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

Mengyuan Zhang[1], Lingyu Wang[1], Sushil Jajodia[2], and Anoop Singhal[3]

[1] Concordia Institute for Information Systems Engineering, Concordia University
{mengy_zh,wang}@ciise.concordia.ca
[2] Center for Secure Information Systems, George Mason University
jajodia@gmu.edu
[3] Computer Security Division, National Institute of Standards and Technology
anoop.singhal@nist.gov

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 (e.g., the network level) are limited to an intuitive and qualitative notion, losing the power of the original concept. In this paper, we lift the attack surface concept to the network level as a security metric for evaluating the resilience of networks against potential zero day attacks. Specifically, we tackle two main challenges as follows. First, we develop models for addressing the incompatibility between the original attack surface model and the need for average across different resources inside a network. Second, we design heuristic algorithms to significantly reduce the complexity of calculating the network attack surface. Finally, we confirm the effectiveness of the proposed algorithms through simulation results.

1 Introduction

For a mission critical computer network (e.g., those that play the role of a nerve system in critical infrastructures, governmental or military organizations, and enterprises), the security administrators usually 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 exploiting previously unknown vulnerabilities. However, while there exist many standards and metrics for measuring the relative severity of known vulnerabilities (e.g., CVSS [21]), the task becomes far more challenging for unknown vulnerabilities, which are sometimes believed to be unmeasurable [19].

To that end, a promising solution is the *attack surface* concept [18], 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 independent

of external factors, such as the disclosure of vulnerabilities or availability of exploits, it naturally covers both known and unknown vulnerabilities [18] and becomes a good candidate for understanding 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 [9], mobile device security [15], automotive security [5], Moving Target Defense (MTD) [13], etc. (a detailed review of related work is provided in Section 5). 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. Adopting such an imprecise notion unavoidably loses most of the original concept's power in formally and quantitatively reasoning about the likelihood of a system to contain vulnerabilities.

In this paper, we address this issue by lifting the original attack surface concept to the network level as a 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 paradox 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 produce less than 0.05 error rate with only 20% of the resources calculated).

The main contribution of this work is twofold. First, to the best of our knowledge, this is the first effort on lifting the attack surface concept to the network level as a formally defined security metric. We believe such a metric may serve as a foundation of many useful analyses for quantitatively designing, evaluating, and improving network security. 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. The remainder of this paper is organized as follows. We first build intuitions through a motivating example and then present the formal models in Section 2. We design heuristic algorithms in Section 3 and evaluate their performance in Section 4. We review related work in Section 5 and discuss limitations and future work before we conclude the paper in Section 6.

1.1 Motivating Example

First, we illustrate the main challenges through a motivating example shown in Figure 1 (the topology is roughly based on [27]). 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 h4 but blocks all inbound requests except those from h2. We also assume our main concern is protecting the Admin Host (h4) containing critical assets. Based on such assumptions, we can easily see that, an attacker at h0 can follow an *attack path*, e.g., $h1 \rightarrow h2 \rightarrow h4$, to compromise h4. Keeping this in mind, we now consider the question: *How could we apply the attack surface concept to such a network to measure its security (e.g., in terms of h4)*?

