Network Attack Surface: Lifting the Concept of Attack Surface to the Network Level for Evaluating Networks' Resilience against Zero-Day Attacks

Mengyuan Zhang, Lingyu Wang, *Member, IEEE,* Sushil Jajodia *Fellow, IEEE,* and Anoop Singhal *Senior Member, IEEE,*

***Abstract*—The concept of attack surface has seen many applications in various domains, e.g., software security, cloud security, mobile device security, Moving Target Defense (MTD), etc. However, in contrast to the original attack surface metric, which is formally and quantitatively defined for a software, most of the applications at higher abstraction levels, such as the network level, are limited to an intuitive and qualitative notion, losing the modeling power of the original concept. In this paper, we lift the attack surface concept to the network level as a formal security metric for evaluating the resilience of networks against zero day attacks. Specifically, we first develop novel models for aggregating the attack surface of different network resources. We then design heuristic algorithms to estimate the network attack surface while reducing the effort spent on calculating attack surface for individual resources. Finally, the proposed methods are evaluated through experiments.**

I. Introduction

For mission critical computer networks (e.g., those that play the role of a nerve system in critical infrastructures, governmental or military systems, and cloud data centers), the security administrators usually need to look beyond traditional security mechanisms, such as firewalls and IDSs. Their worry over the prospect of Advanced Persistent Threat (APT) and hidden malware usually drive them to understand the resilience of their networks against potential zero day attacks (i.e., attacks that involve exploiting previously unknown vulnerabilities). However, while there exist standards and metrics for measuring the relative severity of known vulnerabilities (e.g., CVSS [1]), the task is more challenging for unknown vulnerabilities, which are sometimes believed to be unmeasurable [2].

To that end, a promising solution is the *attack surface* concept [3], which is originally proposed for measuring a software's degree of security exposure along three dimensions, namely, entry and exit points (i.e., methods calling I/O functions), channels (e.g., TCP and UDP), and untrusted data items (e.g., registry entries or configuration files). Since attack surface relies on such intrinsic properties of a software, which are independent of external factors (e.g., the disclosure of vulnerabilities or availability of exploits), it naturally covers both known and unknown vulnerabilities [3] and becomes a good candidate for modeling the threat of zero day attacks.

Evidently, in addition to software security, the concept of attack surface has also seen many applications in other emerging domains, e.g., cloud security [4], mobile device security [5], automotive security [6], Moving Target Defense (MTD) [7], etc. (a detailed review of related work is provided in Section VI). However, in contrast to the original attack surface metric, which is formally and quantitatively defined for a single software, most of the applications at higher abstraction levels (e.g., the network level) are limited to an intuitive and qualitative notion [8]. Adopting such an imprecise notion unavoidably loses most of the original concept's power in formally and quantitatively reasoning about the relative likelihood of different systems to contain vulnerabilities.

In this paper, we address this issue by lifting the original attack surface concept to the network level as a formally defined security metric, namely, *network attack surface*, for evaluating the resilience of networks against potential zero day attacks. There are two main challenges in lifting attack surface to the network level. First, the attack surface model relies on addition for aggregating scores, which is incompatible with the causal relationships among different resources inside a network. Second, there exists a challenge that the only way to avoid the costly calculation of attack surface is to perform that calculation. We devise models and heuristic algorithms to address those challenges, and we confirm the effectiveness of the proposed solutions through experiments (e.g., our algorithms has an error rate of 0.05 when we only calculate the attack surface for 20% of the resources).

The main contribution of this work is twofold. First, to the best of our knowledge, this is the first work to lift the attack surface concept to the network level as a formally defined security metric. It addresses a key limitation of our previous works [9], [10], [11], [12], i.e., different resources are assumed to be equally likely to include unknown vulnerabilities. Second, our simulation results show that the proposed algorithms can produce relatively accurate results with a significant reduction in the costly calculation of attack surface, paving the way for practical applications at the network level.

The rest of the paper is organized as follows. Section II defines the formal models, and Section III designs the heuristic algorithms. Section IV discusses how to instantiate the models, and Section V presents experimental results. Section VI reviews related work, and Section VII concludes the paper.

M. Zhang and L. Wang are with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montreal, QC H3G 1M8, Canada. E-mail: {mengy_zh,wang}@ciise.concordia.ca.

S. Jajodia is with the Center for Secure Information Systems, George Mason University, Fairfax, VA 22030, USA.

A. Singhal is with the Computer Security Division, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA.

II. The Network Attack Surface Model

In this section, we first build intuitions through a motivating example and describe our assumptions in Section II-A. Sections II-B and II-C then convert the attack surface into attack probabilities. Finally, Section II-D aggregates the attack probabilities of different network resources into a single measure of network attack surface.

A. Motivating Example and Assumptions

First, we illustrate the main challenges through a motivating example. Figure 1 depicts the network topology of a fictitious campus network [13]. We assume the *External Firewall* allows all outbound connection requests but blocks all inbound requests to the *Mail Server* (h2) and *File Server* (h3), including those from the *Classroom Computers* (h25); the *Internal Firewall* allows all outbound requests from the *Admin Host* h4 but blocks all inbound requests except those from h2. We also assume our main concern is protecting h4. Based on such assumptions, we can see that, an attacker at h0 can potentially follow an *attack path*, e.g., $h1 \rightarrow h2 \rightarrow h4$, to compromise h4. Keeping this in mind, we consider the question: *How could we apply the attack surface concept [3], which is only defined for each individual software to such a network to measure its overall security (e.g., in terms of h4)*?

Two obvious solutions are to directly apply the metric either by regarding the whole network as a single software system, or by first applying it to each resource separately, and then adding the results together. Since the addition operation is associative, both solutions actually yield the same result, i.e., the total numbers of methods, channels, and untrusted data items, respectively. The main problem here is that such an addition operation is incompatible with the causal relationships between network resources, which can be either conjunctive or disjunctive. For example, in Figure 1, while it makes sense to add up the attack surface of all the Classroom Computers (i.e., a larger number of such computers means the network is more *exposed* to attacks), applying this along an attack path, e.g., $h1 \rightarrow h2 \rightarrow h4$, is less meaningful, since it means a longer attack path, which indicates more attacking steps required from an attacker and hence more security, would yield a larger attack surface meaning less security. Therefore, our first challenge is *how to aggregate the attack surface of network resources while respecting their causal relationships*, which will be the main topic of the remainder of Section II.

The second major challenge lies in the calculation of attack surface, which is well known to be costly since identifying the source code that lies on the attack surface may require domain expertise and manual effort [3], [14]. Therefore, a natural question is whether we can reduce our effort by avoiding calculating attack surface for those resources that do not contribute to the final result. For example, in Figure 1, since our main concern is h4, we only need to calculate attack surface along the path $h1 \rightarrow h2 \rightarrow h4$, which significantly saves the effort by avoiding the calculation for the 25 *Classroom Computers*. However, the problem is not so straightforward in general. In this example, suppose we change the firewall rules such that requests are allowed to be sent from both h2 and h3 to h4 . We now have a challenge that, in order to know which path, $h1 \rightarrow h2 \rightarrow h4$ or $h1 \rightarrow h3 \rightarrow h4$, should be calculated (the criteria for selecting the path will be detailed later in this section) such that we can avoid calculating the other path, we must first calculate and compare the attack surface of both h2 and h3, which defies the purpose because by then we would have effectively calculated both attack paths. Therefore, our second challenge is *how to reduce the effort of calculating attack surface for network resources while keeping the final result sufficiently accurate*, which will be the main topic of Section III.

Assumptions: We make following assumptions in this paper. First, similar to other metrics (e.g., temperature, length, and weight), the attack probability discussed in our model is only intended as a relative measure for comparison between different software; the absolute value is less meaningful and not intended to indicate the exact probability of attacks, which is generally infeasible to obtain in practice. Second, the metric focuses on remote attacks exploiting network services and does not cover other types of threats, e.g., those caused by human errors, social engineering, infected browsers, phishing attacks, etc. (note that, since the consequence of those attacks is some resources become directly accessible to external attackers or malware, those attacks could still be covered in our model, although we do not consider them for simplicity). Third, similar to the original attack surface concept, our metric only provides a general indicator of the network's potential for vulnerabilities but provides no guarantee for such vulnerabilities to actually exist (however, we do examine the correlation between the two through experiments with real world software in Section V-A).

B. CVSS-Based Attack Probability

This section addresses the challenge that the addition operation used in attack surface is incompatible with the causal relationships between network resources, as demonstrated in Section II-A. Our main idea is to convert the attack surface of each software resource into an *attack probability* (the relative likelihood that the software contains at least one exploitable zero day vulnerability), which can then be aggregated for different resources based on their causal relationships. Since attack surface provides an indication of both the severity (represented by the weights, i.e., the access rights and privileges) and the likelihood (represented by the counts, i.e., the total numbers of methods, channels, and untrusted data items) of potential vulnerabilities [3], the conversion will take two steps as follows.

- First, for each group of methods, we explore a mapping between the attack surface and the common vulnerability scoring system (CVSS) [1] to convert the access rights and privileges of attack surface to a CVSS base score.
- Second, at the software level, we aggregate the base scores of different groups of methods into a single attack probability for the entire software.

