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

William Nzoukou*, Lingyu Wang*, Sushil Jajodia† and Anoop Singhal‡
*Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada
Email: {w_nzouko,wang}@ciise.concordia.ca
†Center for Secure Information Systems, George Mason University, Fairfax, VA 22030-4444, USA
Email: jajodia@gmu.edu
‡Computer Security Division, National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899, USA
Email: anoop.singhal@nist.gov

***Abstract*—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 paper, 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.**

I. 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. A major difficulty in securing computer networks in such mission critical systems 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 II 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 [34].

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.

A. 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) [3] 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 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)* or *condition($host_1$, $host_2$)*, 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 safety metric [34] 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 $root(2)$.

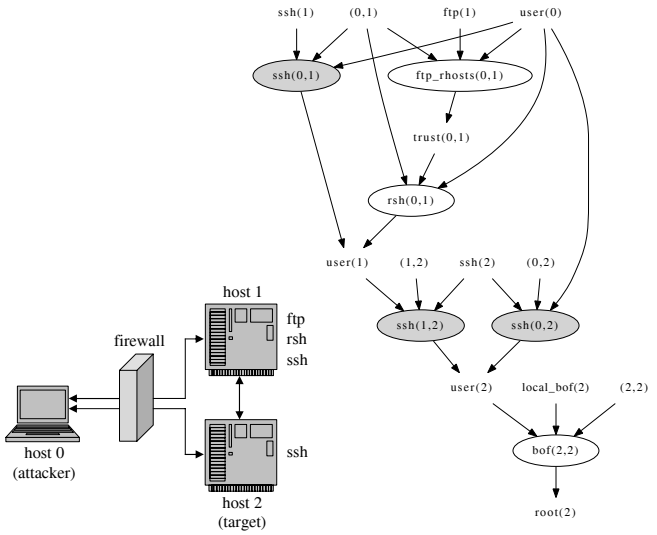

Fig. 1: An Example

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 paper).

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 (since we are counting the number of zero day vulnerabilities [34]) 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 the two zero day exploits involve the same service, the metric score becomes $1 + s_{bof}$ (here two zero day exploits involving the same service and hence vulnerability will be counted only once since attackers may reuse their knowledge and tools [34]), 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 paper, we address this important issue through exploring 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 (note ssh is exploited twice in this case) ,
2) $t_2 = \kappa + min(f(ssh), \kappa') + f(bof)$ for case 2 (where $min()$ means the minimum value; ssh on host 1 is patched so it becomes an unknown vulnerability; $min(f(ssh), \kappa')$ means attackers may either exploit ssh on host 2 or exploit another, unknown vulnerability on host 2, but only the easier one will be counted), and
3) $t_3 = \kappa + \kappa' + f(bof)$ for case 3 (in this case the same unknown vulnerability on both host 1 and host 2 is exploited twice).

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 than case 2) 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, subsequently exploiting it again takes very little extra 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).

B. Contributions

The contributions of this paper 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 paper is organized as follows. Section II discusses related work. Background knowledge is reviewed in Section III. Section IV presents our security metric approaches. A case study and discussions are provided in Section V. Finally, concluding remarks and future research directions are given in Section VI.

II. Related Works

The research on security metrics has attracted increasing attention [15]. 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 [17]. In a later work [21], 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 [1], 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 [29]. More recently, the authors in [13] 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 [30]. Another recent study of several CVSS-based vulnerability metrics shows the correlation between those metrics and the time to compromise of a system [11].

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 [20], a report on the popularity of zero day vulnerabilities among attackers [9], an empirical study on software vulnerabilities' life cycles [32], and more recently an effort on estimating the effort required for developing new exploits [33]. Another related 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 [14].

The concept of mean time to compromise (MTTC) has been studied in the past under other names, such as the Mean Effort to Security Failure (METF) [4], [27]. The current paper takes its inspiration from McQueen's work [19] in which 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. [16] 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 paper, 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 the well known CVSS metric [22] values of specific vulnerabilities, 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 [12], we use Bayesian network to avoid this limitation.

