Cost-Aware Securing of IoT Systems Using Attack Graphs

Beytüllah Yiğit[a,*], Gürkan Gür[b], Fatih Alagöz[a], Bernhard Tellenbach[b]

[a] *Department of Computer Engineering, Bogazici University*
34342 Istanbul, Turkey
Email: {beytullah.yigit, fatih.alagoz}@boun.edu.tr
[b] *Zurich University of Applied Sciences ZHAW*
8401 Winterthur, Switzerland
Email: {gueu,tebe}@zhaw.ch

Abstract

The Internet of Things (IoT) contains a diverse set of sensors, actuators and other Internet-connected devices communicating, processing data and performing a multitude of functions. It is emerging as an integral part of societal infrastructure enabling smart services. However, these connected objects might have various vulnerabilities that can lead to serious security compromises and breaches. Securing and hardening of IoT systems is thus of vital importance. In that regard, attack graphs provide analytical support to prevent multistep network attacks by showing all possible sequences of vulnerabilities and their interactions. Since attack graphs generally consist of a very large number of nodes, it is computationally challenging to analyze them for network hardening. In this paper, we propose a greedy algorithm using compact attack graphs to find a cost-effective solution to protect IoT systems. First, we extract all possible attack paths which reach predetermined critical resources embedded in the network. Then, exploit or initial condition with minimum effective cost is selected to be removed. This cost is calculated as a function of contribution to attack paths (the higher, the better) and removal cost (the lower, the better). This process continues iteratively until the total cost exceeds the allocated budget. The experimental results show that our algorithm scales almost linearly with the network size and it can be applied to large-scale graphs with a very large number of IoT nodes. In addition to network-hardening, our proposal measures the security level of the network in every step to demonstrate the vulnerability grade of the system.

Keywords: Internet of Things, attack graphs, network hardening, security decision support, security metrics, CVSS.

1. Introduction

Information and Communications Technology (ICT) has become an inherent and dispensable part of our daily lives with "anytime-anywhere" Internet communications, smart devices, ubiquitous connectivity and networked infrastructure. This revolution is crystallized as the Internet of Things (IoT) where Internet-connected devices with different levels of computation, communication, sensing, and intelligence capabilities form a cyber-substrate performing various tasks and offering advanced functions. IoT is envisaged to be a key pillar of smart communications and services infrastructure in Future Internet. However, to provide robust, trusted and dependable services in that environment, security is a vital requirement and thereby a perpetual task of ICT management. Therefore, vulnerabilities of IoT systems must be identified to avoid disruption or failure of network-based services.

The criticality of these connected infrastructures has led to the huge body of research focusing on cybersecurity. Accordingly, analysis of security vulnerabilities and how to avoid them are important research topics. In practice, there are various vulnerability scanners such as Nessus [1] and Snort [2] which detect vulnerabilities of each element of an investigated network in isolation. Since such tools do not give any information about interaction of vulnerabilities, attackers can combine different vulnerabilities to breach a seemingly well-guarded network and then can disrupt it. Therefore, the information of not only isolated vulnerabilities, but also their interactions are essential to effectively protect a network from multistep attacks which are combination of correlated vulnerabilities exploited sequentially to reach intended target(s). As a typical example, an attacker compromises a Internet-reachable IoT device as a stepping stone to reach a target in the core of the network with a multistep attack.

Attack graphs provide crucial information to prevent multistep attacks since they show all possible sequences of vulnerabilities and their relationships [3]. In other words, they reveal potential threats against networks by showing all possible attack paths. An attack graph can be a useful tool to extract recommendation for hardening network nodes against inherent vulnerabilities [4]. It can also be instrumental to reckon how secure a network is [5]. However, as the number of devices in an IoT system and their

*Corresponding author

corresponding vulnerabilities increase, an attack graph can become very complex for a security analyst to find which vulnerabilities should be removed. Hence, automated extraction of recommendations from attack graphs and their user accessibility have a practical importance for effective countermeasure enforcement.

While eliminating vulnerabilities to prevent multistep attacks, one should also consider the cost of removal since patches for different remedies usually incur different costs [6]. In that regard, another critical question is how we can minimize the cost of protection while removing all attack paths or reducing their success probabilities since the cost of protection is generally constrained by a limited budget [7]. Moreover, removing all identified vulnerabilities can be infeasible due to the characteristics of IoT systems such as lack of known patches (zero-day attacks), resource limitations, and operational constraints [8]. Although all the identified vulnerabilities should be fixed in the long run, there should be an efficient scheme to decide on an "action plan" based on impact and feasibility in practical systems. On the theoretical side, finding optimal minimum cost solution by using attack graphs is an intractable problem [9], which calls for computationally efficient approaches.

