IoT System Vulnerability Analysis and Network Hardening with Shortest Attack Trace in a Weighted Attack Graph

Yinxin Wan*
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
ywan28@asu.edu

Xuanli Lin*
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
xlin54@asu.edu

Abdulhakim Sabur
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
asabur@asu.edu

Alena Chang
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
ahchang@asu.edu

Kuai Xu
Arizona State University
School of Mathematical and Natural Sciences
Glendale, AZ, USA
kuai.xu@asu.edu

Guoliang Xue
Arizona State University
School of Computing and Augmented Intelligence
Tempe, AZ, USA
xue@asu.edu

ABSTRACT

In recent years, Internet of Things (IoT) devices have been extensively deployed in edge networks, including smart homes and offices. Despite the exciting opportunities afforded by the advancements in the IoT, it also introduces new attack vectors and vulnerabilities in the system. Existing studies have shown that the attack graph is an effective model for performing system-level analysis of IoT security. In this paper, we study IoT system vulnerability analysis and network hardening. We first extend the concept of attack graph to weighted attack graph and design a novel algorithm for computing a shortest attack trace in a weighted attack graph. We then formulate the network hardening problem. We prove that this problem is NP-hard, and then design an exact algorithm and a heuristic algorithm to solve it. Extensive experiments on 9 synthetic IoT systems and 2 real-world smart home IoT testbeds demonstrate that our shortest attack trace algorithm is robust and fast, and our heuristic network hardening algorithm is efficient in producing near optimal results compared to the exact algorithm.

CCS CONCEPTS

• **Security and privacy** → **Network security**; **Vulnerability management**; • **Networks** → **Cyber-physical networks**; **Home networks**.

KEYWORDS

shortest attack trace, weighted attack graph, network hardening, IoT security

*Both authors contributed equally to this research.

ACM Reference Format:
Yinxin Wan, Xuanli Lin, Abdulhakim Sabur, Alena Chang, Kuai Xu, and Guoliang Xue. 2023. IoT System Vulnerability Analysis and Network Hardening with Shortest Attack Trace in a Weighted Attack Graph. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3576842.3582326

1 INTRODUCTION AND BACKGROUND

Recent advances in artificial intelligence, wireless communication, and cloud computing have driven the rapid development of IoT devices. Heterogeneous IoT devices designed for different purposes have brought convenience to users and facilitated home automation. However, design flaws, weak authentication, and software vulnerabilities in IoT devices also introduced a broad range of attack vectors, making IoT devices ideal targets for cyber attacks [2].

To analyze and understand the IoT system, many research works have been focused on measuring the network traffic generated by IoT devices [23, 24], analyzing smart home applications [9], and investigating the vulnerabilities in low-energy wireless communication protocols [15, 18]. Recent studies on detecting abnormal IoT device events in smart homes IoT systems have identified the interactions between IoT devices [6, 11] which are critical for identifying abnormal IoT device events.

To systematically profile and analyze IoT system security, IOTA [8] proposes to utilize attack graphs [16, 19, 21] for modeling and studying the vulnerabilities of the whole IoT system where different IoT devices and IoT applications are deployed. In particular, IOTA generates the exploit-dependency attack graph by identifying the vulnerabilities of individual IoT device as well as the physical dependencies and mobile app dependencies among the IoT devices. IOTA further defines the concept of attack trace, which models how the attack could gradually achieve the attack goal from the primitives and proposes a recursive algorithm for computing a shortest attack trace (SAT) to an attack goal as well as an algorithm for calculating the blast radius of a vulnerability in the attack graph. [22] proposes a risk-based approach to identify critical IoT devices that act as enablers in an attack path. The authors combine CVSS metrics to develop a risk-oriented approach that aims to reduce the number of critical attack paths to enhance the mitigation strategy.

In this paper, we consider a more general attack graph model where each vertex and edge is associated with a non-negative weight. We study attack traces and shortest attack traces in a weighted attack graph and show that the unweighted concepts of attack graphs, attack traces, and shortest attack traces are special cases of the weighted counterparts studied in this paper.

We design a novel algorithm for computing a shortest attack trace in a weighted attack graph. Our algorithm is *robust*. It can properly deal with cycles in the attack graphs [16] and is guaranteed to find a shortest attack trace in polynomial time if the attack graph contains at least one attack trace. In case there is no attack trace in the attack graph, our algorithm will stop properly without entering the infinite loop. The state-of-the-art algorithm [8] does not have this robust property, as demonstrated by examples. Our algorithm is *fast*. It has a worst-case running time of $O(m + n \log n)$, where n and m are the number of vertices and edges in the attack graph, respectively. This is faster than the state-of-the-art algorithm [8].

We study the network hardening problem, where a subset of network elements is selected to be hardened (within a budget constraint) in order to maximally increase the height of a shortest attack trace in the hardened attack graph. Since the height of the shortest attack trace represents the level of difficulty for the attacker to gain access to the attack goal, the network hardening problem is important. We prove that the network hardening problem is NP-hard. We design an exact algorithm (as a baseline) for computing an optimal solution using a novel bounding technique. We also design a polynomial-time heuristic algorithm. While the heuristic algorithm does not guarantee to find an optimal solution, our extensive experimental evaluation on different datasets demonstrates that it can produce relatively good results compared with the exact algorithm.

Our main contributions are the following:

- We study weighted attack graphs and present a polynomial-time algorithm for computing a shortest attack trace in a weighted attack graph. Our algorithm can properly deal with cycles in the attack graph and guarantees producing correct results regardless of whether the attack graph contains an attack trace. In contrast, the state-of-the-art algorithm [8] may require exponential time when the attack graph has no cycles and go into an infinite loop when the attack graph contains cycles.
- Using the height of a shortest attack trace as the metric, we study the network hardening problem. We prove that the problem is NP-hard by a polynomial-time reduction from **Knapsack** [12]. We design an exact algorithm with a novel bounding technique. We also design a polynomial-time heuristic algorithm with good performance.
- We implement all algorithms and evaluate them on 9 synthetic IoT systems and 2 real-world smart home IoT testbeds. Our extensive experiment results have demonstrated that our heuristic network hardening algorithm can efficiently produce near optimal results compared to our exact algorithm.

The remainder of this paper is organized as follows. Section 2 defines basic concepts. Section 3 presents our algorithm for computing a shortest attack trace. Section 4 studies the network hardening problem. Section 5 presents the evaluation results. Section 6 concludes this paper.

2 BASIC CONCEPTS

In this section, we first present the concept of attack graphs in network security. We then discuss how to conduct the system-level security analysis of IoT systems using attack graphs. We show that the concept of attack trace [8] is critical in analyzing the attack graphs. Subsequently, we define weighted attack graphs and discuss attack traces and shortest attack traces in a weighted attack graph.

2.1 Attack Graph

The concept of attack graphs has been widely adopted in cyber security to provide a systematic view of network security and to analyze the vulnerabilities in the system. State-based attack graphs [21] model the network system as a finite state machine where state transitions correspond to the intruder's atomic attacks. However, the size of the state-based attack graphs could scale exponentially to the input size, thus limiting their applications. Logical attack graphs [16, 17], on the other hand, is more efficient as the logical attack graphs can be generated in polynomial time. In this paper, we focus on the logical attack graph model, and we refer to logical attack graphs as attack graphs in the rest of this paper.

We use `MulVAL` [17] to generate attack graphs. The inputs to `MulVAL` are system configurations and interaction rules in Datalog tuples and Datalog rules. Then the `MulVAL` reasoning engine will call the Prolog system XSB [20] to evaluate the interaction rules on input facts and subsequently output the logical attack graph.

The `MulVAL`-generated logical attack graphs are directed graphs and could contain *cycles* [16]. There are three kinds of vertices in the logical attack graph: *primitive fact* vertices, *derived fact* vertices, and *rule* vertices. The attack goal is a special *derived fact* vertex. The edges in the graph represent the "depends on" relation. Each derived fact vertex is dependent on any one of its incoming neighbors. Hence a derived fact vertex is also called an `OR` vertex.

Each rule vertex is dependent on all of its incoming neighbors. Hence a rule vertex is also called an `AND` vertex.

2.2 System-Level IoT Security Analysis

An attack graph conveys critical information regarding the system vulnerabilities, making it a powerful model for methodically measuring and analyzing IoT system security. IOTA [8] formally develops a framework for efficiently constructing attack graphs given the IoT system configurations.

In IOTA, the attack graph of an IoT system is generated using MulVal [17] with Prolog clauses representing the exploit models and device dependencies as inputs. The exploit models are built by scanning the IoT system configurations for individual vulnerability. Three types of IoT device dependencies are identified by IOTA, including app-based dependency, indirect physical dependency, and direct physical dependency. The app-based dependencies are extracted by applying natural language processing (NLP) techniques to the descriptions of smart home apps. IOTA [8] proposes the metrics of the *shortest attack trace* to an attack goal and the *blast radius* of a vulnerability for interpreting the generated attack graph.

A recursive algorithm is proposed to calculate the shortest attack trace to the attack goal in the attack graph.

The authors of [1] study the IoT device deployment problem systematically by considering the whole IoT system with the attack graph. Both the problem of deploying all required IoT devices with minimal security implications and computing the maximal number of IoT devices to be deployed without increasing the security risk of the network are studied in [1]. A heuristic search algorithm for solving both optimization problems is designed by [1].

2.3 Weighted Attack Graph

We extend the concept of attack graphs [16] and propose the concept of weighted attack graphs.

