economics of Information Security*, 2007.

[64] European Network and Information Security Agency. Introduction to return on security investment. http://www.enisa.europa.eu/activities/cert/other-work/introduction-to-return-on-security-investment.

[65] J. Niccolai. Oracle vs sap lawsuit: 27 most popular questions about the tomorrownow case. http://www.computerworlduk.com/in-depth/it-business/3246680/oracle-vs-sap-lawsuit-27-most-popular-questions-about-the-tomorrownow-case/.

[66] D.M. Nicol, W.H. Sanders, and K.S. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1(1):48 – 65, jan.-march 2004.

[67] P. Ning, Y. Cui, and D.S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 245–254, New York, NY, USA, 2002. ACM.

[68] National Institute of Standards and Technology. Directions in security metrics research. *NISTIR 7564*.

[69] National Institute of Standards and Technology. National software reference library. http://www.nsrl.nist.gov.

[70] National Institute of Standards and Technology. National vulnerability database version 2.2. http://nvd.nist.gov/.

[71] National Institute of Standards and Technology. Information security. *NIST Special Publication 800-55 Revision 1*, July 2008.

[72] 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*, CCS '06, pages 336–345, New York, NY, USA, 2006. ACM.

[73] 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*, SSYM'05, pages 8–8, Berkeley, CA, USA, 2005. USENIX Association.

[74] A. Ozment. Bug auctions: Vulnerability markets reconsidered. In *Third Workshop on the Economics of Information Security*, 2004.

[75] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the ACM Quality of Protection*, pages 31–38, 2006.

[76] C. Phillips and L.P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, NSPW '98, pages 71–79, New York, NY, USA, 1998. ACM.

[77] N. Poolsappasit, R. Dewri, and I. Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable Secure Computing*, 9(1):61–74, January 2012.

[78] Rapid7. Metasploit project. `http://www.metasploit.com/`.

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

[80] SANS Institute InfoSec Reading Room. A guide to security metrics. `http://www.sans.org/reading_room/whitepapers/auditing/guide-security-metrics_55`.

[81] M.A. Sasse, S. Brostoff, and D. Weirich. Transforming the 'weakest link—a human/computer interaction approach to usable and effective security. *BT technology journal*, 19(3):122–131, 2001.

[82] S. Schechter. Toward econometric models of the security risk from remote attack. *IEEE Security & Privacy*, 3(1):40–44, January 2005.

[83] G. Schudel and B. Wood. Adversary work factor as a metric for information assurance. In *Proceedings of the 2000 workshop on New security paradigms*, NSPW '00, pages 23–30, New York, NY, USA, 2000. ACM.

[84] M. Shahzad, M.Z. Shafiq, and A.X. Liu. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012.

[85] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security & Privacy*, pages 273 – 284, 2002.

[86] T. Sommestad, H. Holm, and M. Ekstedt. Effort estimates for vulnerability discovery projects. In *Proceedings of the 2012 45th Hawaii International Conference on System Sciences*, HICSS '12, pages 5564–5573, Washington, DC, USA, 2012. IEEE Computer Society.

[87] S. Stuart. Quantitatively differentiating system security. In *The First Workshop on Economics and Information Security*, pages 16–17, 2002.

[88] J. Templeton, S. and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms*, NSPW '00, pages 31–38, New York, NY, USA, 2000. ACM.

[89] M. Tupper and A. Nur Zincir-Heywood. Vea-bility security metric: A network security analysis tool. In *Proceedings of the 2008 Third International Conference on*

Availability, Reliability and Security, ARES '08, pages 950–957, Washington, DC, USA, 2008. IEEE Computer Society.

[90] C. Wang and W.A. Wulf. Towards a framework for security measurement. In *Proceedings of the Twentieth National Information Systems Security Conference*, pages 522–533, 1997.

[91] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 283–296, Berlin, Heidelberg, 2008. Springer-Verlag.

[92] 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.

[93] L. Wang, A. Singhal, and S. Jajodia. Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM workshop on Quality of protection*, QoP '07, pages 49–54, New York, NY, USA, 2007. ACM.

[94] A. Yautsiukhin, R. Scandariato, T. Heyman, F. Massacci, and W. Joosen. Towards a quantitative assessment of security in software architectures. In *Nordic Workshop on Secure IT Systems (NordSec), Copenhagen, Denmark, October 2008*, October 2008.