1) Method Group-Level Conversion: First, we briefly review the concepts of attack surface and CVSS [1]. As illustrated in the first column of Table I, the CVSS defines six base

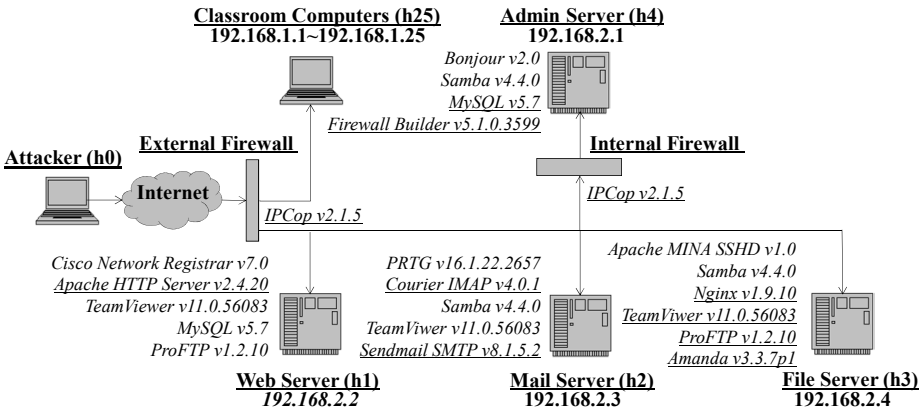


Fig. 1: The Motivating Example

metrics in two groups, and the accessibility group includes the following [1].

- Access Vector (AV): what is required to access this method; Local (L): requiring physical access to the host; Adjacent Network (A): requiring access to adjacent networks, e.g., local subnet; Network (N): remotely exploitable.
- Access Complexity (AC): the complexity of the attack required to access this method; High (H): requiring specialized access conditions, e.g., social engineering or spoofing multiple systems; Medium (M): requiring somewhat specialized access conditions, e.g., non-default configuration; Low (L): requiring no specialized access condition, e.g., default configuration.
- Authentication (Au): the type of authentication required to access this method; Multiple (M): requiring authentication two or more times; Single (S): requiring attacker to login to the system; None (N): authentication not required.

The impact group includes confidentiality impact (C), integrity impact (I), and availability impact (A) (the possible values of each metric and their corresponding numerical scores are also shown in the table) [1]. The second column of Table I shows the different access rights and privileges and their numerical values used as weights in the attack surface metric (the underlined rows will be discussed later).

Since both the accessibility group of CVSS and the access rights of attack surface represent the pre-conditions for exploiting a vulnerability, their values may be mapped together. Similarly, the impact group of CVSS and the privileges of attack surface both represent the post-conditions of exploiting a vulnerability and are mapped together. As an example. the mapping for two IMAP daemons are shown in the last column of Table I (three dimensional attack surface values have been calculated in [3]). Each CVSS vector maps to the corresponding access right or privilege in the same row in the second column.

The mapping is established based on understanding the software, including its channels and untrusted data items (consequently, we will not count those again later when we convert base scores into attack probabilities). First, in the third row, the authenticated access right is mapped to *network* for *access vector* (i.e., *AV:N*), because the UNIX socket in those software has the *local authenticated* access right, which means attackers may obtain the local authenticated access right over the network. Second, we assign *access complexity* to *medium* (i.e., *AC:M*), because the authenticated access right matches the description of the medium access complexity: "The affected configuration is non-default, and is not commonly configured (e.g., a vulnerability present when a server performs user account authentication via a specific scheme, but not present for another authentication scheme)" [1]. Finally, we assign *Authentication* to *single* (i.e., *Au:S*), because the access requires a single authenticated session in those software. Similarly, in the fifth row, the authenticated privilege is mapped to *partial confidentiality impact*, partial integrity impact, and complete availability impact (i.e., *C:P, I:P, A:C*), since the authenticated privilege implies accesses to 13 files in those software, allows modifying some system files or data, and may render the system unusable by deleting critical files.

As shown in Table I, we map all the methods of those two software to corresponding CVSS base metrics, and then calculate the overall base score according to the CVSS formula [1], as shown in Table II. The methods are divided into groups (first column) according to similar privileges (second column) and access rights (third column). The fourth and fifth columns show the total numbers of entry and exit points in each group. The next two columns show the mapped CVSS vector and the calculated base score for each group.

2) Software-Level Conversion: Now that we have calculated the base score for each group of methods, we can convert the attack surface into an *attack probability* as follows. Suppose there are totally g groups of methods in the attack surface. Let b_i and s_i $(1 \leq i \leq g)$ denote the base score and the number of methods of each group, respectively. Assume on average there will exist one zero day vulnerability for every n methods, and the probability for attackers to discover such a vulnerability is p_0[1]. In Equation 1, the base score divided by its range 10 gives the probability that a vulnerability in this group

[1]Note that, here n and p_0 are both intended as normalizing constants, since their true values are certainly impossible to obtain in practice; as long as those values stay constant across different software, the equation will yield a relative metric sufficient for comparing the exploitability of different software based on both the severity, represented by the base scores b_i, and counts, represented by the number of methods s_i, of potential zero day vulnerabilities.

CVSS (Base Metric Group)	Attack Surface (Methods)			Vectors
AV:[L:0.395,A:0.646,N:1.0]	Access Rights	anoymous	1	[AV:N,AC:L,Au:N]
AC:[H:0.35,M:0.61,L:0.71]		unauthenticated	1	[AV:N,AC:L,Au:N]
Au:[M:0.45,S:0.56,N:0.704]		authenticated	3	[AV:N,AC:M,Au:S]
		admin	4	[AV:A,AC:H,Au:M]
C:[N:0.0,P:0.275,C:0.66]	Privileges	authenticated	3	[C:P,I:P,A:C]
I:[N:0.0,P:0.275,C:0.66]		cyrus	4	[C:C,I:C,A:C]
A:[N:0.0,P:0.275,C:0.66]		root	5	[C:C,I:C,A:C]

TABLE I: Mapping Attack Surface to CVSS Base Metrics for Courier IMAP Server v4.1.0 and Cryus IMAP Server v2.2.10 (the Attack Surface Values Borrowed from [3])

Method Group	Privilege	Access Rights	DEP	DExp	Vector	Base Score	Attack Probability
Courier							
M1	root	unauthenticated	28	17	[AV:1.0,AC:0.71,Au:0.704,C:0.66,I:0.66,A:0.66]	10	0.000315
M2	root	authenticated	21	10	[AV:1.0,AC:0.61,Au:0.56,C:0.66,I:0.66,A:0.66]	8.5	0.000184
M3	authenticated	authenticated	113	28	[AV:1.0,AC:0.61,Au:0.56,C:0.275,I:0.275,A:0.66]	7.5	0.000809
Cyrus							
M1	cyrus	unauthenticated	16	17	[AV:1.0,AC:0.71,Au:0.704,C:0.66,I:0.66,A:0.66]	10	0.000132
M2	cyrus	authenticated	12	21	[AV:1.0,AC:0.61,Au:0.56,C:0.66,I:0.66,A:0.66]	8.5	0.000112
M3	cyrus	admin	13	22	[AV:0.646,AC:0.35,Au:0.45,C:0.66,I:0.66,A:0.66]	6.3	0.0000882
M4	cyrus	anonymous	12	21	[AV:1.0,AC:0.71,Au:0.704,C:0.66,I:0.66,A:0.66]	10	0.000132

TABLE II: Method Groups and Their Base Scores for Courier IMAP Server v4.1.0 and Cyrus IMAP Server v2.2.10 (the Attack Surface Values Borrowed from [3])

is exploitable; multiplying this with p_0 gives the probability that the method can be both discovered and exploited; s_i/n gives the number of vulnerabilities out of those s_i methods in this group; the equation therefore gives the probability p that the software contains at least one exploitable zero day vulnerability.

$$p = 1 - \prod_{i=1}^{g} (1 - p_0 \frac{b_i}{10})^{\frac{s_i}{n}} \quad (1)$$

Example 1. *Assuming $n = 30$ and $p_0 = 0.08$, we can calculate p for both software as follows. For Courier, $p = 1-(1-0.08*10/10)^{45/30}*(1-0.08*8.5/10)^{31/30}*(1-0.08*7.5/10)^{141/30} = 0.384$, and similarly for Cyrus, $p = 0.273$.*

C. Graph-Based Attack Probability

In Section II-B, the attack probability obtained using Equation 1 does not yet capture the relationships among different dimensions of attack surface. To address this issue, we combine different dimensions of attack surface by taking attackers' point of view. Specifically, a remote attack over the network (which is the focus of this paper, as mentioned in Section II-A) would typically involve all three dimensions, i.e., using communication channels to access and invoke methods in order to manipulate untrusted data items and fulfill the attacking goal. This observation shows that there exist causal relationships between the three dimensions of attack surface, which will be modeled in two steps as follows.