Definition 1 (Weighted Attack Graph). A weighted attack graph is a directed graph $G = (V_p, V_d, V_r, E, w, g)$, where V_p, V_d and V_r denote the set of *primitive fact* vertices (source vertices), the set of *derived fact* vertices (OR vertices), and the set of *rule* vertices (AND vertices), respectively; $E \subseteq \{(V_p \cup V_d) \times V_r\} \cup \{V_r \times V_d\}$ is the set of directed edges; $w(v) \geq 0$ is the *weight* for a vertex v; $w(e) \geq 0$ the *weight* for an edge $e \in E$; $g \in V_d$ is the attacker's goal. □

In the above definition, the notation $w(\cdot)$ is *overloaded*: $w(v)$ denotes the weight of vertex $v \in V_p \cup V_d \cup V_r$, while $w(e)$ denotes the weight of edge $e \in E$. It should be noted that this notation overloading simplifies the presentation without creating ambiguity, as its meaning is clear from the context.

Let G be a weighted attack graph. We use V_p^G to denote the set of primitive fact vertices in G, V_d^G to denote the set of derived fact vertices in G, V_r^G to denote the set of rule vertices in G, E^G to denote the set of edges in G, g^G to denote the goal vertex in G, and w^G to denote the weight function in G. We also use V^G to denote the set of all vertices, i.e., $V^G = V_p^G \cup V_d^G \cup V_r^G$. When the graph G is clear from the context, we may drop the superscript and use the notations $V_p, V_d, V_r, V, E, g, w$.

The concept of *weighted attack graph* contains the concept of (unweighted) *attack graph* studied in the literature [8, 16, 19, 21] as a special case. When we restrict $w(v) = 0, \forall v \in V$ and $w(e) = 1, \forall e \in E$, the weighted attack graph reduces to the classic (unweighted) attack graph. For this reason, *we will use the notations of weighted attack graph and (unweighted) attack graph interchangeably, unless specified otherwise.*

Note that not all vulnerabilities are the same in terms of penetrability and ubiquity. Therefore, the inclusion of vertex and edge weights in attack graphs can more accurately characterize the vulnerabilities of the IoT system. This is the main motivation for us to study weighted attack graphs in this paper. We will discuss the impact and cost of networking hardening in Section 4.

2.4 Attack Trace

Given an attack graph, an attacker can have different attacking strategies in order to achieve the attack goal. To model the attacker's attacking strategies, we adopt the concept of attack trace introduced in IOTA [8], which can accurately profile the dependency relationships between vertices in an attack graph. Note that IOTA [8] defines attack traces for unweighted attack graphs. In this paper, we extend the concept of attack traces to weighted attack graphs in Definition 2.

Definition 2 (Attack Trace in a Weighted Attack Graph). Let G be a weighted attack graph. Let $t \in V^G$ be any vertex in G. An *attack trace to vertex t* in G, denoted by $T^{G,t}$, is a subgraph of G that satisfies the following properties:

1. Let v be any vertex in $T^{G,t}$. If $v \in V_d^G$, then the in-degree of v in $T^{G,t}$ is 1. In other words, for any OR vertex v in $T^{G,t}$, $T^{G,t}$ contains exactly one of the edges in $\{(u,v)|(u,v) \in E^G\}$.
2. Let v be any vertex in $T^{G,t}$. If $v \in V_r^G$, then the in-degree of v in $T^{G,t}$ is equal to the in-degree of v in G. In other words, for any AND vertex v in $T^{G,t}$, $T^{G,t}$ contains all edges in the set $\{(u,v)|(u,v) \in E^G\}$.
3. Let v be any vertex in $T^{G,t}$. If the in-degree of v in $T^{G,t}$ is 0, then $v \in V_p^G$. In other words, every source vertex in $T^{G,t}$ is a primitive fact vertex.
4. Vertex t is the only vertex in $T^{G,t}$ with out-degree 0 in $T^{G,t}$.

The *height* of the attack trace $T^{G,t}$, denoted by $\mathcal{H}(T^{G,t})$, is the length of the longest path in $T^{G,t}$, where the length of a path is the summation of the weights of the vertices and edges on the path. An attack trace to vertex g is called an attack trace of G. We may use T^G to denote $T^{G,g}$ since the goal vertex g is unique in G. □

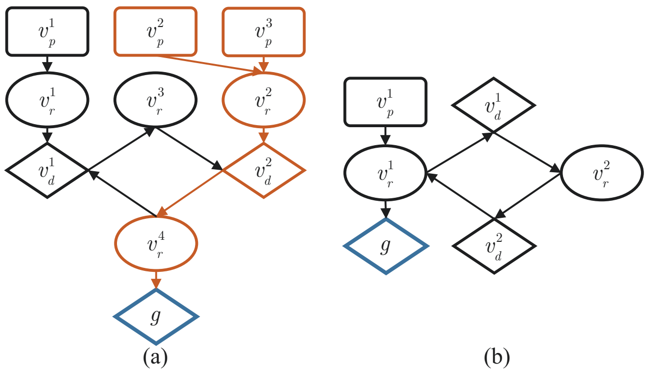


Figure 1: (a) An attack graph with two attack traces. (b) An attack graph with no attack trace.

Example. To illustrate the concept of attack trace in a weighted attack graph, consider the attack graph G in Fig. 1(a). We assume that $w(v) = 0$ for each vertex v and $w(e) = 1$ for each edge in G. There are two attack traces to node g in the attack graph G:

1. Attack trace $T_1^{G,g}$ with vertices $\{v_p^2, v_p^3, v_r^2, v_d^2, v_r^4, g\}$, and edges $\{(v_p^2, v_r^2), (v_p^3, v_r^2), (v_r^2, v_d^2), (v_d^2, v_r^4), (v_r^4, g)\}$. This attack trace is highlighted in orange in Fig. 1(a).
2. Attack trace $T_2^{G,g}$ with vertices $\{v_p^1, v_r^1, v_d^1, v_r^3, v_d^2, v_r^4, g\}$ and edges $\{(v_p^1, v_r^1), (v_r^1, v_d^1), (v_d^1, v_r^3), (v_r^3, v_d^2), (v_d^2, v_r^4), (v_r^4, g)\}$

The height of attack trace $T_1^{G,g}$ is 4, while the height of attack trace $T_2^{G,g}$ is 6.

Definition 3 (Shortest Attack Trace). Let G be a weighted attack graph. Let $t \in V^G$ be any vertex in G. Let $T_{sh}^{G,t}$ be an attack trace in G. $T_{sh}^{G,t}$ is said to be a shortest attack trace to vertex t, if $\mathcal{H}(T_{sh}^{G,t}) \leq \mathcal{H}(T^{G,t})$ for any attack trace $T^{G,t}$ to vertex t. A shortest attack trace to vertex $g \in V^G$ (the attacker's goal) is called a shortest attack trace of G. □

The concept of (unweighted) attack trace and shortest attack trace was introduced in [8] for unweighted attack graphs, where the height of an attack trace is defined as the longest path (measured by hop-count) in the attack trace. One can verify that the height of an attack trace defined above is the same as the height of an attack trace defined in [8] when $w(v) = 0, \forall v \in V$ and $w(e) = 1, \forall e \in E$.

A given attack graph may have multiple nonidentical shortest attack traces. However, for any two shortest attack traces T_1 and T_2 of attack graph G, their heights must be equal, i.e., $\mathcal{H}(T_1) = \mathcal{H}(T_2)$. To simplify the presentation without confusion, we will use T_{sh} to denote a shortest attack trace.

Following a shortest attack trace is the *optimal* strategy for the attacker, i.e., it requires the *least* effort/time for the attacker to gain control of the attack goal. Therefore $\mathcal{H}(T_{\text{sh}})$ is the least effort/time that the attacker needs to spend to gain control of the attack goal.

3 COMPUTING A SHORTEST ATTACK TRACE: PERSPECTIVE OF THE ATTACKER

In this section, we present a novel polynomial time algorithm for computing a shortest attack trace in a weighted attack graph. Our algorithm is inspired by the approach in IOTA [8].

3.1 A Novel Algorithm for Computing an SAT

Our algorithm for computing a shortest attack trace is Algorithm 1, named `SAT`(G). `SAT`(G) takes an attack graph G as input. It either stops in Line 27, claiming that there is no attack trace in G, or stops in Line 37, outputting a shortest attack trace T_{sh}.

Before illustrating `SAT`(G) with examples, we explain its main steps in the following. Line 1 creates an empty priority queue PQ such as the Fibonacci heap [10]. We use a color system for the vertices. A vertex v is `WHITE` before it is inserted into PQ, `GRAY` when it is on PQ, and `BLACK` after it is extracted from PQ. The in-degree of v is recorded in $v.in$. The vertex attribute $v.done$ denotes the number of edges into v that have been traversed by the algorithm. If v is an `OR` vertex, $v.height$ is initialized to ∞, and decreased to the *minimum* of the heights of all attack traces to v computed so far. If v is an `AND` vertex, $v.height$ is initialized to $-\infty$, and increased to the *maximum* of $\{u.height + w(u,v) + w(v) | u.color = \texttt{BLACK}\}$. When all incoming neighbors of v are extracted from PQ, a shortest attack trace to vertex v is computed, and its height is stored in $v.height$. These attributes for $v \in V_d \cup V_r$ are initialized in Lines 2-7.

