k-Zero Day Safety: Measuring the Security Risk of Networks against Unknown Attacks

Lingyu Wang[1], Sushil Jajodia[2], Anoop Singhal[3], and Steven Noel[2]

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

Abstract. The security risk of a network against unknown zero day attacks has been considered as something unmeasurable since software flaws are less predictable than hardware faults and the process of finding such flaws and developing exploits seems to be chaotic [12]. In this paper, we propose a novel security metric, *k-zero day safety*, based on the number of unknown zero day vulnerabilities. That is, the metric simply counts how many unknown vulnerabilities would be required for compromising a network asset, regardless of what vulnerabilities those might be. We formally define the metric based on an abstract model of networks and attacks. We then devise algorithms for computing the metric. Finally, we show the metric can quantify many existing practices in hardening a network.

1 Introduction

Today's critical infrastructures and enterprises increasingly rely on networked computer systems. Such systems must thus be secured against potential network intrusions. However, before we can improve the security of a network, we must be able to measure it, since *you cannot improve what you cannot measure*. A network security metric is desirable since it will allow for a direct measurement of how secure a network currently is, and how secure it would be after introducing new security mechanisms or configuration changes. Such a capability will make the effort of network hardening a science rather than an art.

Emerging efforts on network security metrics (Section 5 will review related work) typically assign numeric scores to vulnerabilities as their relative exploitability or likelihood. The assignment is usually based on known facts about each vulnerability (e.g., whether it requires an authenticated user account). However, such a methodology is no longer applicable when considering zero day vulnerabilities about which we have no prior knowledge or experience. In fact, a major criticism of existing efforts on security metrics is that unknown zero day vulnerabilities are unmeasurable [12]. First, the knowledge about a software system itself is not likely to help because unlike hardware faults, software flaws leading to vulnerabilities are known to be much less predictable. Second, modeling adversaries is not feasible either, because the process of finding flaws and developing exploits is believed to be chaotic. Third, existing metrics for known vulnerabilities are not helpful, because they measure the difficulty of *exploiting* a known vulnerability but not that of *finding* a zero day vulnerability.

The incapability of measuring unknown zero day vulnerabilities can potentially diminish the value of security mechanisms since an attacker can *simply step outside the implementation and do as he pleases* [12]. What is the value of a more secure configuration, if it is equally susceptible to zero day attacks? We thus fall into the agnosticism that security is not quantifiable until we can fix all security flaws (by then we certainly do not need any security metric, either).

We propose a novel security metric, *k-zero day safety*, to address this issue. Instead of attempting to measure *which* zero day vulnerability is more likely, our metric counts *how many* distinct zero day vul-

nerabilities are required to compromise a network asset [1]. A larger number will indicate a relatively more secure network, since the likelihood of having more unknown vulnerabilities all available at the same time, applicable to the same network, and exploitable by the same attacker, will be lower. Based on an abstract model of networks and attacks, we formally define the metric and prove it to satisfy the three algebraic properties of a metric function. We then design algorithms for computing the metric. Finally, we show the metric can quantify many existing practices in network hardening and discuss practical issues in instantiating the model.

The contribution of this work is twofold. First, to the best of our knowledge, this is the first effort capable of quantifying the security risk of a network against unknown zero day attacks. Second, we believe the metric would bring about new opportunities to the evaluation, hardening, and design of secure networks.

In the rest of this paper, we first build intuitions through a running example. We then present a model and define the metric in Section 2, design and analyze algorithms in Section 3, discuss network hardening and model instantiation in Section 4, review related work in Section 5, and finally conclude the paper in Section 6. The proof of theorems can be found in Appendices.

1.1 Motivating Example

The left-hand side of Figure 1 shows a toy example where host 1 provides an HTTP service ($http$) and a secure shell service (ssh), and host 2 provides only ssh. The firewall allows traffic to and from host 1, but only connections originated from host 2. Assume the main security concern is over the root privilege on host 2. Clearly, if all the services are free of known vulnerabilities, a vulnerability scanner or attack graph will both lead to the same conclusion, that is, the network is secure (an attacker on host 0 can never obtain the root privilege on host 2), and no additional network hardening effort is necessary.
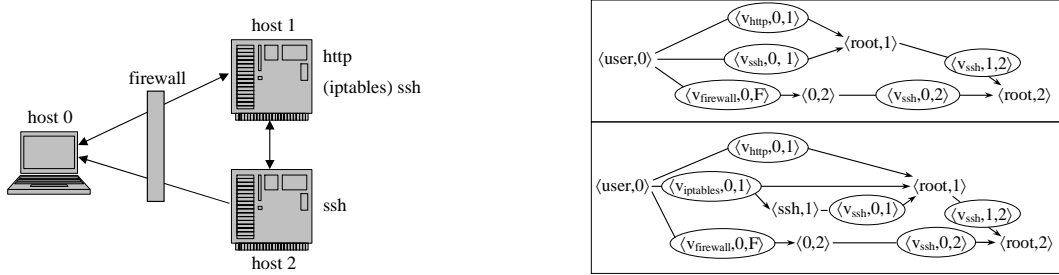

Fig. 1. Network Configuration and Sequences of Zero Day Attacks

However, we shall reach a different conclusion by considering *how many distinct zero day attacks the network can resist*. The upper-right corner of Figure 1 shows three sequences of zero day attacks leading to $\langle root, 2 \rangle$ (each pair denotes a condition and each triple inside oval denotes the exploitation of a zero day vulnerability): An attacker on host 0 can exploit a zero day vulnerability in either $http$ or ssh on host 1 to obtain the root privilege; using host 1 as a stepping stone, he/she can exploit a zero day vulnerability in ssh on host 2 to reach $\langle root, 2 \rangle$; alternatively, he/she can exploit a zero day vulnerability in the firewall (e.g., a weak password in its Web-base remote administration interface) to re-establish the blocked connection to host 2 and then exploit ssh on host 2. The network can resist at most one zero day attack since the second sequence only requires one unique zero day vulnerability in ssh (on both host 1 and 2).

Now consider hardening the network by using iptables rules ($iptables$) to allow only specific hosts, not including host 0, to connect to ssh on host 1. The lower-right corner of Figure 1 shows four sequences of zero day attacks (the two new sequences indicate exploiting a zero day vulnerability in $iptables$ to either connect to ssh, or obtain the root privilege, on host 1). It can be observed that all four sequences now require

---

[1] In our model, an asset is a general concept that may encompass one or more aspects of security, such as confidentiality, integrity, and availability.

two distinct zero day vulnerabilities. The seemingly unnecessary hardening effort thus allows the network to resist one more zero day attack. The hardened network can be considered relatively more secure, since the likelihood of having more zero day vulnerabilities available at the same time, in the same network, and exploitable by the same attacker, will be lower [2]. Therefore, the number of distinct zero day vulnerabilities can be used to measure the relative security risk of different networks, which may otherwise be indistinguishable by existing techniques. Those discussions, however, clearly oversimplify many issues, which will be addressed in the rest of this paper.

2 Modeling k-Zero Day safety

In this section, we define the k-zero day safety metric based on an abstract model of network components. We shall delay to Section 4 the discussion of practical issues in instantiating the abstract model based on a real world network.

