An Efficient Framework for Evaluating the Risk of Zero-Day Vulnerabilities

Massimiliano Albanese[1], Sushil Jajodia[1,2(✉)], Anoop Singhal[3], and Lingyu Wang[4]

[1] Center for Secure Information Systems, George Mason University, 4400 University Dr, Fairfax, VA 22030, USA
jaodia@gmu.edu
[2] The MITRE Corporation, 7515 Colshire Dr, McLean, VA 22102, USA
[3] Computer Security Division, NIST, 100 Bureau Dr, Gaithersburg, MD 20899, USA
[4] Concordia Institute for Information Systems Engineering, Concordia University, 1515 Sainte-Catherine St W, Montreal, QC H3G 2W1, Canada

Abstract. Computer systems are vulnerable to both known and zero-day attacks. Although known attack patterns can be easily modeled, thus enabling the definition of suitable hardening strategies, handling zero-day vulnerabilities is inherently difficult due to their unpredictable nature. Previous research has attempted to assess the risk associated with unknown attack patterns, and a metric to quantify such risk, the k-zero-day safety metric, has been defined. However, existing algorithms for computing this metric are not scalable, and assume that complete zero-day attack graphs have been generated, which may be unfeasible in practice for large networks. In this paper, we propose a framework comprising a suite of polynomial algorithms for estimating the k-zero-day safety of possibly large networks efficiently, without pre-computing the entire attack graph. We validate our approach experimentally, and show that the proposed solution is computationally efficient and accurate.

Keywords: Zero-day attacks · Vulnerability analysis · Attack graphs

1 Introduction

In today's networked systems, attackers can leverage complex interdependencies among network configurations and vulnerabilities to penetrate seemingly well-guarded networks. Besides well-known weaknesses, attackers may leverage unknown (zero-day) vulnerabilities, which developers are not aware of. In-depth analysis of network vulnerabilities must consider attacker exploits not merely

The work presented in this paper is supported in part by the National Institutes of Standard and Technology under grant number 70NANB12H236, by the Army Research Office under MURI award number W911NF-09-1-0525, and by the Office of Naval Research under award number N00014-12-1-0461. The work of Sushil Jajodia was also supported by the MITRE Sponsored Research Program.

© Springer-Verlag Berlin Heidelberg 2014
M.S. Obaidat and J. Filipe (Eds.): ICETE 2013, CCIS 456, pp. 322–340, 2014.
DOI: 10.1007/978-3-662-44788-8_19

in isolation, but in combination. Attack graphs reveal such threats by enumerating potential paths that attackers can take to penetrate networks [1,2]. This helps determine whether a given set of network hardening measures provides safety of given critical assets. However, attack graphs can only provide qualitative results, and this renders resulting hardening recommendations ineffective or far from optimal, as illustrated by the example discussed in Sect. 3.1.

To address these limitations, traditional efforts on network security metrics typically assign numeric scores to vulnerabilities as their relative exploitability or likelihood, based on known facts about each vulnerability. However, this approach is clearly not applicable to zero-day vulnerabilities due to the lack of prior knowledge or experience. In fact, a major criticism of existing efforts on security metrics is that zero-day vulnerabilities are unmeasurable due to the less predictable nature of both the process of introducing software flaws and that of discovering and exploiting vulnerabilities [3]. Recent work addresses the above limitations by proposing a security metric for zero-day vulnerabilities, namely, k-zero-day safety [4]. Intuitively, the metric is based on the number of distinct zero-day vulnerabilities that are needed to compromise a given network asset. A larger such number indicates relatively more security, because it will be less likely to have a larger number of different unknown vulnerabilities all available at the same time, applicable to the same network, and exploitable by the same attacker. However, as shown in [4], the problem of computing the exact value of k is intractable. Moreover, [4] assumes the existence of a complete attack graph, but generating attack graphs for large networks is usually infeasible in practice [5]. These facts comprise a major limitation in applying this metric or any other similar metric based on attack graphs.

In this paper, we propose a suite of efficient algorithms to address this limitation and thus enable zero-day analysis of practical importance to be applied to networks of realistic sizes. Therefore, the major contribution of this work is to provide a practical solution to a problem which was previously considered intractable. We start from the problem of deciding whether a given network asset is at least k-zero-day safe for a given value of k [4], but then we go beyond this basic problem and provide a more complete analysis. First, we drop the assumption that the zero-day vulnerability graph has been precomputed, and combine on-demand attack graph generation with the evaluation of k-zero-day safety. Second, we identify an upper bound on the value of k using a heuristic algorithm that integrates attack graph generation and zero-day analysis. Third, when the upper bound on k is below an admissible threshold, we compute the exact value of k by reusing the computed partial attack graph. Section 4 formally states the three related problems we are addressing in this paper, and shows their role within the framework. To the best of our knowledge, this is the first attempt to define a comprehensive and efficient approach to zero-day analysis.

The paper is organized as follows. Section 2 discusses related work. Section 3 recalls some preliminary definitions and provides a motivating example. Then Sect. 4 discusses the limitations of previous approaches and provides a formal statement of the problems addressed in our work. Section 5 describes our

approach to efficient evaluation of k zero-day safety. Finally, Sect. 6 reports experimental results, and Sect. 7 gives some concluding remarks.

2 Related Work

Existing standardization efforts, such as the Common Vulnerability Scoring System (CVSS) [6] and the Common Weakness Scoring System (CWSS) [7], provide standard ways for security analysts and vendors to rank known vulnerabilities or software weaknesses using numerical scores. These efforts provide a practical foundation for research on security metrics, but are designed for individual vulnerabilities and do not address the combined effect of multiple vulnerabilities. Early work on security metrics include a Markov model-based metric for estimating the time and efforts required by adversaries [8], and a metric based on lengths of shortest attack paths [9]. The main limitation of these approaches is that they do not consider the relative severity or likelihood of different vulnerabilities. Another line of work adapts the PageRank algorithm to rank states in an attack graph based on the relative likelihood of attackers' reaching these states when they progress along different paths in a random fashion [10]. Other recent work uses specially marked attack trees [11] or more expressive attack graphs [12] in order to find the easiest attack paths. A Mean Time-to-Compromise metric based on the predator state-space model (SSM) captures the average time required to compromise network assets [13]. A probabilistic approach defines a network security metric as attack probabilities and derives such probabilities from CVSS scores [14]. Several important issues in calculating probabilistic security metrics, such as dependencies between attack sequences and cyclic structures, are addressed in [15].