In Lines 8-9, each vertex $v \in V_p$ is inserted into PQ, with $v.height$ set to its final value of $w(v)$. The main body of the algorithm is the while-loop in Lines 10-26, where the vertex on PQ with the minimum height value is extracted. Whenever a vertex u is extracted from PQ, edges in the form (u,v) are traversed and the attributes of v are updated accordingly. An `OR` vertex v is inserted into PQ as soon as an attack trace $T^{G,v}$ is computed (note that $T^{G,v}$ does not have to be a shortest attack trace to v). An `AND` vertex v is inserted into PQ as soon as a shortest attack trace $T^{G,v}$ is computed. If the goal vertex g is extracted, the algorithm goes to Line 28 to output the computed shortest attack trace T_{sh} and exits in Line 37. If PQ becomes empty before g is inserted into it, the algorithm exits in Line 27, claiming that G does not contain an attack trace.

Algorithm 1: `SAT`(G)

```
   Input: $G = (V_p, V_d, V_r, E, w, g)$: an attack graph
   Output: $T_{\text{sh}}$: a shortest attack trace to $g$ in $G$.
1  Create an empty priority queue $PQ$;
2  for $\forall v \in V_d$ do
3      $v.type \leftarrow 1$; $v.in \leftarrow 0$; $v.done \leftarrow 0$; $v.height \leftarrow \infty$;
        $v.color \leftarrow$ WHITE;
4  for $\forall v \in V_r$ do
5      $v.type \leftarrow 2$; $v.in \leftarrow 0$; $v.done \leftarrow 0$; $v.height \leftarrow -\infty$;
        $v.color \leftarrow$ WHITE;
6  for $\forall e = (x, y) \in E$ do
7      $y.in \leftarrow y.in + 1$;
8  for $\forall v \in V_p$ do
9      $v.type \leftarrow 0$; $v.height \leftarrow w(v)$; Insert $v$ to $PQ$;
        $v.color \leftarrow$ GRAY;
10 while $PQ \neq \emptyset$ do
11     $u \leftarrow$ ExtractMin$(PQ)$; $u.color \leftarrow$ BLACK;
12     if $(u == g)$ goto 28; // Shortest attack trace found
13     for $\forall v \in u.adj$ with $v.color \neq$ BLACK do
14         $temp \leftarrow u.height + w(u, v) + w(v)$;
15         if $(v.type == 1)$ then
16             if $(v.done == 0)$ then
17                 $v.height \leftarrow temp$; $v.parent \leftarrow u$;
18                 Insert $v$ to $PQ$; $v.color \leftarrow$ GRAY;
19             else if $(temp < v.height)$ then
20                 $v.height \leftarrow temp$; $v.parent \leftarrow u$;
                       // DecreaseKey at vertex $v$ triggered
21             $v.done \leftarrow v.done + 1$;
22         if $(v.type == 2)$ then
23             $v.height \leftarrow \max\{v.height, temp\}$;
24             $v.done \leftarrow v.done + 1$;
25             if $v.done == v.in$ then
26                 Insert $v$ to $PQ$; $v.color \leftarrow$ GRAY;
27 stop: There is no attack trace in the attack graph $G$; // Exit 1
28 Create an empty FIFO queue $Q$; Insert $g$ to $Q$;
29 Create an empty set $T_{\text{sh}}$ of edges in the shortest attack trace;
30 while $Q \neq \emptyset$ do
31     $v \leftarrow Dequeue(Q)$;
32     if $(v.type == 1)$ then
33         $u \leftarrow v.parent$; Insert $u$ to $Q$; $T_{\text{sh}} \leftarrow T_{\text{sh}} \cup \{(u, v)\}$;
34     if $(v.type == 2)$ then
35         for $\forall u \in V$ such that $(u, v) \in E$ do
36             Insert $u$ to $Q$; $T_{\text{sh}} \leftarrow T_{\text{sh}} \cup \{(u, v)\}$;
37 Output $T_{\text{sh}}$; // Exit 2: $T_{\text{sh}}$ is a shortest attack trace
```

3.2 Walk-through Examples

We use the examples in Fig. 1 to illustrate Algorithm 1. The attack graph in Fig. 1(a) has two attack traces. The attack graph in Fig. 1(b) has no attack trace. We assume that $w(v) = 0$ for each vertex v and $w(e) = 1$ for each edge in Fig. 1. Hence, the weighted attack graphs reduce to unweighted attack graphs.

When Algorithm 1 is applied to the attack graph in Fig. 1(a), the main steps are as follows. (S1) An empty priority queue PQ is initialized and the primitive fact vertices v_p^1, v_p^2, v_p^3 are inserted into PQ with $v_p^1.height = 0$, $v_p^2.height = 0$, $v_p^3.height = 0$. (S2) v_p^1

is extracted from PQ. Edge (v_p^1, v_r^1) is traversed,and $v_r^1.height$ is increased from $-\infty$ to $v_p^1.height + w(v_p^1, v_r^1) + w(v_r^1) = 1$. Since we have traversed all edges going into v_r^1, the AND vertex v_r^1 is inserted into PQ. (S3) v_p^2 is extracted from PQ. Edge (v_p^2, v_r^2) is traversed, and $v_r^2.height$ is increased from $-\infty$ to $v_p^2.height + w(v_p^2, v_r^2) + w(v_r^2) = 1$. Unlike the previous step, the AND vertex v_r^2 is not inserted into PQ at this moment, since we have not traversed all edges going into v_r^2 yet. (S4) v_p^3 is extracted from PQ. Edge (v_p^3, v_r^2) is traversed, $v_r^2.height$ remains unchanged since $v_p^3.height + w(v_p^3, v_r^2) + w(v_r^2)$ is not larger than $v_r^2.height$. However, since we have now traversed all edges going into v_r^2, the AND vertex v_r^2 is inserted into PQ. (S5) v_r^1 is extracted from PQ. Edge (v_r^1, v_d^1) is traversed, and $v_d^1.height$ is decreased from ∞ to 2. The OR vertex v_d^1 is inserted into PQ. (S6) v_r^2 is extracted from PQ. The OR vertex v_d^2 is inserted into PQ with $v_d^2.height = 2$. (S7) v_d^1 is extracted from PQ. The AND vertex v_r^3 is inserted into PQ with $v_r^3.height = 3$. (S8) v_d^2 is extracted from PQ. The AND vertex v_r^4 is inserted into PQ with $v_r^4.height = 3$. (S9) v_r^3 is extracted from PQ. The edge (v_r^3, v_d^2) is traversed. Since v_d^2 has been extracted from PQ, the attributes at v_d^2 are not updated. (S10) v_r^4 is extracted from PQ. The OR vertex g is inserted into PQ with $g.height = 4$. (S11) g is extracted from PQ. The algorithm outputs the attack trace $\{(v_r^4, g), (v_d^2, v_r^4), (v_r^2, v_d^2), (v_p^2, v_r^2), (v_p^3, v_r^2)\}$ and stops in Line 37. The height of the computed shortest attack trace is 4. *This example is representative of cases where the attack graph contains at least one attack trace.*

When Algorithm 1 is applied to the attack graph in Fig. 1(b), the main steps are as follows. (S1) An empty priority queue PQ is initialized and the primitive fact vertex v_p^1 is inserted into PQ with $v_p^1.height = 0$. (S2) v_p^1 is extracted from PQ. The edge (v_p^1, v_r^1) is traversed, and $v_r^1.height$ is increased from $-\infty$ to $v_p^1.height + w(v_p^1, v_r^1) + w(v_r^1) = 1$. Since we have not traversed all edges going into v_r^1 (edge (v_d^2, v_r^1) has not been traversed) yet, the AND vertex v_r^1 is not inserted into PQ at this moment. (S3) Algorithm 1 finds $PQ = \emptyset$ and goes to Line 27. It stops, claiming that there is no attack trace in the attack graph shown in Fig. 1(b). *This example is representative of cases where the attack graph does not contain any attack trace.*

3.3 Analysis of the Algorithm

In this section, we analyze the properties of the algorithm. In Theorem 1, we analyze the worst-case running time of the algorithm. In Theorems 2-3, we prove the correctness of the algorithm.

Theorem 1. Let n be the number of vertices in G and m be the number of edges in G. The worst-case running time of Algorithm 1 is $O(m + n \log n)$. □

Proof. The time spent on Lines 1-9 is $O(n+m)$ as we spend $O(1)$ time for each vertex and each edge. The while-loop from Line 10 to Line 26 performs at most n ExtractMin operations, at most n Insertion operations, and at most m DecreaseKey operations. If we use the Fibonacci heap [10] to implement the priority queue, this portion has a worst-case running time $O(m + n \log n)$. Line 28 takes $O(1)$ time. The while-loop from Line 30 to Line 36 takes $O(m)$ time, as there are at most m edges in T. Hence the worst-case time complexity of Algorithm 1 is $O(m + n \log n)$. □

Remarks. At the high level, our algorithm for computing a shortest attack trace follows the same principle as the algorithm in [8]. However, there are subtle differences that may affect the worst-case running time. The algorithm in [8] uses a top-down approach (without momoization). Our algorithm uses a bottom-up approach, where no instance is solved more than once. Our algorithm can compute a shortest attack trace whenever an attack trace exists, and can recognize if an attack trace does not exist, in polynomial time (refer to Theorems 1 and 2).

Theorem 2. If the attack graph G contains an attack trace, then Algorithm 1 correctly computes a shortest attack trace, with the edges in the shortest attack trace stored in T_{sh}. In this case, the algorithm exits in Line 37. □

Proof. Our proof relies on the following claims:

(a) Let $v \in V^G$ be any vertex in G. The following is always true throughout the execution of the algorithm: $v.color$ is WHITE if and only v has not been inserted into the priority queue PQ, $v.color$ is GRAY if and only v is on the priority queue PQ, and $v.color$ is BLACK if and only v has been extracted from the priority queue PQ.