2.1 The Network Model

Definition 1 gives an abstract model of network components relevant to measuring zero day attacks (all notations will later be summarized in Table 1). The model will allow us to formally define and reason about the proposed metric.

Definition 1 (Network). *Our network model has the following components:*

- *H, S, and P, which denotes the set of hosts (computers and networking devices), services, and privileges, respectively.*
- *$serv(.) : H \rightarrow 2^S$ and $priv(.) : H \rightarrow 2^P$, which denotes a function that maps each host to a set of services and that of privileges, respectively.*
- *$conn \subseteq H \times H$, and $\preceq \subseteq priv(h) \times priv(h)$, which denotes a connectivity relation and a privilege hierarchy relation, respectively.*

Here hosts are meant to also include networking devices because such devices are vulnerable to zero day attacks, and a compromised device may re-enable accesses to blocked services (e.g., the firewall in Figure 1). Also, tightly-coupled systems (e.g., a server hosting multiple replicas of a virtual host under the Byzantine-Fault Tolerance algorithm [3]) should be regarded as a single host, since we shall only consider causal relationships between hosts.

A service in the model is either remotely accessible over the network, in which case called a *remote service*, or used to disable a remote service or network connection, in which case called a *security service*. The model does not include services or applications that can only be exploited locally for a privilege escalation (modeling such applications may not be feasible at all considering that an attacker may install his/her own applications after obtaining accesses to a host). On the other hand, the model includes remote services and connectivity currently disabled by security services, since the former may be re-enabled through zero day attacks on the latter (e.g., ssh behind $iptables$ in Figure 1).

In the model, privileges are meant to include those under which services are running and those that can potentially be obtained through a privilege escalation. The purpose of including the latter is not to model privilege escalation itself but to model the strength of isolation techniques (e.g., sandboxing or virtual machines) that may prevent such an escalation, as we shall elaborate shortly.

Example 1. In Figure 1, we have

---

[2] This likelihood would decrease exponentially in the number of vulnerabilities if such vulnerabilities can be modeled as i.i.d. random variables, but we shall not assume any specific model since the process of developing exploits is believed to be chaotic [12].

- $H = \{0, 1, 2, F\}$ (F denotes the firewall),
- $conn = \{\langle 0, F \rangle, \langle 0, 1 \rangle, \langle 0, 2 \rangle, \langle 1, F \rangle, \langle 1, 0 \rangle, \langle 1, 2 \rangle, \langle 2, F \rangle, \langle 2, 0 \rangle, \langle 2, 1 \rangle\}$ (we include $\langle 0, 2 \rangle$ since it can be enabled by a zero day attack on the firewall),
- $serv(1) = \{http, ssh, iptables\}$, $serv(2) = \{ssh\}$, and $serv(F) = \{firewall\}$ ($firewall$ is a security service and it disables connection $\langle 0, 2 \rangle$),
- $priv(1) = priv(2) = \{user, root\}$.

2.2 The Zero Day Attack Model

The very notion of *unknown* zero day vulnerability means we cannot assume any vulnerability-specific property, such as the likelihood or severity. We can, however, assume generic properties common to vulnerabilities, as in Definition 2.

Definition 2 (Zero Day Vulnerability). *A zero day vulnerability is a vulnerability whose details are unknown except that it satisfies the following* [3].

1. *It cannot be exploited unless*
    (a) *a network connection exists between the source and destination hosts,*
    (b) *a remote service with the vulnerability exists on the destination host,*
    (c) *and the attacker already has a privilege on the source host.*
2. *Its exploitation can potentially yield any privilege on the destination host.*

The assumptions essentially depict a worst-case scenario about the pre- and post-conditions, respectively, of exploiting a zero day vulnerability. That is, a particular zero day vulnerability may in reality require stronger pre-conditions while implying weaker post-conditions than those stated above. This fact ensures our metric to always yield a conservative result (the metric can be extended to benefit from weaker assumptions when they can be safely made). For a similar purpose, we shall assign one zero day vulnerability to each service although in reality a service may have more vulnerabilities (note that a more conservative result of a metric is one that requires less zero day vulnerabilities).

We more formally state above assumptions in Definition 3 and 4. In Definition 3, the zero day exploit of a privilege will act as a placeholder when we later model isolation techniques. In Definition 4, unlike the exploit of a known vulnerability which has its unique pre- and post-conditions, all zero day exploits share the same hard-coded conditions, as assumed above. Also note that the zero day exploit of each security service has additional post-conditions, which indicates the exploit will re-enable the disabled conditions. For zero day exploits of a privilege p, the pre-conditions include the privilege of every service, unless if that privilege already implies p (in which case including it will result in redundancy). This follows from our assumption that a zero day exploit may potentially lead to any priviledge.

Definition 3 (Zero Day Exploit). *For each $h \in H$ and $x \in (serv(h) \cup priv(h))$, denote by v_x a zero day vulnerability. A zero day exploit is the triple*

- $\langle v_s, h, h' \rangle$ *where $\langle h, h' \rangle \in conn$ and $s \in serv(h')$, or*
- $\langle v_p, h, h \rangle$ *where $p \in priv(h)$.*

Definition 4 (Condition). *Denote by E_0 the set of all zero day exploits, C_0 the set of conditions ($conn \cup \{\langle x, h \rangle : h \in H, x \in serv(h) \cup priv(h)\}$), and define functions $pre(.) : E_0 \to C_0$ and $post(.) : E_0 \to C_0$ as*

---

[3] While we shall focus on such a restrictive model of zero-day vulnerabilities in this paper, an interesting future direction is to extend the model to address other types of zero-day vulnerabilities, such as a time bomb whose exploitation does not require a network connection.

- $pre(\langle v_s, h, h' \rangle) = \{\langle h, h' \rangle, \langle s, h' \rangle, \langle p_{min}, h \rangle\}$ *for each* $s \in serv(h)$, *where* p_{min} *is the least privilege on* h.
- $pre(\langle v_p, h, h \rangle) = \{p_s : s \in serv(h), \neg(p \preceq p_s)\}$ *for each* $p \in priv(h)$.
- $post(\langle v_s, h, h' \rangle) = \{p_s\}$ *for each remote service* s *with privilege* p_s.
- $post(\langle v_s, h, h' \rangle) = \{p_s\} \cup C_s$ *for each security service* s, *where* C_s *is the set of conditions disabled by* s.
- $post(\langle v_p, h, h \rangle) = \{\langle p, h \rangle\}$ *for each* $p \in priv(h)$.

In Definition 5, a zero day attack graph is composed by relating both exploits of known vulnerabilities and zero day exploits through common pre- and post-conditions. In a zero day attack graph, the exploits of known vulnerabilities can be considered as *shortcuts* that help attackers to satisfy a condition with less zero day exploits. Therefore, exploits of known vulnerabilities here may also be a trust relationship or a misconfigured application, as long as they serve the same purpose of a shortcut for bypassing zero day exploits.