In this paper, we present a cost-aware network hardening solution with limited budget by using compact attack graphs for IoT systems. Firstly, all attack paths to critical resources are extracted from the attack graph and their success probabilities are calculated. The security analyst can set length and likelihood thresholds to exclude certain attack paths whose lengths exceed the length threshold or the success probability is lower than the likelihood threshold. Our algorithm uses *Common Vulnerability Scoring System (CVSS)* [10] exploitability scores while calculating success probabilities of attack paths. Then, security of the network is measured with a novel method by considering those success probabilities. This accompanying feature is beneficial for network security assessment and situation awareness. Subsequently, the most critical vulnerability is selected according to the cost and its contribution to the attack paths. This selection process continues iteratively until the network is completely secure or the allocated budget is consumed.

In this work, we elaborate on network hardening, compact attack graphs and analysis of network hardening schemes in IoT via contributions in the following key threads:

- We propose a heuristic-based and cost-controlled network hardening solution in IoT with cost minimization by using compact attack graphs (Sect. 3) and illustrate its operation using a case study (Sect. 4).
- We devise attack path length and attack path success probability thresholds to utilize budget effectively by eliminating unlikely and long attack paths (Sect. 3).
- We propose a novel method to measure security level metric of an analyzed network with the help of *CVSS* scores (Sect. 3).
- We provide a comprehensive complexity and protection cost analysis of our approach. This aspect of our work also provides a framework for comparative analysis of such schemes for network security managers (Sect. 5).

In the next section, we introduce essential background for attack graphs. Our novel network hardening method, *Cost- and Budget-Aware Network Hardening for IoT (COBANOT)*, is described in Section 3 followed by a case study to illustrate its operation in Section 4. In Section 5, experimental results are presented to examine the time complexity and minimum cost efficiency of *COBANOT*. For illustrating the current state of the literature and how our work differs, Section 6 reviews related work on attack graphs and IoT security. In Section 7, we focus on limitations of our approach for a balanced view. Finally, Section 8 discusses future research directions and concludes the paper.

2. Preliminaries on Compact Attack Graphs

Multistep and multi-host attacks must be considered to analyze the security of a network effectively since an attacker can penetrate the network by hopping from one device to another. However, the representation and analysis of attack graphs for ICT infrastructures composed of more than just a few components can be difficult. For example, in the work by Sheyners *et al.*, nodes represent the state of the entire infrastructure after each step of an attack [11]. While the state is polynomial in the size of the infrastructure, the possible number of states is exponential. Even when we assume that not all states can be reached when considering the attack graph, the potential state explosion makes this approach impractical except for small-scale infrastructures. A way to circumvent this problem is to use compact attack graphs instead of state-based ones. They have polynomial complexity in terms of total number of vulnerabilities and security conditions [12]. Hence compact attack graphs can be generated for large IoT systems containing a very large number of nodes. Considering the advantages they offer in terms of complexity, we utilize compact attack graphs for security modeling and analysis in our work. Moreover, there are various automated tools proposed like MulVAL and NetSPA to produce attack graphs in polynomial time [13, 14].

Compact attack graphs are directed graphs which consist of two node types: *exploits* and *security conditions* [15]. An exploit is a vulnerability from which an attacker can benefit to obtain higher privileges. Security conditions show required preconditions or post conditions for exploits. There are no direct edges among exploits and among security conditions. However, exploits and security conditions are connected with two types of directed edges. The first type points to an exploit from a security condition. It represents a *require* relation such that execution of the exploit requires that the security condition must be satisfied. The

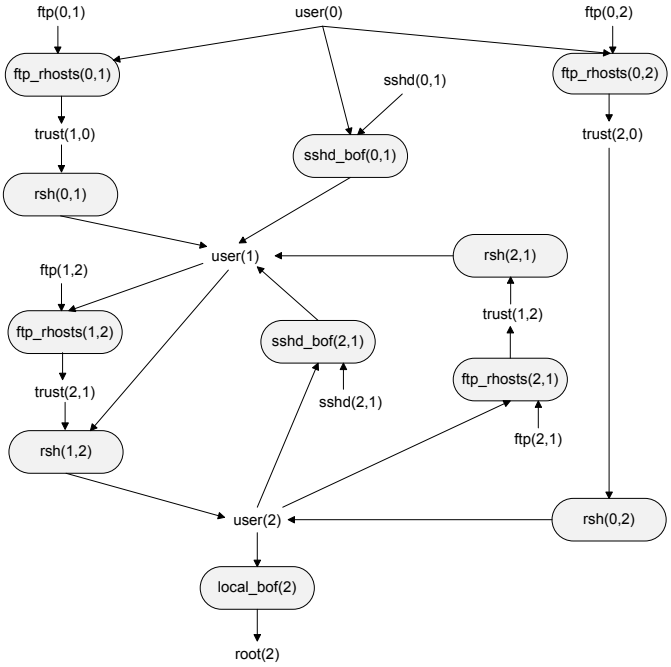

Figure 1: An attack graph (AG1) with four vulnerability types, eleven exploits and seven initial conditions.

second edge type points to a security condition from an exploit. This type represents an *imply* relation such that execution of the exploit will satisfy the security condition. Attack graphs are formally described in Definition 1.1. We shall also assume this notion of attack graphs in our work.