(b1) Let v be any OR vertex in G. The value $v.height$ is monotonically non-increasing during the execution of the algorithm.

(b2) Let v be any OR vertex in G. The first time $v.height$ is reduced from ∞ to a real number, v is inserted into PQ, and $v.height$ is the height of *some* (not necessarily the shortest) attack trace to vertex v; When v is extracted from PQ, $v.height$ is the height of a shortest attack trace to vertex v.

(c1) Let v be any AND vertex in G. The value $v.height$ is monotonically non-decreasing during the execution of the algorithm.

(c2) Let v be any AND vertex in G. When v is inserted into PQ, $v.height$ is the height of a shortest attack trace to vertex v. The value $v.height$ will not be changed again after v is inserted into PQ. When v is extracted from PQ, $v.height$ is the height of a shortest attack trace to vertex v.

(d) If vertex α is extracted from PQ before vertex β is extracted, then $\alpha.height$ (at the time α is extracted) is less than or equal to $\beta.height$ (at the time β is extracted).

Claim (a) follows directly from the algorithm. We note that vertex v becomes GRAY and gets inserted into PQ only in Line 9 (for $v \in V_p$), Line 18 (for $v \in V_d$), and Line 26 (for $v \in V_r$); and vertex v gets extracted from PQ and becomes BLACK only in Line 11.

To prove Claim (b1), we note that for $v \in V_d^G$, $v.height$ is initialized to ∞ in Line 5, and changed (to a smaller value) in Line 17 and Line 20. To prove Claim (c1), we note that for $v \in V_r^G$, $v.height$ is initialized to $-\infty$ in Line 7, and changed (to a larger value) in Line 23.

To prove Claim (d), we note that vertices is extracted from PQ by the ExtractMin operation. Therefore, when α is extracted, the height of α is the smallest among all vertices on PQ. Since the vertex weights and edge weights are all non-negative, for any vertex v that has not been extracted from PQ before vertex α, we will not have $v.height < \alpha.height$ when $v.done \geq 1$. This proves Claim (d).

Claims (b2) and (c2) can be proved using mathematical induction. Let v be an `OR` vertex. The first time an incoming neighbor of v is extracted from PQ, the height of v is decreased from ∞ to a real number (which is the height of an attack trace to v). This value may be further reduced when other incoming neighbors of v are extracted from PQ. When all incoming neighbors of v are extracted from PQ, the height of v will no longer be decreased.

Let v be an `AND` vertex. $v.height$ is initialized to $-\infty$ in Line 5. Every time an incoming neighbor v' of v is extracted, $v.height$ will be set to the maximum of its current value and $v'.height + w(v', v) + w(v)$. Therefore, when all incoming neighbors of v are extracted from PQ, the $v.height$ is the height of a shortest attack trace to vertex v. The above analysis, together with (d), proves (b2) and (c2).

Now assume that there is an attack trace in G. Since the number of attack traces in G is finite, there must be a shortest attack trace. Let h_{opt} be the height of the shortest attack trace to g. Following the above analysis, vertex v will be inserted into and extracted from PQ provided that the height of the shortest attack trace to v is *smaller than* the h_{opt}. Hence, the attack goal g will be inserted into and extracted from PQ. Therefore, Algorithm 1 exits at Line 37. □

Theorem 3. If the attack graph G contains no attack trace, then Algorithm 1 stops in Line 27. □

Proof. The algorithm stops after $O(m + n \log n)$ basic operations. If it stops at Line 37, it must have computed a shortest attack trace. The only other place for the algorithm to stop is Line 27. Therefore if there is no attack trace to g, Algorithm 1 must stop at Line 27. □

4 OPTIMAL NETWORK HARDENING: PERSPECTIVE OF THE DEFENDER

In Section 3, we presented an efficient algorithm for computing a shortest attack trace. The best strategy for the attacker to gain access to the attack goal g is to launch an attack along a shortest attack trace. Therefore, the height of a shortest attack trace, $\mathcal{H}(T_{\text{sh}})$, is a good metric to measure the *hardness* for the attacker to gain access to its attack goal g. The larger the value of $\mathcal{H}(T_{\text{sh}})$, the less vulnerable the system is.

Assume that we can harden a network element z (a vertex or an edge) to increase the value of $w(z)$. Then we can increase the height of the shortest attack trace by hardening some selected network elements. However, hardening a network element comes with a cost. Also, some network elements are hardenable, while others are not. Therefore, a natural question to ask is: *What is the best strategy to harden the network with a given budget constraint*? This section is devoted to answering the above question.

4.1 The Network Hardening Problem

A *network element* in this paper refers to either a vertex or an edge in G. Since there are $n = |V|$ vertices and $m = |E|$ edges in G, we have a total of $N = n + m$ network elements. We define a one-to-one mapping η from the set of integers $\{1, 2, \ldots, N\}$ to the set of network elements $V \cup E$ such that $\eta(k) \in V$ for $k = 1, 2, \ldots, n$, and $\eta(k) \in E$ for $k = n+1, n+2, \ldots, n+m$. We use a binary-valued array $\phi[1 : N]$ to indicate whether a network element is hardenable. In particular, $\phi[k] = 1$ if $\eta(k)$ is hardenable, and $\phi[k] = 0$ otherwise.

Let $k \in [1, N]$ be an integer. If $\phi[k] = 1$, we can harden network element $\eta(k)$ with a cost of $c[k] > 0$ to increase the weight of $\eta(k)$ from $w(\eta(k))$ to $w(\eta(k)) + \delta[k]$. If $\phi[k] = 0$, $\eta(k)$ is not hardenable. For convenience, we define $\delta[k] = 0$ and $c[k] = \infty$ in this situation.

Definition 4. Let the attack graph G be given, together with mapping η and network hardening information ϕ, δ, and c. A binary-valued array $X[1 : N]$ is said to be a feasible hardening strategy for budget $B \geq 0$, if $X[k] \leq \phi[k],\ k = 1, 2, \ldots, N$ and $\sum_{k=1}^{N} X[k] \times c[k] \leq B$. □

The meaning of the array $X[1 : N]$ as a hardening strategy is the following. For $k = 1, 2, \ldots, N$, network element $\eta(k)$ is hardened if and only if $X[k] = 1$. Since $X[k] \leq \phi[k]$, only hardenable network elements will be hardened. Since $\sum_{k=1}^{N} X[k] \times c[k] \leq B$, the total cost for hardening does not exceed the given budget B.

We use $G(X)$ to denote the hardened attack graph corresponding to X, and use $T_{\text{sh}}(X)$ and $\mathcal{H}(T_{\text{sh}}(X))$ to denote the shortest attack trace of $G(X)$ and the height of $T_{\text{sh}}(X)$, respectively. The decision version and the optimization version of the network hardening problem are formally defined in the following.

Definition 5 (Decision Hardening Problem). Let the attack graph G be given, together with network hardening information η, ϕ, δ, and c. Let $\mathbb{B}$ be the budget for network hardening, and $\mathbb{H}$ be the desired level of network hardness. The decision network hardening problem (**DHP**) asks whether there exists a feasible hardening strategy X such that $\mathcal{H}(T_{\text{sh}}(X)) \geq \mathbb{H}$. When the answer is `YES`, the problem also asks for the corresponding hardening strategy X. □

Definition 6 (Optimization Hardening Problem). Let the attack graph G be given, together with network hardening information η, ϕ, δ, and c. Let $\mathbb{B}$ be the budget for network hardening. The optimization network hardening problem (**OHP**) asks for a feasible hardening strategy X_{opt} such that $\mathcal{H}(T_{\text{sh}}(\text{X}_{\text{opt}})) \geq \mathcal{H}(T_{\text{sh}}(X))$ for every feasible hardening strategy X. □

OHP can be formulated as the following optimization problem.

$$\texttt{maximize}\ \mathcal{H}(T_{\text{sh}}(X)) \tag{1}$$

$$\texttt{s.t.}\ X[k] \in \{0, 1\},\ k = 1, 2, \ldots, N, \tag{2}$$

$$X[k] \leq \phi[k],\ k = 1, 2, \ldots, N, \tag{3}$$

$$\sum_{k=1}^{N} X[k] \times c[k] \leq \mathbb{B}. \tag{4}$$

In Section 4.2, we will study the computational complexity of the network hardening problem. In Sections 4.3 and 4.4, we will present optimal and heuristic algorithms, respectively, for solving the **OHP** problem.

4.2 Hardness of the Problem

We present a polynomial-time reduction from **Knapsack** [12] to **DHP**. An instance of **Knapsack** is given by a finite set U, a *size* $s(u) \in Z^+$ and a *reward* $r(u) \in Z^+$ for each $u \in U$, a *size constraint* $\mathbb{B} \in Z^+$, and a *reward goal* $\mathbb{R} \in Z^+$. It asks for a subset $S \subseteq U$ such that

$$\sum_{u \in S} s(u) \leq \mathbb{B} \text{ and } \sum_{u \in S} v(u) \geq \mathbb{R}. \tag{5}$$

Theorem 4 (Hardness of Network Hardening). The network hardening problems **DHP** and **OHP** are both NP-hard. □

Proof. Let $\mathcal{I}_1 = (U, s(\cdot), r(\cdot), \mathbb{B}, \mathbb{R})$ be an arbitrary instance of **Knapsack**, where $U = \{u_1, u_2, \ldots, u_K\}$. If K is an even integer, we construct a corresponding instance $\mathcal{I}_1' = (U', s'(\cdot), r'(\cdot), \mathbb{B}', \mathbb{R}')$ with $|U'| = |U| + 1$ such that

- $U' = U \cup \{u_{K+1}\}$,
- $s'(u_k) = s(u_k)$, $1 \le k \le K$,
- $s'(u_{K+1}) = \min\{s(u_k) | 1 \le k \le K\}$,
- $r'(u_k) = r(u_k)$, $1 \le k \le K$,
- $r'(u_{K+1}) = 1 + \sum_{k=1}^{K} r(u_k)$,
- $\mathbb{B}' = \mathbb{B} + s(u_{K+1})$,
- $\mathbb{R}' = \mathbb{R} + r(u_{K+1})$.