Definition 5 (Zero Day Attack Graph). *Given the set of exploits of known vulnerabilities* E_1 *and their pre- and post-conditions* C_1, *let* $E = E_0 \cup E_1$, $C = C_0 \cup C_1$, *and extend* $pre(.)$ *and* $post(.)$ *to* $E \to C$ *(as the union of relations). The directed graph* $G = \langle E \cup C, \{\langle x, y \rangle : (y \in E \land x \in pre(y)) \lor (x \in E \land y \in post(x))\}\rangle$ *is called a zero day attack graph.*

In Definition 6, the notion of *initial condition* serves two purposes. First, it includes all conditions that are not post-conditions of any exploit (which is the usual interpretation of the notion). Second, it is meant to also include conditions that may be satisfied as the result of insider attacks or user mistakes. In another word, the effect of such attacks or mistakes is modeled as the capability of satisfying post-conditions of an exploit without first executing the exploit [4]. Also note that in the definition, an attack sequence is defined as a total order, which means multiple attack sequences may lead to the same asset. However, this is not a limitation since our metric will not require the attack sequence to be unique, as we shall show.

Instead of the usual way of modeling an asset as a single condition, we take a more general approach. The logical connectives $\land$, $\lor$, and $\neg$ respectively model cases where multiple conditions must be satisfied altogether to cause a damage (e.g., the availability of a file with multiple backups on different hosts), cases where satisfying at least one condition will cause the damage (e.g., the confidentiality of the aforementioned file), and cases where conditions are not to be satisfied during an attack (for example, conditions that will trigger an alarm). The asset value is introduced as the relative weight of independent assets.

Definition 6 (Initial Condition, Attack Sequence, and Asset). *Given a zero day attack graph* G,

- *the set of initial conditions is given as any* $C_I \subseteq C$ *satisfying* $C_I \supseteq \{c : (\forall e \in E)(c \notin post(e))\}$,
- *an attack sequence is any sequence of exploits* $e_1, e_2, \ldots, e_j$ *satisfying* $(\forall i \in [1, j])\, (\forall c \in pre(e_i))\, (c \in C_I) \lor (\exists x \in [1, i-1]\, c \in post(e_x))$,
- *an asset* a *is any logical proposition composed of conditions and the logic connectives* $\land$, $\lor$, *and* $\neg$ *for which an asset value* $v(a)$ *is given through a function* $v(.) : A \to [0, \infty)$ *where* A *denotes the set of all assets, and*
- *define a function* $seq(.) : A \to 2^Q$ *as* $seq(a) = \{e_1, e_2, \ldots, e_j : a \in post(e_j)\}$ *where* Q *denotes the set of all attack sequences.*

Example 2. Figure 2 shows the zero day attack graph of our running example,

---

[4] In a broader sense, we should improve robustness of the model such that it will fail gracefully when assumptions fail, which is beyond the scope of this paper.

– if we do not consider insider attacks or user mistakes, the following attack sequences will lead to the asset $\langle root, 2 \rangle$.
   1. $\langle v_{http}, 0, 1 \rangle, \langle v_{ssh}, 1, 2 \rangle, \langle v_{root}, 2, 2 \rangle$
   2. $\langle v_{iptables}, 0, 1 \rangle, \langle v_{ssh}, 1, 2 \rangle, \langle v_{root}, 2, 2 \rangle$
   3. $\langle v_{iptables}, 0, 1 \rangle, \langle v_{ssh}, 0, 1 \rangle, \langle v_{ssh}, 1, 2 \rangle, \langle v_{root}, 2, 2 \rangle$
   4. $\langle v_{firewall}, 0, F \rangle, \langle v_{ssh}, 0, 2 \rangle, \langle v_{root}, 2, 2 \rangle$
– if we consider insider attacks on host 1, only sequence $\langle v_{ssh}, 1, 2 \rangle, \langle v_{root}, 2, 2 \rangle$ and the fourth attack sequence above will be needed to compromise $\langle root, 2 \rangle$.
– if we consider a different asset $\langle root, 1 \rangle \wedge \langle root, 2 \rangle$, then only the first three attack sequences above can compromise this asset.
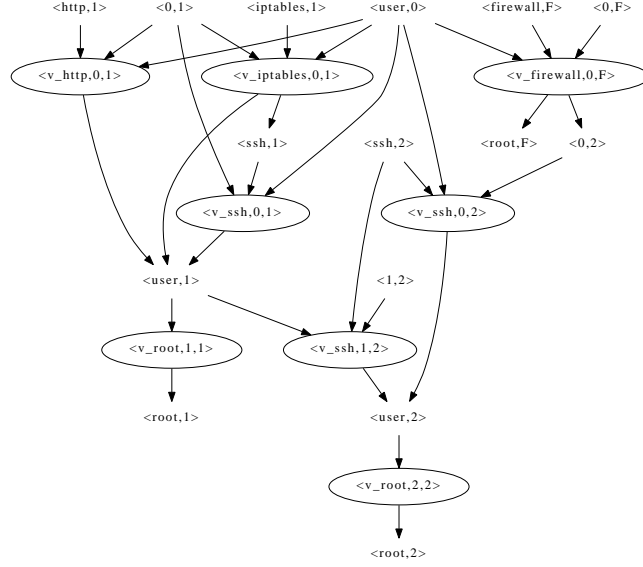

Fig. 2. An Example of Zero Day Attack Graph

2.3 The k-Zero Day Safety Model

In Definition 7, the relation $\equiv_v$ models two distinct cases in which two zero day exploits should only be counted once. First, both exploits involve the same zero day vulnerability. Second, the exploit of a service is related to the exploit of a privilege to indicate that the former will directly yield the privilege due to the lack of isolation between the two (note that we do not model the details of any involved privilege escalation). A probability can be associated to relation $\equiv_v$ to indicate the degree of similarity or isolation, when such information is available. Although the relationship between exploits has distinct meanings in those two cases, the effect of such a relationship towards our metric will be the same. Therefore, the relation $\equiv_v$ models such relationships in a unified way.

Given two sets of zero day exploits, the function $k0d(.)$ counts how many exploits in their symmetric difference are distinct (that is, these exploits cannot be related through $\equiv_v$). In particular, if one of the sets is empty, then the function $k0d(.)$ will yield the number of distinct zero day exploits in the other set. When a probabilistic approach is adopted in defining the relation $\equiv_v$, the function $k0d(.)$ can be revised to give the expected value (mean). The reason of defining the function over the symmetric difference of two sets is given in Theorem 1.

Definition 7 (Relation $\equiv_v$ and Metric Function $k0d(.)$).

6

- *Define a relation $\equiv_v \subseteq E_0 \times E_0$ such that $e \equiv_v e'$ indicates either e and e' are exploits of the same zero day vulnerability, or $e = \langle v_s, h_1, h_2 \rangle$, $e' = \langle v_p, h_2, h_2 \rangle$ and exploiting s yields p. We say e and e' are distinct if $e \not\equiv_v e'$.*
- *Define a function $k0d(.) : 2^{E_0} \times 2^{E_0} \rightarrow [0, \infty]$ as $k0d(F, F') = max(\{ |F''| : F'' \subseteq (F \triangle F'), (\forall e_1, e_2 \in F'') (e_1 \not\equiv_v e_2) \})$ where $|F''|$ denotes the cardinality of F'', $max(.)$ denotes the maximum value in a set, and $F \triangle F'$ denotes the symmetric difference (that is, $(F \setminus F') \cup (F' \setminus F)$).*

Theorem 1. *The function $k0d(.)$ is a metric.*

In Definition 8, we apply the metric $k0d(.)$ to assets, sets of assets, and a network. First, $k0d(a)$ indicates the minimum number of distinct zero day exploits required to compromise a. This number is unique for each asset, although multiple attack sequences may compromise the asset. The empty set in the definition can be interpreted as the conjunction of all initial conditions (which can always be compromised without any zero day exploit). Second, the metric is applied to a set of independent assets by taking the weighted average with asset values as the weight. Finally, by applying the metric to all assets, we obtain a measurement of a network's resistance to potential zero day attacks.