Definition 1.1. Given a set of exploits E, a set of security conditions S, a require relation $R_r \subseteq S \times E$, and an imply relation $R_i \subseteq E \times S$, an attack graph AG is the directed graph $AG(E \cup S, R_r \cup R_i)$ ($E \cup S$ represents the vertex set and $R_r \cup R_i$ represents the edge set) [9].

A sample attack graph is shown in Fig. 1. There are three users in this network: an attacker, *user 0*, and two victim users (devices), *user 1* and *user 2*. For the identifiers, numbers in parenthesis specify related users. For example, *root(2)* indicates that attackers have root privilege on *user 2*. Similarly, execution of *ftp.rhosts* vulnerability from *user 0* to *user 1* is shown as *ftp_rhosts(0,1)*. In that figure, plain text items represent security conditions while rounded boxes represent exploits.

Security conditions are of two different types: *initial conditions* and *intermediate conditions*. An initial condition is not a postcondition of any exploits but a necessary precondition for them. Accessibility rules of hosts or network configuration can be examples of initial conditions like *user(0)* and *ftp(0,1)* in Fig. 1. Intermediate conditions are both preconditions and postconditions of some exploits. Condition *trust(1,0)* in Fig. 1 represents a trust relationship between the attacker (*user(0)*) and *user 1*, and is an example of intermediate condition.

An attack path or attack scenario is a multistep attack which consists of a combination of correlated exploits performed consecutively to breach critical resources of a network. As an example, attacker *(user(0))* can obtain root privileges on *user 2* by executing attack path *"ftp_rhosts(0, 2)* $\rightarrow$ *rsh(0,2)* $\rightarrow$ *local_bof(2)"* in the example network. To do this, firstly a trust relationship is established between *user 0* (attacker) and *user 2* (the condition *trust(2,0)*) by exploiting *ftp.rhosts* vulnerability on *user 2* (the exploit *ftp_rhosts(0,2)*). Then attackers obtain user privilege on *user 2* (the condition *user(2)*) using rsh login attack (the exploit *rsh(0,2)*). After that, attackers own root privilege on *user 2* (the condition *root(2)*) via local buffer overflow attack on that device (the exploit *local_bof(2)*).

There are two ways to eliminate an attack path. First one is to disable one of the security conditions in the attack path. However, intermediate conditions are typically not controlled directly by the security analyst [9]. Therefore, initial conditions are left as the only option to disable security conditions. Patching one of the exploits in the attack path is the second option. However, we have to keep in mind that both initial conditions and vulnerabilities cannot be fixed in all cases since some IoT devices cannot be patched properly due to constrained hardware and practical issues such as lack of omnipresent device management capabilities [16]. In addition to that, there are circumstances such as zero-day attacks where patches are not available for some vulnerabilities. Furthermore, disabling an initial condition may seriously affect service availability of the system. As an example, disabling *ftp(0,2)* by stopping ftp service can be unacceptable for a server which processes and relays sensory IoT data to other consumer nodes. Hence, our approach *COBANOT* allows marking of designated exploits and initial conditions such that they are ignored when selection of exploits or initial conditions for network hardening is performed. In other words, security analyst can mark exploits and initial conditions which cannot be removed from the system due to some constraint or rationale.

3. Cost- and Budget-Aware Network Hardening for IoT (*COBANOT*)

By traversing on attack graph, we can find all attack paths which end in given critical devices of a system. If all these attack paths are removed, IoT devices in the network become secure against potential attacks. There are plenty of options while eliminating all attack paths since different exploits and initial conditions can be selected to achieve this goal. However, we need to consider the cost factor of elimination to harden network with minimum burden. Some exploits or initial conditions may also be unfitting for removal as discussed in Section 2. On the other hand, cost consideration induces budget constraint since the total cost of making network completely secure can be unacceptable. Especially, large IoT systems consisting of thousands of devices with poor vendor security practices pose a serious obstacle against completely securing such a system with limited budget. Nevertheless, likelihood of attacks against critical resources of the system can