1) Method Group-level Conversion: First, we divide methods into groups based on the pair $\langle access\ right, privilege \rangle$, such that the methods in the same group require the same access right and lead to the same privilege. As an example, the first column of Table II gives the group name for each method group. We will simply use M1 in Courier to refer to the group of methods restricted by unauthenticated access right and lead to root privilege in Courier in following discussions. In group M1 of Courier, attackers only need to exploit one method out of 45 to gain the corresponding privilege. However, attackers may exploit multiple methods in one group, e.g., due to the lack of sufficient knowledge about such methods. Taking this into consideration, we define the attack likelihood of one group of methods as the probability of compromising at least one method out of the group. Suppose we have totally s_i methods in one group, and let b and p_0 denote the base score and the probability for attackers to discover one method, respectively. In Equation 2, the base score divided by its range 10 gives the probability of finding a method in a software application to be exploitable; multiplying this with p_0 gives the probability that the method can be both discovered and exploited (as mentioned before, p_0 is only intended as a normalizing constant).

$$p = 1 - (1 - p_0 * \frac{b}{10})^{s_i} \quad (2)$$

Example 2. *To compare Courier and Cyrus, we take p_0 as the ratio of choosing one method per thousand lines of the source code. The number of lines of source code for Courier and Cyrus are 138,283 and 236,321, respectively [15]. Therefore, we have $p_0 = 0.00723$ for Courier and $p_0 = 0.00423$ for Cyrus. We can calculate p for M2 for both software applications as follows. For Courier, $p = 1-(1-0.00723*\frac{8.5}{10})^{31} = 0.174$, and similarly M2 in Cyrus, $p = 0.112$.*

2) Software-Level Conversion: In order to aggregate the attack probabilities for the entire software, we first need a model of the relationships among the three dimensions of attack surface. Our model is syntactically equivalent to an *attack graph* [16] (we will therefore omit its formal definition) although our model focuses on resources inside a software instead of known vulnerabilities inside a network. As an example, Figure 2 depicts the *attack surface graph* for both Courier and Cryus based on the information given in Table III. Each square box in the figure represents a resource in attack surface (e.g., TCP, SSL, and UNIX socket, which are channels in attack surface, are represented as the connectivity for the software applications); the edges point from the pre-conditions to corresponding resources (e.g., $\langle TCP\ connection \rangle$ and

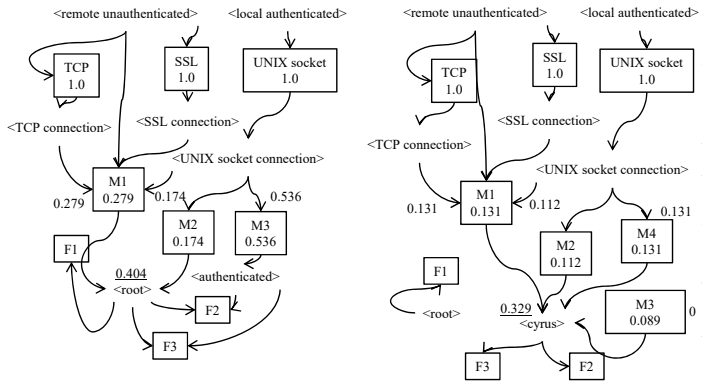


Fig. 2: The Attack Surface Graph of Courier (Left) and Cryus (Right)

$\langle remote\ unauthenticated \rangle$ to M1) or from resources to their post-conditions (e.g., M1 to $\langle root \rangle$).

Example 3. *In Figure 2 (Left), the remote unauthenticated privilege is required to establish TCP connections. After a connection is established, an attacker could invoke the method under the same privilege graph, e.g., M1. Then, the attacker could gain the root privilege after accessing M1, which provides sufficient access right to access all the file groups.*

TABLE III: Channels and Untrusted Data items [3]

Courier Channels		Untrusted Data items		
Type	Access Rights	Group	Type	Access Rights
TCP	remote unauthenticated	F1	file	root
SSL	remote unauthenticated	F2	file	authenticated
UNIX socket	local authenticated	F3	file	world
Cyrus Channels		Untrusted Data Items		
TCP	remote unauthenticated	F1	file	root
SSL	remote unauthenticated	F2	file	cyrus
UNIX socket	local authenticated	F3	file	cyrus

As shown in the attack surface graph, channels, which are modeled as the resources with initially satisfied pre-conditions (initial conditions), can be directly accessed by attackers. Methods can be invoked by attackers only if the corresponding channels are associated with an equivalent or higher privilege. For example, we consider that attackers passing from the channel UNIX socket is able to access M1 since UNIX socket has *local authenticated* privilege which is higher than the required access right of M1, *unauthenticated*). Similarly, when sending untrusted date items, the privileges gained from methods should be equivalent or higher than the access right of untrusted data items. For example, as shown in Table III, *root* is required to send data to F1 in Courier, which means M3 with *authenticated* does not have sufficient access right to send data to F1.

Based on the attack surface graph, the overall *attack probability* can be calculated using Bayesian inferences. The overall *attack probability* for courier can be calculated as 0.404 and that for Cryus as 0.329, as depicted in Figure 2.

D. Aggregating Attack Probabilities inside a Network

Now that we have converted the attack surface of each software to its attack probability, we can easily aggregate such probabilities for all the network resources into a single measure of *network attack surface*. We consider two different ways for such aggregation, the shortest path-based approach [9] and the Bayesian network (BN)-based approach [10], which reflect the worst case scenario (i.e., attackers take the shortest attack path) and the average case scenario, respectively.

To illustrate the idea, Figure 3 shows a partial *resource graph* [10] for our example[2]. Specifically, each pair in plaintext is a security-related condition, e.g., connection $\langle source, destination \rangle$ or privilege $\langle privilege, host \rangle$, and each triple inside a box is a zero day exploit $\langle resource, source, destination \rangle$. The number inside each box is the corresponding attack probability.

Example 4. *In Figure 3, for the shortest path-based approach [9], we can calculate the attack probability for the shortest path indicated by the dashed line,* $\langle IPCop, 0, F \rangle \rightarrow \langle Courier, 0, 2 \rangle \rightarrow \langle FirewallBuilder, 2, 4 \rangle$*, the probability for attackers to reach* $\langle user, 4 \rangle$ *can be calculated as* $p = 0.48 * 0.384 * 0.04 = 0.0074$ *(the attack probabilities are obtained using the method introduced in Section II-B). For the BN-based approach [10], the attack probability of each resource is regarded as the conditional probability that the corresponding resource can be exploited given that its pre-conditions are all satisfied. Bayesian inferences then indicate the probability for attackers to reach* $\langle user, 4 \rangle$ *is* $p_{goal} = 0.016$.

More formally, the following formally defines the concept of network attack surface.

Definition 1 (Network Attack Surface)**.** *Given a network with the set of resources* R*, the attack probability* $p(r)$ *as defined in Equation 1 or Equation 2 for each* $r \in R$*, the resource graph* G *and a given condition* $c_g \in G$*,*

- *let* AP *denote the collection of all attack paths in* G *ending at* c_g*, and for each* $ap \in AP$*, let* $R(ap)$ *denote the set of resources involved in* ap *and denote* $p(ap) = \prod_{r \in R(ap)} p(r)$*. We call* $max(\{p(ap) : ap \in AP\})$ *(where* $max(.)$ *returns the maximum value of a set) the worst case network attack surface w.r.t.* c_g*.*
- *let* $B = (G', \theta)$ *be a BN, where* G' *is* G *annotated with the attack probabilities and* θ *is the set of parameters of the BN, and let* C_I *be the set of conditions without parents in* G'*, we call* $p = P(c_g \mid \forall_{c \in C_I} c = True)$ *the average case network attack surface w.r.t.* c_g*.*

Discussions Our network attack surface metric addresses a key limitation of the existing k*-zero day safety metric* [9], i.e., it cannot discriminate different resources based on their relative attack probabilities (consequently the metric simply counts the number of such resources). On the other hand, although our network attack surface metric is defined as

[2]Note that, although the resource graph demonstrated here has a similar syntax as the attack surface graph discussed in the previous section, they work at different abstraction levels, i.e., the former models network-level resources and the latter models resources inside a software.

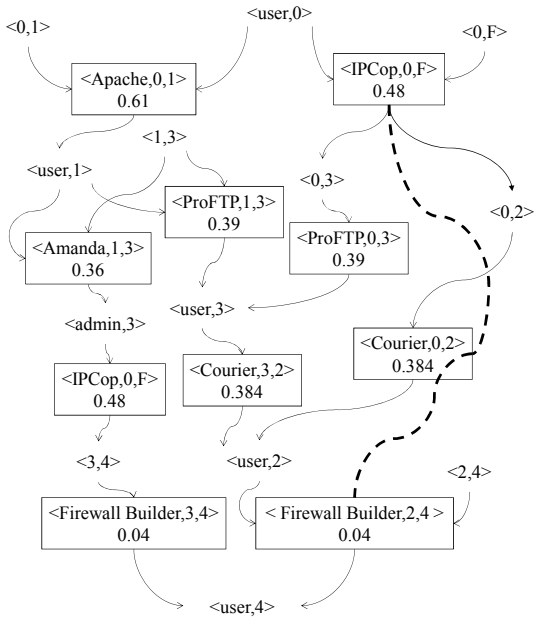


Fig. 3: The Resource Graph

probabilities, those can be easily converted into other forms for different applications. For example, given the network attack surface p as a probability, we can convert p back into the equivalent number of zero day vulnerabilities along an attack path as $\log_{0.08} p$ (here 0.08 is a nominal probability for zero day vulnerabilities based on CVSS base metrics, as described in [10]); such a simple count-based metric may be helpful for interpretation and comparison purposes (we will use this method in our algorithms and simulations). As another example, we can also convert p into the equivalent number of methods s with a given base score b, by inverting Equation 1 as: $s = n \log_{1-p_0}(1-p)$. We can therefore evaluate the network as a single software system with an attack surface composed of s methods with the base score b (which can also be mapped back to access rights and privileges if necessary).

III. Heuristic Algorithms for Calculating Network Attack Surface

In this section, we propose heuristic algorithms to reduce the effort of calculating attack surface for individual resources in evaluating the network attack surface. We state the problem in Section III-A and design heuristics algorithms in Section III-B.