Definition 8 (*k-Zero Day Safety*). *Given a zero day attack graph G, the set of initial conditions C_I, and the set of assets A,*

- *for any $a \in A$, we use $k0d(a)$ for $min(\{k0d(q \cap E_0, \phi) : q \in seq(a)\})$ where $min(.)$ denotes the minimum value in a set and q stands for both a sequence and a set. For any $k \in [0, kod(a))$, we say a is k-zero day safe.*
- *given any $A' \subseteq A$, we use $k0d(A')$ for $\sum_{a \in A'} (k0d(a) \cdot v(a)) / \sum_{a \in A'} v(a)$. For any $k \in [0, kod(A'))$, we say A' is k-zero day safe.*
- *in particular, when $A' = A$, we say the network is k-zero day safe.*

Example 3. For the running example, suppose all exploits of services involve distinct vulnerabilities except $\langle v_{ssh}, 0, 1 \rangle$, $\langle v_{ssh}, 1, 2 \rangle$, and $\langle v_{ssh}, 0, 2 \rangle$. Assume ssh and $http$ are not protected by isolation but $iptables$ is protected. Then, the relation $\equiv_v$ is shown in the left-hand side of Table 1 where 1 indicates two exploits are related and 0 the opposite (or, by adopting a probabilistic approach, these can be regarded as the probabilities associated with the relation $\equiv_v$).

	$\langle v_{iptables}, 0, 1 \rangle$	$\langle v_{http}, 0, 1 \rangle$	$\langle v_{ssh}, 0, 1 \rangle$	$\langle v_{root}, 1, 1 \rangle$	$\langle v_{ssh}, 1, 2 \rangle$	$\langle v_{firewall}, 0, F \rangle$	$\langle v_{ssh}, 0, 2 \rangle$	$\langle v_{root}, 2, 2 \rangle$
$\langle v_{iptables}, 0, 1 \rangle$	1	0	0	0	0	0	0	0
$\langle v_{http}, 0, 1 \rangle$	0	1	0	1	0	0	0	0
$\langle v_{ssh}, 0, 1 \rangle$	0	0	1	1	1	0	1	0
$\langle v_{root}, 1, 1 \rangle$	0	1	1	1	0	0	0	0
$\langle v_{ssh}, 1, 2 \rangle$	0	0	1	0	1	0	1	1
$\langle v_{firewall}, 0, F \rangle$	0	0	0	0	0	1	0	0
$\langle v_{ssh}, 0, 2 \rangle$	0	0	1	0	1	0	1	1
$\langle v_{root}, 2, 2 \rangle$	0	0	0	0	1	0	1	1

Notation	Explanation
H, h	A set of hosts, a host
S, s	A set of services, a service
P, p	A set of privileges, a privilege
$serv(.)$	Services on a host
$priv(.)$	Privileges on a host
$conn$	Connectivity
v_s, v_p	Zero day vulnerability
$\langle v_x, h, h' \rangle$	Zero day exploit
$pre(.), post(.)$	Pre- and post-conditions
G	Zero day attack graph
C_I	Initial conditions
$e_1, e_2, \ldots, e_j$	Attack sequence
A	Assets
$seq(a)$	Attack sequences compromising a
$\equiv_v$	Relation of non-distinct exploits
$k0d(.)$	The k-zero day safety metric

Table 1. An Example of Relation $\equiv_v$ (Left) and the Notation Table (Right)

3 Computing k-Zero Day Safety

This section presents algorithms for computing the k-zero day safety.

3.1 Computing the Value of k

To compute the k-zero day safety of a network, we first derive a logic proposition of each asset in terms of exploits. Then, each conjunctive clause in the disjunctive normal form (DNF) of the derived proposition will correspond to a minimal set of exploits that jointly compromise the asset. The value of k can then be decided by applying the metric $k0d(.)$ to each such conjunctive clause.

More precisely, we interpret a given zero day attack graph as a logic program by regarding each exploit or condition as a Boolean variable and by having a logic proposition $c \leftarrow .$ for each initial condition c, a proposition $e \leftarrow \bigwedge_{c \in pre(e)} c$ and a set of propositions $\{c \leftarrow e : c \in post(e)\}$ for each pre- and post-condition relationship, respectively. We can then apply Procedure $k0d\_Bwd$ shown in Figure 3 to obtain the value of k. The main loop (lines 1-8) computes the k-zero day safety for each asset. The results of all iterations are aggregated as the final output (line 9). The inner loop (lines 3-6) repetitively applies the afore-mentioned logic propositions to derive a formula, which is converted into its DNF (line 7) from which the k-zero day safety is computed (line 8).

Procedure $k0d\_Bwd$
Input: A zero day attack graph G, a set of assets A with the valuation function $v(.)$
Output: A non-negative real number k
Method:
1.   **For** each asset $a \in A$
2.   **Let** L be the logic proposition representing a
3.   **While** at least one of the following is possible, do
4.   **Replace** each initial condition c with $TRUE$
5.   **Replace** each condition c with $\bigvee_{e \in \{e' : c \in post(e')\}} e$
6.   **Replace** each non-negated exploit e with $e \wedge (\bigwedge_{c \in pre(e)} c)$
7.   **Let** $L_1 \vee L_2 \vee \ldots L_n$ be the DNF of L
8.   **Let** $k_a = min(\{k0d(F_i \cap E_0, \phi) : F_i$ is the set of non-negated exploits in $L_i, 1 \leq i \leq n\})$
9.   **Return** $\sum_{a \in A}(k_a \cdot v(a))/\sum_{a \in A} v(a)$

Fig. 3. Computing the Value of k

Complexity The procedure's worst-case complexity is exponential in the size of the zero day attack graph. Specifically, the complexity is dominated by the size of the derived proposition L and its DNF; both may be exponential. Indeed, Theorem 2 shows that the problem of computing k-zero day safety is NP-hard.

Theorem 2. *Given a zero day attack graph and an asset a, finding an attack sequence $q \in seq(a)$ to minimize $k0d(q \cap E_0, \phi)$ is NP-complete.*

Note that the intractability result here only implies that a single algorithm is not likely to be found to efficiently determine k for all possible inputs (that is, arbitrary zero day attack graphs). However, efficient solutions still exist for practical purposes. We shall examine such cases in the following.

3.2 Determining k-Zero Day Safety for a Given Small k

For many practical purposes, it may suffice to know that every asset in a network is k-zero day safe for a given value of k, even though the network may in reality be k'-zero day safe for some unknown $k' > k$ (note that we have shown determining k' to be intractable). We now describe a solution whose complexity is polynomial in the size of a zero day attack graph if k is a constant compared to this size. Roughly speaking, we attempt to compromise each asset with less than k distinct zero day exploits through a forward search of limited depth. The asset is not k-zero day safe if any branch of the search succeeds, and vice versa.