Since $s(u_{K+1}) \le s(u_k)$ for $1 \le k \le K$ and $r(u_{K+1}) > \sum_{k=1}^{K} r(u_k)$, $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_1'$ has a solution. In addition, any solution to $\mathcal{I}_1'$ must be the union of S and $\{u_{K+1}\}$, where S is a solution to $\mathcal{I}_1$. Therefore, without loss of generality, we may assume that for instance $\mathcal{I}_1$, K is an odd integer.

Given instance $\mathcal{I}_1$ of **Knapsack**, we construct a corresponding instance $\mathcal{I}_2$ of **DHP** in the following. The weighted attack graph is $G = (V_p, V_d, V_r, E, g, w)$, where $V_p = \{v_1\}$, $V_d = \{v_3, v_5, v_7, \ldots, v_K\}$, $V_r = \{v_2, v_4, v_6, \ldots, v_{K-1}\}$, $E = \{(v_{k-1}, v_k) | k = 2, 3, \ldots, K\}$, $g = v_K$, and $w(v_k) = 1$ for each $k = 1, 2, \ldots, K$ and $w(e) = 1$ for each $e \in E$. There are K vertices and $K-1$ edges in G, leading to $N = 2K - 1$ network elements. Define η, ϕ, δ, and c such that

$$\eta(k) = v_k,\ \phi[k] = 1, \qquad k = 1, 2, \ldots, K, \tag{6}$$

$$\delta[k] = r(u_k),\ c[k] = s(u_k), \qquad k = 1, 2, \ldots, K, \tag{7}$$

$$\eta(K+k) = (v_k, v_{k+1}),\ \phi[K+k] = 0, \qquad k = 1, 2, \ldots, K-1, \tag{8}$$

$$\delta[K+k] = 0,\ c[K+k] = \infty, \qquad k = 1, 2, \ldots, K-1. \tag{9}$$

Define the hardening budget to be $\mathbb{B}$, and the desired hardness level to be $\mathbb{H} = \mathbb{R} + (K+1)$. Then $\mathcal{I}_1$ has a solution if and only if **DHP** has a solution. In addition, if X is a solution to **DHP**, then $S = \{\eta(k) | X[k] = 1\}$ is a solution to **Knapsack** and vice versa. Since **Knapsack** is NP-hard, we have proved that **DHP** is NP-hard. Since **OHP** is the optimization version of **DHP**, **OHP** is also NP-hard. □

4.3 An Exact Algorithm

We design a branch and bound algorithm for computing an optimal solution to **OHP**. The algorithm is listed in Algorithm 2. While the branch and bound algorithm paradigm has been known for a long time, *the bounding technique in our algorithm is novel.* Its effectiveness will be demonstrated in our evaluation results.

Theorem 5. Algorithm 2 (ExactBnB) always computes an optimal hardening strategy. □

Proof. Algorithm 2 traverses a decision tree with 2^K leaf vertices, while cutting branches that do not contain a better solution whenever we know it. We use B to denote the *residual budget*, which is the initial budget $\mathbb{B}$ minus the sum of the costs of the subset of network elements selected to be hardened. In general, given the values of $X[j]$ for $1 \le j \le k-1$ such that $X[j] \le \phi[j]$ for $1 \le j \le k-1$ and $\sum_{j=1}^{k-1} X[j] \times c[j] \le \mathbb{B}$, we decide whether to explore or cut the branch with $X[k] = 1$, and the branch with $X[k] = 0$.

Algorithm 2: ExactBnB$(G, N, \eta, \phi, \delta, c, \mathrm{H_{best}}, \mathrm{X_{best}}, \mathbb{B})$

```
Input: $G = (V, E, g)$ is a weighted attack graph; $N = |V| + |E|$ is
       the number of network elements; $\eta : \{1, 2, \ldots, N\} \mapsto V \cup E$
       is a one-to-one mapping; $\phi[k]$ is 1 if $\eta[k]$ is hardenable, 0
       otherwise; $\delta[k]$ and $c[k]$ are the *added strength* and
       *hardening cost* for network element $\eta[k]$ when $\phi[k]$ is 1; $\mathbb{B}$
       is the budget constraint for network hardening; $\mathrm{X_{best}}$ is
       some hardening strategy; $\mathrm{H_{best}}$ is the height of the shortest
       attack trace after the network is hardened using $\mathrm{X_{best}}$.
Output: An optimal hardening strategy given by $\mathrm{X_{best}}[k]$, $j = 1, 2,$
        $\ldots, N$ together with the height of the shortest attack trace
        after hardening using the optimal strategy.
Create an array $X[1:N]$ and an array $c_{\mathrm{sum}}[1:N+1]$;
$c_{\mathrm{sum}}[N+1] \leftarrow 0$;
for $k = N, N-1, \ldots, 1$ do
    if $(\phi[k] == 1)$ then
        $c_{\mathrm{sum}}[k] \leftarrow c_{\mathrm{sum}}[k+1] + c[k]$;
    else
        $c_{\mathrm{sum}}[k] \leftarrow c_{\mathrm{sum}}[k+1]$;
$B \leftarrow \mathbb{B}$;
Branch1$(B, 1, X, \mathrm{X_{best}}, \mathrm{H_{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}})$;
Branch0$(B, 1, X, \mathrm{X_{best}}, \mathrm{H_{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}})$;
Output $\mathrm{H_{best}}$ and $\mathrm{X_{best}}$.
```

Algorithm 3: Branch1$(B, k, X, \mathrm{X_{best}}, \mathrm{H_{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}})$

```
Input: Current $\mathrm{H_{best}}$ and feasible values for $X[1:k-1]$
Output: Explore the branch for $X[k] = 1$
if $(k > N$ or $\phi[k] == 0$ or $c[k] > B)$ then return;
$X[k] \leftarrow 1$; $B \leftarrow B - c[k]$;
if $(k == N$ or $c_{\mathrm{sum}}[k+1] \le B)$ then
    $X[j] \leftarrow \phi[j],\ j = k+1, k+2, \ldots, N$;
    $\mathrm{H_{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
    if $(\mathrm{H_{new}} > \mathrm{H_{best}})$ then
        $\mathrm{X_{best}} \leftarrow X$; $\mathrm{H_{best}} \leftarrow \mathrm{H_{new}}$;
else
    $X[j] \leftarrow 0,\ j = k+1, k+2, \ldots, N$;
    $\mathrm{H_{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
    if $(\mathrm{H_{new}} > \mathrm{H_{best}})$ then
        $\mathrm{X_{best}} \leftarrow X$; $\mathrm{H_{best}} \leftarrow \mathrm{H_{new}}$;
    Branch1$(B, k+1, X, \mathrm{X_{best}}, \mathrm{H_{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}})$;
    Branch0$(B, k+1, X, \mathrm{X_{best}}, \mathrm{H_{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}})$;
$B \leftarrow B + c(k)$;
```

The Branch for $X[k] = 1$ does not exist if $\phi[k] = 0$ or $\sum_{j=1}^{k} X[j] \times c[j] > \mathbb{B}$. If $\sum_{j=1}^{k} X[j] \times c[j] + \sum_{j=k+1}^{N} c[j] \times \phi[j] \le \mathbb{B}$, the best solution in the branch corresponding to setting $X[k] = 1$ is to set $X[j] = \phi[j]$ for $j = k+1, \ldots, N$. In a nutshell, the algorithm never cuts a branch that contains a better solution than the current best solution. Therefore, the algorithm always computes an optimal solution to **OHP**. □

4.4 A Heuristic Algorithm

Since **OHP** is NP-hard, we design a polynomial-time greedy heuristic algorithm for computing a solution to **OHP**. This is listed in

Algorithm 4: Branch0($B, k, X, X_{\text{best}}, H_{\text{best}}, G, N, \eta, \phi, \delta, c, c_{\text{sum}}$)

```
Input: Current $H_{best}$ and feasible values for $X[1:k-1]$
Output: Explore the branch for $X[k] = 0$
1  if ($k > N$) then return;
2  $X[k] \leftarrow 0$;
3  if ($k == N$ or $c_{sum}[k+1] \leq B$) then
4      $X[j] \leftarrow \phi[j],\ j = k+1, k+2, \dots, N$;
5      $H_{new} \leftarrow \mathcal{H}(T_{sh}(X))$;
6      if ($H_{new} > H_{best}$) then
7          $X_{best} \leftarrow X$; $H_{best} \leftarrow H_{new}$;
8  else
9      $X[j] \leftarrow 0,\ j = k+1, k+2, \dots, N$;
10     $H_{new} \leftarrow \mathcal{H}(T_{sh}(X))$;
11     if ($H_{new} > H_{best}$) then
12         $X_{best} \leftarrow X$; $H_{best} \leftarrow H_{new}$;
13     Branch1($B, k+1, X, X_{best}, H_{best}, G, N, \eta, \phi, \delta, c, c_{sum}$);
14     Branch0($B, k+1, X, X_{best}, H_{best}, G, N, \eta, \phi, \delta, c, c_{sum}$);
```

Algorithm 5. While our greedy heuristic algorithm does not guarantee to find an optimal solution, extensive evaluation results show that the heuristic algorithm produces close-to-optimal solutions.