Most existing work on network security metrics has focused on previously known vulnerabilities [3]. A few exceptions include an empirical study on the total number of zero-day vulnerabilities available on a single day [16], a study on the popularity of zero-day vulnerabilities [17], and an empirical study on software vulnerabilities' life cycles [18]. Another recent effort ranks different applications by the relative severity of having one zero-day vulnerability in each application [19], which has a different focus than our work. Closest to our work, recent work on k-zero-day safety defines a metric based on the number of potential unknown vulnerabilities in a network [4].

In this paper, we address the complexity issues associated with the metric proposed in [4], and propose a set of polynomial algorithms for estimating the k-zero-day safety of possibly large networks efficiently. The proposed zero-day attack graph model borrows the compact model given in [2] – based on the *monotonicity assumption* – while incorporating zero-day vulnerabilities.

3 Preliminaries

Attack graphs represent prior knowledge about vulnerabilities, their dependencies, and network connectivity. With a *monotonicity* assumption, an attack graph

can record the dependencies among vulnerabilities and keep attack paths implicitly without losing any information. The resulting attack graph has no duplicate vertices and hence has a polynomial size in the number of vulnerabilities multiplied by the number of connected pairs of hosts.

Definition 1 (Attack Graph). *Given a set of exploits E, a set of security conditions C, a* require *relation $R_r \subseteq C \times E$, and an* imply *relation $R_i \subseteq E \times C$, an attack graph G is the directed graph $G = (E \cup C, R_r \cup R_i)$, where $E \cup C$ is the vertex set and $R_r \cup R_i$ the edge set. For an exploit e, we call the conditions related to e by R_r and R_i as its pre- and post-conditions, denoted using functions $pre : E \rightarrow 2^C$ and $post : E \rightarrow 2^C$, respectively.*

We denote an exploit as a triple $\langle v, h_s, h_d \rangle$, indicating an exploitation of vulnerability v on the destination host h_d, initiated from the source host h_s. A security condition is a pair $\langle c, h_d \rangle$ – that indicates a satisfied security-related condition c on host h_d, such as the existence of a vulnerability – or a pair $\langle h_s, h_d \rangle$ – that indicates connectivity between hosts h_s and h_d. Initial conditions are a special subset of security conditions that are initially satisfied, whereas intermediate conditions are those that can only be satisfied as post-conditions of some exploits.

Definition 2 (Initial Conditions). *Given an attack graph $G = (E \cup C, R_r \cup R_i)$,* initial conditions *refer to the subset of conditions $C_i = \{c \in C | \nexists e \in E \text{ s.t. } (e, c) \in R_i\}$, whereas* intermediate conditions *refer to the subset $C \backslash C_i$.*

3.1 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. Therefore, our zero-day vulnerability model is based on following generic properties that are common to most vulnerabilities. Specifically, a zero-day vulnerability is a vulnerability whose details are unknown except that its exploitation requires a network connection between the source and destination hosts, a remotely accessible service on the destination host, and that the attacker already has a privilege on the source host. In addition, we assume that the exploitation can potentially yield any privilege on the destination host. These assumptions intend to depict a worst-case scenario about the pre- and post-conditions of a zero-day exploit, and are formalized as the first type of zero-day exploit in Definition 3, whereas the second type represents subsequent privilege escalation.

Definition 3 (Zero-day Exploit). *We define two types of zero-day exploits,*

- *for each remote service s, we define a zero-day vulnerability v_s such that the zero-day exploit $\langle v_s, h, h' \rangle$ has three pre-conditions, $\langle s, h' \rangle$ (existence of service), $\langle h, h' \rangle$ (connectivity), and $\langle p, h \rangle$ (attacker's existing privilege); this zero-day exploit has one post-condition $\langle p', h' \rangle$ where p' is the privilege of service s on h'.*

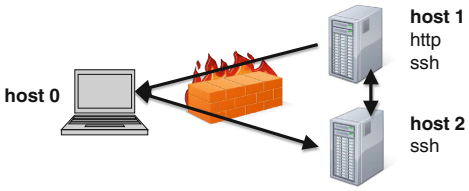


Fig. 1. Example of network configuration.

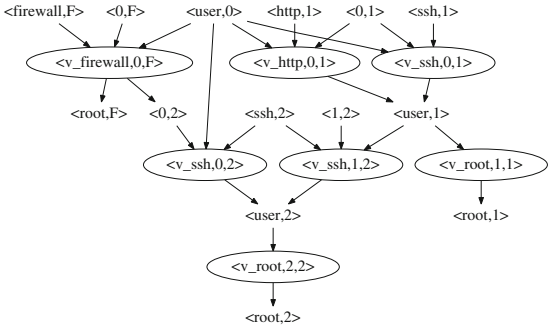


Fig. 2. Example of zero-day attack graph.

– *for each privilege p, we define a zero-day vulnerability v_p such that the zero-day exploit $\langle v_p, h, h\rangle$ has its pre-conditions to include all privileges of remote services on h, and its post-condition to be p on h.*

We use E_0 and C_0 to denote the set of all zero-day exploits and the set of all their pre- and post-conditions respectively, and we extend the functions $pre()$ and $post()$ accordingly.

We are now ready to assemble all known and zero-day exploits via their common pre- and post-conditions into a zero-day attack graph.

Definition 4 (Zero-Day Attack Graph). *Given an attack graph $G = (E \cup C, R_r \cup R_i)$, a set E_0 of zero-day exploits, a set C_0 of pre and post-conditions of exploits in E_0, a zero-day attack graph G^* is the directed graph $G^* = (E^* \cup C^*, R_r^* \cup R_i^*)$, where $E^* = E \cup E_0$, $C^* = C \cup C_0$, $R_r^* = R_r \cup \{(c, e) \mid e \in E_0 \wedge c \in pre(e)\}$, and $R_i^* = R_i \cup \{(e, c) \mid e \in E_0 \wedge c \in post(e)\}$.*

Figure 1 shows a simple network configuration including three hosts. Host 0 is the user's machine used to launch attacks, whereas host 1 and host 2 are machines within the perimeter of the enterprise network we are seeking to protect. Host 1 provides an HTTP service (http) and a secure shell service (ssh), whereas host 2 provides only ssh. The firewall allows traffic to and from host 1, but only connections originated from host 2. In this example, we 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. However, we may reach a different conclusion by hypothesizing the presence of zero-day vulnerabilities and considering how many distinct zero-day exploits the network can resist.