III. Preliminaries

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

A. CVSS

The Common Vulnerability Scoring System (CVSS) is a widely adopted standard [22] for assigning numerical scores to vulnerabilities for their relative severity. CVSS scores are readily available through public vulnerability databases (e.g., the NVD [26]). Briefly speaking (more details can be found in [22]), 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.

$$BS = round(((0.6 * I) + (0.4 * Expl) - 1.5) * f(I))$$
$$I = 10.41 * (1 - (1 - CI) * (1 - II) * (1 - AI))$$
$$f(I) = 0 \text{ if } I = 0,\ 1.176 \text{ otherwise}$$
$$Expl = 20 * AV * AC * AU$$

where AV, AC, AU, CI, II and AI are respectively the access vector (e.g., accessible from network), access complexity (e.g., user privilege required), authentication (e.g., no authentication needed), confidentiality impact, integrity impact and availability impact.

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

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

where E, RL and RC are respectively the exploitability (e.g., exploit code available), remediation level (e.g., official fix available), and report confidence (e.g., vulnerability confirmed by vendor).

B. 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* denoted by $R_r(e)$ (that is, $\{c \in C \mid c\ R_r\ e\}$), 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.

C. Bayesian Networks

Bayesian Networks (BN) are probabilistic graphical models used to represent knowledge about an uncertain domain[2]. 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:

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

IV. 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.

A. 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 assumes 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 sum 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

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

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, and $p(c)$ 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.

B. 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 1: Figure 2 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 2, 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 Conditional Probability Tables (CPTs) [2] on the right side of Figure 2 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

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

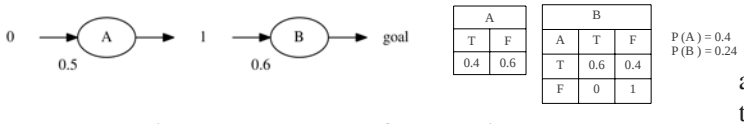

Fig. 2: An Example of Calculating MTTC

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.

1) Step 1: Probability of Exploiting Vulnerabilities Independently: We consider two cases, exploits of known vulnerabilities and zero day exploits, respectively.

a) 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 (ranging from 0 to 10) of the exploited vulnerability, denoted as $cvss(e)$.

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

b) 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 I (note this is intended to be a general guideline and a different interpretation of the relationships may be possible).

0day	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 I: Vulnerabilities status and corresponding CVSS temporal metrics (details of the metrics can be found in [22])

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 $(II = P, CI = AI = N)$ (details of the metrics can be found in [22]). We choose these values since they maximize the overall base score and a large group of vulnerabilities in the NVD share these characteristics [8]. 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) [26]. Adding temporal metrics according to the CVSS formula [22] gives the probability of successfully executing a zero day vulnerability (without considering its pre-conditions) as

$$p(e = T| \wedge_{c \in R_r(e)} \ c = T) = 0.08 \text{ if } e \text{ is zero day} \qquad (2)$$

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 [12]). 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).

Example 2: Table II 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 $rsh(0,1)$. ⊡

$ssh(0,1)$				
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 II: Examples of CPT tables for the running exa

Next, we use the Bayesian Network to find the probe that conditions are satisfied and exploits executed while t into consideration all their relationships. We denote this ability by $p(node = T)$ or $p(node)$. This step is similar method we introduced in [6] except that we now includ zero day exploits as well.

3) Step 3: Calculating $P_r(e)$: The last step is to c the probability of a successful attacker executing each e inside minimal attack sequences leading to the goal, de by $p_r(e \wedge c)$ (or $p_r(e)$) for each exploit e. To find the $p_r(e)$, we perform a backward traversal of the graph, st from the goal condition. We estimate the ratio of succ attackers that have arrived at the current node from each parents. Different assumptions may be made for this pu as demonstrated in the following example.

Example 3: Figure 3 shows an 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 calculate (details omitted due to space limitations) 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 (details omitted due to space limitations) 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 choose to do. For example (those cases are not intended to be exhaustive),

1) if we assume attackers always prefer to exploit the easiest vulnerability (that they are able to exploit), 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 (proportional to the scores). The time to compromise is equal to

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

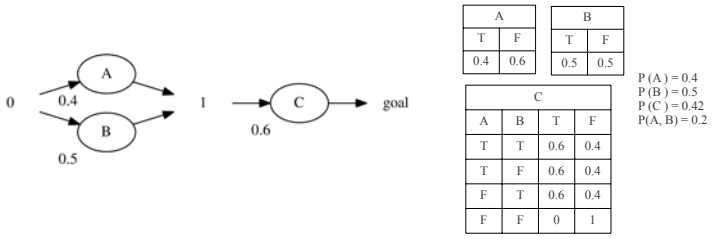

Fig. 3: An Example of Calculating MTTC

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.