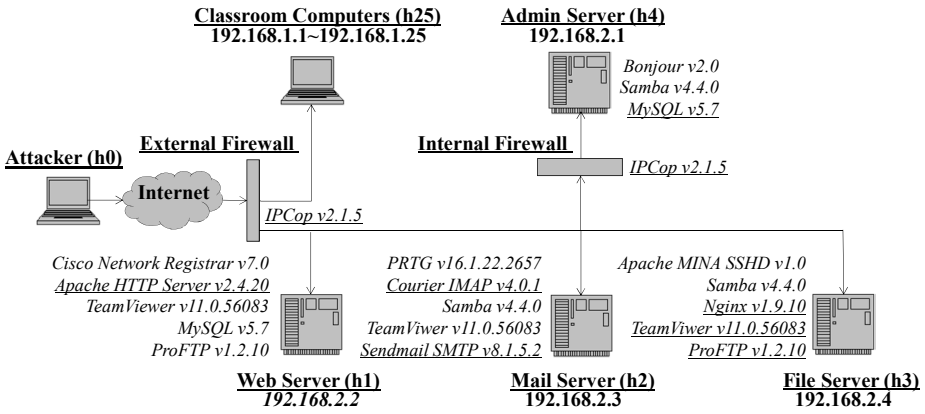


Fig. 1: The Motivating Example

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 yield the total numbers of methods, channels, and untrusted data items, respectively (more details are given in Section 2). 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), doing this along an attack path, e.g., $h1 \rightarrow h2 \rightarrow h4$, is less meaningful, because it means a longer attack path would yield a larger attack surface (less secure), but a longer path usually requires more effort from attackers (more secure), which is a contradiction. 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 Section 2.

The second major challenge lies in the calculation of attack surface, which is well known to be costly since identifying the code that lies on the attack surface requires expertise and significant manual effort [18, 29]. 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 for 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 the above example, suppose we change the firewall rules such that requests from both h2 and h3 to h4 are allowed. We now have a paradox 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 in Section 2) 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 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 3.

2 The Network Attack Surface Model

In this section, we lift the attack surface concept to the network level in two steps. First, Section 2.1 converts the attack surface of a software to its attack probability. Second, Section 2.2 aggregates the attack probabilities of network resources into a single measure of network attack surface.

2.1 Converting Attack Surface to 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 1.1. Our main idea is to convert the attack surface of each software resource into an *attack probability*, which reflects the relative likelihood that the software contains at least one exploitable zero day vulnerability [4]. 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 [18], 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) [21] 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.

Method Group-Level Conversion First, we briefly review the concepts of attack surface and CVSS. As illustrated in the first column of Table 1, the CVSS defines six base metrics in two groups, the accessibility group including access vector (AV), access complexity (AC), and authentication (Au), and the impact group including 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) [21]. The second column of Table 1 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 hence are mapped together. The exact mapping for those two IMAP daemons are shown in the last column of Table 1. Each CVSS vector maps to the corresponding access right or privilege shown 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). For example, in the third row, the authenticated access right is mapped to network for access vector (i.e., *AV:N*), because the

[4] Note the attack probability here is only intended as a relative metric for comparison purposes, instead of the actual probability of attacks which is generally infeasible to obtain in practice.

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 1: Mapping Attack Surface to CVSS Base Metrics for Courier IMAP Server v4.1.0 and Cryus IMAP Server v2.2.10

UNIX socket in those software has the local authenticated access right, which means attackers may obtain the local authenticated access right over the network. Also, 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)" [21]. 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.

Note that, since this mapping is based on the understanding of access rights, privileges, and the software, different administrators may end up assigning the mappings in different and incomparable ways. However, since metrics are relative, and meant for comparing similar configurations of the same network, the results would still be meaningful as long as the mapping is consistent across different configurations.

Based on the mapping shown in Table 1, 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 [21], as shown in Table 2. 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.

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* representing the relative likelihood of the software to be exploitable through at least one zero day vulnerability. 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 (n of p_0 are both intended as normalizing constants; see below for more discussions). In Equation 1, the base score divided by its range 10 gives the probability that a vulnera-

Method Group	Privilege	Access Rights	DEP	DExp	Vector	Base Score
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
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
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
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
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
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
M4	cyrus	anonymous	12	21	[AV:1.0,AC:0.71,Au:0.704,C:0.66,I:0.66,A:0.66]	10

Table 2: Method Groups and Their Base Scores for Courier IMAP Server v4.1.0 and Cyrus IMAP Server v2.2.10

bility in this group 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. Note that, the true values of parameters n and p_0 are certainly impossible to obtain in practice, so those are only intended to be normalizing constants chosen to ensure a reasonable value for p. As long as those values stay constant between different software, the equation will yield a relative value 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.

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

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

2.2 Aggregating Attack Probabilities inside a Network

Now that we have converted the attack surface of a resource to its attack probability, we can easily aggregate the attack surface of all network resources into a single *network attack surface* value. We consider two different ways for aggregating the attack surface of resources in the network, the shortest path-based approach [31] and the Bayesian network (BN)-based approach [36], which reflect the worst case scenario (i.e., with respect to attackers following the easiest attack path) and the average case scenario (i.e., with respect to any attacker), respectively.