Specifically, the zero-day attack graph of this example is depicted in Fig. 2, where each triple inside an oval denotes a zero-day exploit and a pair denotes a condition. In this attack graph, we can observe three sequences of zero-day exploits leading to $root(2)$. First, an attacker on host 0 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, or the attacker can exploit a zero-day vulnerability in either http or ssh on host 1 to obtain the user privilege and then, using host 1 as a stepping stone, the attacker can further exploit a zero-day vulnerability in ssh on host 2 to reach $root(2)$. Since this last sequence (ssh on host 1 and then ssh on host 2) involves one zero-day vulnerability in the ssh service on both hosts, this network can resist at most one zero-day attack. Contrary to the previous belief that further hardening this network is not necessary, this zero-day attack graph shows that further hardening may indeed improve the security. For example, suppose we limit accesses to the ssh service on host 1 using a personal firewall or iptables rules, such that an arbitrary host 0 cannot reach this service from the Internet. We can then imagine that the new attack graph will only include sequences of at least two different zero-day vulnerabilities (e.g., the attacker must first exploit the personal firewall or iptables rules before exploiting ssh on host 1). This seemingly unnecessary hardening effort thus can help the network resist one more zero-day attack.

4 Problem Statement

The exact algorithm for computing the k-zero-day safety metric presented in [4] first derives a logic proposition of each asset in terms of exploits by traversing the attack graph backwards. Each conjunctive clause in the disjunctive normal form (DNF) of the derived proposition corresponds to a minimal set of exploits that jointly compromise the asset. The value of k can then be decided by applying the metric $k0d()$ – which counts the number of distinct zero-day vulnerabilities – to each such conjunctive clause. Although the logic proposition can be derived efficiently, converting it to its DNF may incur an exponential explosion. In fact, the authors of [4] show that the problem of computing the k-zero day safety metrics is NP-hard in general, and then focus on the solution of a more practical problem. They claim that, 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 determining k' is intractable). Then, they 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. However, there are cases in which it is not satisfactory to just know $k' > k$, but more accurate estimations or exact calculation of the value of k is desired. Moreover, those analyses are all based on complete zero-day attack graphs, but for really large networks, it may even be infeasible to generate the zero-day attack graph in the first place. The metric then becomes impractical in such cases since there is little we can say about the value of k.

The aforementioned intractability result means no polynomial algorithm will likely exist for computing the exact value of k. However, in this section we show that a decision process may still allow security administrators to obtain good estimations about k, and to calculate the exact value of k when it is practically feasible. Our main objectives are three-fold. First, all the algorithms involved in the decision process will be efficient and have polynomial complexity. Second, all the algorithms will adopt an on-demand approach to attack graph generation, which will only generate partial attack graphs necessary for the analysis. Third, subsequent algorithms will reuse the partial attack graph already generated earlier in the decision process, thus further improving the overall efficiency. With those optimizations, we can provide a better understanding of zero-day vulnerabilities even for relatively large networks. Specifically, in most practical scenarios, security administrators may simply want to assess *whether the network or specific assets are secure enough.* In such cases, knowing that k is larger than or equal to a given lower bound l may be sufficient. However, once it has been confirmed that $k > l$, a security administrator may want to know *whether it is possible to compute the exact value of* k. Since the problem of computing the exact value of k is intractable, this may only be possible for relatively small values of k. Therefore, we need to estimate whether k is less than a practical upper bound that represents available computational power. Finally, if this is true, then we can proceed to *calculate the actual value of* k *in an efficient way.* In the following, we formalize the three related problems that form the basis of the above decision process. We describe a solution to each of these problem in the next section.

Problem 1 (Lower Bound). Given a network N, a goal condition c_g, and a small integer l, determine whether $k \geq l$ is true for N with respect to c_g.

Our goal is to identify a lower bound on the value of k. This problem is analogous to the practical problem addressed in [4], but we do not assume the entire attack graph is available. We simply assume that the network is defined in terms of initial conditions C_i and known and unknown exploits E^*.

Problem 2 (Upper Bound). Given a network N, a goal condition c_g, and an integer u, find an upper bound u on the value of k with respect to c_g.

Our goal is to identify an upper bound on the value of k. We show that, using a heuristic approach, it is feasible to compute a good upper bound in polynomial time. If the value of u is below a threshold u^*, it may then be feasible to compute the exact value of k.

Problem 3 (Exact Value). Given a network N, and a goal condition c_g such that $l \leq k \leq u \leq u^*$ is true for N with respect to c_g, find the exact value of k.

In other words, when the value of k is known to be bounded and the upper bound is small enough, we will compute the exact value of k, leveraging the upper bound u for pruning, and reusing the partial attack graph generated during previous steps of the decision process. Figure 3 shows the role of these three problems in the overall decision process.

5 Proposed Solution

5.1 Solution for Problem 1

The existing solution for this problem assumes that the entire zero-day attack graph is available [4], which is impractical since generating such an attack graph may be infeasible for large networks. Paths with up to l zero-day vulnerabilities are generated and evaluated using the metric. The idea behind our solution is to combine an exhaustive forward search of limited depth with partial attack graph generation, so that only attack

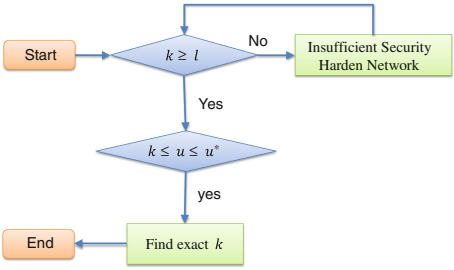


Fig. 3. Flowchart of the decision process.

We use connectivity information to hypothesize zero-day exploits (see Definition 3) and guide the generation of the graph.

Algorithm *k0dLowerBound* (Algorithm 1) takes as input a set C_i of initial conditions on hosts, the set E^* of known and zero-day exploits, an integer $l \in \mathbb{N}$ representing the desired lower bound on the value of k, and a goal condition $c_g \in C^*$. It returns a partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, and a truth value indicating whether $k \geq l$.

For ease of presentation, we consider problems with a single goal condition. The generalization to the case where multiple target conditions need to be considered at the same time is straightforward and is discussed below. Given a set C_g of goal conditions, we can add a dummy exploit e_g, such that e_g has each $c_i \in C_g$ as a precondition. Then, we can add a dummy goal condition c_g as the only postcondition of e_g. It is clear that the minimum number of zero-day

Algorithm 1. $k0dLowerBound(C_i, E^*, l, c_g)$.