Algorithm 5: Greedy($G, N, \eta, \phi, \delta, c, \mathbb{B}$)

```
Input: $G, N, \eta, \phi, \delta, c$: as in Algorithm 2; $\mathbb{B}$: hardening budget.
Output: A feasible hardening strategy $X_{grd}$.
1  Create a binary-valued array $X_{grd}[1:N]$;
2  $X_{grd}[k] \leftarrow 0,\ k = 1, 2, \dots, N$;
3  $H_{grd} \leftarrow \mathcal{H}(T_{sh}(X_{grd}))$; $B \leftarrow \mathbb{B}$; $done \leftarrow 0$;
4  while ($done == 0$) do
5      $done \leftarrow 1$; $R_{best} \leftarrow 0$; $k_{best} \leftarrow 0$;
6      for $k = 1, 2, \dots, N$ do
7          if ($X_{grd}[k] < \phi[k]$ and $c[k] \leq B$) then
8              $X_{grd}[k] \leftarrow 1$;
9              $H_{new} \leftarrow \mathcal{H}(T_{sh}(X_{grd}))$; $R_{new} \leftarrow \frac{H_{new} - H_{grd}}{c[k]}$;
10             if ($R_{new} > R_{best}$) then
11                 $done \leftarrow 0$; $R_{best} \leftarrow R_{new}$; $k_{best} \leftarrow k$;
12             $X_{grd}[k] \leftarrow 0$;
13     if ($done == 0$) then
14         $X_{grd}(k_{best}) \leftarrow 1$; $B \leftarrow B - c[k_{best}]$; $H_{grd} \leftarrow \mathcal{H}(T_{sh}(X_{grd}))$;
15 output $X_{grd}$ and $H_{grd}$;
```

Theorem 6. Algorithm 5 always finds a feasible hardening strategy. Its worst-case running time is $O(K^2(m + n\log n))$, where $n = |V|$, $m = |E|$, and K is the number of hardenable network elements. □

Proof. Algorithm 5 only hardens hardenable network elements, and never hardens a subset of network elements whose aggregated cost exceeds the given budget. Therefore, it always produces a feasible hardening strategy. The algorithm performs $O(K)$ iterations, where each iteration requires the computation of $O(K)$ shortest attack traces. Therefore, the worst-case running time of the algorithm is $O(K^2(m + n\log n))$. □

4.5 Related Work on Network Hardening

Network hardening is an important problem in cyber security, and the attack graph is an elegant tool to measure system-level vulnerabilities for performing network hardening [5, 7, 13, 26, 27]. [13] proposes heuristic approaches to perform network hardening by patching a selected subset of vertices in the attack graph with the lowest cost. However, only primitive fact vertices can be patched in [13], thus their model is not general. Similarly, [5] assumes that only primitive fact vertices are patchable at different costs. It models the network hardening problem as finding the optimal patching strategy that minimizes the cost and residual damage to the system. However, [5] applies the attack tree model [3] instead of the attack graph, which has a simpler structure but makes strong assumptions about attackers' abilities. Durkota *et al.* [7] model the defender's hardening strategy as finding the optimal way to add honeypots to a networked system, by which the defender can detect and mitigate attacks at certain costs. They proposed several heuristic algorithms, albeit with limited scalability.

Another set of works takes the probabilistic metric [25] into account when studying the network hardening problem. The network hardening framework proposed by [26] assigns probabilities to the edges in attack graphs to incorporate the attack graph and the Hidden Markov Model (HMM). Both attack cost and defense cost are modeled in [26] for conducting cost-benefit security analysis. The cost-aware IoT network hardening solution in [27] assigns a cost for removing exploits and initial conditions in the system, and applies a greedy algorithm for finding a cost-effective solution to harden the system. Since all paths to the attack goal need to be calculated to decide which exploits or initial conditions to be removed, it faces scalability issues when the attack graph is large.

Our work has similar motivations of assigning costs and weights to the elements in the attack graphs as that of existing research. However, our problem is more general where all vertices and edges are hardenable. Instead of measuring the attacker's capability as the probability of reaching a specific state, e.g., conquering the attack goal, we assume that the attacker always follows the shortest attack trace to launch an attack. This assumption is intuitive and reasonable, and profiles the attacker's attacking strategy more accurately compared to the probabilistic model.

5 PERFORMANCE EVALUATION

To evaluate our proposed network hardening algorithms, we implemented both algorithms and tested them in two types of scenarios. We introduce the network topology in Section 5.1 and the derivation of the parameters in the attack graph of the evaluation in Section 5.2. We present evaluation results in Section 5.3.

5.1 Network Topologies

Following the attack graph generation strategy in [8], we generated 11 (unweighted) attack graphs from the devices and app configurations in IoT systems using `MulVAL` [17]. The basic flow of the attack graph generation is as follows. First, we profile an IoT system by obtaining the details of IoT devices installed in the system, including brand, model, network protocol, and firmware version. We also verify if there are companion apps to these devices. We then query the CVE database [4] and gather any vulnerabilities reported about the installed devices in the IoT system. To build dependency relationships in the attack graph, we use natural language processing (NLP) techniques to process the descriptions and automation rules in the devices' companion apps. We then combine the vulnerability information and the dependency information and use `MulVAL` to generate the attack graph for the system.

We studied 9 synthetic IoT systems based on real IoT devices and apps (systems with numbered labels) and 2 real-world IoT systems from our smart home testbeds (`System A` and `System B`). The IoT devices deployed in Systems A and B are listed in Table 1. In total, 11 system configurations were studied and converted to attack graphs. With an attack graph generated for a system, we then apply the approaches discussed in Section 5.2 to obtain weighted attack graphs with both `vul-only` parameters and `random` parameters for evaluation.

Table 1: IoT devices deployed in smart home testbeds

Device Name	Device Type	Protocols	Home
Amazon Smart Plug	Plug	WiFi	A
Amcrest ProHD	Camera	WiFi	A
Arlo Q Camera	Camera	WiFi	A
Arlo Ultra	Camera	WiFi	A
August Doorbell Cam Pro	Doorbell	WiFi	A
August Lock	Lock	WiFi	A
Blink XT2	Camera	WiFi	A
Chamberlain Garage Control	Sensor	WiFi	B
D-Link Water Sensor	Sensor	WiFi	A & B
Gosund WiFi Smart Socket	Plug	WiFi	A
Kangaroo Motion Sensor	Sensor	WiFi	A & B
Philips Hue	Bulb	WiFi & Zigbee	A
Reolink Camera	Camera	WiFi	A
Ring Doorbell	Doorbell	WiFi	A & B
Ring Spotlight	Spotlight	WiFi	A
Schlage WiFi Deadbolt	Lock	WiFi	A
Sengled SmartLED	Bulb	WiFi	A
Smart Life Contact Sensor	Sensor	WiFi	A & B
SmartThings Hub	Hub	WiFi & Zigbee	A
Tessan Contact Sensor	Sensor	WiFi	A & B
TP-Link Bulb	Bulb	WiFi	A & B
TP-Link Plug	Plug	WiFi	A & B
WeMo Plug	Plug	WiFi	A

5.2 Evaluation Scenarios

We evaluated our algorithms on two types of scenarios:

(α) Only vulnerability vertices are hardenable. A vulnerability vertex is signified by the `vulExists` keyword in the vertex's description. Their weights w, costs c, and added strength δ are systematically derived from the relevant CVE information, detailed below. Under this scenario, we study a variation of the OHP problem. Instead of trying to maximize the height of the shortest attack trace (SAT), we aim to make the system secure by patching out critical vulnerabilities while minimizing the total costs associated with patching the system. We call this type of parameter assignment the `vul-only` type.

(β) Hardenable network elements are chosen randomly. Initial weights, costs, and added strength for all network elements are randomly generated. For this scenario, we follow OHP problem defined in Equations (1)-(4) to obtain a hardening strategy X under budget $\mathbb{B}$, where the SAT for the hardened attack graph $\mathcal{H}(T_{sh}(X))$ is maximized. We designate this type of parameter assignment as the `random` type.

We now detail our approach to parameter assignment in both types of scenarios.

Hardenable elements ϕ: For scenario (α), we define $\phi[i] = 0$ for $i = |V| + 1, |V| + 2, \ldots, |V| + |E|$, and $\phi[i] = 1$ if and only if $\eta(i)$ is a vulnerability vertex in the attack graph G, $i = 1, 2, \ldots, |V|$. For scenario (β), we choose hardenable elements randomly. $K = \sum_{i=1}^{N} \phi[i]$ is the number of hardenable elements. For each attack graph, we evaluate the algorithms with $K = 16, K = 24$, and $K = 32$.

Weights w: For scenario (α), weights for vulnerability vertices are calculated based on the CVSS exploitability score of the CVE number associated with each vertex. Let $CVSS_{expl}(v)$ be this exploitability score, $w(v) = (\max(CVSS_{expl}) - CVSS_{expl}(v)) \times 2.5$, where $\max(CVSS_{expl})$ is 4, the maximum exploitability subscore in the CVSS 3.x scoring system. Note that the higher the exploitability score is, the easier it is to breach a vulnerable device. Thus, to represent the relative difficulty to breach the device, it is necessary to deduct $CVSS_{expl}(v)$ from the maximum exploitability score. We multiply the final result by 2.5 to scale it to the range [0, 10] to be consistent with the the weights in scenario (β). For network elements that are not hardenable, we assign weights of zero to them. For scenario (β), weights for vertices and edges are real numbers randomly chosen within the range $[0, 10]$.