A. The Problem statement

The calculation of attack surface is becoming more practical due to ongoing efforts on automating or approximating the calculation [14]. However, calculating the attack surface of a software can still be costly [3], [14] mostly due to the manual effort and expertise required for analyzing the source code of the software in order to extract both the counts (e.g., the total number of methods calling I/O functions) and weights (e.g., the access rights and privileges). Moreover, even with automated techniques, the calculation will likely remain a costly process due to the ever increasing size of modern software. For example, all the software mentioned in our running example in Figure 1 have a large number of lines of source

Procedure *Mpath-Topo_Heuristic*
Input: Resource graph G, parameter M, and budget N
Output: a sequence of resources P
Method:
1. **Let** $P = \phi$ be a sequence of resources
2. **Let** MS be the sequence of M paths with the least numbers of exploits in G, with the paths sorted ascendingly based on such numbers, and the resources inside each path topologically sorted
3. **Let** $T = G \setminus MS$, topologically sorted
4. **While** $N > 0$
5. **If**$|\ MS\ | > 0$
6. **Append** the first resource r in MS to P
7. **Remove** r from MS
8. **Else If** $|\ T\ | > 0$
9. **Append** the first resource r in T to P
10. **Remove** r from T
11. **Let** N= N-1
12. **Return** P

Procedure *Keynode_Heuristic*
Input: Resource graph G, $p_0, p_1 \in [0, 1]$, and budget N
Output: a sequence of resources P
Method:
1. **Let** $P = \phi$ be a sequence of resources
2. **Let** $KN = \phi$ be a sequence of resources
3. **Let** p be the network attack surface calculated based on assigning p_0 to all the resources in G
4. **For** each resource r in G
5. **Calculate** p again on G with p_1 assigned to r
6. **If** p changes
7. **Add** r to KN
8. **Sort** KN based on topological order
9. **While** $N > 0$
10. **If** $|\ KN\ | > 0$
11. **Append** the first resource r in KN to P
12. **Remove** r from KN
13. **Else If** $|\ G \setminus KN\ | > 0$
14. **Append** the first resource r to P
15. **Remove** r from G
16. **Let** N= N-1
17. **Return** P

Fig. 4: The Heuristic Algorithms

code: Nginx (171,567), IPCop (271,645), Apache(1,800,402), MySql (2,731,107), Linux Kernel (18,766,825), and Google Chrome (14,137,145).

Therefore, we investigate the problem of evaluating the network attack surface metric proposed in previous section while reducing the effort of calculating the attack surface of individual resources. Clearly, there will be a tradeoff between the cost (i.e., the percentage of network resources whose attack surface is calculated), and the error in the calculated network attack surface result (due to the estimation of attack surface for the non-calculated network resources). Specifically, suppose the true value of the network attack surface is p_{true} and the calculated value is p_{cal} (all values described in this section will be count-based, as described at the end of Section II-D), we would like to minimize the error $\frac{|p_{true}-p_{cal}|}{p_{true}}$ while calculating the attack surface for no more than a given percentage of resources (the budget).

Note that, although the above may seem to be a standard optimization problem, this is not really the case since the objective function $\frac{|p_{true}-p_{cal}|}{p_{true}}$ contains an unknown value p_{true} that can only be obtained by calculating the attack surface for all the resources (which defies the very purpose of reducing the cost). Also, since the problem of finding the shortest path is already NP-hard [9], which is a special case of our problem (when the budget is unlimited), the latter is also intractable. Therefore, we study heuristic algorithms in the coming section.

B. The Heuristic Algorithms

Our main observation is that, since we can only calculate a certain percentage of resources under a given budget, what determines the error is the order of calculation among all resources, e.g., the error would be minimized if we first calculate all the resources appearing on the shortest attack path (however, recall that the shortest path is unknown before the calculation, as demonstrated in Section II-A). Therefore, we first consider several simple heuristics for choosing resources in the right order, e.g., by exploring the structural properties of a resource graph. We will then combine those heuristics into algorithms and evaluate their performance through simulations later in Section V. We will focus on the worst case network attack surface, as given in Definition 1, while leaving the average case network attack surface to future work.

a) Random Choose: The most obvious solution is to simply choose resources in a completely random fashion, namely, the *random choose* heuristic. This provides a baseline for comparison with other heuristic algorithms. For example, in Figure 3, if our budget is to calculate the attack surface of at most two resources, then among the $\binom{6}{2} = 15$ possible choices, the worst result is $p = 0.46$ with an error rate of 0.76, whereas the best result is $p = 1.73$ with error rate 0.109. Clearly, this heuristic may lead to a solution that is far from optimal.

b) Frequency Choose: The key idea is that, since the same resource may appear on multiple hosts inside a network, calculating the attack surface for the most frequently seen resources will provide the most information with the same cost. For example, in Figure 3, *IPCop*, *Firewall Builder*, *Courier* and *ProFTP* all appear twice among totally 10 exploits. Therefore, if our budget is two, then calculating any two of them will unveil $4/10$ of the exploits (the best result is $p = 1.73$ with an error rate of 0.109 by calculating *Firewall Builder* and *Courier*, and the worst result is $p = 0.60$ with an error rate 0.69 by calculating *IPCop* and *ProFTP*).

c) Topological Order: The idea is that, since the nodes closer to the first and last nodes of a resource graph (in the sense of a topological sorting) tend to be shared among more attack paths (e.g., the last two exploits are shared by all paths in Figure 3), it may help to choose resources based on a topological order among the exploits. We consider both the *topological order* and the *reversed topological order* heuristics, which choose resources in the same, and opposite order as topological sorting, respectively. For example, in Figure 3, suppose our budget is two, the topological order heuristic will choose *Apache* and *IPCop* (the result is $p = 0.60$ with error rate 0.69) while the reversed topological order will choose *Firewall Builder* and *Courier* (the result is $p = 1.73$ with error rate 0.109).

d) Shortest Path: This heuristic starts the calculation with the path with the least number of exploits, which, although not always the right path in terms of the final result, may serve as a good starting point. For example, in Figure 3, if our budget is two, then the shortest path heuristic will choose *Courier* and *Firewall Builder* on the dashed line path (the result is $p = 1.73$ with error rate 0.109). In this particular example, this path happens to be the right path for the final result, so a larger budget will produce more accurate result.

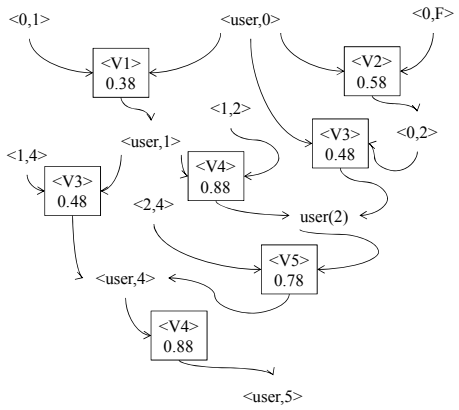


Fig. 5: An Example of Applying Mpath-Topo and Keynode Heuristic Algorithms

Although the above simple heuristics may not produce good results when each of them is used alone, combining these may lead to algorithms with good performance. The following presents two such algorithms, whose performance will be confirmed through simulations in Section V.

e) Mpath-Topo Heuristic Algorithm: This algorithm combines the topological order and shortest path heuristics as follows. First, we choose M (an integer parameter) shortest paths ranked by the number of unique exploits. We next apply the topological order heuristic to sort resources along each path, as well as those not on those paths. The algorithm is more clearly depicted on the left-hand side of Figure 4. Lines 1 to 3 initialize the algorithm and the main loop between lines 4 and 11 chooses resources for calculation as described above.

Example 5. *In Figure 5, we have three paths with five distinct exploits; assuming* $M = 2$ *and* $N = 2$*, we have* $MS = \{V1, V4, V3, V5\}$ *and* $P = \{V1, V4\}$*; the final result is* $p = 0.51$ *with error rate* 0.04*.*

f) Keynode Heuristic Algorithm: This heuristic algorithm is based on the idea that a resource is more important for determining the final value p of network attack surface if changing its value results in significant changes, e.g., a change in the path selected for calculating the final result, or a change in the currently calculated result of p. We combine this heuristic with the topological order heuristic to form the algorithm depicted on the right-hand side of Figure 4 (here we only show the change in p, which can be replaced with the change in the optimal path, and we will evaluate both algorithms in the coming section).

Example 6. *Suppose we choose* $p_0 = 0.08$ *and* $p_1 = 1$*. In Figure 5, we initially calculate* $p = 5.12 * 10^{-4}$ *and then calculate* p *again by assigning* p_1 *to each resource, e.g., changing* V1 *from* p_0 *to* p_1 *leads to* $p = 0.0064$ *so* V1 *is a key node. Similarly, we can obtain the key node sequence as* $KN = \{V1, V4, V3, V5\}$*. If our budget* $N = 2$*, then* $V1$ *and* $V4$ *will be chosen and the result is* $p = 0.51$ *with error rate* 0.04*.*

IV. Instantiating The Network Attack Surface Metric

This section provides a case study based on 34 real world software and discusses various practical issues in instantiating the proposed network attack surface metric.

A. Case Study

To demonstrate how to apply the proposed metric, we revisit our motivating example shown in Figure 1. The following three stages correspond to the models introduced in Sections II-B, II-C, and II-D, respectively.

1) The CVSS-Based Attack Probability: The information for instantiating the CVSS-Based attack probability listed in Table IV are collected as follows.

- Attack surface: All the 34 software applications we have analyzed are based on C or C++ language. The methods that call I/O functions (from standard C library [17]) need to be identified as the entry/exit points in attack surface [3]. To this end, we have implemented a script to automatically identify methods from the call graphs which are generated from *cflow* [18] starting from the *main* function. The information about channels is gathered from the application documentations and manually verified. The connection functions and methods sometimes can also be found in the developing documentations, e.g., Amanda can be connected to in four different ways, namely UDP, TCP, RSH and SSH [19]. For simplicity, we only consider one type of untrusted data items in our case study, which is file, so all the exit points related to files are captured as the modification to files.
- Access right and privileges: We annotate source code to identify privilege-related functions. For example, in Amanda, function $access\_init$ is used to authenticate user access rights from *unauthenticated* to *authenticated*; therefore, the methods appearing before this function has *unauthenticated* access right and those appearing afterwards have *authenticated*. Also, the function set_root_privs is used to escalate the privileges, which implies the methods invoked afterwards have root privilege. Default privilege-related functions [20], such as setreuid, seteuid, setuid, setfsuid and suid, are also annotated in source code.
- Mapping table: With the information collected from previous steps, it is easy to map attack surface to CVSS base metrics, as already detailed in Section II-B.