Input: Set C_i of initial conditions, set E^* of known and zero-day exploits, integer $l \in \mathbb{N}$ representing the desired lower bound on k, and goal condition $c_g \in C^*$.
Output: Partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, and a truth value indicating whether $k \geq l$.

1: $C \leftarrow C_i$
2: $E \leftarrow \emptyset$
3: $C_{new} \leftarrow C_i$
4: **for all** $c \in C_i$ **do**
5: $\quad \pi(c) \leftarrow \emptyset$
6: **end for**
7: **while** $C_{new} \neq \emptyset$ **do**
8: $\quad E_{new} \leftarrow \{e \in E \mid pre(e) \subseteq C \wedge pre(e) \cap C_{new} \neq \emptyset\}$ // Unvisited exploits reachable from C
9: $\quad$ **for all** $e \in E_{new}$ **do**
10: $\quad\quad$ **for all** $c \in pre(e)$ **do**
11: $\quad\quad\quad R_r \leftarrow R_r \cup \{(c, e)\}$ // Add an edge from c to e
12: $\quad\quad$ **end for**
13: $\quad\quad \{c_1, \ldots, c_m\} \leftarrow \{c \in C \mid (c, e) \in R_r\}$
14: $\quad\quad \pi(e) \leftarrow \{P_1 \cup \ldots \cup P_m \cup \{e\} \mid P_i \in \pi(c_i)\}$
15: $\quad\quad \pi(e) \leftarrow \{P \in \pi(e) \mid k0d(P) < l\}$ // Prune paths with l or more zero-day vulnerabilities
16: $\quad$ **end for**
17: $\quad C_{new} \leftarrow \emptyset$
18: $\quad$ **for all** $e \in E_{new}$ s.t. $\pi(e) \neq \emptyset$ **do**
19: $\quad\quad$ **for all** $c \in post(e)$ **do**
20: $\quad\quad\quad R_i \leftarrow R_i \cup \{(e, c)\}$ // Add an edge from e to c
21: $\quad\quad\quad C_{new} \leftarrow C_{new} \cup \{c\}$
22: $\quad\quad\quad$ **if** $c \equiv c_g$ **then**
23: $\quad\quad\quad\quad$ **return** $G, false$
24: $\quad\quad\quad$ **end if**
25: $\quad\quad\quad \pi(c) \leftarrow \bigcup_{e \in E \mid (e,c) \in R_i} \pi(e)$
26: $\quad\quad$ **end for**
27: $\quad$ **end for**
28: $\quad C \leftarrow C \cup C_{new}$
29: $\quad E \leftarrow E \cup E_{new}$
30: **end while**
31: **return** $G, true$

exploits needed to reach all the conditions in C_g corresponds to the minimum number of zero-day exploits needed to reach the dummy goal condition c_g. In fact, as c_g is reachable only from the dummy exploit e_g, all the preconditions of e_g must be satisfied, therefore all the actual goal conditions in C_g must be reached.

Lines 1–6 of algorithm *k0dLowerBound* simply initialize the sets of conditions and exploits in the partial attack graph, the set C_{new} of newly satisfied conditions, and the mapping $\pi : E \cup C \rightarrow 2^{2^E}$ which associates each exploit or condition with a set of attack paths leading to it, where an attack path is a set of exploits. By default, $\pi(c) = \emptyset$ for all $c \in C_i$. The set C_{new} will initially contain all the initial conditions, whereas in each subsequent iteration of the algorithm it will contain the conditions implied by exploit visited in that iteration. The main loop at Lines 7–30 iterates until the goal condition is reached (Lines 22–24) or the set of newly satisfied conditions becomes empty – which means that no path with fewer than l distinct zero-day vulnerabilities can reach the goal condition. In the first case the algorithm returns $false$ (i.e., $k < l$), otherwise it returns $true$ (i.e., $k \geq l$).

Line 8 defines the set E_{new} of unvisited exploits reachable from C. An exploit is *unvisited* if at least one of its preconditions is in C_{new}. For each $e \in E_{new}$, Lines 10–12 add edges from all preconditions of e to e itself, and Lines 13–14 compute partial attack paths leading to and including e. Finally, Line 15 prunes all attack paths with l or more distinct zero-day vulnerabilities. As an exploit needs all the preconditions to be satisfied, an attack path for e is constructed by combining an attack path to each precondition.

Once all the newly visited exploits have been processed and added to the attack graph, the algorithm considers the new conditions that are implied by such exploits. For each $e \in E_{new}$ such that at least one partial path reaching e has $k0d(P) < l$, and each condition c in $post(e)$, Lines 20–21 add an edge from e to c to the graph and update C_{new} (which was reset on Line 17), and Line 25 computes the set $\pi(c)$ of attack paths leading to c as the union of the sets of attack paths leading to each of the exploit implying c, unless c is the goal condition, in which case the algorithm terminates.

Example 1. When applied to the graph in Fig. 2, Algorithm *k0dLowerBound* (Algorithm 1) will proceed by each horizontal level of conditions and exploits, from top to bottom, until it reaches the second level of exploits (i.e., $\langle v_{ssh}, 0, 2\rangle$, $\langle v_{ssh}, 1, 2\rangle$, and $\langle v_{root}, 1, 1\rangle$). Suppose l is given to be 2, then obviously all the paths up to now will be pruned by Line 15 (since each of them includes two distinct zero-day vulnerabilities, failing the condition $k0d(P) < l$), except the path $\langle v_{ssh}, 0, 1\rangle$, $\langle v_{ssh}, 1, 2\rangle$ (which includes only one vulnerability v_{ssh}). Therefore, the next loop on Lines 18–27 will be skipped for exploit $\langle v_{ssh}, 0, 2\rangle$ and $\langle v_{root}, 1, 1\rangle$ (meaning the partial attack graph generation stops at those exploits), but it continues from exploit $\langle v_{ssh}, 1, 2\rangle$ (the final result will depend on whether we assume $\langle v_{ssh}, 1, 2\rangle$ directly yields $\langle root, 2\rangle$).