Specifically, Figure 4 shows the recursive Procedure $k0d\_Fwd$ with two base cases (lines 1-2 and 3-4, respectively) and one recursive case (lines 5-9). In the first base case, the procedure returns $FALSE$

when asset a can be compromised with less than k distinct zero day exploits in T_e. The Sub-Procedure $k0d\_Reachable$ expands T_e with all reachable known exploits since they do not count in terms of the $k0d(.)$ metric. In the second base case, the procedure returns $TRUE$ when the set T_e already has more than k distinct zero day exploits (regardless of whether a can be satisfied with T_c). Note that a for loop is used in the main procedure and a while loop in the sub-procedure because we need to exhaustively follow through all possible sequences of zero day exploits, whereas we can safely ignore the order of adjacent known exploits.

Procedure $k0d\_Fwd$
Input: A zero day attack graph G, an asset a, a real number $k > 0$, $T_e = \phi$, $T_c = C_I$
      $//T_e$ and T_c include the exploits and conditions visited so far, respectively
Output: $TRUE$, if $k0d(a) > k$; $FALSE$, otherwise
Method:

1. **If** $k0d\_reachable(T_e, T_c) \wedge k0d(T_e) < k$ **Sub-Procedure** $k0d\_Reachable$
2.   **Return** $FALSE$ **Input:** T_e, T_c
3. **ElseIf** $k0d(T_e) \geq k$ **Output:** $TRUE$ or $FALSE$
4.   **Return** $TRUE$ **Method:**
5. **Else**   10.   **While** $(\exists e \in E_1 \setminus T_e)(pre(e) \subseteq T_c)$
6.   **For** each $e \in E_0 \setminus T_e$ satisfying $pre(e) \subseteq T_c$ 11.   **Let** $T_e = T_e \cup \{e\}$
7.   **If** $\neg\, k0d\_Fwd(G, a, k, T_e \cup \{e\}, T_c \cup post(e))$ 12.   **Let** $T_c = T_c \cup post(e)$
8.   **Return** $FALSE$ 13. **Return** $(\bigwedge_{c \in T_c} c \to a)$
9.   **Return** $TRUE$

Fig. 4. Determining k-Zero Day Safety for a Given k

The main procedure enters the recursive case only when T_e includes less than k distinct zero day exploits and a cannot be satisfied with T_c. In this case, the Sub-Procedure $k0d\_Reachable$ must have already added all known exploits and their post-conditions to T_e and T_c, respectively. Now the main procedure iteratively visits each zero day exploit e reachable from T_c (line 6), and starts a recursive search from e (line 7). If no such e exists, the procedure will return $TRUE$ indicating the end of a sequence is reached (line 9). If any branch of the search succeeds, $FALSE$ will be recursively returned to indicate a is not k-zero day safe (line 8); otherwise, $TRUE$ is returned (line 9).

Note that a for loop is used in the main procedure, whereas a while loop in the sub-procedure. The reason is that while all possible sequences of zero day exploits must be searched through, the order of adjacent known exploits is not important. For the former, a for loop is sufficent since each reachable zero day exploit e is not added to T_e but used for recursion (line 7) so the number of exploits visited at line 6 is fixed. For the latter, each known exploit is immediately added to T_e (line 11) so the set of known exploits visited at line 10 may continually expand after each iteration, which requires a while loop.

More precisely, it adds to T_e every known exploit e that can be executed with the set of conditions T_c (initially set to be the set initial conditions C_I) satisfied (line 10-11). The post-conditions of e are then added to T_c (line 11), which again may lead to more known exploits to be added to T_e. This loop will terminates when no more known exploits can be added. Then, $TRUE$ will be returned if satisfying all the conditions in T_c implies satisfying (the logic propostion representing) asset a; $FALSE$ will be returned, otherwise (line 12).

Complexity To find reachable known exploits from E_1, the sub-procedure will check the pre-conditions of each known exploit, which takes time $O(|C| \cdot |E_1|)$. This will be repeated upon adding an exploit to T_e and its post-conditions to T_c. Therefore, $k0d\_Reachable$ takes time $O(|C| \cdot |E_1|^2)$, which is also the complexity for the base cases of the main procedure since it dominates the complexity of other steps. For the recursive case, we have the recurrence formula $t = O(|C| \cdot |E_1|^2) + |E_0| \cdot t'$ where t and t' denote the complexity of the recursive case and that of each recursive call. Since the recursive case cannot be entered unless $k0d(T_e) < k$ and each recursive call will add one more zero day exploit to T_e, the maximum layers of recursion can be written as $l = max(\{|q| : q$ is an attack sequence satisfying $k0d(q, \phi) < k + 1\})$. Solving the recurrence formula, we have that $t = |C| \cdot |E_1|^2 \cdot |E_0|^l$. Therefore, the complexity is polynomial in the size of the zero day attack graph if k is a constant.

This procedure can be modified to compute the exact value of k by removing both base cases (lines 1-4) so the search will only stop at the end of each attack sequence. We can thus find q minimizing $k0d(q, \phi)$ for each asset a, and average the results to be the value of k. While the worst-case complexity of (the modified version of) Procedure $k0d\_Fwd$ and that of Procedure $k0d\_Bwd$ are both exponential, the two may be relatively more efficient for different cases. For example, Procedure $k0d\_Bwd$ may be more efficient when the given set of assets cover most attack sequences .

Specifically, Procedure $k0d\_Bwd$ is more focused in the sense that all the visited attack sequences will lead to an asset a, whereas the above modified version of Procedure $k0d\_Fwd$ may waste time on many irrelevant attack sequences that cannot satisfy any asset (note that although the latter may be very efficient if it happens to search an attack sequence q that minimizes $k0d(q, \phi)$, such a best-case scenario is not meaningful for complexity comparison). Therefore, the latter may be efficient only when the given set of assets cover most attack sequences. On the other hand, while the complexity of Procedure $k0d\_Bwd$ is exponential in the size of the whole zero day attack graph, that of Procedure $k0d\_Fwd$ is only in the number of zero day exploits (due to the fact that the latter ignores the logic relationships between known exploits). Therefore, the latter may be more efficient if $|E_0| << |E_1|$ is true.

3.3 Computing k-Zero Day Safety as Shortest Paths in a DAG

Although it is intractable to compute k for arbitrary zero day attack graphs, efficient solutions may exist for those satisfying special properties. We now show such a case where the problem can be reduced to that of finding shortest paths in a directed acyclic graph (DAG). Roughly speaking, we make two assumptions: First, most exploits will only require one condition on the remote host (e.g., when a host is only used as a stepping stone, the condition could be a user privilege on that host); second, zero day exploits will be distinct unless they are on the same or adjacent hosts. Next, we informally describe our method while leaving the detailed algorithm and complexity analysis to Appendix C.

The first assumption implies that we can derive a logical proposition (as in Procedure $k0d\_Bwd$) separately for each host. In the resultant DNF, each conjunctive clause will include at most one condition involving a remote host, which means the asset can be expressed as a disjunction of conditions (without considering exploits). We can thus repeat the same reasoning by regarding each such condition as an asset on the involved remote host. Since the relationships between all conditions are now disjunctive, we can regard each condition as the vertex of a DAG (recall that cycles will be avoided) with their disjunctive relationships as edges, and exploits in the same conjunctive clause as edge weights.