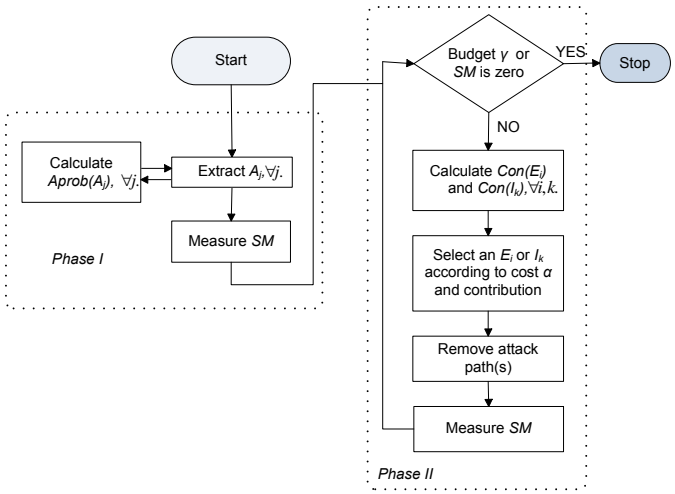

Figure 2: The flowchart of *COBANOT*.

be reduced even if it cannot be totally eliminated in such a case. To that end, the security metric of the system is calculated to determine how secure the system is and then a security analyst can decide how much protection needs to be performed.

In this paper, we propose *Cost- and Budget-Aware Network Hardening for IoT (COBANOT)* which achieves minimal cost by considering budget and cost factors in an iterative way. The flowchart of *COBANOT* is shown in Fig. 2. Accordingly, Table 1 shows the parameters used in our algorithm and analysis. Our proposal consists of two phases. In the first phase, all attack paths are extracted from the attack graph with a backward algorithm. Meanwhile, the success probabilities of all attack paths are computed. Then, the security metric *SM* is calculated. In the second phase, the contributions of each exploit and initial condition to attack paths are calculated. According to the cost of removal α and the contribution to attack paths, an exploit or an initial condition is selected for removal. Finally, attack paths which include selected exploit or initial condition are removed and the security metric is re-calculated. This second phase continues until network is "completely" secure, which means that there is no attack path for an intruder to breach the critical IoT assets or the available budget is depleted.

To find all attack paths which terminate at given critical resources, a backward algorithm is used in the first phase. Thus, exploits which cannot reach the target are never explored. We also benefit from forward algorithm logic and discard attack paths which do not start from the attacker node. Therefore, the complexity of second phase is reduced since only necessary and relevant attack paths are considered for elimination. Additionally, attack graphs can also have cyclic paths. However, intruders do not usually choose cyclic paths as noted in [15]. In backward algorithm, while extracting one attack path, if an exploit which was already included by that attack path is visited again, extraction of the path is stopped to avoid a

Table 1: Analysis parameters

Abbreviation	**Explanation**
E_i	An exploit
I_k	An initial condition
A_j	An attack path
SM	Security Metric
$Eprob(E_i)$	Likelihood of an exploit
$Aprob(A_j)$	Likelihood of an attack path
$Con(E_i)$	Contribution of an exploit
$Con(I_k)$	Contribution of an initial condition
β	Effective cost
α	Cost of removal for exploits and initial conditions
γ	Remaining budget
θ	Exploit Diversity Constant
φ	For an exploit E_i, φ shows number of exploits whose type is equal to E_i and exploited before E_i in an attack path A_j
M	The maximum number of exploits which point to a condition in the attack graph
n	The path length threshold

cyclic path. Hence we do not need to remove cycles in the attack path, but just detect and avoid them during the process.

Similar to cyclic paths, long attack paths would not be preferred by attackers. Typically, attackers are not expected to use attack paths which take more than 4 steps [15, 18]. In *COBANOT*, while extracting attack paths, a security analyst can assign thresholds for length and success probability to attack paths. During the extraction of attack paths with backward algorithm, if the path length exceeds given length or its likelihood is lower than the likelihood threshold, extraction of that path is halted. Hence, complexity of Phase I is reduced by not traversing for long and unlikely attacks paths. Additionally, the complexity of Phase II is also reduced since the number of attack paths which pass to Phase II is decreased. Moreover, a security analyst can prevent easier and thus more likely intrusions by considering them in the first place.