C. 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.

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.

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:
$\mathcal{PS} = \{U : U \subseteq \{e_i | c \in R_i(e_i)\}\}$
foreach *set* $s \in \mathcal{PS}$ **do**
  $p(s) = p(s = T \mid \forall u \in \mathcal{PS}$ st $s \subset u,\ u = F)$
end
$p_r(e_i) = p(\{e_i\})$;
foreach *set* $s \in \mathcal{PS}$ *st* $(e_k \in s) \wedge (|s| > 1)$ **do**
  if *attackers choose easiest vulnerability* **then**
    $p_r(e_i) = p_r(e_i) + p_r(c)$ if $p(e_i) < p(e_j)\ \forall e_j \neq e_i \in \{e_j\}$
  end
  else
    $p_r(e_i) = p_r(e_i) + \frac{p(e_i)}{\sum p(e_j)} p(s),\ e_j \in s$i
  end
end
$p_r(e_i) = p_r(e_i) * \prod p(r_i)$;
return $p_r(e_i)$

Algorithm 1: Computing $p_r(e_i)$

a) 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 [26]) or exploit databases (e.g., the Metasploit DB [31]). 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 [18] as follows. If we denote by m the total number of software entities in existence (e.g., this can be estimated using the number of software entities included in the National Software Reference Library [25]), x the total number of software entities 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 [31]), then by applying search theory, the probability that an exploit code exists can be estimated as (note here we are considering a case in which no exploit-specific information is available so the probability applies to all exploits):

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

As to the average time spent by an attacker when exploit code is available, we enhance McQueen's approach [19] 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 [26], which can certainly be further fine-tuned based on specific applications' needs).

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

b) 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

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

c) Combining Both Cases: Based on the two cases' results, the mean time to exploit a known vulnerability can be estimated as

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

Example 4: Using our method, the time to exploit the *ftp* vulnerability in Figure 1 can be estimated as:

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

$boxdot$

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 [33]. 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.

a) 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 [9], [28]. 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 [24]. 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 [18] for an estimation. If we denote by m the total number of software entities in existence, x the total number of software entities on the host being examined, and k' the total number of zero day exploits (which may be estimated from statistical information [20]), 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 [23], 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 (where 32 days is the mean time to obtain a zero day vulnerability, and the $cvss(e)$ value can be estimated similarly as in Section IV-B1).

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

b) 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 [20], 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 [33] 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)

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

c) Combining Both Cases: The time to exploit a zero day vulnerability can thus be estimated as

$$t = p_3 t_3 + p_4 t_4$$
$$t = (32 + \frac{10}{cvss(e)}) + (33 + 4.8\frac{10}{cvss(e)})e^{-xk'/m}\ days \quad (4)$$

Example 5: Using our method, the time to exploit the *ssh* vulnerability on host 1 in Figure 1 can be estimated as:

$$t(ssh) = (32+\frac{10}{0.6})+(33+4.8\frac{10}{0.6})e^{-2*491/7083} = 140.5\ days$$

⊡

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

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

V. Case Study and Simulation

In this section, we first demonstrate how our metrics framework may be applied through two case studies. We then present a simulation result on the performance of the methods.

A. Case Study

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 $t_1 = 10/BaseScore$ (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 [26]. 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 III 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 III: Results

Next, we look at how our metric can be used to compare different network configurations (as shown in Figure 4). 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 4a shows a simple network consisting of a target behind a firewall running three services. Figure 4a 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 4b 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 are now denied. Figure 4c 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.