In the weighted DAG, determining the value of k amounts to finding the *shortest* path along which the function $k0d(.)$ applied to all zero day exploits will yield the minimum value. During a backward search, we keep two parts of a distance for each edge: For those zero day exploits that may later be related to others through $\equiv_v$, we keep them in a set since the function $k0d(.)$ can not yet be applied; for other exploits, we only keep the result value of applying $k0d(.)$. The second assumption above essentially ensures that the first part of the edge distance will not grow quickly. The shortest distance can then be obtained using a standard algorithm [5], taking polynomial time (more precisely, the complexity is shown to be $|H|^4 \cdot |E_0|$ in Appendix C).

4 Discussions

In this section, we demonstrate the power of our metric through an example application, network hardening, and discuss issues in instantiating the model.

Network Hardening Using the Metric Based on the proposed metric, *network hardening* can be defined as making a network k-zero day safe for a larger k. Such a concept generalizes the existing qualitative approach

in [23], which essentially achieves $k > 0$. Moreover, the metric immediately imply a collection of hardening options. To see this, we first unfold k based on the model:

$$k = k0d(A) = \sum_{a \in A} (k0d(a) \cdot v(a)) / \sum_{a \in A} v(a) \qquad (1)$$

$$k0d(a) = min(\{k0d(q \cap E_0, \phi) : q \in seq(a)\}) \qquad (2)$$

$$k0d(q \cap E_0, \phi') = max(\{\ |F|\ : F \subseteq q \cap E_0, (\forall e_1, e_2 \in F)\ (e_1 \not\equiv_v e_2)\}) \qquad (3)$$

$$seq(a) = \{e_1, e_2, \ldots, e_j : a \in post(e_j), \qquad (4)$$

$$(\forall i \in [1, j])\ (\forall c \in pre(e_i))\ (c \in C_I) \lor (\exists x \in [1, i-1]\ c \in post(e_x))\} \qquad (5)$$

Therefore, k can be increased by:

– Increasing the diversity of services such that exploits will involve more distinct zero-day vulnerabilities (and no longer related by $\equiv_v$) in Equation (3).
– Strengthening isolation techniques for a similar effect as above.
– Disabling initial conditions (e.g., removing a service or a connection) in C_I to yield longer attack sequences in above line (5) (part of Equation (4)).
– Enforcing more strict access control policies to lessen the risk of insider attacks or user mistakes (thus removing conditions from C_I in line (5)).
– Protecting assets with backups (conjunction of conditions) and detection efforts (negation of conditions) to yield a longer sequence in Equation (4).
– Introducing more security services to regulate accesses to remote services in such a way that a longer sequence can be obtained in Equation (4).
– Patching known vulnerabilities such that less shortcuts for bypassing zero day exploits yield a longer sequence in Equation (4).
– Prioritizing the above options based on the asset values in Equation (1) and shortest attack sequences in Equation (2).

Clearly, these hardening options closely match current practices, such as the so-called *layered defense*, *defense in depth*, *security through virtualization*, and *security through diversity* approaches. However, their effectiveness [5] can now be *quantified* in a simple, intuitive way; their cost can now be more easily justified, not based upon speculation or good will, but simply with a larger k.

Instantiating the Model Since the proposed metric and algorithms are based on an abstract model of networks, how to instantiate the model for given networks is an equally important (and admittedly difficult) issue. We now address several key aspects of the issue while leaving more research to future work.

– While instantiating the model, an uncertain situation can be dealt with by either taking a conservative assumption under which the metric yields a lower k (e.g., any host should be included unless it is believed to be absolutely immune from zero day attacks) or by taking a probabilistic approach (e.g., we have discussed how associating a probability to relation $\equiv_v$ can help to model the degree of similarity in vulnerabilities and strength of isolation). Our future work will further explore such probabilistic approaches.
– An extremely conservative assumption may yield a trivial result (e.g., no network is 1-zero day safe, if insider attacks are considered possible on every host). While such an assumption may be the safest, it is also the least helpful in terms of improving the security since nothing would be helpful.
– The remote services and network connectivity must be identified by examining hosts' configuration. A network scanning is insufficient since it will not reveal services or connectivity currently disabled by security services (e.g., *ssh* behind *iptables* in Figure 1). The model is thus more concerned about the existence, instead of the current reachability, of a service or host.

---

[5] None of options can always guarantee improved security, which is why we need a metric.

- A zero day attack graph cannot be obtained by injecting zero day exploits into an existing attack graph of known vulnerabilities. The reason is that some unreachable exploits may be discarded in generating an attack graph of known vulnerabilities [1], whereas such exploits may indeed serve as shortcuts for bypassing zero day exploits in a zero day attack graph.
- The model itself does not provide a means for determining which conditions are likely to be subject to insider attacks or user mistakes, which should be determined based on knowledge about access control polices (which users are allowed to do what on which hosts) and how trustworthy each user is.

5 Related Work

Standardization efforts on vulnerability assessment include the Common Vulnerability Scoring System (CVSS) [14] which measures vulnerabilities in isolation. The NIST's efforts on standardizing security metrics are also given in [15] and more recently in [21]. The research on security metrics has attracted much attention lately [9]. Earlier work include the a metric in terms of time and efforts based on a Markov model [4]. More recently, several security metrics are proposed by combining CVSS scores based on attack graphs [22, 7]. The minimum efforts required for executing each exploit is used as a metric in [2, 17]. A mean time-to-compromise metric is proposed based on the predator state-space model (SSM) used in the biological sciences in [11]. The cost of network hardening is quantified in [23]. Security metrics are also developed for specific applications, such as IDSs [10] and distributed trust managment [19].

More closely related to our work, *attack surface* measures how likely a software is vulnerable to attacks based on the degree of exposure [16]. Our work borrows from attack surface the idea of focusing on interfaces, instead of internal details, of a system. However, we apply the idea to a network of computer systems instead of a single software system. Parallel to the study of security metrics, fault tolerance algorithms rely on replication and diversity to improve the availability of services [3]. Our metric provides a means for measuring the effectiveness of systems running such algorithms in the context of a network. Our work is partially inspired by the well known data privacy metric k-anonymity [18] which measures the amount of privacy using an integer regardless of specific application semantic. In our study, we adopt the graph-based representation of attack graphs proposed in [1], which avoids the state explosion problem that may face a model checking-based approach [20].

To the best of our knowledge, few work exist on measuring zero day attacks. An empirical study of the total number of zero day vulnerabilities available on a single day is given based on existing data [13]. If such a result can be obtained or estimated in real time, it may be incorporated into our metric by dynamically adjusting the value of k (a larger k is needed when more vulnerabilities are available). Another recent effort orders different applications in a system by the seriousness of consequences of having a single zero day vulnerability [8]. In contrast to our work, it has a different focus (applications instead of networks) and metric (seriousness of consequences instead of number of vulnerabilities).