Each vulnerability has different difficulty level for an attacker to exploit. Therefore, vulnerabilities have different likelihoods of successful occurrence which could be estimated by average time or computational complexity required to breach the network [19]. However, this estimation task is not trivial for a non-expert person. In that regard, an alternative approach based on empirical data can be adopted and success probabilities of exploits $Eprob(E_i)$ can be extracted from *CVSS* exploitability scores [12, 20, 21, 22]. As of now, the most widely used vulnerability

Table 2: *CVSS* attributes used to calculate base metric exploitability scores [17]

CVSS attribute	Value
Attack Vector (AV)	Physical (P) Local (L) Adjacent Network (A) Network (N)
Access Complexity (AC)	High (H) Low (L)
Privileges Required (PR)	High (H) Low (L) None (N)
User Interaction (UI)	Required(R) None (N)

Table 3: Temporal exploitability score values and descriptions [17]

Value	Description
Unproven (U)	No exploit code is available or the exploit is entirely theoretical.
Proof-of-concept (P)	Proof of concept exploit code or an attack demonstration that is not practical for most systems is available.
Functional (F)	Functional exploit code is available. The code works in most situations where the vulnerability exists.
High (H)	The vulnerability can be exploited by automated code and mobile code like a worm or virus.
Not Defined (X)	A signal to ignore this score.

scoring system is *CVSS* [23]. *CVSS* scores for known and reported vulnerabilities can be found in NVD (National Vulnerability Database) [24]. First exploitability score is a subscore of base metric group and it consists of *attack vector (AV)*, *access complexity (AC)*, *privileges required (PR)* and *user interactions (UI)* metric. Second one is a subscore of temporal metrics. Both of these exploitability scores can be used separately to find the success probability of an exploit. However, NVD currently does not provide temporal scores.

Table 2 shows *CVSS* attributes used for base metric exploitability scores. More detailed description can be found in [17]. The base exploitability score is calculated as:

$$\textit{Exploitability Score} = 8.22 \times AV \times AC \times PR \times UI, \quad (1)$$

Base exploitability score range from 0 to 10. Hence we can use base exploitability scores as $Eprob(E_i)$ by normalizing it with division by 10 like [25]. As an example, base exploitability score of *local_bof* is 10.0 since its AV is exploitable from *Network (N)* called as remotely exploitable, AC is *Low (L)*, UI is *None (N)* and PR is *None (N)* which means attackers do not need to authenticate. Therefore, $Eprob(E_i)$ of *local_bof* becomes 1.0 if we use base exploitability score to obtain $Eprob(E_i)$ of *local_bof*.

Temporal metrics change over time since exploits evolve, they are disclosed and automated or patches for them are developed. The temporal exploitability score (*Exploit Code Maturity*) shows the current state of exploit techniques or code availability. *Exploit Code Maturity* can also be a good indicator to determine path length threshold since if there are well-known vulnerabilities with automation available through tools/frameworks, the path length threshold might be much larger. Table 3 shows values and their corresponding scores for temporal metric exploitability scores [17]. We use the mapping {Unproven $\rightarrow$ 0.1, Proof-of-concept $\rightarrow$ 0.5, Functional $\rightarrow$ 0.8, High $\rightarrow$ 1.0} for success likelihoods. Apparently, it is reasonable that as exploit code is more accessible, success likelihood becomes higher.

We use success probabilities of exploits $Eprob(E_i)$ to calculate likelihood of attack paths. As the length of a path increases, likelihood of execution of that attack path generally decreases. However, there may be longer attack paths which consist of vulnerabilities easy to exploit and have higher success probability than shorter attack paths which include vulnerabilities difficult to exploit. In that regard, likelihood of an attack path A_j, $Aprob(A_j)$, can be calculated by multiplying success probabilities of exploits which are present in that attack path.

Exploit diversity in an attack path also affects the likelihood of that attack path. When an attacker exploits a vulnerability while executing an attack path, it is more likely to have a higher success probability for exploiting the same vulnerability type. Let's consider attack path "*ftp_rhosts(0,1)* $\rightarrow$ *rsh(0,1)* $\rightarrow$ *rsh(1,2)* $\rightarrow$ *local_bof(2)*" for instance. When attacker exploits *rsh(0,1)*, it is likely that *rsh(1,2)* is exploited more easily. In other words, successful exploitation of *rsh(0,1)* increases the success probability of *rsh(1,2)* since attacker has already acquired knowledge to exploit the *rsh* vulnerability. Hence if an attacker tries to find a way to crack a device by using a vulnerability, it is possible that it can crack other devices which have the same vulnerability. This feature can be seen in the recent case of Mirai malware which set out to exploit specific vulnerabilities to breach a massive number of IoT systems [26]. An attacker can easily find IoT device vulnerabilities from search engines such as SHODAN [27] and capture hundreds of thousands of device by using the same vulnerability like the access vulnerability exposed in Belkin WeMo devices [16].