The complexity of Algorithm *k0dLowerBound* (Algorithm 1) is clearly dominated by the steps for extending the paths on Lines 13–15. Specifically, the loop at Line 7 will run at most $\mid C \mid$ times; the nested loop at Line 9 will run $\mid E \mid$ times; steps 13–15 will involve at most $\mid E \mid^l$ paths each of which has maximum possible length of $\mid E \mid$. Therefore, the overall complexity is $O(\mid C \mid \cdot \mid E \mid \cdot \mid E \mid^l \cdot \mid E \mid) = O(\mid C \mid \cdot \mid E \mid^l)$, which is polynomial when l is given as a constant (compared to attack graph size).

5.2 Solution for Problem 2

In this section, we propose a solution to Problem 2. As we did for the previous algorithm, instead of building the entire attack graph, we only build the portions of the attack graph that are most promising for finding an upper bound on the value of k. In order to avoid the exponential explosion of the search space – which includes all the sets of exploits leading to the goal condition – we design an heuristic algorithm that maintains only the best partial paths with respect to the $k0d$ metric.

Algorithm *k0dUpperBound* (Algorithm 2) builds the attack graph forward, starting from initial conditions. A key advantage of building the attack graph

Algorithm 2. *k0dUpperBound*(C_i, E^*, c_g).

Input: Set C_i of initial conditions, set E^* of known and zero-day exploits, and goal condition $c_g \in C^*$.

Output: Partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, mapping $\pi : C \cup E \rightarrow 2^{2^E}$, mapping $zdu : C \rightarrow \mathbb{N}$, and upper bound u on the value of k.

1: $C \leftarrow C_i$
2: $R_r \leftarrow \emptyset$
3: $R_i \leftarrow \emptyset$
4: **for all** $c \in C_i$ **do**
5: $\quad \pi(c) \leftarrow \emptyset$
6: $\quad zdu(c) \leftarrow 0$
7: **end for**
8: $E \leftarrow \{e \in E^* \mid pre(e) \subseteq C\}$
9: **for all** $e \in E$ **do**
10: $\quad$ **for all** $c \in pre(e)$ **do**
11: $\quad\quad R_r \leftarrow R_r \cup \{(c, e)\}$
12: $\quad$ **end for**
13: $\quad \pi(e) \leftarrow \{\{e\}\}$
14: $\quad zdu(e) \leftarrow k0d(\{e\})$
15: **end for**
16: $\langle E_1, \ldots, E_n \rangle \leftarrow rankedPartition(E)$
17: $i \leftarrow 0$
18: **while** $c_g \notin C \wedge i \leq n$ **do**
19: $\quad i \leftarrow i + 1$
20: $\quad u \leftarrow DFS(E_i)$
21: **end while**
22: **return** $G, \pi(), zdu(), u$

forward is that intermediate solutions are indeed estimates of the upper bound on k for intermediate conditions. In fact, in a single pass, algorithm *k0dUpperBound* can estimate an upper bound on k with respect to any condition in C. To limit the exponential explosion of the search space, intermediate solutions can be pruned – based on some pruning strategy – whereas this would not be possible for an algorithm building the attack graph backwards.

The algorithm takes as input the set C_i of initial conditions on hosts, the set E^* of known and zero-day exploits, and a goal condition $c_g \in C^*$. The algorithm returns an upper bound u on the value of k, and also computes a partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, a mapping $\pi : C \cup E \rightarrow 2^{2^E}$ which associates each node in the partial attack graph with attack paths leading to it, and a mapping $zdu : C \rightarrow \mathbb{N}$ which associates each node in the partial attack graph with an estimate of the upper bound on k. In this section, we assume that Algorithm 2 starts from initial conditions, but modifying the algorithm to reuse partial attack graphs generated by previous execution of Algorithm 1 is straightforward, and can be done as shown for algorithm *k0dValue* (Algorithm 5).

Lines 1–8 simply initialize all the components of the partial attack graph. Line 1 adds the initial conditions to the set C of security conditions in the partial attack graph. As the algorithm builds the attack graph, new conditions will be added to C. Lines 2–3 initialize the *require* and *imply* relationships as empty sets. For each $c \in C_i$, Lines 5–6 set $\pi(c)$ to $\emptyset$ – meaning that no exploit is needed to reach initial conditions, as they are satisfied by default – and $zdu(c)$ to 0 – meaning that no zero-day exploit is needed to reach initial conditions. Finally, Line 8 sets E to the set of exploits reachable from conditions in C.

For each exploit $e \in E$, Lines 10–12 add edges to e from each of its preconditions, Line 13 associates e with the only set of exploits leading to it, that is $\{e\}$, and Line 14 computes $zdu(e)$ as the number of distinct zero-day vulnerabilities in $\{e\}$, that is $k0d(\{e\})$[1].

Line 16–21 try to find an attack path reaching the goal condition with the lowest possible number of distinct zero-day vulnerabilities. Since we use an heuristic approach to prune the search space, the number of distinct zero-day vulnerabilities in such path is naturally an upper bound on the minimum number k of zero-day vulnerabilities needed to reach the goal. Line 16 uses Algorithm *rankedPartition* (Algorithm 3) to rank exploits in E by increasing value of $zdu(e)$ and partition the set into ranked subsets. Then, Lines 18–21 iteratively explore the partial attack graph in a depth-first manner, by using the recursive algorithm DFS (Algorithm 4), starting from the set of exploits E_1 with the smallest values of $zdu()$.

Algorithm 3. *rankedPartition*(E').

Input: Set E' of exploits.
Output: Partition P_E of E
1: $E_r \leftarrow \langle e_1, \ldots, e_{|E'|} \rangle$ s.t. $(\forall i, j \in [1, |E'|])(i \leq j \Rightarrow zdu(e_i) \leq zdu(e_j))$
2: $P_E \leftarrow \langle E_1, \ldots, E_n \rangle$ s.t. $(\forall i, j \in [1, n])(i \leq j \Rightarrow (\forall e' \in E_i, e'' \in E_j)(zdu(e') \leq zdu(e'')))$
3: **return** P_E

Algorithm *rankedPartition* (Algorithm 3) takes as input a set of exploits E' and returns a partition P_E of E. Line 1 sorts exploits in E by increasing value of $zdu(e)$. Then, Line 2 partitions E into an ordered set of sets $E_1, \ldots, E_n$, such that for each $i \leq j \leq n$ all exploits in E_i have smaller values of $zdu()$ than any exploit in E_j.