6 Conclusion

We have proposed k-zero day safety as a novel security metric for measuring the relative security of networks against potential zero day attacks. In doing so, we have transformed the unmeasureability of unknown vulnerabilities from a commonly perceived obstacle to an opportunity for security metrics. While the general problem of computing the metric is intractable, we have demonstrated that practical security issues can be formulated and solved in polynomial time. For future work, we shall extend the model to address various limitations mentioned in this paper; we shall also integrate the proposed algorithms into existing attack graph-based security tools so to validate their real world effectiveness.

References

1. P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of CCS'02*, 2002.
2. D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st Workshop on Quality of Protection*, 2005.
3. M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
4. M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.
5. E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269271, 1959.
6. J. Doob. *Measure Theory*. Springer-Verlag, 1994.
7. M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of ACM workshop on Quality of protection*, 2008.
8. K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *Proceedings of ACSAC'09*, pages 117–126, Washington, DC, USA, 2009. IEEE Computer Society.
9. A. Jaquith. *Security Merics: Replacing Fear Uncertainity and Doubt*. Addison Wesley, 2007.
10. W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 130, Washington, DC, USA, 2001. IEEE Computer Society.
11. D. J. Leversage and E. J. Byres. Estimating a system's mean time-to-compromise. *IEEE Security and Privacy*, 6(1):52–60, 2008.
12. J. McHugh. Quality of protection: Measuring the unmeasurable? In *Proceedings of the 2nd ACM workshop on Quality of protection (QoP'06)*, pages 1–2, 2006.
13. M. McQueen, T. McQueen, W. Boyer, and M. Chaffin. Empirical estimates and observations of 0day vulnerabilities. *Hawaii International Conference on System Sciences*, 0:1–12, 2009.
14. P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy Magazine*, 4(6):85–89, 2006.
15. National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133, 1985.
16. J. W. P. Manadhata. An attack surface metric. Technical Report CMU-CS-05-155, 2005.
17. J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38, New York, NY, USA, 2006. ACM Press.
18. P.Samarati. Protecting respondents' identities in microdata release. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pages 1010–1027, 2001.
19. M. Reiter and S. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, 5 1999.
20. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
21. M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo. Security metrics guide for information technology systems. NIST Special Publication 800-55, 2003.
22. L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'08)*, 2008.
23. L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 11 2006.

Appendix A: Proof of Theorem 1

Proof: This is to prove, for all $F, F', F'' \subseteq E_0$, the following hold [6].

1. $k0d(F, F') = 0$ iff $F = F'$: This is straightforward since $k0d(F, F') = 0$ iff $F \triangle F' = \phi$, and the latter is equivalent to $F = F'$.

2. $k0d(F, F') = k0d(F', F)$: This property is satisfied by the symmetric difference.

3. $k0d(F, F') + k0d(F', F'') \geq k0d(F, F'')$: Denote by $tmp(G)$ the function $max(\{ |G'| : G' \subseteq G, \forall e_1, e_2 \in G' (e_1 \not\equiv_v e_2)\})$. First, the symmetric difference satisfies the triangle inclusion relation $F \triangle F'' \subset (F \triangle F') \cup (F' \triangle F'')$ [6]. So, $tmp((F \triangle F') \cup (F' \triangle F'')) \geq tmp(F \triangle F'')$ holds. Next, we only need to show $tmp(F \triangle F') + tmp(F' \triangle F'') \geq tmp((F \triangle F') \cup (F' \triangle F''))$ is true. It suffices to show the function $tmp(.)$ to be subadditive, that is, $tmp(G) + tmp(G') \geq tmp(G \cup G')$ holds for any $G, G' \subseteq E_0$. This follows from the fact that if the relation $e \equiv_v e'$ holds for any $e, e' \in G$ (or $e, e' \in G'$), it also holds in $G \cup G'$ (the converse is not necessarily true).

$\square$

Appendix B: Proof of Theorem 2

Proof: First, the problem is NP, since whether a given sequence of exploits q satisfies $q \in seq(a) \wedge k0d(q \cap E_0, \phi) = k$ can be easily determined in polynomial time in the size of the zero day attack graph.

Next, we reduce the NP-hard problem of finding the minimum attack (that is, an attack sequence with the minimum number of exploits) in attack graph [1, 20] to the current problem. First of all, the reduction is not trivial. More precisely, the reduction cannot be trivially achieved by simply replacing each known exploit with a zero day exploit in a given attack graph of know exploits, because, unlike the former, the latter has a fixed number of hard-coded pre- and post-conditions that may prevent them from fitting in the position of a known exploit.

We construct a zero day attack graph G' by injecting a zero day exploit before each known exploit. Specifically, first let $G' = G$. Then, for each known exploit e of a service s from a source host h_1 to a different destination host h_2, we inject a zero day exploit e' with the post-conditions $\{\langle s, h_2 \rangle, p_{useless}\}$ where $p_{userless}$ is a privilege designed not to be the pre-condition of any exploit (e' can be interpreted as exploiting a vulnerability in a security service, such as a personal firewall, that blocks accesses to the service s on h_2 from h_1). We then have the following two facts. First, executing e requires e' to be executed first; conversely, if e' needs to be executed, then the only reason must be to satisfy the condition $\langle s, h_2 \rangle$ and consequently execute e. That is, any attack sequence in G' will include either both e and e', or none of them. Second, among the three conditions in $pre(e') = \{\langle s', h_2 \rangle, \langle h_1, h_2 \rangle, \langle p_{least}, h_1 \rangle\}$, the first is an initial condition and the last two are also members of $pre(e)$. Therefore, the injection of e' does not change the logical structure of the attack graph (more precisely, G and G' are isomorphic if we regard e and e' as a single exploit and ignore the initial condition $\langle s', h_2 \rangle$).

Next, for each known exploit e involving the same source and destination host h, we replace e with a zero day exploit e' and a known exploit e'' satisfying that $post(e'') = post(e)$, $pre(e'') = pre(e) \setminus \{\langle p, h \rangle\} \cup \{\langle p', h \rangle\}$ where $\langle p, h \rangle \in pre(e)$ and $\{\langle p', h \rangle\}$ are two privileges. We also let $post(e') = \{\langle p', h \rangle\}$, and design the relation $\equiv_v$ in such a way that e' is not related to any other zero day exploits in h through $\equiv_v$. We then have two similar facts as above. First, any attack sequence in G' will include either both e and e', or none of them. Second, the injection of e' does not change the logical structure of the attack graph.