To illustrate the idea, Figure 2 shows a partial *resource graph* [31] for our example, which is syntactically equivalent to an attack graph, but models zero day attacks instead of known vulnerabilities. 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 probability inside each box is the attack probability of the corresponding resource.

Example 2. In Figure 2, for the shortest path-based approach, we can calculate the attack probability for the shortest path indicated by the dashed line, $\langle IPCop, 0, F \rangle \rightarrow$

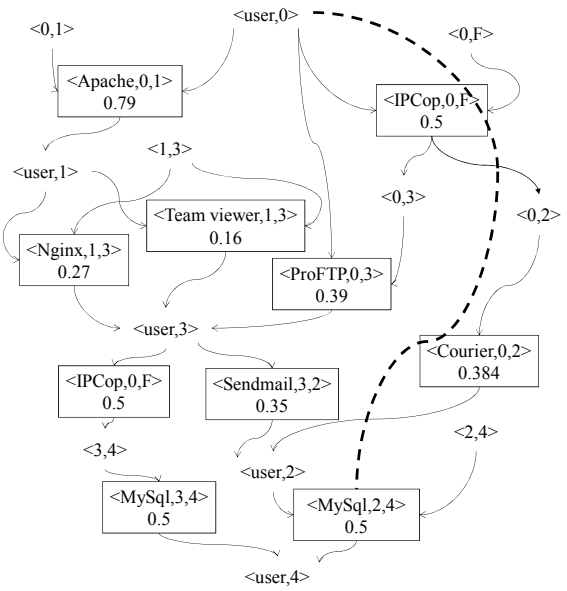


Fig. 2: The Resource Graph with Attack Probability for the Network in Figure 1

$\langle Courier, 0, 2\rangle \rightarrow \langle MySql, 2, 4\rangle$, the probability can be calculated as $p = 0.5 * 0.384 * 0.5 = 0.096$. Note that our approach here addresses a key limitation of the existing k-zero day safety metric (which also adopts a shortest path-based approach) [31], i.e., it cannot discriminate different resources based on their relative attack probabilities.

Example 3. For the BN-based approach, we can simply regard Figure 2 as a Bayesian network, with the attack probability of each resource regarded as the conditional probability that the corresponding resource can be exploited given that its pre-conditions are all satisfied, and then perform probabilistic inference using the BN [36]. In this example, we can calculate the probability for attackers to reach $\langle user, 4\rangle$ as $p_{goal} = 0.236$.

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 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 (the BN is more precisely defined in [7] and details are omitted here due to space limitations), 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.

We note that, although the network attack surface above is defined as probabilities, those can potentially be converted into other forms for different interpretations. For example, given the network attack surface p as a probability, we can easily 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). Also, we can convert p back into an equivalent number of zero day vulnerabilities as $\log_{0.08} p$ (here 0.08 is a nominal probability for zero day vulnerabilities based on CVSS base metrics as described in [36]), which is a simple count-based metric helpful for interpretation and comparison purposes (we will use this method in our algorithms and simulations).

3 Heuristic Algorithms for Calculating Network Attack Surface

In this section, we propose heuristic algorithms to reduce the effort in evaluating the network attack surface. We first state the problem in Section 3.1, and then introduce several simple heuristics in Section 3.2 and design algorithms based on such heuristics in Section 3.3.

3.1 The Problem

Calculating the attack surface of a software is well known to be costly [18, 29] mostly due to the manual work 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). On the other hand, the calculation of attack surface is becoming more practical due to ongoing efforts on automating or approximating the calculation [29]. Nonetheless, we believe although the calculation is practical with automated techniques, it will still remain a costly process due to the ever increasing size of modern software [5].

Therefore, we investigate the problem of evaluating the network attack surface while reducing the effort of calculating the attack surface of individual resources. 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. 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. Specifically, given a network with the set of resources R and suppose the true value of the network attack surface is p_{true} and the calculated value is p_{cal} (we assume all the values described in this section are count-based, as described at the end of Section 2.2), we would like to

[5] For example, the number of lines of software mentioned in our running example in Figure 1 are as follows: 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).