Hence, *COBANOT* has a tuning parameter named exploit diversity constant and denoted as θ to adjust this increase in success probability according to the system re-

quirement. In *COBANOT*, security analyst can assign values between 0.0 and 1.0 to θ which has default value 1.0. Value 0 means that success probability of an exploit is not changed according to diversity. As θ increases, exploiting the same vulnerability type while executing an attack path becomes easier. If θ is assigned to 1.0, when attacker exploits a vulnerability in an attack path, the success probability of that vulnerability type becomes 1.0 which means that attacker can always utilize that exploit type successfully after a single successful exploitation. Furthermore, if θ is 1.0, a vulnerability type counts only once in an attack path in order to normalize its effect on the length. For instance, length of "*ftp_rhosts(0,1)* $\rightarrow$ *rsh(0,1)* $\rightarrow$ *rsh(1,2)* $\rightarrow$ *local_bof(2)*" counts as three if θ is 1.0.

Therefore, $Aprob(A_j)$ can be calculated as:

$$Aprob(A_j) = \prod min(1.0, (Eprob(E_i) + \theta \times \varphi \times (1 - Eprob(E_i)))), \quad (2)$$

where $E_i \in A_j$, *min* is the minimum function and φ is a variable which shows the number of exploits whose type is equal to E_i and exploited before E_i. In this formula, *min* is used since success probability of an exploit cannot be larger than one. Each time the same vulnerability type is exploited in an attack path, its success probability increases by an amount of $\theta \times (1 - Eprob(E_i))$.

At the end of first phase, *SM* is calculated to determine security risk of the system. *SM* is a way to capture the attack surface based on a set of known vulnerabilities and initial conditions. Hence, the security analyst can use it to get a better understanding of the threat posed by this attack surface and it might help to drive requests for budget allocations to reduce the attack surface in practical system management. In our proposal, likelihoods of extracted attack paths are added to compute *SM*. In this way, the number of attack paths and their likelihoods are combined to create our security metric. We define security metric *SM* as:

$$SM = \sum Aprob(A_j), \quad (3)$$

where A_j is the attack path which reaches the target device(s).

As mentioned before, there are two options to eliminate attack paths: disabling initial conditions and removing (patching) exploits. In the second phase of *COBANOT*, elimination of paths is done iteratively via the selection of an exploit or initial condition in each step. In *COBANOT*, there are two criteria for selection: the cost of removal α and the contribution to attack paths *Con(x)*. Existence of each exploit and initial condition increases the success probability of the intruder or security risk of the network, i.e. *SM*, and the resulting contribution is a measure of this addition. In other words, the contribution shows us how much the success probability of intruder is affected from removal of an exploit or an initial condition. Contribution of an exploit $Con(E_i)$ can be found by adding likelihoods of all attack paths which includes the corresponding exploit. Contribution of an initial condition $Con(I_k)$ can be found by adding contributions of all exploit(s) which are enabled by the corresponding initial condition. Therefore, $Con(E_i)$ and $Con(I_k)$ can be calculated as:

$$Con(E_i) = \sum Aprob(A_j), \quad (4)$$

where $E_i \in A_j$, and

$$Con(I_k) = \sum Con(E_i), \quad (5)$$

where I_k enables E_i.

Cost of removal α represents the patching cost for exploits and the disabling cost for initial conditions. These costs can originate from diverse actions or security investments such as buying a new firewall, stopping a service or upgrading firmware of the IoT devices. Hence, it is rather context-dependent and assumed to be estimated by security experts and given as input to the algorithm. Parameters α and *Con(x)* are unified as the effective cost β. This latter parameter simply shows how much security risk is reduced per cost. With higher contribution rate, one unit cost provides more reduction in security risk or security metric *SM*. Effective cost β is defined as:

$$\beta = \alpha / Con(x), \quad (6)$$

where x can be an exploit or an initial condition.

β is calculated for each exploit and initial condition. Our mechanism chooses an exploit or an initial condition for removal in each step according to the following rule:

$$\min(\beta) \wedge (\alpha \leq \gamma) \quad (7)$$

Hence, *COBANOT* chooses an E_i or I_k with minimum β among all E_i and I_k whose cost α does not exceed remaining budget γ. If an exploit is selected, attack paths which include that selected exploit are removed. Since the selected exploit is patched, attack paths including it will be no longer available for attackers. If an initial condition is selected, attack paths which include exploit(s) which were enabled by selected initial condition are removed. Thus, the selected initial condition or exploit eliminates its contribution to attack paths and reduces *SM*. By considering β for removal, critical assets in the IoT network are secured with minimal cost. Selecting and using minimum effective cost ensures that highest reduction in security risk is achieved per unit cost in each step. Then *SM* is re-measured and second phase continues until *SM* is zero or γ is less than minimum α.

4. A Case Study for *COBANOT* Operation

In this section, we utilize a network example, AG1, from the literature to elaborate on the operation of *COBANOT*. The AG1 is a well-known network example in the study of attack graphs [9, 12, 15, 28].