Based on the above construction, given any asset a, for any attack sequence $q' \in seq(a)$ in G', the known exploits in q also form an attack sequence $q \in seq(a)$ in G (note that a will always be the post-condition of known exploits due to our construction). Moreover, if we design $\equiv_v$ in such a way that no two zero day

exploits are related by $\equiv_v$, then we have $\mid q \mid = k0d(q' \cap E_0, \phi)$. Therefore, for any non-negative integer k, finding q' in G' to minimize $k0d(q' \cap E_0, \phi)$ will immediately yield q in G that also minimizes $\mid q \mid$, and the latter is essentially the minimum attack problem. This shows the former to be an NP-hard problem and concludes the proof. $\qquad\square$

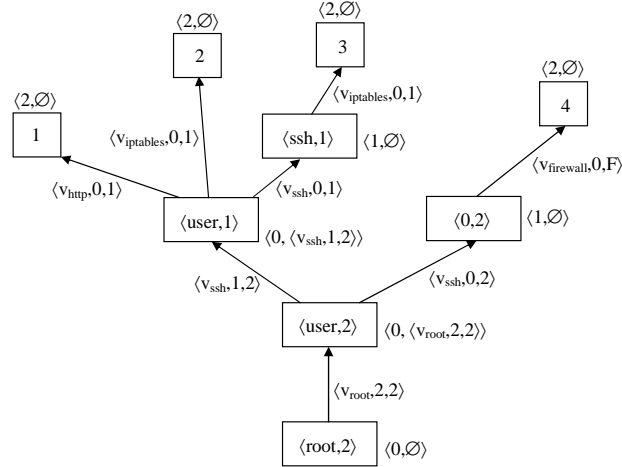## Appendix C: A Procedure for Computing k-Zero Day Safety as Shortest Paths in a DAG

In Figure 5, Procedure $k0d\_Shortest$ provides a more precise description of the method informally discussed in Section 3.3. Sub_Procedure $k0d\_Graph$ is used to build a DAG based on a given zero day attack graph and asset. First, the sub-procedure derives a logical proposition of the asset in terms of exploits and conditions using the same statements as in Procedure $k0d\_Backward$ (line 17), but stops whenever the DNF of the logic proposition includes at most one condition in each conjunctive clause (lines 18-19). The sub-procedure then adds each such conjunctive clause to the result DAG (line 20) by regarding each condition as a vertex pointed to by the asset (lines 21-22), and the set of exploits in the same conjunctive clause as the edge weight (line 23). The sub-procedure then recursively expands on each such condition (line 24). If a conjunctive clause does not include a condition (meaning that only initial conditions are required), a dummy vertex is added to represent the collection of deleted initial conditions (line 26-27).

Procedure $k0d\_Shortest$
Input: A zero day attack graph G, an asset L
Output: A non-negative real number k
Method:
1. **Let** G_s be a directed acyclic graph (DAG) with a vertex L and $elabel$ be an empty array
2. **Let** $\langle G_s, elabel \rangle = k0d\_Graph(G, L, G_s, elabel)$
3. **Let** $vlist$ be any topological sort of G_s
4. **Let** $dist_L = \{\langle 0, \phi \rangle\}$ and $dist_x = \{\langle \infty, \phi \rangle\}$ for any other vertex x
5. **While** $vlist$ is not empty, do
6.   **Delete** the first vertex u from $vlist$
7.   **For** each outgoing edge $\langle u, v \rangle$ of u
8.   **Let** $elist$ be the set of all edges reachable from v
9.   **For** each $\langle x, y \rangle \in dist_u$
10.   **Let** $y' = \{e : e \in y \cup elabel[\langle u, v \rangle], \exists e' \in elist \; \exists e'' \in elabel[e'] \; e \equiv_v e''\}$
11.   **Let** $x' = x + k0d((y \cup elabel[\langle u, v \rangle] \setminus y') \cap E_0, \phi)$
12.   **Let** $dist_v = dist_v \cup \langle x', y' \rangle$
13.   **While** $(\exists \langle x, y \rangle, \langle x', y' \rangle \in dist_v)(x \geq (x' + k0d(y' \cap E_0, \phi)))$
14.   **Delete** $\langle x, y \rangle$ from $dist_v$
15. **Return** $min(\{x : \langle x, \phi \rangle \in dist_d, d \text{ is a dummy vertex }\})$

Sub_Procedure $k0d\_Graph$
Input: A zero day attack graph G, an asset L, a DAG G_s, an array $elabel$
Output: Updated G_s and $elable$
Method:
16. **Do**
17.   (Lines 4-6 of Procedure $k0d\_Backward$)
18.   **Let** L be its DNF
19.   **While** there exists a conjunctive clause l in L including more than one condition
20. **for** each conjunctive clause l in L
21.   **If** l includes a condition c
22.   **Add** vertex c and edge $\langle L, c \rangle$ to G_s
23.   **Let** $elabel[\langle L, c \rangle]$ be the set of exploits in l
24.   **Let** $\langle G_s, elabel \rangle = k0d\_Graph(G, c, G_s, elabel)$ //Recursive call with c as L
25.   **Else**
26.   **Add** a dummy vertex d and edge $\langle L, d \rangle$ to G_s
27.   **Let** $elabel[\langle L, d \rangle]$ be the set of exploits in l
28. **Return** G_s

Fig. 5. Computing k-Zero Day Safety as Shortest Paths in a DAG

The main procedure then imitates a standard algorithm for finding the shortest path in a DAG [5]. More specifically, the vertices are processed based on a topological sort (line 3); and the distance of the source

vertex is initialized as 0 while that of other vertices as infinity (line 4); upon processing a vertex (line 5-6), each of its neighbors (line 7) will be updated with potentially shorter distances via the current vertex (lines 9-14). Finally, the main procedure returns the minimum shortest distance from the asset to a dummy vertex (representing initial conditions) as the result k.

The following modifications to the standard shortest distance algorithm are designed to take into account zero day exploits related by $\equiv_v$. First, instead of a single number, each distance is now a set of pairs $\langle x, y \rangle$ where x denotes the result of applying $k0d(.)$ to exploits that will not later be related to others by $\equiv_v$, whereas y denotes the set of zero day exploits that may later be related to others. More than one pairs may be necessary for a distance since they are incomparable due to the latter. Second, the reachable edges are collected in order to determine whether an exploit may later be related to others by $\equiv_v$ (line 8). Third, instead of simply calculating the minimum distance, both parts of each distance pair must be computed based on the distance of current vertex and the edge weight (line 10-11). The new distance pair will then immediately be added (line 12). Finally, after all distance pairs are added, the set of distance pairs is examined again to remove those that cannot be the minimum distance even when considering the effect of relation $\equiv_v$ (line 13-14).

Example 4. For our running example, Figure 6 illustrates the execution of Procedure $k0d\_Shortest$. Each edge is labeled with the edge weight $elabel$ and each vertex with the distance $dist$.

Fig. 6. An Example of Executing Procedure $k0d\_Shortest$

Complexity The complexity of the procedure will depend on how well the aforementioned assumptions hold on a given zero day attack graph. First, the complexity of Sub-Procedure $k0d\_Graph$ is exponential in the number of exploits and conditions involved in the loop at lines 16-19. Therefore, if the first assumption perfectly holds, this loop will always terminate after processing a single host. If we regard the number of exploits and conditions on each host as a constant, then the complexity of the sub-procedure will be linear in the number of hosts (that is, a constant time is required for deriving and processing L for each host). Second, the complexity of the main procedure depends on the size of the distance of each vertex. If the second assumption holds perfectly such that each distance has a negligible size, then the complexity of the main procedure will be dominated by processing the reachable edges in $elist$ and their labels $elabel$ (line 10). Since each edge in G_s is visited exactly once by the main loop and the size of $elist$ is linear in the

number of such edges, the processing of $elist$ takes quadratic time in the number of edges in G_s, which is roughly $O(|H|^4)$ (by the first assumption, each host corresponds to a constant number of vertices in G_s). Finally, multiplying this by the size of $elabel$, we have the complexity $|H|^4 \cdot |E_0|$.