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 the case, because the objective function $\frac{|p_{true}-p_{cal}|}{p_{true}}$ contains an unknown value p_{true}, whose calculation would imply calculating the attack surface for all resources and defy the very purpose of reducing the cost. Also, since the problem of finding the shortest path is already NP-hard [31], which is a special case of our problem with unlimited budget, the latter is also intractable. Therefore, we study heuristic algorithms in the coming section.

3.2 The Heuristics

The 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. Therefore, this section first considers a few straightforward heuristics for choosing the resources in the right order, e.g., by exploring the structural properties of a resource graph. We will then combine those heuristics into better algorithms in the coming section and evaluate their performance later in Section 4.

Random Choose The most obvious solution is probably to simply choose resources in a completely random fashion, namely, the *random choose* heuristic. Although the random choose algorithm is likely far from optimal, it provides a baseline for comparison with other heuristic algorithms we will propose. For example, in Figure 2, if our budget is to calculate the attack surface of at most two resources, then among the $\binom{8}{2} = 28$ possible choices, the worst result is $p = 0.46$ with an error rate of 0.51, whereas the best result is $p = 0.7$ with error 0.24.

Frequency Choose The idea of this heuristic 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 2, we can see both *IPCop* and *MySql* appear twice among totally 10 exploits. Therefore, if our budget is two, then calculating both of them will unveil $4/10$ of the exploits (the result is $p = 0.7$ with an error rate of 0.24).

Topological Order The idea here is that, since the nodes closer to the beginning and end of a resource graph tend to be shared among more attack paths (e.g., the last two exploits are shared by all paths in Figure 2), 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 from the beginning and from the end, respectively. For example, in Figure 2, suppose our budget is two, the topological order heuristic may choose *Apache* and *IPCop* while the reversed topological order may choose *MySql* and *Sendmail* (the results would both be $p = 0.58$ with error 0.375).

Shortest Path This heuristic starts the calculation with resources on the path with the least number of exploits (e.g., the path depicted in dashed line in Figure 2), which, although not always the right path in terms of the final result, may serve as a good starting point. For example, in Figure 2, if our budget is two, then the shortest path

heuristic will choose *IPCop* and *MySql* on the dashed line path (the result is $p = 0.70$ with error 0.24). In this particular example, this path happens to be the right path for calculating the final result, so a larger budget will potentially produce more accurate result.

3.3 The Algorithms

The above heuristics may not produce good results when each of them is used alone, but combining them leads to algorithms with good performance, as we will show in this section.

Mpath-Topo Heuristic Algorithm: This algorithm combines the above topological order and shortest path heuristics as follows. First, we apply the shortest path heuristic to choose M (an integer parameter) shortest paths, which are ranked based on the number of exploits, as the starting points. Since there is no order between resources along each such path, we next apply the topological order heuristic to sort all paths, as well as those not on such paths. The algorithm is more clearly depicted on the left-hand side of Figure 3.

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**$\mid MS \mid > 0$
6. **Append** the first resource r in MS to P
7. **Remove** r from MS
8. **Else If** $\mid T \mid > 0$
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** $\mid KN \mid > 0$
11. **Append** the first resource r in KN to P
12. **Remove** r from KN
13. **Else If** $\mid G \setminus KN \mid > 0$
14. **Append** the first resource r to P
15. **Remove** r from G
16. **Let** N= N-1
17. **Return** P

Fig. 3: Mpath-Topo (Left) and Keynode (Right) Heuristic Algorithms

Example 4. In Figure 2, assuming $M = 2$, we have $MS = IPCop, Courier, MySql, ProFTP, Sendmail$ and $T = Apache, Nginx, Team\ viewer$. If our budget $N = 2$, then $P = IPCop, Courier$, and the final result is $p = 0.61$, with error 0.34.

Keynode Heuristic Algorithm This heuristic algorithm is based on the idea that a resource is more important in determining the final network attack surface value p, if changing its value may result in significant changes, e.g., a change in the shortest path

(the path selected for calculating the final result, which is different from the one mentioned in the above shortest path heuristic) , or a change in the currently calculated result of p. We then combine this heuristic with the topological order heuristic to form the algorithm depicted on the right-hand side of Figure 3 (here we only show the change in p, which can be replaced with the change in the shortest path, and we will evaluate both algorithms in the coming section).