Added strength δ: In scenario (α), hardening a network element means that the vulnerability is eliminated from the network and the attacker can no longer utilize the vulnerability. Therefore, we define the added strength for a hardened vulnerability vertex v to be a very large number $\mathbb{H}$, i.e. $\delta(v) = \mathbb{H}$. We seek a minimum-cost hardening strategy so that the height of a shortest attack trace in the hardened attack graph is greater than or equal to $\mathbb{H}$. For scenario (β), $\delta(z)$ for each hardenable element z is randomized to be a real number in $[0.05, 2.00]$ times $w(z)$.

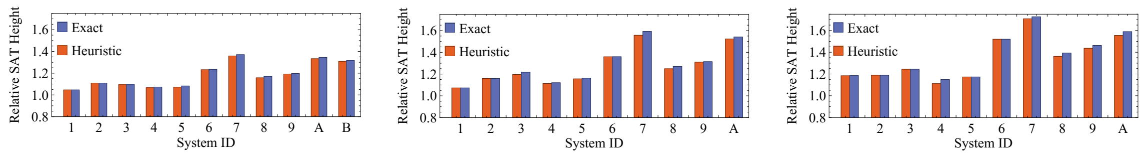
Hardening cost c: In scenario (α), because every vulnerability is resulted from a weakness, we correlate the origin of the vulnerability, i.e., the weakness, and examine how many identified vulnerabilities are affected by the same weakness. The hardening cost $c(v)$ for a vulnerability vertex v is then defined as the ratio between the vertex's CWE [14] score relative to the number of all CWEs as follows: $c(v) = \frac{R_{CWE}(v)}{T_{CWE}} \times 10$, where $R_{CWE}(v)$ is the rank of the CWE for vulnerability v and T_{CWE} is the total number of CWEs. The resulting cost will be in the range [0-10]. This cost represents the amount of effort that is needed to remediate a vulnerability.

For scenario (β), $c(z)$ for a hardenable element z is chosen to be a real number in $[0.30, 1.50]$ times $\delta(z)$.

Note that for scenario (β), the generated parameters are not meant to represent realistic network configurations or actual effects of hardening, but rather to test whether our algorithms are effective for any general case of network hardening problems. Also for a given attack graph, we fix the values of w, c, and δ once they are generated, and we vary the hardenable network elements for each test case in scenario (β).

5.3 Evaluation Results

We present the evaluation results for attack graphs generated in both scenarios (and their associated hardening problems). All evaluation was done on a workstation with i9-12900 16-core CPU @ 5.1GHz, 64GB system memory, and Ubuntu 22.04 system.



(a) $K = 16$ (b) $K = 24$ (c) $K = 32$

Figure 2: Relative SAT heights for the exact and the heuristic algorithms across different systems.

For the `vul-only` type of graphs, we follow the parameter assignment described in Section 5.2. There is only one test case per attack graph, since the vulnerability vertices in an attack graph and their CVSS assessments are fixed. For the `random` type of graphs, define ρ the percentage of the budget $\mathbb{B}$ over the cost of all hardenable elements, i.e., $\rho = \frac{\mathbb{B}}{\sum_{i=1}^{N} \phi[i] \times c[i]}$. We further create two kinds of test suites.

(i) The suite with a fixed budget percentage ρ, but different attack graphs: We fix ρ to 0.3 so that the budget $\mathbb{B}$ will be 0.3 times the total cost of all hardenable elements in an attack graph, and we create test cases with all 11 attack graphs. For each attack graph, we further create cases where $K = 16, 24$, and 32, respectively. For each K we generate 10 different test cases. Recall that the network parameters are set once they are generated, so each test case with the same attack graph differs only in their selection of hardenable network elements and budget $\mathbb{B}$.

(ii) The suite with a fixed attack graph, but different budget percentages ρ: We fix `System 9` as the target attack graph, but vary ρ to be one of the following values: [0, 15, 30, 50, 70, 85, 100]. We generate cases with zero budget assigned all the way up to the cases where the budget is enough to harden all network elements. Like in (i), we create test cases where $K = 16, 24$, and 32, respectively. For each K we also generate 10 different test cases.

For each test suite, we run both the exact algorithm (Algorithm 2) and the heuristic algorithm (Algorithm 5). Note that the exact algorithm will give the optimal solution X_{opt} where the height of the attack trace $\mathcal{H}(T_{sh}(X_{opt}))$ is maximized given the budget $\mathbb{B}$.

5.3.1 Results for Scenario (α). We apply the algorithms on `vul-only` graphs to find a hardening strategy X_{opt} that secures the system while having the lowest total cost. We say that a system is secure when the height of the SAT in the hardened attack graph is greater than or equal to $\mathbb{H}$. To ensure the hardening strategy X_{opt} has the minimum cost, we perform a bisection search on the budget $\mathbb{B}$ and find the minimal $\mathbb{B}$ that affords a strategy to secure the system.

Table 2 lists several key results from the experiments. We note that for all 11 systems, the procedure defined above was able to find a hardening strategy to secure the system within 0.01 seconds.

The results also reveal that in an attack graph, there can be a large number of hardenable elements (vulnerabilities), but not all of them affect the overall security of the system equally. In some cases, one can patch only a small subset of all vulnerable vertices and still be able to secure the system. We list `Orig. Cost` the combined cost to harden all vulnerable vertices, and `Opt. Cost` the optimized cost to secure the system after performing the cost optimization procedure. In all attack graphs in our test where $K > 1$, it is possible to cut down the costs to secure the system by a significant amount, resulting in as much as 96.7% savings in hardening cost as in the case of `System 1`.

Table 2: Evaluation results on applying network hardening techniques to patch vulnerabilities in the system

System ID	K	Running Time	Strategy Found?	Original Cost	Optimal Cost	Cost Saving
System 1	25	$0.0036s$	Yes	182.6	6.1	96.7%
System 2	17	$0.0009s$	Yes	127.7	8.2	93.6%
System 3	14	$0.0008s$	Yes	105.7	8.2	92.2%
System 4	9	$0.0006s$	Yes	74.5	19.8	73.4%
System 5	9	$0.0004s$	Yes	68.3	9.6	85.9%
System 6	4	$< 0.0001s$	Yes	31.2	9.5	69.6%
System 7	4	$< 0.0001s$	Yes	38.0	9.5	75.0%
System 8	4	$< 0.0001s$	Yes	28.3	5.5	80.6%
System 9	1	$< 0.0001s$	Yes	9.8	9.8	N/A
Testbed A	6	$< 0.0001s$	Yes	50.6	10.0	80.2%
Testbed B	2	$< 0.0001s$	Yes	15.1	6.6	56.3%

5.3.2 Results for Scenario (β)-(i). In Fig. 2, we present the relative SAT heights for hardening strategies generated by both the exact and the heuristic algorithms. This illustrates the improvements of the heights of the SAT after running both algorithms and performing the hardening strategy returned by the algorithms. Note that for `Testbed B`, we have results for $K = 16$ and 21 only, as the system has only 21 network elements.

The exact algorithm is guaranteed to return an optimal hardening strategy X_{opt} within the budget $\mathbb{B}$. Note that the optimal strategy is not necessarily unique, and the algorithm does not guarantee the strategy is of minimum cost. Due to the "bound" procedure in the algorithm, anytime when the residual budget is enough to harden all the remaining elements, the algorithm will do so if the resulting height of the SAT is better. This cuts down the running time but can result in more spending than necessary. This is the reason that the exact algorithm sometimes produces a more expensive strategy than the heuristic strategy, even if they achieve the same SAT height. The data shows that the algorithm exhibits an exponential running time with regard to K. However, due to the branch and bound strategy in the algorithm, the running time in general is much faster than a brute-force approach.

On the other hand, the heuristic algorithm achieves a comparable performance to the exact algorithm in many attack graphs. In some cases, the heuristic algorithm finds the same optimal solution that the exact algorithm finds. Averaging all test cases, the solution that the heuristic algorithm produces is approximately 96.81% as good as our exact algorithm produces. In other words, the heuristic strategy yields on average 96.81% of the increase in the height of SAT that an optimal strategy can do. In addition, the heuristic algorithm runs quickly even in the largest cases, finishing within 0.0002 seconds for all test cases.