2) The Graph-Based Attack Probability: To instantiate the graph-based attack probabilities, we collect the following information.

- p_0: Different measurements can be used for the size of software applications, e.g., the number of lines, the number of functions, or the number of files in the source code. Our case study is based on the total number of functions in a software, which is obtained from the call graph. For example, the Firewall Builder has 552 functions, and Amanda has 34,768 functions.
- Goal condition: We use the root privilege (or maximum privilege if root is not applicable) as goal conditions in our case study.

3) The Network Attack Surface: Once the attack probabilities for individual resources are calculated, as shown in Table IV, we can instantiate the network attack surface metric by collecting following additional information.

- Connectivity: This is obtained from the network topology, as shown in Figure 1.
- Security conditions: The access rights for each applications are used as pre-conditions, and the privileges are used as post-conditions. For example, Amanda could lead to root privilege [19], whereas Firewall Builder can only lead to authenticated privilege [21]. In addition, by studying the existing vulnerabilities of those applications, we obtain other security-related conditions. For example, Apache has a vulnerability (CVE-2016-1240) allowing local users to gain root privilege.
- Critical assets: In our study, we consider $\langle user, 4\rangle$ as the critical asset (system administrators can choose critical assets based on their priority).

The results of our case study have already been discussed in Section II. The lessons learned will be summarized next as general discussions for instantiating the metric.

B. Discussions on Instantiating the Proposed Metric

We discuss practical issues in instantiating the network attack surface metric in the following. We focus on open source projects in this section, and we will discuss how to estimate the attack surface of closed source software applications and study the impact of non-calculable software applications in Section V.

CVSS-Based attack probability: As demonstrated in our case study, to instantiate the CVSS-Based attack probability, the key challenge is to collect information about each dimension (channels, methods, and untrusted data items) of the attack surface, the access right and privileges, and the mapping between attack surface and CVSS base metrics. First, to calculate the attack surface, existing tools, e.g., *cflow* [18], can be used to generate call graphs for source code written in C language. Automated scripts can then be developed to identify the entry/exit points as the functions that call input/output functions. Channels and untrusted data items can be identified from documentations or observed at runtime [3]. Second, the privileges for the three dimensions can be identified based on a set of *uid-setting* system calls which associate with change of privileges [20]. The access right requires the study of authentication functions in source code, e.g., the methods can be invoked only after user authentications should be considered as *authenticated* access rights [3]. Finally, establishing the mapping with CVSS vectors requires domain experts to assign numeric values based on the relationships between attack surface results and CVSS base metrics. The aggregated attack surface base score can be calculated using the CVSS base score calculator [22].

Graph-Based Attack Probability: To instantiate the graph-based attack probability, we need to collect additional information as follows. First, we need to identify the size of each

Method Group	Privilege	Access Rights	Count	Vector	Base Score	Attack Probability
Amanda						
M1	unauthenticated	unauthenticated	834	[AV:1.0,AC:0.71,Au:0.704,C:0,I:0,A:0]	0	0
M2	root	unauthenticated	672	[AV:1.0,AC:0.71,Au:0.704,C:0.66,I:0.66,A:0.66]	10	0.0191
M3	authenticated	authenticated	1953	[AV:1.0,AC:0.61,Au:0.56,C:0.275,I:0.275,A:0.66]	7.5	0.0415
M4	root	authenticated	297	[AV:1.0,AC:0.61,Au:0.56,C:0.66,I:0.66,A:0.66]	8.5	0.00723
Firewall Builder						
M1	unauthenticated	unauthenticated	46	[AV:1.0,AC:0.71,Au:0.704,C:0,I:0,A:0]	10	0.082
M2	authenticated	authenticated	28	[AV:1.0,AC:0.61,Au:0.56,C:0.275,I:0.275,A:0.66]	7.5	0.037

TABLE IV: Method Groups and Their Base Scores for Amanda and Firewall Builder

TABLE V: Amanda's Channels and Untrusted Data items

Amanda Channels			Untrusted Data items		
Type	Access Rights	Count	Type	Access Rights	Count
TCP	remote unauthenticated	2	file	root	27
SSL	remote unauthenticated	2	file	authenticated	6
RSH	remote authenticated	1	file	unauthenticated	27
SSH	remote unauthenticated	1			
TCP	authenticated	2			
UDP	authenticated	2			
Firewall Builder Channels			Untrusted Data Items		
TCP	remote unauthenticated	2	file	authenticated	22
UDP	remote unauthenticated	2			
IP	local authenticated	1			

software, e.g., in the number of lines of source code or the number of functions. It is easy to find such information about open source projects, e.g., through the Open Hub [15], or to calculate it using simple scripts. Second, the goal condition can be identified by examining which privileges can be obtained by exploiting the software.

Network Attack Surface: To instantiate the network attack surface metric, we need to collect following additional information. The connectivity information can be obtained either from existing network topology or using network scanners, e.g., Nessus [23]. To identify security conditions required for accessing hosts or resources, the configuration of firewalls and hosts, and the local policies, e.g., authentication, may needed to be examined. Some security conditions associated with the resources can also be derived during the instantiation of attack probabilities of individual software. Finally, critical assets can be assigned based on organization's specific needs and priority.

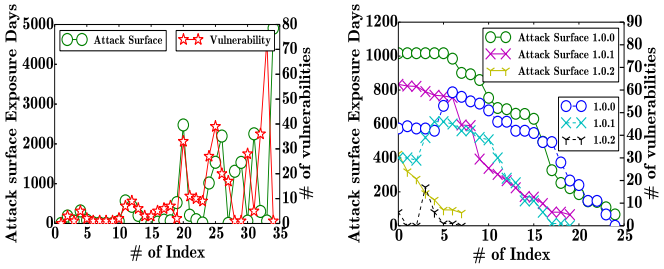
V. Experimental Results

In this section, we first examine the correlation between our models introduced in Section II and the vulnerabilities of real world software. We then conduct simulations to evaluate the performance of our heuristic algorithms proposed in Section III.

A. Correlation between Attack Surface and Vulnerabilities

Since our model for converting attack surface to attack probability in Section II is based on the hypothesis that attack surface reflects a software's likelihood of having vulnerabilities, we investigate this correlation by conducting experiments with real software. We examine the correlation both for different software and for different versions of the same software.

First, we examine 34 popular software applications and their correlation results are presented in Figure 6(a). The main criteria in choosing those software applications are as follows. First, we need to ensure their attack surface can be calculated with reasonable effort (e.g., written in C or C++ and with source codes of reasonable sizes to facilitate the manual analysis required for calculating attack surface). Second, we only choose software applications with existing vulnerabilities listed in the NVD vulnerability database to facilitate our experiments. The name of each software can be found in the Appendix based on its index. We manually study the source code of each software in order to calculate the attack surface, and subsequently convert the result into attack probability using the method mentioned in Section II-B. In Figure 6(a), the left y-axis and the green line with round markers show the attack surface (converted to attack probability) multiplied by the days of exposure (i.e., the number of days since the software was released) of each software since vulnerabilities take time to be discovered even though the attack surface of the software remains the same over time. The right y-axis and the red line with star markers show the number of vulnerabilities found for the same software in NVD [24].



(a) (b)

Fig. 6: Correlation between Attack Surface and the Number of Vulnerabilities for Different Software (a) and Different Versions of OpenSSL (b)

Results and Implications: From the results, we can see that there is a positive correlation between the number of vulnerabilities and the attack surface multiplied by exposure days for most of the software (specifically, 25 out of 34). The correlation is unclear for the last few software (after index number 25). We believe the reason lies in other related factors affecting vulnerability discovery, e.g., the market share of a software, popularity of a software among attackers, and the security expertise level of typical users of a software. For example, index 33 is *freetype*, a popular software development library used for rendering font-related operations, which is widely used by modern video games, Opera for Wii, and many other projects [25]. Such a widely used software is usually more attractive for attackers to discover vulnerabilities, and hence becomes an outlier in our results. As another example, index 34 is *Amanda*, a network-based backup system, which has only one vulnerability, even though its attack surface multiplied by exposure days is relative large. We believe the reason could be that such a backup system is usually hosted in enterprise networks and operated by administrators with more

security expertise and awareness, which may have rendered the software less attractive to attackers.

Second, we examine 53 different versions of OpenSSL along 3 version branches, 1.1.0, 1.0.1, and 1.0.2, respectively, and the results are presented in Figure 6(b). The study of different versions of the same software reduces the influence of aforementioned unrelated factors in discovering the vulnerabilities (e.g., market share). The index indicates the version numbers in chronologically order. From the results, we can see that the number of vulnerabilities has a similar trend with the attack surface multiplied by exposure days for all three branches. The branch with larger values for attack surface also has more vulnerabilities. For all three branches, we can see the maximum number of vulnerabilities always appears somewhere in the middle of the branch likely because, with a major change of version branch, it takes time for user adoption and also for attackers to change the focus.

The above experiments, although are still of a limited scale, show a promising result supporting our hypothesis that there is a positive correlation between the attack surface and the number of vulnerabilities. Our future work will expand the scope and scale of the experiments.

B. Performance of Heuristic Algorithms

In this section, we study the performance of our proposed heuristic algorithms in general. Since there currently does not exist any publicly available dataset of resource graphs, we generate synthetic resource graphs by starting from small but realistic seed graphs like the one shown in Fig. 3 and then injecting random nodes and edges in a random but realistic fashion (e.g., each exploit can only have a few pre- and post-conditions). All the results are collected using a computer equipped with a 3.0 GHz CPU and 8GB RAM in the Python environment under Ubuntu 14.04 LTS.