Example 5. Here we choose $p_0 = 0.08$ and $p_1 = 1$. In Figure 2, we initially calculate $p = 5.12 * 10^{-4}$. We then calculate p again by assigning p_1 to each resource. For example, with *IPCop* changed from p_0 to p_1, we have $p = 0.0064$, so *IPCop* is a key node. Similarly, we can obtain the key nodes sequence as $KN = IPCop, Courier, MySql$. If our budget $N = 2$, then $IPCop$ and $Courier$ will be chosen and the result $p = 0.61$ with error 0.34.

We will evaluate the performance of those heuristics and algorithms, including both the accuracy and running time, in the coming section.

4 Experimental Results

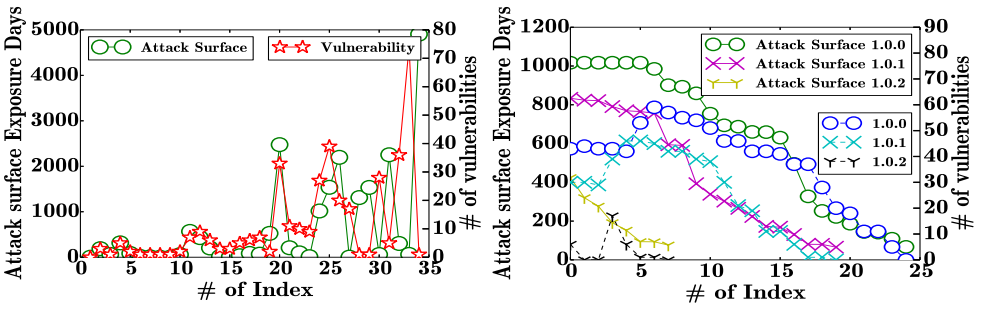
In this section, we first support our model for converting attack surface to attack probability with experimental results on the correlation between attack surface and vulnerabilities based on real software. We then conduct simulations to evaluate the performance of our heuristic algorithms proposed in Section 4.

4.1 Correlation between Attack Surface and Vulnerabilities

Since our model for converting attack surface to attack probability (presented in Section 2.1) 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 and their correlation results are presented in Figure 4(a). The name of each software can be found in the Appendix based on its index number. 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 2.1. In Figure 4(a), the left y-axis and the green line show the attack surface (converted to attack probability) multiplied by the days of exposure 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 show the number of vulnerabilities found for the same software in NVD [23].

From the results, we can see that there is a positive correlation between the number of vulnerabilities and 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



(a) (b)

Fig. 4: Correlation between Attack Surface and the Number of Vulnerabilities for Different Software (a) and Different Versions of OpenSSL (b)

among attackers, and the security expertise level of typical users of a software. For example, the index number 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 [2]. 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, the index number 34 is *Amanda*, a network-based backup system, which has only one vulnerability, even though its attack surface * 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 make the software less attractive to attackers.

Second, we examined 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 4(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 * exposure days for all three branches. The branch with larger attack surface * exposure days also has more vulnerabilities. The new versions inside each branch always have less vulnerabilities while attack surface * exposure days are also smaller. 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 version branch 1.0.2 is newly released since January 2015, so the attack surface * exposure days is not sufficient to create visible trends.

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 ongoing work will significantly expand the scope and scale of the experiments.