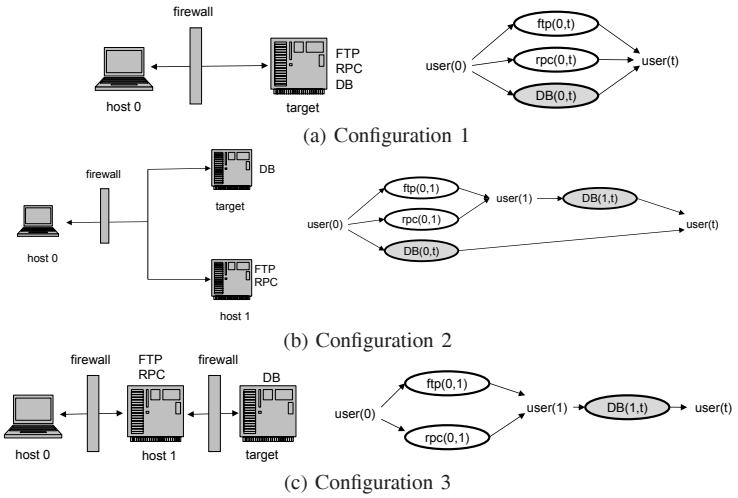

Fig. 4: An Example of Different Configurations

B. Simulation

The simulation is conducted using the Python Language and libraries including the OpenBayes[7] and Pygraphviz[10]. To render the graphs, we use GraphViz visualization package[5]. The experiments were performed inside an Intel Core I7 computer with 8Gb of RAM. The computer is running Ubuntu 12.04 LTS. The attack graphs were generated using the python programs. First, a seed graph with 20 nodes obtained from real world attack graphs is obtained. Then, conditions and exploits were randomly added to grow the seed graph to a desired size.

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 5 presents the results of our experiments. The curves represent time taken to perform the activities named in the legend. 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.

VI. Conclusion and Future Work

In this paper, 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 our 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. Our future work will look at other applications of the metrics, and will also conduct simulations to compare the proposed

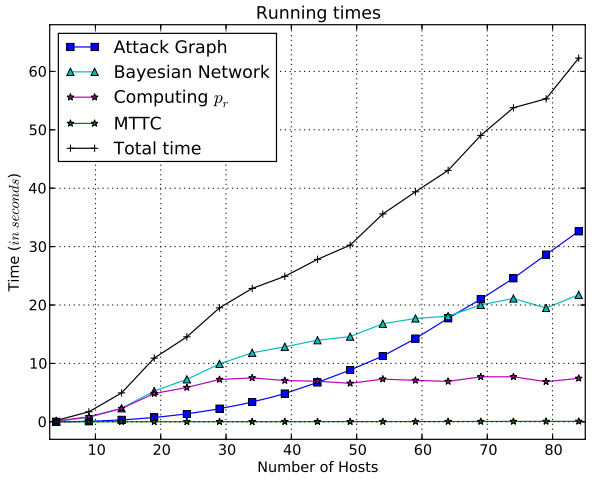

Fig. 5: Simulation Result

metrics to other existing approaches to further demonstrate the advantages of our approach.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable comments. This work is supported in part by the National Institutes of Standard and Technology under grant number 70NANB12H236, Army Research Office under MURI award number W911NF-09-1-0525 and DURIP Award W911NF-11-1-0340, by the Office of Naval Research under award number N000141210461, and by Natural Sciences and Engineering Research Council of Canada Discovery Grant.

REFERENCES

[1] BALZAROTTI, D., MONGA, M., AND SICARI, S. Assessing the risk of using vulnerable components. In *Proceedings of the 1st ACM QoP* (2005).

[2] BEN-GAL, I. Bayesian networks. http://www.eng.tau.ac.il/~bengal/BN.pdf.

[3] CORPORATION, T. M. Common vulnerabilities and exposures (cve). http://cve.mitre.org/.

[4] DACIER, M., DESWARTE, Y., AND KAÂNICHE, M. Information systems security. Chapman & Hall, Ltd., London, UK, UK, 1996, ch. Models and tools for quantitative assessment of operational security, pp. 177–186.

[5] ELLSON, J., GANSNER, E., KOUTSOFIOS, L., NORTH, S., WOODHULL, G., DESCRIPTION, S., AND TECHNOLOGIES, L. Graphviz open source graph drawing tools. In *Lecture Notes in Computer Science* (2001), Springer-Verlag, pp. 483–484.