Table 4: Success probability and cost value for each exploit type in AG1

Exploits E_i	Success Probability $Eprob(E_i)$	Cost α
ftp_rhosts	0.8	5
rsh	0.8	15
local_bof	1.0	25
sshd_bof	0.8	12

Attack graph of sample network 1 (AG1) is shown in Fig. 1. Plain text items represent security conditions while rounded boxes represent exploits in the figure. More details can be found in [11, 29]. There are eleven exploits and seven initial conditions in the attack graph. Table 4 shows the success probability and the patching cost for each exploit type. All initial conditions have the same disabling cost, which is 10 units and *user(0)* cannot be disabled since it is an attacker. The budget allocated for network protection is 25 units. Success probability values are extracted from *CVSS* base exploitability scores as discussed in Section 3. The values for budget and cost of removal are assumed to be given by security experts. However, in this example, budget and cost of removal values are assigned by us appropriately and randomly in order to illustrate the operation of *COBANOT* more clearly. The unit for cost values can be a money currency such as US dollars or average time as man-hours or some other representation of cost according to user requirements.

Attack path length threshold n and attack path likelihood threshold are set to 4 and 0.01 for this case study, respectively. Also, the diversity constant θ is assigned as 0.5. Table 5 shows four attack paths for intruders which try to obtain root privileges on *host 2* and their corresponding likelihoods. The backward algorithm finds nine attack paths. However, four of them, one example being *ftp_rhosts(2,1)* $\rightarrow$ *rsh(2,1)* $\rightarrow$ *rsh(1,2)* $\rightarrow$ *local_bof(2)*, are excluded since they do not start from the attacker node (*user(0)*). Thus, our algorithm combines advantages of both backward and forward algorithm. Also, *ftp_rhosts(0,1)* $\rightarrow$ *rsh(0,1)* $\rightarrow$ *ftp_rhosts(1,2)* $\rightarrow$ *rsh(1,2)* $\rightarrow$ *local_bof(2)* is excluded since its length is 5. In this case, the attack path likelihood threshold does not help to reduce the number of attack paths since likelihoods of all attack paths is greater than 0.01.

The success probability of an attack path can be calculated by multiplying success probabilities of all exploits which are included in that attack path. For instance, likelihood of first attack path A_1 is $Aprob(A_1) = 0.8 \times 0.8 \times (0.8 + (1 - 0.8) \times 0.5) \times 1.0 = 0.576$. Attack path A_1 includes two *rsh* (*rsh(0,1)* and *rsh(1,2)*) vulnerabilities. Hence, success probability of second *rsh* vulnerability (*rsh(1,2)*) increases to $(0.8 + 1 \times (1 - 0.8) \times 0.5) = 0.9$ because of diversity constant θ being equal to 0.5. If there was another *rsh* vulnerability in this attack path, its success probability would become $(0.8 + 2 \times (1 - 0.8) \times 0.5) = 1.0$.

For the initial setting, SM equals to $0.576 + 0.512 + 0.64 + 0.64 = 2.368$ and is measured by the sum of success probabilities of all four attack paths. SM and budget γ are not zero hence we can execute the second phase of our algorithm. In second phase, we give an example from $COBANOT$ calculations but do not show all of them for the sake of brevity.

In the first iteration, effective cost β is calculated for each exploit and initial condition. Then an exploit or initial condition which has minimum effective cost and a cost of removal smaller than remaining budget is selected for removal. Contribution of an exploit is measured by summing likelihood of attack paths which include that exploit. Contribution of an initial condition is measured by summing likelihood of attack paths which include the exploit(s) enabled by that initial condition. To calculate β, cost of removal α is divided by that contribution as shown in (6). Table 6 shows effective cost and contribution values of exploits and initial conditions for each step. In the table, green, blue and yellow colors (decreasing darkness in grayscale) show chosen exploit or initial condition for iteration 1, 2 and 3, respectively. If an exploit or an initial condition does not contribute to SM, which means that it is not used by attacker, its effective cost is not calculated. For instance, contribution of *ftp_rhosts(0,2)* is 0.64 since only A_4 includes it. Hence, the effective cost of exploit *ftp_rhosts(0,2)* is equal to $5/0.64 = 7.8125$. The cost of *ftp_rhosts(0,2)* is not larger than the budget and it has the minimum effective cost. Hence, *ftp_rhosts(0,2)* is selected for removal and attack path A_4 is removed. Budget γ reduces to $25 - 5 = 20$ and new SM value is $0.576 + 0.512 + 0.64 = 1.728$. Then SM and γ are not zero and we go on with the second iteration.