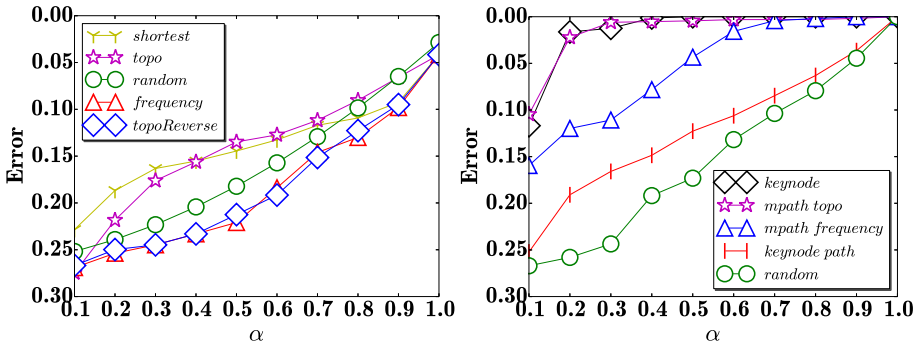
4.2 Performance of Heuristic Algorithms

In this section, we study the performance of our proposed heuristic algorithms via simulations. All simulation 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. All the resource graphs are created from small seed graphs based on realistic networks (e.g., the one shown in Figure 1), by increasing the number of hosts and resources in a random but realistic fashion.

The objective of the first two simulations is to evaluate the error rate of our simple heuristics (presented in Section 3.2) and heuristic algorithms (presented in Section 3.3). The error rate is defined in the same way as in the previous section ($\frac{|p_{true}-p_{cal}|}{p_{true}}$ where both p_{true} and p_{cal} are count-based values, as described at the end of Section 2.2). 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 numbers, is that evaluating a larger network naturally implies a larger budget will be require so a relative value will be more meaningful.

Figure 5(a) shows the error vs. the percentage of calculated resources (α) for simple heuristics and Figure 5(b) shows it for the heuristic algorithms. The y-axis is shown in reversed scale in both figures in order to show the increasing accuracy of those algorithms for a larger α. Figure 5(c) depicts the processing time of the algorithms. In all simulations, for each configuration, we repeat 500 times to obtain the average results.



(a) (b)

Fig. 5: The Cost vs. Error for Simple Heuristics (a) and for Heuristic Algorithms (b)

Results and Implications: From Figure 5 (a), we have following observations. First of all, with the increase of α, the error generally decreases, and when α increases to 1, which means we calculate all the resources in the network, the error of all the heuristics reaches 0 as expected. The green line with round markers is the baseline for comparison, which represents the results of the random choose heuristic and 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

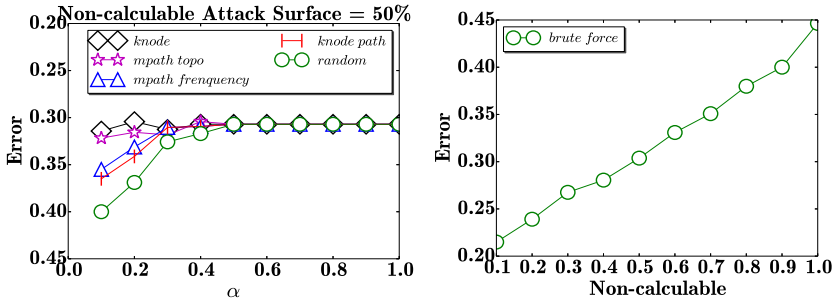
reversed topological order tends to choose resources equally among all the paths, since the paths converge towards the end of the graph. On the other hand, the topological order heuristic chooses from beginning nodes, which might converge into one path and give better results. The most accurate one in Figure 5(a) is the shortest path heuristic algorithm, which combines the topological order and shortest path heuristics together. The error rate of this algorithm becomes flat when it finishes calculating the shortest path and starts to calculate other resources.

Figure 5(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. Here the mpath frequency and mpath topo algorithms are the combination of m-shortest path heuristic with the frequency choose heuristic and the topological order heuristic, respectively. From the results we can see that the mpath topo algorithm has less error than mpath frequency. For the keynode heuristic algorithm, we tested 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 two have very different error rate, 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.

4.3 The Impact of Non-Calculatable Resources

To calculate the attack surface of network resources we need to have access to the source code. However, many resources are closed source applications for which calculating the attack surface would be infeasible. Therefore, the objective of this set of simulations is to examine the impact of non-calculable resources. Here we assume a brute force algorithm which calculates all the attack surface as the baseline for comparison. We assign the value 0.68 (which is the attack probability converted from the average value of all CVSS scores in NVD [23]) to those non-calculable resources. Figure 6(a) shows the performance of our algorithms when half of the resources are non-calculatable. In Figure 6(b), we use two brute force algorithms to study the impact of increasing ratio of non-calculable resources of which the first calculates all the attack surface values as a base line for comparison, and the second simply assigns 0.68 to non-calculable resources but calculates for all other resources.