Algorithm DFS (Algorithm 4) takes as input a set E_{start} of exploits and a goal condition $c_g \in C^*$, and returns an upper bound u on the value of k. We assume that the partial attack graph and the two mappings $\pi()$ and $zdu()$ are global variables.

For each $e \in E_{start}$ and each $c \in post(e)$, (i) Lines 4–5 add an edge from e to c, and update the set C_{new} of newly reached conditions, (ii) Line 6 computes the set $\pi(c)$ of attack paths leading to c as the union of the sets of attack paths leading to each exploit implying it, and (iii) Line 7 computes an estimate $zdu(c)$ of the upper bound on k with respect to c as the smallest $zdu(P)$ over all paths P in $\pi(c)$. If c is the goal condition, then the algorithm returns $zdu(c)$.

If none of the conditions in C_{new} is the goal condition, then Line 14 defines a new set E_{new} of unvisited exploits reachable from C, which has been updated to include all conditions reached from E_{start}. An exploit is *unvisited* if at least one of its preconditions is in C_{new}. If no new exploit is enabled, then the algorithm return $+\infty$ (Line 16), meaning that the goal condition cannot be reached from

[1] For exploits directly reachable from initial conditions, $zdu(e)$ is either 1, if e is a zero-day exploit, or 0, otherwise.

Algorithm 4. $DFS(E_{start}, c_g)$.

Input: Set E_{start} of exploits and goal condition $c_g \in C^*$
Output: Upper bound u on the value of k.

```
 1: C_new ← ∅
 2: for all e ∈ E_start do
 3:     for all c ∈ post(e) do
 4:         R_i ← R_i ∪ {(e, c)}
 5:         C_new ← C_new ∪ {c}
 6:         π(c) ← ⋃_{e∈E|(e,c)∈R_i} π(e)
 7:         zdu(c) ← min_{P∈π(c)} k0d(P)
 8:         if c = c_g then
 9:             return zdu(c)
10:         end if
11:     end for
12: end for
13: C ← C ∪ C_new
14: E_new ← {e ∈ E | pre(e) ⊆ C ∧ pre(e) ∩ C_new ≠ ∅}
15: if E_new = ∅ then
16:     return +∞
17: end if
18: for all e ∈ E_new do
19:     E ← E ∪ {e}
20:     for all c ∈ pre(e) do
21:         R_r ← R_r ∪ {(c, e)}
22:     end for
23:     {c_1, ..., c_m} ← {c ∈ C | (c, e) ∈ R_r}
24:     π(e) ← {P_1 ∪ ... ∪ P_m ∪ {e} | P_i ∈ π(c_i)}
25:     π(e) ← top(π(e), t)
26:     zdu(e) ← min_{P∈π(e)} k0d(P)
27: end for
28: ⟨E_1, ..., E_n⟩ ← rankedPartition(E_new)
29: i ← 0
30: while c_g ∉ C ∧ i ≤ n do
31:     i ← i + 1
32:     u ← DFS(E_i)
33: end while
34: return u
```

the branch of the attack graph explored in the current iteration of the algorithm. Otherwise, for each $e \in E_{new}$, (i) Lines 19–22 add e and edges to e from each of its preconditions to the partial attack graph, (ii) Lines 23–24 compute the set $\pi(e)$ of partial attack paths ending with e in the same way we have described for algorithm *k0dLowerBound*, (iii) Line 25 prunes $\pi(e)$ by maintaining only the top b partial attack paths with respect to the $k0d()$ metric, and (iv) Line 26 computes an estimate $zdu(e)$ of the upper bound on k with respect to e as the smallest $zdu(P)$ over all paths P in $\pi(e)$.

Finally, Line 28 uses algorithm *rankedPartition* (Algorithm 3) to rank exploits in E by increasing value of $zdu(e)$ and partition the set. Then, Lines 30–33 iteratively explore the partial attack graph in a depth-first manner, by recursively calling algorithm DFS, starting from the set of exploits E_1 with the smallest values of $zdu()$.

Example 2. When applied to the graph of Fig. 2, algorithm *k0dUpperBound* (Algorithm 2) will first consider exploits E reachable from the initial conditions (i.e., the first level of exploits, namely $\langle v_{firewall}, 0, F\rangle$, $\langle v_{http}, 0, 1\rangle$, $\langle v_{ssh}, 0, 1\rangle$), and will rank them by increasing value of $zdu()$. Then, assume that algorithm

rankedPartition (Algorithm 3) partitions the set of exploits into subsets of size 1. As each exploit e on the first level has $zdu(e) = 1$, algorithm *k0dUpperBound* will continue building the graph starting from any such exploit. If we assume it will start from $\langle v_{firewall}, 0, F\rangle$, then its post-condition $\langle 0, 2\rangle$ will be added to the graph. Subsequent recursive calls of algorithm DFS will add $\langle v_{ssh}, 0, 1\rangle$, $\langle user, 2\rangle$, $\langle v_{root}, 2, 2\rangle$, and $\langle root, 2\rangle$, thus reaching the goal condition and returning $u = 2$. As seen in the previous example, the actual value of k in this scenario is 1, so $u = 2$ is a reasonable upper bound, which we were able to compute efficiently by building only a partial attack graph.

The complexity of Algorithm *k0dUpperBound* (Algorithm 2) is clearly dominated by the recursive execution of algorithm DFS (Algorithm 4), which in the worst case – due to the adopted pruning strategy – has to process t partial attack paths for each node in the partial attack graph. Therefore, the complexity is $O(t \cdot (\mid C \mid + \mid E \mid))$, which is linear in the size of the graph when t is constant.

5.3 Solution for Problem 3

When the upper bound on the value of k is below a practical threshold u^*, we would like to compute the exact value of k, which is intractable in general. Our solution consists in performing a forward search, similarly to algorithm *k0dLowerBound*, starting from the partial attack graphs computed in previous steps of the decision process discussed in Sect. 4. To limit the search space, compared to a traditional forward search, and avoid the generation of the entire attack graph, we leverage the upper bound computed by algorithm *k0dUpperBound* to prune paths not leading to the solution. In fact, although the value of k is known to be no larger than u, there still may be many paths with more the u distinct zero-day vulnerabilities, and we want to avoid adding such paths to the attack graph. Algorithm *k0dValue* (Algorithm 5) is indeed very similar to algorithm *k0dLowerBound*. Therefore, for reasons of space, we only highlight the main differences in our discussion. First, the algorithm takes as input partial attack graphs, instead of starting from initial conditions. Thus, Line 1 computes C_{new} as the set of pre-conditions of unvisited exploits (i.e., exploits not added yet to the attack graph). Second, Line 10 prunes all attack paths with more than u distinct zero-day vulnerabilities. Finally, when the goal condition is reached, the algorithm computes the exact value of k as the smallest $k0d(P)$ over all paths P in $\pi(c_g)$.