The objective of the first two simulations is to evaluate the error rate of our heuristics (presented in Section III-B). The error rate is defined in the same way as in Section II-D. The cost is defined as the percentage of resources whose attack surface is calculated, and denoted as α. The reason we choose the percentage of resources instead of the absolute number is that evaluating a larger network naturally implies a larger budget will be required.

Figure 7(a) shows the error vs. the percentage of calculated resources (α) for simple heuristics and Figure 7(b) shows the same for the heuristic algorithms. The y-axis is shown in reversed scale in both figures to show the increasing accuracy for a larger α. Figure 7(c) depicts the processing time of the algorithms. In all simulations, for each configuration, we repeat 500 times to obtain the average results.

Results and Implications: From Figure 7 (a), we have following observations. First of all, with the increase of α, the error generally decreases as expected (e.g., $\alpha = 1$ means we calculate all the resources). The green line with round markers is the baseline for comparison, which represents the results of the random choose heuristic. The error of this heuristic reduces almost linearly in both simulations. The frequency choose heuristic represented by the red line with vertical markers has the worst error among all the heuristics. The reason is that, the repetition of a resource does not necessarily mean the importance of this resource in determining the final result. The blue line with square and purple line with star represents the reversed topological order heuristic and the topological order heuristic, respectively. Both heuristics start worse than the random heuristic, and the reverse topological order stays worse than the random heuristic, but the topological order heuristic reduces and later becomes better than random. The reason is that, the reversed topological order tends to choose resources equally among all the paths, since the paths converge towards the end of the graph. On the other hand, the topological order heuristic chooses from initial nodes, which might converge into one path and give better results. The most accurate is the shortest path heuristic algorithm, which combines the topological order and shortest path heuristics. The error rate of this algorithm becomes flat when it finishes calculating the shortest path and starts to calculate other resources.

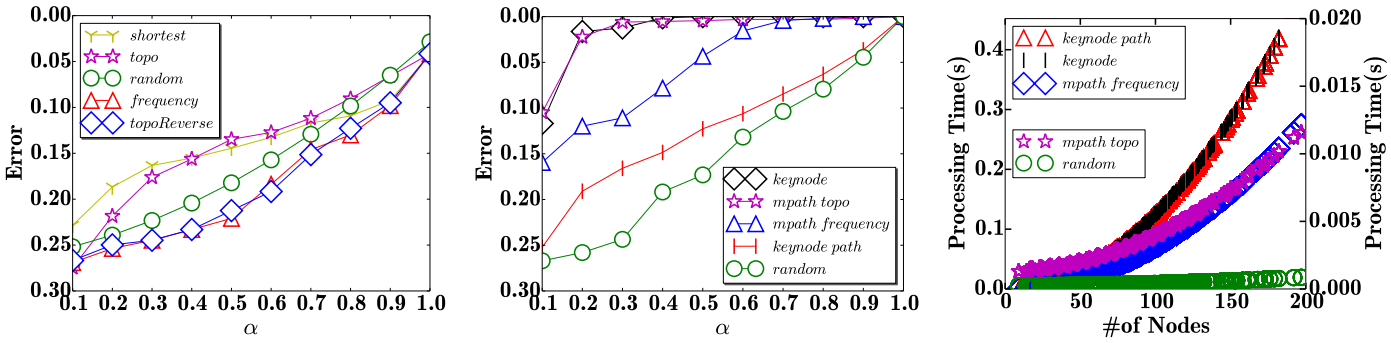
Figure 7(b) depicts the error rate of the heuristic algorithms combining multiple heuristics. We can see that the keynode and the mpath topo algorithms produce very good results, e.g., less than 0.05 error rate with only 20% of resources calculated. Such results show a promising solution for obtaining relatively accurate network attack surface results without incurring too much cost for calculation. From the results we can see that the mpath topo algorithm has less error than mpath frequency. For the keynode heuristic algorithm, we test two different variations, one based on the change of shortest path and the other based on the change of the calculated result. From the results, we can see that those have very different error rates, because the result-based keynode algorithm tends to gather the resources in the shortest path, whereas the path-based algorithm tends to avoid such resources.

Figure 7 (c) depicts the processing time. From the results, we can see that the keynode path and keynode result algorithms have almost the same processing time, because the majority of processing is used to preselect the keynode set. The processing time for mpath frequency is higher than mpath topo, because each iteration generates new m-shortest paths and we need to reorder frequently. For mpah topo, we only gather m-shortest paths once and then order them by the topological order. The random choose heuristic has the lowest processing time as expected. Overall, we can conclude that the mpath topo algorithm is the best choice in terms of both error rate and processing time.

C. Comparison with k-Zero Day Safety

In this section, we compare the proposed network attack surface metric (calculated based on the brute force algorithm) and the result of the keynode heuristic algorithm (with 10% of the calculation effort) to the existing *k*-zero day safety metric [9]. The objective is to demonstrate the significance of discriminating different network resources based on their attack surface, which is a key contribution of our metric in contrast to *k*-zero day safety (which simply regards all network resources as equally likely to have unknown vulnerabilities) .

First, we study the difference between *k*-zero day safety and network attack surface with the increasing length of



(a) (b) (c)

Fig. 7: The Cost vs. Error for Simple Heuristics (a) and for the Heuristic Algorithms (b), and the Processing Time (c)

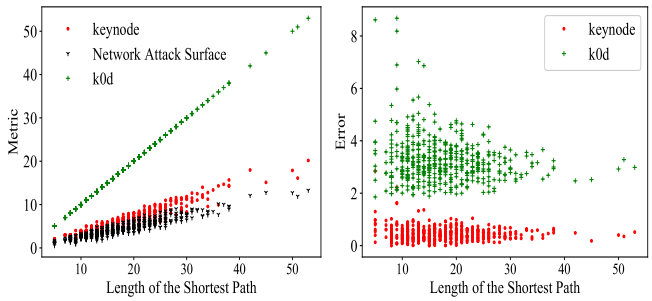


Fig. 8: The Comparison between the *k*-Zero Day Safety Metric and the Network Attack Surface Metric (a), the Error Rate (b) with the increasing length of the shortest path

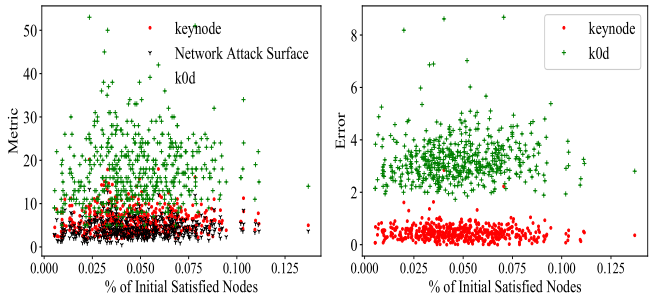


Fig. 9: The Comparison between the *k*-Zero Day Safety Metric and the Network Attack Surface Metric (a), the Error Rate (b) with the increasing percentage of initially satisfied nodes

the shortest path. Figure 8(a) shows the metric results, and Figure 8(b) shows the error rate for the *k*-zero day safety metric and keynode heuristic algorithm (both in contrast to the network attack surface metric).

Results and Implications: As can be seen in Figure 8(a), although in general the metric values all increase in the length of shortest path, the network attack surface metric (as well as the keynode heuristic algorithm) and the *k*-zero day safety metric do not follow the same trend. Specifically, the increase of the network attack surface metric is significantly slower than that of the *k*-zero day safety metric. The absolute difference between the two metrics increases with the length of the shortest path. This can be explained by the fact that, by simply counting the number of distinct resources on the shortest path, the *k*-zero day safety essentially ignores the difference between the attack surface of different resources, and consequently yields an upper bound of the network attack surface metric. The implications of those results are as follows. First, the error introduced in the *k*-zero day safety metric may become significant, especially for larger and well guarded networks with relatively long shortest paths (which are the main target of our work). For smaller networks or networks that are not well guarded (e.g., campus networks) with shorter shortest paths, the absolute difference between the two metrics may be smaller but the relative error rate may still be significant, as shown in Figure 8(b). Finally, with only 10% of calculation effort, our keynode heuristic algorithm could already estimate the network attack surface value with reasonably small error, which shows its potential as a relatively accurate estimation requiring far less effort.

Figure 9 shows similar results for the metrics vs. the increasing percentage of the initial satisfied nodes in resource graphs (i.e., initially exploitable resources, which provides another indicator about how well guarded the network is). Figure 9(a) demonstrate the absolute values of metrics, while Figure 9(b) shows the error rate.

Results and Implications: From Figure 9, we can have the following observations. First, both network attack surface and *k*-zero day safety metrics are less dependent on the initially satisfied nodes as they did in the previous case. This is because the initial satisfied nodes may not be directly correlated with the length of the shortest path. Second, despite the lack of clear trends, similar comparison results between the metrics can still be observed. The larger errors and error rates between *k*-zero day safety metric and the network attack surface metric (as well as the heuristic algorithm) demonstrate the fact that the *k*-zero day safety metric may become significantly less accurate, whereas our keynode heuristic algorithm provides a more accurate estimation.

D. The Impact of Non-Calculatable Resources

Instead of calculating attack surface for a smaller subset of software in the network, as studied in previous section, estimating the attack surface of a software by only considering a subset of its resources (e.g., entry/exit points), is another technique to reduce the total cost. For example, the Microsoft research team has proposed attack surface approximation method based on stack trace analysis [14]. In a trial on Windows 8, the authors discover that the approximation selects

only 48.6% of the software but includes 94.6% of the known vulnerabilities. In this section, we would like to evaluate the impact of this idea through simulations.