[6] FRIGAULT, M., WANG, L., SINGHAL, A., AND JAJODIA, S. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection* (New York, NY, USA, 2008), QoP '08, ACM, pp. 23–30.

[7] GAITANIS, K., AND COHEN, E. Openbayes 0.1.0. http://pypi.python.org/pypi/OpenBayes/0.1.0.

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

[9] GREENBERG, A. Shopping for zero-days: A price list for hackers' secret software exploits. *Forbes* (23 March 2012).

[10] HAGBERG, A., SCHULT, D., AND SWART, P. pygraphviz. http://networkx.lanl.gov/pygraphviz/.

[11] HOLM, H., EKSTEDT, M., AND ANDERSSON, D. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable Secure Computing 9*, 6 (Nov. 2012), 825–837.

[12] HOMER, J., OU, X., AND SCHMIDT, D. A sound and practical approach to quantifying security risk in enterprise networks. *Technical report, Kansas State University, Computing and Information Sciences Department* (2009).

[13] IDIKA, N., AND BHARGAVA, B. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on Dependable and Secure Computing 9* (2012), 75–85.

[14] INGOLS, K., CHU, M., LIPPMANN, R., WEBSTER, S., AND BOYER, S. Modeling modern network attacks and countermeasures using attack graphs. In *Proceedings of ACSAC'09* (2009), pp. 117–126.

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

[16] LEVERSAGE, D., AND JAMES, E. Estimating a system's mean time-to-compromise. *IEEE Security & Privacy 6*, 1 (jan.-feb. 2008), 52 –60.

[17] LI, W., AND VAUGHN, R. 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* (Washington, DC, USA, 2006), CCGRID '06, IEEE Computer Society, pp. 26–.

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

[19] MCQUEEN, M., BOYER, W., FLYNN, M., AND BEITEL, G. Time-to-compromise model for cyber risk reduction estimation. In *Quality of Protection*, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds., vol. 23 of *Advances in Information Security*. Springer US, 2006, pp. 49–64. 10.1007/978-0-387-36584-8_5.

[20] MCQUEEN, M., MCQUEEN, T., BOYER, W., AND CHAFFIN, M. Empirical estimates and observations of 0day vulnerabilities. *Hawaii International Conference on System Sciences 0* (2009), 1–12.

[21] MEHTA, V., BARTZIS, C., ZHU, H., CLARKE, E., AND WING, J. Ranking attack graphs. In *Recent Advances in Intrusion Detection 2006* (2006).

[22] MELL, P., SCARFONE, K., AND ROMANOSKY, S. Common vulnerability scoring system. *IEEE Security & Privacy 4*, 6 (2006), 85–89.

[23] MILLER, C. 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).

[24] NICCOLAI, J. 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/.

[25] OF STANDARDS, N. I., AND TECHNOLOGY. National software reference library. http://www.nsrl.nist.gov.

[26] OF STANDARDS, N. I., AND TECHNOLOGY. National vulnerability database version 2.2. http://nvd.nist.gov/.

[27] ORTALO, R., DESWARTE, Y., AND KAÂNICHE, M. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Trans. Softw. Eng. 25*, 5 (Sept. 1999), 633–650.

[28] OZMENT, A. Bug auctions: Vulnerability markets reconsidered. In *Third Workshop on the Economics of Information Security* (2004).

[29] PAMULA, J., JAJODIA, S., AMMANN, P., AND SWARUP, V. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the ACM Quality of Protection* (2006), pp. 31–38.

[30] POOLSAPPASIT, N., DEWRI, R., AND RAY, I. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable Secure Computing 9*, 1 (Jan. 2012), 61–74.

[31] RAPID7. Metasploit project. http://www.metasploit.com/.

[32] SHAHZAD, M., SHAFIQ, M., AND LIU, A. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)* (2012).

[33] SOMMESTAD, T., HOLM, H., AND EKSTEDT, M. Effort estimates for vulnerability discovery projects. In *Proceedings of the 2012 45th Hawaii International Conference on System Sciences* (Washington, DC, USA, 2012), HICSS '12, IEEE Computer Society, pp. 5564–5573.

[34] WANG, L., JAJODIA, S., SINGHAL, A., AND NOEL, S. 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).