6 Experimental Results

In this section, we present the results of experiments we conducted to validate our approach. Specifically, our objective is three-fold. First, we evaluated the performance of the proposed algorithms in terms of processing time in order to confirm that they are efficient enough to be practical. Second, we evaluated the percentage of nodes included in the generated partial attack graph compared to the full attack graph, which shows the degree of savings, in terms of both

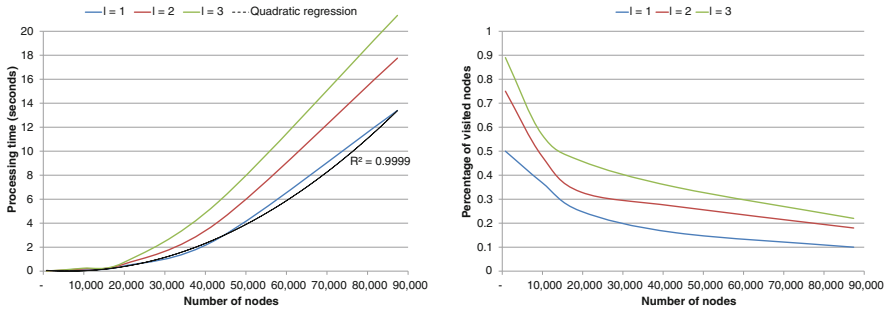
time and storage, that may be achieved through our on-demand generation of attack graphs. Third, we also evaluated the accuracy of estimations made using algorithm *k0dUpperBound* compared to the real results obtained using a brute force approach.

Algorithm 5. $k0dValue(G, E^*, u, c_g)$.

Input: Partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, set E^* of known and zero-day exploits, integer $u \in \mathbb{N}$ representing the upper bound on the value of k computed by algorithm *k0dUpperBound*, and goal condition $c_g \in C^*$.
Output: Updated Partial zero-day attack graph $G = (E \cup C, R_r \cup R_i)$, and the exact value of k.

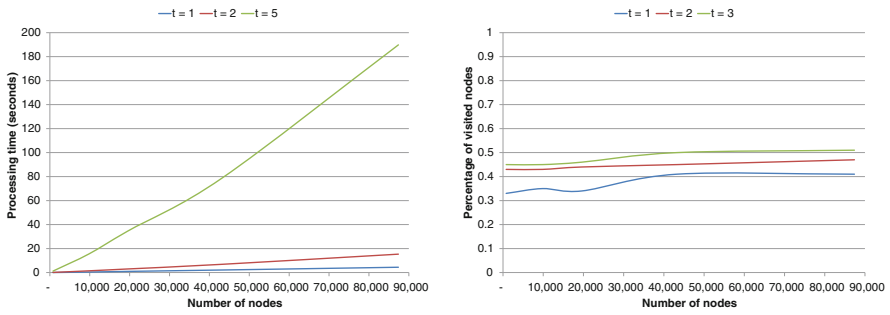
$C_{new} \leftarrow \{c \in C \mid \nexists e \in E, (c, e) \in R_r\}$
while $C_{new} \neq \emptyset$ **do**
  $E_{new} \leftarrow \{e \in E \mid pre(e) \subseteq C \wedge pre(e) \cap C_{new} \neq \emptyset\}$ // Unvisited exploits reachable from C
  for all $e \in E_{new}$ **do**
    for all $c \in pre(e)$ **do**
      $R_r \leftarrow R_r \cup \{(c, e)\}$ // Add an edge from c to e
    end for
    $\{c_1, \ldots, c_m\} \leftarrow \{c \in C \mid (c, e) \in R_r\}$
    $\pi(e) \leftarrow \{P_1 \cup \ldots \cup P_m \cup \{e\} \mid P_i \in \pi(c_i)\}$
    $\pi(e) \leftarrow \{P \in \pi(e) \mid k0d(P) \leq u)$ // Prune paths with more than u zero-day vulnerabilities
  end for
  $C_{new} \leftarrow \emptyset$
  for all $e \in E_{new}$ s.t. $\pi(e) \neq \emptyset$ **do**
    for all $c \in post(e)$ **do**
      $R_i \leftarrow R_i \cup \{(e, c)\}$ // Add an edge from e to c
      $C_{new} \leftarrow C_{new} \cup \{c\}$
      $\pi(c) \leftarrow \bigcup_{e \in E \mid (e,c) \in R_i} \pi(e)$
      if $c \equiv c_g$ **then**
        return $G, min_{P \in \pi(c)} k0d(P)$
      end if
    end for
  end for
  $C \leftarrow C \cup C_{new}$
  $E \leftarrow E \cup E_{new}$
end while

First, we show that, as expected, algorithm *k0dLowerBound* is polynomial for given small values of l. Specifically, Fig. 4(a) shows that the running time of algorithm *k0dLowerBound* grows almost quadratically in the size of attack graphs. It is also clear that the actual running time is quite reasonable even for relatively large graphs (e.g., it only takes about 20 s to determine $k > 3$ for a graph with 80,000 nodes). We can also observe that, although the value of l affects the average running time of the algorithm, such effect is not dramatic for such small values of l (which may be sufficient in most practical cases). This experiment confirms that algorithm *k0dLowerBound* is efficient enough for realistic applications. Next, we show how generating partial attack graphs may lead to savings in both time and storage cost. Specifically, Fig. 4(b) shows the percentage of nodes that are generated by algorithm *k0dLowerBound* in performing the analysis. We can see that such a percentage will decrease while the size of attack graphs increases, which is desirable since this reflects higher amount of savings for larger graphs. It is also clear that although a higher value



(a) Processing time (b) Percentage of nodes

Fig. 4. Processing time and percentage of nodes for algorithm $k0dLowerBound$ vs. number of nodes in the full attack graph for different values of l.



(a) Processing time (b) Percentage of nodes

Fig. 5. Processing time and percentage of nodes for algorithm $k0dUpperBound$ vs. number of nodes in the full attack graph for different values of t.

of l will imply less savings (more nodes need to be generated), in most cases the savings are significant (e.g., less than half of the nodes are generated in most cases). This experiment confirms the effectiveness of our on-demand approach to generating attack graphs.