Table 3: Evaluation results of our exact and heuristic network hardening algorithm

System ID	#Vertices	#Edges	Height	K	Budget	Heuristic Algorithm			Exact Algorithm			
						Cost	Height	Time	Cost	Height	Ratio	Time
System 1	338	577	124.83	16	23.20	5.78	130.60	0.0004s	17.12	130.60	21.70%	0.1121s
				24	36.34	9.88	133.90	0.0011s	32.08	133.90	12.03%	15.02s
				32	50.50	24.02	147.77	0.0020s	46.58	147.97	8.01%	2802s
System 2	209	310	77.94	16	21.14	11.58	86.46	0.0003s	17.50	86.46	21.21%	0.0527s
				24	34.01	19.90	90.38	0.0006s	27.49	90.38	8.86%	5.495s
				32	42.84	18.59	89.56	0.0010s	38.66	89.56	6.62%	1041s
System 3	162	234	79.06	16	22.04	7.55	86.52	0.0002s	18.56	86.52	17.44%	0.0328s
				24	28.40	15.73	94.57	0.0004s	25.63	96.25	12.54%	6.052s
				32	36.21	23.05	98.34	0.0007s	30.67	98.39	7.31%	873.7s
System 4	130	182	228.02	16	26.05	19.72	243.12	0.0002s	17.42	244.63	20.21%	0.0281s
				24	40.32	34.70	253.87	0.0004s	29.51	255.46	9.83%	3.571s
				32	51.44	43.28	257.55	0.0006s	46.68	260.47	5.21%	454.5s
System 5	117	173	110.29	16	27.91	12.55	118.20	0.0001s	18.86	119.40	25.68%	0.0338s
				24	47.07	22.78	127.49	0.0003s	38.07	128.20	15.95%	5.493s
				32	59.77	29.22	129.37	0.0005s	54.59	129.37	10.92%	900.9s
System 6	37	39	94.80	16	21.95	17.82	116.77	$<0.0001s$	17.40	117.07	8.72%	0.0028s
				24	33.31	28.47	128.84	0.0001s	29.03	128.90	3.60%	0.2950s
				32	44.68	41.18	114.07	0.0001s	41.59	144.07	1.81%	39.02s
System 7	36	35	72.11	16	24.19	21.59	97.96	$<0.0001s$	18.10	98.82	16.06%	0.0052s
				24	33.56	30.73	112.28	0.0001s	28.95	114.82	9.03%	0.7210s
				32	42.44	39.03	123.17	0.0002s	35.64	124.50	6.38%	34.40s
System 8	35	44	106.01	16	21.67	16.73	122.71	$<0.0001s$	17.88	124.12	27.07%	0.0076s
				24	31.31	28.33	128.84	0.0001s	26.18	134.68	12.22%	0.8846s
				32	44.78	40.79	145.04	0.0001s	41.71	147.30	7.32%	135.1s
System 9	25	26	165.84	16	24.25	22.82	197.94	$<0.0001s$	17.72	198.51	28.90%	0.0041s
				24	38.21	37.12	217.24	$<0.0001s$	31.28	217.95	22.78%	0.8392s
				32	52.42	51.69	238.24	0.0001s	50.96	242.62	16.78%	154.1s
Testbed A	37	42	61.91	16	15.22	13.38	82.55	$<0.0001s$	12.59	83.28	25.93%	0.0090s
				24	24.16	22.34	94.30	0.0001s	19.89	95.43	11.18%	0.9399s
				32	27.01	25.51	96.29	0.0002s	23.94	98.40	8.72%	12.00s
Testbed B	10	11	55.36	16	15.19	12.68	72.45	$<0.0001s$	12.71	72.87	34.38%	0.0026s
				21	18.87	17.16	80.75	$<0.0001s$	12.86	80.75	26.97%	0.0671s

Table 3 lists the results in this test suite. For each system, we show the number of vertices and edges in the attack graph. The `Height` column indicates the height of the SAT in the graph before any hardening. For each algorithm, we show the cost of the hardening strategies along with the heights of the SAT in the hardened graph. We also list the time taken to execute one test case. For the exact algorithm, we added a ratio of the number of SAT queries made by the algorithm over that of the brute force solution (which is 2^K). Each (minor) row is averaged results over 10 test cases.

5.3.3 *Results for Scenario (β)-(ii).*

We study also the behavior of the algorithms using the same attack graph (hence identical parameters) but different budget constraints. Fig. 3 shows the relative heights of the SAT of both algorithms compared to that of the original graph when $K = 16, 24$, and 32. While reaffirming the results found in Section 5.3.2, the results also display that the difference between relative SAT heights is higher for medium budgets and tapers off towards the extremes.

Intuitively, when the budget is low, neither of the two algorithms can do much to harden the network. As the budget increases, the disparity between the two algorithms begins to manifest. However, once the budget is sufficiently large, the two algorithms can elect to harden most elements in the network, resulting again a similar performance.

The figures also suggest a diminishing return as more budget is added. Indeed, as the budget percentage ρ increases, the gains in SAT heights slows down, to a point where virtually no improvement is made between $\rho = 0.85$ and $\rho = 1.00$.

6 CONCLUSIONS

In this paper, we study the problem of analyzing IoT system vulnerabilities and network hardening. We first design a novel algorithm for computing a shortest attack trace in a weighted attack graph. We demonstrate that our algorithm is more robust and faster than the state-of-the-art [8]. In particular, our algorithm can handle cycles in the attack graph properly, and works correctly regardless of whether the attack graph contains an attack trace or not. We then formulate the network hardening problem. We prove that the network hardening problem is NP-hard, and design an exact algorithm and a polynomial-time heuristic algorithm to solve it. Extensive evaluations show that our exact algorithm can compute optimal solutions for reasonably sized problems, and our heuristic algorithm can efficiently produce near optimal results.

ACKNOWLEDGMENTS

This research was supported in part by NSF grants 2007469 and 2007083. The information reported herein does not reflect the position or the policy of the funding agency.

REFERENCES

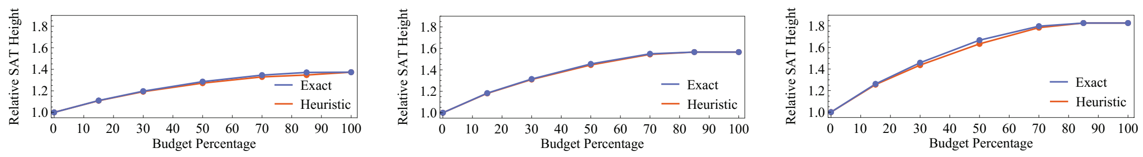
[1] Noga Agmon, Asaf Shabtai, and Rami Puzis. 2019. Deployment optimization of IoT devices through attack graph analysis. In *Proc. of ACM WiSec.*

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security evaluation of home-based IoT deployments. In *Proc. of IEEE S&P.*

[3] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. 2002. Scalable, graph-based network vulnerability analysis. In *Proc. of ACM CCS.*

[4] CVE. Common vulnerabilities and exposures. https://www.cve.org

[5] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. 2007. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proc. of ACM CCS.*



(a) $K = 16$ (b) $K = 24$ (c) $K = 32$

Figure 3: Relative SAT heights for the exact and the heuristic algorithms with different budget percentages.

[6] Wenbo Ding, Hongxin Hu, and Long Cheng. 2021. IOTSAFE: Enforcing safety and security policy with real IoT physical interaction discovery. In *Proc. of NDSS*.

[7] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, Christopher Kiekintveld, and Michal Pěchouček. 2019. Hardening networks against strategic attackers using attack graph games. *Computers & Security* 87 (2019), 101578.

[8] Zheng Fang, Hao Fu, Tianbo Gu, Pengfei Hu, Jinyue Song, Trent Jaeger, and Prasant Mohapatra. 2022. IOTA: A framework for analyzing system-level security of IoTs. In *Proc. of ACM/IEEE IoTDI*.

[9] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. FlowFence: Practical data protection for emerging IoT application frameworks. In *Proc of USENIX Security*.

[10] Michael L Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)* 34, 3 (1987), 596–615.

[11] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. HAWatcher: Semantics-aware anomaly detection for appified smart homes. In *Proc. of USENIX Security*.

[12] Michael R Garey and David S Johnson. 1979. *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman and Company.

[13] Tania Islam and Lingyu Wang. 2008. A Heuristic approach to minimum-cost network hardening using attack graph. In *2008 New Technologies, Mobility and Security*.

[14] MITRE. CWE - Common weakness enumeration. https://cwe.mitre.org/

[15] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. 2017. Insecure to the touch: Attacking ZigBee 3.0 via touchlink commissioning. In *Proc. of ACM WiSec*.

[16] Xinming Ou, Wayne F Boyer, and Miles A McQueen. 2006. A scalable approach to attack graph generation. In *Proc. of ACM CCS*.

[17] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. 2005. MulVAL: A logic-based network security analyzer. In *Proc. of USENIX Security Symposium*.

[18] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT goes nuclear: Creating a ZigBee chain reaction. In *Proc. of IEEE S&P*.

[19] Abdulhakim Sabur, Ankur Chowdhary, Dijiang Huang, and Adel Alshamrani. 2022. Toward scalable graph-based security analysis for cloud networks. *Computer Networks* 206 (2022), 108795.

[20] Konstantinos Sagonas, Terrance Swift, and David S. Warren. 1994. XSB as a deductive database. In *Proc. of ACM SIGMOD*.

[21] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. 2002. Automated generation and analysis of attack graphs. In *Proc. of IEEE S&P*.

[22] Ioannis Stellios, Panayiotis Kotzanikolaou, and Christos Grigoriadis. 2021. Assessing IoT enabled cyber-physical attack paths against critical systems. *Computers & Security* 107 (2021), 102316.

[23] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2019. PingPong: Packet-level signatures for smart home device events. In *Proc. of NDSS*.

[24] Yinxin Wan, Kuai Xu, Feng Wang, and Guoliang Xue. 2021. Characterizing and mining traffic patterns of IoT devices in edge networks. *IEEE Transactions on Network Science and Engineering* 8, 1 (2021), 89–101.

[25] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. 2008. An attack graph-based probabilistic security metric. In *Proc. of IFIP Annual Conference on Data and Applications Security and Privacy*.

[26] Shuzhen Wang, Zonghua Zhang, and Youki Kadobayashi. 2013. Exploring attack graph for cost-benefit security hardening: A probabilistic approach. *Computers & Security* 32 (2013), 158–169.

[27] Beytüllah Yiğit, Gürkan Gür, Fatih Alagöz, and Bernhard Tellenbach. 2019. Cost-aware securing of IoT systems using attack graphs. *Ad Hoc Networks* 86 (2019), 23–35.