Results and Implications: From the results we can see that, the trends of algorithms stay the same in the first simulation, with generally a larger error due to the non-calculable resources. When α is close to 0.4, the error becomes stable at 0.3, which matches the corresponding result in Figure 6(b), and the differences between algorithms become invisible. In Figure 6(b), the error increases almost linearly with the increase of the percentage of non-calculable resources. When non-calculable resources reach 100%, our metric essentially becomes the k-zero day safety metric [31], which still provides a useful measure even though it no longer discriminates different resources. Finally, we will also discuss future directions on calculating attack surface on binaries for closed source applications in Section 6.



(a) (b)

Fig. 6: The Error vs. α of Algorithms with 50% Non-Calculable Resources (a) and The Percentage of Non-Calculable Resources vs. Error (b)

5 Related Work

The concept of attack surface is originally proposed for specific software and requires domain-specific expertise to formulate and implement [10]. Later on, the concept is generalized using formal models and becomes applicable to all software [24]. Furthermore, it is refined and applied to large scale software, and its calculation can be assisted by automatically generated call graphs [25, 18]. Attack surface has attracted significant attentions over the years. It is used as a metric to evaluate Android's message-passing system [15], in kernel tailing [16], and also serves as a foundation in Moving Target Defense, which basically aims to change the attack surface over time [13, 12]. Others aim to expand the scope of this concept in other domains, such as the six-way attack surfaces between users, services, and cloud systems [9], and the approximation of attack surface for modern automobiles [5]. 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[4]. Stack traces from user crash reports is used to approximate attack surface automatically [29]. Despite such tremendous interest in the attack surface concept, to the best of our knowledge, little work exists on formally defining attack surface at the network level. 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 [35]. 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.

As to security metrics in general, there exist standardization efforts on vulnerability assessment including the Common Vulnerability Scoring System (CVSS) [21], which measures vulnerabilities in isolation. The NIST's efforts on standardizing security metrics are also given in [22] and more recently in [28]. The research on security metrics has attracted much attention lately [14]. Earlier work include the a metric in terms of time and efforts based on a Markov model [6]. More recently, several security metrics are proposed by combining CVSS scores based on attack graphs [30, 8]. The minimum efforts required for executing each exploit is used as a metric in [3, 26]. A mean time-

to-compromise metric is proposed based on the predator state-space model (SSM) used in the biological sciences in [17]. 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 [20], an effort on ordering different applications in a system by the seriousness of consequences of having a single zero day vulnerability [11], and more recently the k-zero day safety model [32, 31] and the network diversity model [33, 36] both attempt to model the risk of zero day vulnerabilities, but their common limitation is the lack of capability in distinguishing different resources' likelihood of having such vulnerabilities, which is the main contribution of this paper.

6 Limitations and Conclusion

An intuitive notion of attack surface at the network level has prevented applications from inheriting the precise 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. The correlation between attack surface and vulnerabilities was validated through our preliminary experimental results. We have also shown through algorithm design and simulations that the cost of calculating attack surface for network resources could be saved without losing too much accuracy. The limitations of our work and future directions are as follows.

- First, our experiments on the correlation between attack surface and vulnerabilities are still of relatively small scale and scope. Our future work will strengthen the results reported in this paper, and take into consideration other factors, such as market share data.
- Second, there lack automated and mature tools for assisting the calculation of attack surface, which can potential hinder the application of this concept at a higher abstraction level. 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 adapt emerging tools on binary analysis, such as library function identification and clone detection, in order to make estimating attack surface for binaries a reality.
- Fourth, we have not considered the average case network attack surface in the study of heuristic algorithms, and this will be a future direction.