Similarly, we now show that algorithm $k0dUpperBound$ is polynomial for given small parameters. Specifically, Fig. 5(a) shows that the running time of algorithm $k0dUpperBound$ grows linearly in the size of attack graphs. The value of t represents the number of partial solutions maintained at each step (i.e., the degree of approximation). It is clear that the actual running time is very reasonable even for large graphs (e.g., it only takes less than 20 s for a graph with almost 90,000 nodes). However, we can also observe that the degree of approximation (the value of t) will significantly affect the growth of the average running time of the algorithm, which shows a natural trade-off between accuracy and cost. Next, we also show how generating partial attack graphs may lead to savings for this algorithm. Specifically, Fig. 5(b) shows the percentage of nodes

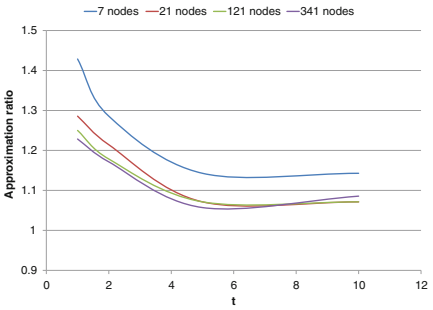


Fig. 6. Approximation ratio of $k0dUpperBound$ vs. t for different graph sizes.

that are generated by algorithm $k0dUpperBound$ in performing the analysis. We can see that such a percentage remains relatively stable across different graph sizes. That is, although the absolute number of generated nodes increases for larger graphs, the ratio remains almost constant, which partially justifies the linear running time of the algorithm. It is also clear that in most cases the savings are significant (less than half of the nodes are generated in most cases). This experiment again confirms the effectiveness of our on-demand attack graph generation.

Finally, we show the accuracy of algorithm $k0dUpperBound$. Specifically, Fig. 6 shows the approximation ratio (i.e., the result u obtained using the algorithm divided by the real value of k obtained using a brute force method) in the approximation parameter t. We can see that, as expected, such a ratio decreases when more partial results are kept at each step, resulting in higher accuracy (and higher cost as well). Overall, the approximation ratio is acceptably low even for a small t (e.g., the result is only about 1.4 times the real value of k when $t = 1$). We can also observe that larger graphs tend to have more accurate results, which is desirable since the analysis actually becomes relevant for larger graphs. Since algorithm $k0dValue(G, E^*, u, c_g)$ is similar to algorithm $k0dLowerBound$ except that it reuses, instead of generating, attack graphs, we expect its running time to be similar to (lower than) that of the latter and thus experiments are omitted here for reasons of space.

7 Conclusions

In this paper, we have studied the problem of efficiently estimating and calculating the k-zero-day safety of large networks. We presented a decision framework comprising three polynomial algorithms for establishing lower and upper bounds of k and for calculating the actual value of k, while generating only partial attack graphs in an on-demand manner. Experimental results confirm the efficiency and effectiveness of our algorithms. Although we have focused on the k-zero-day safety metric in this paper, we believe our techniques can be easily extended to other useful analyses related to attack graphs. Other future work include

fine-tuning the approximation algorithm through various ways for ranking the partial solutions and evaluating the solution on diverse network scenarios.

References

1. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P 2002), Berkeley, CA, USA, pp. 273–284 (2002)
2. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, DC, USA, pp. 217–224 (2002)
3. McHugh, J.: Quality of protection: Measuring the unmeasurable? In: Proceedings of the 2nd ACM Workshop on Quality of Protection (QoP 2006), Alexandria, VA, USA, ACM, pp. 1–2 (2006)
4. Wang, L., Jajodia, S., Singhal, A., Noel, S.: k-zero day safety: measuring the security risk of networks against unknown attacks. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 573–587. Springer, Heidelberg (2010)
5. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004), Fairfax, VA, USA, ACM, pp. 109–118 (2004)
6. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. IEEE Secur. Priv. **4**, 85–89 (2006)
7. The MITRE Corporation: Common Weakness Scoring System (CWSS^{TM}) Version 0.8 (2011). http://cwe.mitre.org/cwss/
8. Dacier, M.: Towards quantitative evaluation of computer security. Ph.D. thesis. Institut National Polytechnique de Toulouse (1994)
9. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the New Security Paradigms Workshop (NSPW 1998), Charlottesville, VA, USA, pp. 71–79 (1998)
10. Mehta, V., Bartzis, C., Zhu, H., Clarke, E.: Ranking attack graphs. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 127–144. Springer, Heidelberg (2006)
11. Balzarotti, D., Monga, M., Sicari, S.: Assessing the risk of using vulnerable components. In: Gollmann, D., Massacci, F., Yautsiukhin, A. (eds.) Quality of Protection. Advances in Information Security, vol. 23, pp. 65–77. Springer, Heidelberg (2006)
12. Pamula, J., Jajodia, S., Ammann, P., Swarup, V.: A weakest-adversary security metric for network configuration security analysis. In: Proceedings of the 2nd ACM Workshop on Quality of Protection (QoP 2006). Advances in Information Security, Alexandria, VA, USA, Springer, vol. 23, pp. 31–68 (2006)
13. Leversage, D.J., Byres, E.J.: Estimating a system's mean time-to-compromise. IEEE Secur. Priv. **6**, 52–60 (2008)
14. Wang, L., Islam, T., Long, T., Singhal, A., Jajodia, S.: An attack graph-based probabilistic security metric. In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 283–296. Springer, Heidelberg (2008)
15. Homer, J., Ou, X., Schmidt, D.: A sound and practical approach to quantifying security risk in enterprise networks, Technical report. Kansas State University (2009)

16. McQueen, M.A., McQueen, T.A., Boyer, W.F., Chaffin, M.R.: Empirical estimates and observations of 0day vulnerabilities. In: Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS 2009), Waikoloa, Big Island, HI, USA (2009)
17. Greenberg, A.: Shopping for Zero-Days: A Price List for Hackers' Secret Software Exploits. Forbes, New York (2012)
18. Shahzad, M., Shafiq, M.Z., Liu, A.X.: A large scale exploratory analysis of software vulnerability life cycles. In: Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), Zurich, Switzerland, pp. 771–781 (2012)
19. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC 2009), Honolulu, HI, USA, pp. 117–126 (2009)