In addition, fully calculating the attack surface of a software may become infeasible for either closed source software or very large open source software. For example, statistical results show 84.34% of desktop operating systems are Windows [26], and even some open source software may become too large for the calculation, e.g., Debian's source lines of code (SLOC) increases from 55-59 millions (Debian 2.2 in 2000) to 419 million (Debian 7.0 in 2012). Therefore, we have divided software applications into three categories in terms of the feasibility of calculating attack surface as follows.

- *Non-Calculable Resources:* The software which do not have accessible source code and hence their attack surface cannot be calculated.
- *Partially Calculable Resources:* The open source software with too large SLOCs for fully calculating the attack surface. A feasible solution is to estimate the attack surface by only considering part of the software.
- *Fully Calculable Resources*: The small to medium open source software for which it is generally feasible to generate call graphs and fully calculate the attack surface.

The first simulation studies the impact of partially-calculable and non-calculable resources. The error rate is defined in the same way as in the previous simulations. The budget is defined as the percentage of effort allowed to spend on calculating attack surface in one network, denoted as α. The partially-calculable rate is defined as the percentage of attack surface calculated for a resource, denoted as β. Calculating β percent of a software application may result in an approximated value of the attack surface. Compared to the true value of attack probability, the approximated value may be either lower or higher because the chosen subset of source code may include either less or more I/O function calls. Therefore, in this simulation, we set an estimation range for the approximated attack probability value. Assuming the true attack probability in a software application is p, the estimation range is defined as $[(1-\beta)*p,\ min((2-\beta)*p, 1)]$, which ranges from $(1-\beta)*p$ lower than the true value to $(1-\beta)*p$ higher. An approximated attack probability value is randomly generated from the estimation range.

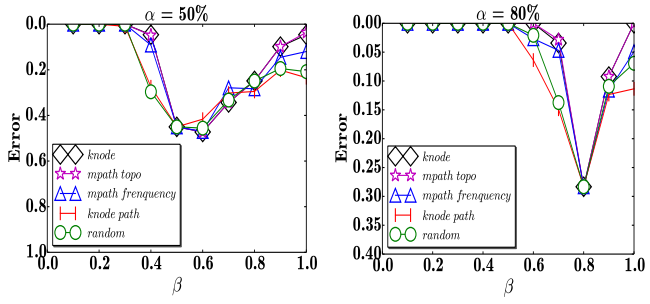


Fig. 10: The Error of the Algorithms with $\alpha = 50\%$ (a) and $\alpha = 80\%$ (b)

Unlike in the previous simulations, α only represents the percentage of effort to calculate attack surface in this simulation, since the percentage of resources whose attack surface are calculated will depend on both α and β. After calculating α percent of the resources in one network, we will still be able to calculate $\alpha - \alpha * \beta$ percent of attack surface with the remaining budget. The overall percentage of calculated resources can be written as $\sum_{i=0}^{\infty} \alpha * (1-\beta)^i$, which is a geometric series with the constant ratio $(1-\beta)$ yielding the final result of $\frac{\alpha}{\beta}$. Notice that when $\alpha > \beta$, we have extra budget calculate attack surface. In this case, we apply the extra budget to fully calculate the remaining resources according to the algorithms' order of calculation.

Results and Implications: In Figure 10(a), $\alpha = 50\%$ means the effort budget is 50% of the overall effort to fully calculate every attack surface in the network, so with $\beta < 50\%$ ($\frac{\alpha}{\beta} > 1$), we have extra budget to calculate more resources according to the algorithms' order. A smaller value of β means more extra budget will be left for full calculation after the initial step of partial calculation is done. Comparing to $\alpha = 50\%$ in Figure 7(a) and Figure 7(b), the error rates are smaller when β is smaller than 30%. This is mostly because the effort spent on the partial calculation provides a rough ranking of the resources, and the remaining efforts are used to fully calculate those resources that contribute the most to the final result of network attack surface. When β increases to 40%, the error rates of algorithms become worse than those under $\alpha = 50\%$ in Figure 7(a) and Figure 7(b), because the remaining efforts are not sufficient to fully calculate the resources on the shortest path.

We can also see the error rate increases until $\alpha = \beta$, which is the worst case in Figure 10(a) since $\alpha = \beta$ means all the attack probabilities used to calculate the final result are partially calculated (the attack probability value for each resource falls in the estimation range mentioned earlier). Similar trends can be observed in Figure 10(b) when $\beta \leq \alpha = 80\%$. We can see that the error rate is 0.5 when $\beta = 50\%$ because the approximation range is set as 50% lower to 50% higher than the exact attack surface (the maximum value for the upper bound is 1). The error rate is 0.28 when $\alpha = \beta = 80\%$ which is still close to our approximation range of 20% lower to 20% higher than the true value of attack probability.

Unlike the increasing trend of the error rates when $\beta \leq 50\%$, the error rates are decreasing when $\beta > \alpha$ for all the algorithms in both Figure 10(a) and Figure 10(b), because the approximated attack probability is closer to the true value of attack probability when more source code are used for calculating attack surface. When $\beta = 100\%$, this simulation is effectively the same as the previous simulation.

While the previous simulations show the relationships of error rates and β with fixed α for the algorithms, Figure 11(a) presents the relationships of error rate and α with fixed $\beta = 60\%$. When $\alpha \leq \beta$, only α percent of the resources can be partially calculated, and the error rate decreases with the percentage of calculated resources. When $\alpha > \beta$, extra budget can be applied again to fully calculate resources according to the algorithms' chosen order, and the error rate is thus different for different algorithms (knode and mpath topo yield the best chosen order). Figure 11(b) shows the overall relationships

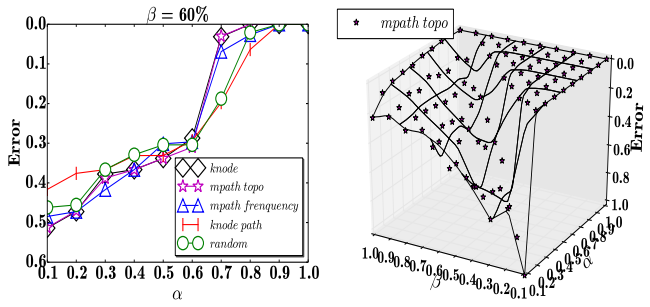
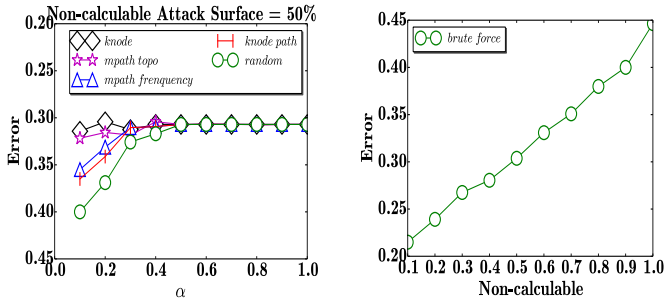


Fig. 11: (a) The Error of the Algorithms with $\alpha = 50\%$ (b) The Error in α and β

among α, β and the error rate. The concave upward part in the 3D graph corresponds to the special case of $\alpha = \beta$, as previously discussed.

Finally, the next simulation focuses on the impact of *non-calculable* resources. The definition of α is the same as in the first simulation, i.e., the percentage of resources whose attack surface is calculated. We assign the attack probability for non-calculable resources based on the average value among the CVSS scores of all vulnerabilities in the National Vulnerability Database (NVD) [24]), which is 0.68, in this simulation.



(a) (b)

Fig. 12: The Error vs. α of Algorithms with 50% *Non-Calculable* Resources (a) and the Percentage of *Non-Calculable* Resources vs. Error (b)

Results and Implications: Figure 12(a) shows the impact on our algorithms when 50% of the resources are *non-calculable* in a network. The error rate decreases till $\alpha = 50\%$, while error rate remains the same when $\alpha > 50\%$. We can see the algorithms help to reduce the error rate and the knode and mpath topo algorithms give the best performance in all the simulations. Next, Figure 12(b) studies the impact of *non-calculable* resources compared to a brute force algorithm (i.e., regardless of the budget, calculating 100% for every calculable attack surface). The error rate increases linearly with the increasing ratio of non-calculable resources in the network. When non-calculable resources reach 100%, our metric essentially becomes equivalent to the k-zero day safety metric [9] (which is a special case of our metric in which no attack surface is calculated).

VI. Related Work

The concept of attack surface is originally proposed for specific software, e.g., Windows, and requires domain-specific expertise to formulate and implement [27]. Later on, the concept is generalized using formal models and becomes applicable to all software [28]. Furthermore, it is refined and applied to large scale software, and its calculation can be assisted by automatically generated call graphs [3], [29]. Attack surface has attracted significant attentions over the years. It is used as a metric to evaluate Android's message-passing system [5], in kernel tailing [30], and also serves as a foundation in Moving Target Defense, which basically aims to change the attack surface over time so to make attackers' job harder [7], [31]. Others aim to expand the scope of this concept to other domains, such as the six-way attack surfaces between users, services, and cloud systems [4], and the approximation of attack surface for modern automobiles [6]. The study on automating the calculation of attack surface is another interesting domain, e.g., COPES uses static analysis from bytecode to calculate attack surface and to secure permission-based software [32]. Stack traces from user crash reports is used to approximate attack surface automatically [14]. The correlation between attack surface and vulnerabilities has also been investigated, such as using attack surface entry points and reachability to assess the risk of vulnerability [33]. A study about the relationship between attack surface and the vulnerability density is given in [34], although the result is only based on two releases of Apache HTTP Server, which gives little clue to the general existence of such a correlation. The so-called attack graph surface introduced in [35] is inspired by attack surface but it focuses on identifying a critical set of attack paths in an attack graph, which is complementary to our work in the sense that we can employ this technique to efficiently identify the shortest path, whereas our metric may potentially be applied to the critical attack paths. Despite such interest in attack surface, to the best of our knowledge, most existing works that apply the concept to a higher abstraction level are still limited to intuitive and informal notions, and this is the first formal attack surface metric at the network level.