DISCLAIMER

References

1. Computer world uk. http://www.computerworlduk.com/blogs/open-enterprise/open-source-has-won-3592314//.
2. Freetype. lhttps://en.wikipedia.org/wiki/FreeType.
3. D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st ACM QoP*, 2005.
4. Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin 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*, pages 274–277. ACM, 2012.
5. Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.
6. M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.
7. M. Frigault and L. Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of The 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.
8. M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of 4th ACM QoP*, 2008.
9. Nils Gruschka and Meiko Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *2010 IEEE 3rd international conference on cloud computing*, pages 276–279. IEEE, 2010.
10. M. Howard, J. Pincus, and J. Wing. Measuring relative attack surfaces. In *Workshop on Advanced Developments in Software and Systems Security*, 2003.
11. K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *Proceedings of ACSAC'09*, pages 117–126, 2009.
12. S. Jajodia, A.K. Ghosh, V. S. Subrahmanian, V. Swarup, C. Wang, and X.S. Wang. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. Springer, 2012.
13. S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang, and X.S. Wang. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*. Springer, 1st edition, 2011.
14. A. Jaquith. *Security Merics: Replacing Fear Uncertainity and Doubt*. Addison Wesley, 2007.
15. David Kantola, Erika Chin, Warren He, and David 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*, pages 69–80. ACM, 2012.
16. Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann, and Rüdiger Kapitza. Attack surface metrics and automated compile-time os kernel tailoring. In *NDSS*, 2013.
17. D.J. Leversage and E.J. Byres. Estimating a system's mean time-to-compromise. *IEEE Security and Privacy*, 6(1):52–60, 2008.
18. P.K. Manadhata and J.M. Wing. An attack surface metric. *IEEE Trans. Softw. Eng.*, 37(3):371–386, May 2011.
19. J. McHugh. Quality of protection: Measuring the unmeasurable? In *Proceedings of the 2nd ACM QoP*, pages 1–2, 2006.
20. M.A. McQueen, T.A. McQueen, W.F. Boyer, and M.R. Chaffin. Empirical estimates and observations of 0day vulnerabilities. *Hawaii International Conference on System Sciences*, 0:1–12, 2009.

21. P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6):85–89, 2006.
22. National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133, 1985.
23. National vulnerability database. available at: http://www.nvd.org, May 9, 2008.
24. J. Wing P. Manadhata. Measuring a system's attack surface. Technical Report CMU-CS-04-102, 2004.
25. J. Wing P. Manadhata. An attack surface metric. Technical Report CMU-CS-05-155, 2005.
26. 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*, pages 31–38, 2006.
27. Allan Reid, Jim Lorenz, and Cheryl A Schmidt. *Introducing Routing And Switching In The Enterprise, CCNA Discovery Learning Guide*. Cisco Press, 2008.
28. M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo. Security metrics guide for information technology systems. NIST Special Publication 800-55, 2003.
29. Christopher Theisen, Kim Herzig, Patrick Morrison, Brendan Murphy, and Laurie Williams. Approximating attack surfaces with stack traces. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 199–208. IEEE Press, 2015.
30. 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.
31. 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*, 11(1):30–44, 2013.
32. 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)*, pages 573–587, 2010.
33. 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*, pages 494–511, 2014.
34. Awad A Younis and Yashwant 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)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.
35. Awad A Younis, Yashwant K Malaiya, and Indrajit 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*, pages 1–8. IEEE, 2014.
36. 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 (TIFS)*, 11(5):1071–1086, 2016.

7 appendix

Index	Software
1	Libfm-1.2.3
2	apcupsd-3.14.13
3	sox-14.4.2
4	w3m-0.5.3
5	squashfs4.3
6	libtirpc-1.0.1
7	ultradefrag
8	fwbuilder-5.1.0.3599
9	dosbox-0.74
10	gnucash-2.6.7
11	tcl8.6.4
12	icinga-1.10.1
13	fuse-2.9.4
14	mcrypt-2.6.8
15	pnp4nagios-0.6.25
16	expat-2.1.0
17	flac-1.3.1
18	lcms2-2.7
19	e2fsprogs-1.42.13
20	libpng-1.6.19
21	unzip610b
22	mpg123-1.22.4
23	ganglia-3.7.2
24	nagios-4.1.1
25	clamav-0.98.7
26	net-snmp-5.4.5.pre1
27	flex-2.6.0
28	vice-2.4
29	gnuplot-5.0.1
30	zabbix-2.4.7
31	optipng-0.7.5
32	pcre2-10.20
33	freetype-2.6
34	amanda-3.3.7p1

Table 3: Tested Software