The research on security metrics in general has attracted much attention lately [8]. There exist standardization efforts on vulnerability assessment including the Common Vulnerability Scoring System (CVSS) [1], which measures vulnerabilities in isolation. The NIST's efforts on standardizing security metrics are also given in [36] and more recently in [37]. Earlier work include the a metric in terms of time and efforts based on a Markov model [38]. More recently, several security metrics are proposed by combining CVSS scores based on attack graphs [39], [40]. The minimum efforts required for executing each exploit is used as a metric in [41], [42]. A mean time-to-compromise metric is proposed based on the predator state-space model (SSM) used in the biological sciences in [43]. While those metrics are mostly developed for known vulnerabilities, fewer work are capable of dealing with zero day attacks. A few exceptions include an empirical study of the total number of zero day vulnerabilities available on a single day based on existing data [44] and an effort on

ordering different applications in a system by the seriousness of consequences of having a single zero day vulnerability [45]. More recently, the k-zero day safety model [9], [11] and the network diversity model [10], [12] both attempt to model the risk of zero day vulnerabilities, but their common limitation is the lack of capability in distinguishing between different resources based on their relative likelihood of having such vulnerabilities. In other words, those works essentially regard all resources as equally likely to include zero day vulnerabilities. This is a limitation since different software may have significantly different attack surface and thus the likelihood of such vulnerabilities may also differ. The key contribution of this paper is exactly to address this limitation through lifting the attack surface concept to the network level. On the other hand, this paper leverages the resource graph concept and the Bayesian model in [10], [12] and the k-zero day safety model in [9], [11] (attack surface is not mentioned in those works).

VII. Limitations and Conclusion

An intuitive notion of attack surface at the network level has prevented applications from inheriting the formal and quantitative reasoning power of the original attack surface metric. In this paper, we have designed methods for lifting this concept to the network level as a formal security metric for measuring networks' resilience against zero day attacks. Specifically, we have shown two ways for converting the attack surface of each individual software into an attack probability and subsequently aggregating such attack probabilities into a single measure of network attack surface based on the causal relationships between different resources. We have also presented heuristic algorithms which can evaluate the network attack surface while limiting the effort of calculating the attack surface for individual software within a given budget. To evaluate the proposed models, we have studied the correlation between attack surface and vulnerabilities using real world software, and our experimental results show a positive correlation does exist between the two. To evaluate the proposed heuristic algorithms, we have shown through simulations that the network attack surface metric can be accurately estimated by calculating the attack surface for only a small percentage of resources.

The following discusses limitations and future directions.

- First, our experiments on the correlation between attack surface and vulnerabilities are still of relatively small scale and scope; a future direction is to expand these and to consider also other factors, such as market share data and exploit information.
- Second, there lack automated and mature tools for assisting the calculation of attack surface. One of our ongoing work is the development of an automated tool for calculating the attack surface for open source software.
- Third, the calculation of attack surface requires source code and thus is not applicable to closed source software. An interesting future direction is to address this through adapting binary analysis techniques (e.g., clone detection).
- Fourth, we have not considered the average case network attack surface in the study of heuristic algorithms, and this will be a future direction.

Acknowledgements. Authors with Concordia University are partially supported by the Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035. Sushil Jajodia was supported in part by the National Institute of Standards and Technology grants 60NANB16D287 and 60NANB18D168, National Science Foundation under grant IIP-1266147, Army Research Office under grant W911NF-13-1-0421, and Office of Naval Research under grant N00014-15-1-2007.

References

[1] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85–89, 2006.
[2] J. McHugh, "Quality of protection: Measuring the unmeasurable?," in *Proceedings of the 2nd ACM QoP*, pp. 1–2, 2006.
[3] P. Manadhata and J. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol. 37, pp. 371–386, May 2011.
[4] N. Gruschka and M. Jensen, "Attack surfaces: A taxonomy for attacks on cloud services," in *2010 IEEE 3rd international conference on cloud computing*, pp. 276–279, IEEE, 2010.
[5] D. Kantola, E. Chin, W. He, and D. Wagner, "Reducing attack surfaces for intra-application communication in android," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 69–80, ACM, 2012.
[6] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, *et al.*, "Comprehensive experimental analyses of automotive attack surfaces.," in *USENIX Security Symposium*, San Francisco, 2011.
[7] S. Jajodia, A. Ghosh, V. Swarup, C. Wang, and X. Wang, *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*. Springer, 1st ed., 2011.
[8] M. Pendleton, R. Garcia-Lebron, J. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, vol. 49, pp. 62:1–62:35, Dec. 2016.
[9] L. Wang, S. Jajodia, A. Singhal, P. Cheng, and S. Noel, "k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 1, pp. 30–44, 2013.
[10] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 1071–1086, 2016.
[11] 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)*, pp. 573–587, 2010.
[12] L. Wang, M. Zhang, S. Jajodia, A. Singhal, and M. Albanese, "Modeling network diversity for evaluating the robustness of networks against zero-day attacks," in *Proceedings of ESORICS'14*, pp. 494–511, 2014.
[13] A. Reid, J. Lorenz, and C. A. Schmidt, *Introducing Routing And Switching In The Enterprise, CCNA Discovery Learning Guide*. Cisco Press, 2008.
[14] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, "Approximating attack surfaces with stack traces," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pp. 199–208, IEEE Press, 2015.
[15] "Open hub." available at:https://www.openhub.net/, April 19, 2017.
[16] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.
[17] C. ANSI, "Iso/iec 9899: Tc2.www.open-std.org/jtc1/sc22," *WG14/www/docs*, no. 1256, 2005.
[18] "Gnu cflow." available at :http://www.gnu.org/software/cflow/, April 10,2016.
[19] "Amanda protocol." available at:http://wiki.zmanda.com/index.php/Developer_documentation, April 19, 2017.
[20] H. Chen, D. Wagner, and D. Dean, "Setuid demystified," in *USENIX Security Symposium*, pp. 171–190, 2002.

[21] "Firewall builder." available at: http://www.fwbuilder.org/4.0/documentation.shtml, April 19, 2017.
[22] "Common vulnerability scoring system version 2 calculator." available at:https://nvd.nist.gov/cvss/v2-calculator?, April 19, 2017.
[23] "Nessus network security scanner." available at:http://www.tenable.com/products/nessus-vulnerability-scanner, April 19,2017.
[24] "National vulnerability database." available at: http://www.nvd.org, May 9, 2008.
[25] "Freetype." available at:https://en.wikipedia.org/wiki/FreeType.
[26] "Stat counter." available at:http://gs.statcounter.com/os-market-share/desktop/worldwide.
[27] M. Howard, J. Pincus, and J. Wing, "Measuring relative attack surfaces," in *Workshop on Advanced Developments in Software and Systems Security*, 2003.
[28] P. Manadhata and J. Wing, "Measuring a system's attack surface." Technical Report CMU-CS-04-102, 2004.
[29] P. Manadhata and J. Wing, "An attack surface metric." Technical Report CMU-CS-05-155, 2005.
[30] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schröder-Preikschat, D. Lohmann, and R. Kapitza, "Attack surface metrics and automated compile-time os kernel tailoring.," in *NDSS*, 2013.
[31] S. Jajodia, A. Ghosh, V. S. Subrahmanian, V. Swarup, C. Wang, and X. Wang, *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. Springer, 2012.
[32] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus, "Automatically securing permission-based software by reducing the attack surface: An application to android," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 274–277, ACM, 2012.
[33] A. A. Younis, Y. K. Malaiya, and I. Ray, "Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability," in *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pp. 1–8, IEEE, 2014.
[34] A. A. Younis and Y. K. Malaiya, "Relationship between attack surface and vulnerability density: A case study on apache http server," in *Proceedings on the International Conference on Internet Computing (ICOMP)*, p. 1, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.
[35] H. Huang, S. Zhang, X. Ou, A. Prakash, and K. Sakallah, "Distilling critical attack graph surface iteratively through minimum-cost sat solving," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, (New York, NY, USA), pp. 31–40, ACM, 2011.
[36] National Institute of Standards and Technology, "Technology assessment: Methods for measuring the level of computer security." NIST Special Publication 500-133, 1985.
[37] M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo, "Security metrics guide for information technology systems." NIST Special Publication 800-55, 2003.
[38] M. Dacier, "Towards quantitative evaluation of computer security." Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.
[39] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Proceedings of the 22nd IFIP DBSec*, 2008.
[40] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic bayesian network," in *Proceedings of 4th ACM QoP*, 2008.
[41] D. Balzarotti, M. Monga, and S. Sicari, "Assessing the risk of using vulnerable components," in *Proceedings of the 1st ACM QoP*, 2005.
[42] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup, "A weakest-adversary security metric for network configuration security analysis," in *Proceedings of the ACM QoP*, pp. 31–38, 2006.
[43] D. Leversage and E. Byres, "Estimating a system's mean time-to-compromise," *IEEE Security and Privacy*, vol. 6, no. 1, pp. 52–60, 2008.
[44] M. McQueen, T. McQueen, W. Boyer, and M. Chaffin, "Empirical estimates and observations of 0day vulnerabilities," *Hawaii International Conference on System Sciences*, vol. 0, pp. 1–12, 2009.
[45] 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*, pp. 117–126, 2009.