In the second iteration, the contribution of exploit *sshd_bof(0,1)* which is included by A_2 and A_3 is $0.512 + 0.640 = 1.152$. Since initial condition *sshd(0,1)* enables only *sshd_bof(0,1)*, it has the same contribution. Hence, the effective cost of *sshd(0,1)* is equal to $10/1.152 = 8.6805$. The cost of *sshd(0,1)* is not larger than the budget and it has the minimum effective cost. Hence, *sshd(0,1)* is selected for removal, and A_2 and A_3 which include *sshd_bof(0,1)* are removed. Budget γ reduces to $20 - 10 = 10$ and new SM is 0.576 (likelihood of A_1). Since SM and γ are not zero, we go on with the third iteration. In this iteration, *ftp_rhosts(0,1)* is selected for removal. Consequently, with total cost of 20, we harden network such that there is no attack path for an intruder to breach *host 2* with root privilege. If only initial conditions are taken into account while protecting the network as in [15, 30, 31, 32, 33], the hardening cost would be 30 instead of 20 whereas using only initial condition provides additional gain of completely securing of network elements. However, these former approaches are not adaptive and do not let the security analyst control the cost of hardening in a flexible way, i.e. cost-aware and limited budget operation focusing on the most critical assets. Therefore, considering exploits for removal as in

Table 5: Attack paths in AG1 and their success probabilities

A_j	Attack Paths	Success Probability of A_j $Aprob(A_j)$
A_1	ftp_rhosts(0,1) → rsh(0,1) → rsh(1,2) → local_bof(2)	0.576
A_2	sshd_bof(0,1) → ftp_rhosts(1,2) → rsh(1,2) → local_bof(2)	0.512
A_3	sshd_bof(0,1) → rsh(1,2) → local_bof(2)	0.640
A_4	ftp_rhosts(0,2) → rsh(0,2) → local_bof(2)	0.640

Table 6: Contribution and effective cost β values for AG1

Graph	Step 1		Step 2		Step 3	
Elements	$Con(x)$	β	$Con(x)$	β	$Con(x)$	β
ftp_rhosts(0,1)	0.576	8.680	0.576	8.680	0.576	8.680
rsh(0,1)	0.576	26.041	0.576	26.041	0.576	26.041
ftp_rhosts(1,2)	0.512	9.765	0.512	9.765	0	-
rsh(1,2)	1.728	8.680	1.728	8.680	0.576	26.041
local_bof(2)	2.368	10.557	1.728	14.467	0.576	43.402
sshd_bof(0,1)	1.152	10.416	1.152	10.416	0	-
rsh(0,2)	0.640	23.437	0	-	0	-
ftp_rhosts(0,2)	0.640	7.812	0	-	0	-
ftp(0,1)	0.576	17.361	0.576	17.361	0.576	17.361
ftp(0,2)	0.640	15.625	0	-	0	-
sshd(0,1)	1.152	8.680	1.152	8.680	0	-
ftp(1,2)	0.512	19.531	0.512	19.531	0	-
ftp_rhosts(2,1)	0	-	0	-	0	-
rsh(2,1)	0	-	0	-	0	-
sshd_bof(2,1)	0	-	0	-	0	-
sshd(2,1)	0	-	0	-	0	-
ftp(2,1)	0	-	0	-	0	-

our technique helps us to converge to the minimum cost requirement in a budget-aware manner.

Let us also consider what happens if budget is 12. The first iteration is the same except now the budget reduces to $12 - 5 = 7$. Hence, in second iteration, *sshd(0,1)* cannot be chosen this time since the cost of *sshd(0,1)* is 10. Therefore, *ftp_rhosts(0,1)* is chosen for removal and A_1 is removed. New SM is $0.512 + 0.64 = 1.152$ and there are two attack paths , A_2 and A_3, to breach. However, there is no more hardening option left since the remaining budget is only 2. In this case, *COBANOT* finds a favorable budget-limited network hardening solution without securing critical assets completely.

5. Complexity/Cost Analysis and Experimental Results

In this section, algorithm and run-time complexity of Phase I and Phase II are analyzed to render the performance of *COBANOT*. Extensive simulations are conducted and reported to demonstrate its performance characteristics. The efficiency of minimum cost solution is investigated by comparing total protection cost of *COBANOT* with optimal minimum cost solution.

5.1. Complexity Analysis of COBANOT

COBANOT is a greedy heuristic-based scheme, since finding optimal solution is an intractable problem for minimum-cost network hardening [9]. Phase I of *COBANOT* extracts at most $O(M^n)$ attack paths in $O(M^{n-1})$ time [15]. Here, M is the maximum number of exploits which point to a condition in the attack graph and n is the path length threshold [15]. An initial condition or an exploit is selected in each iteration of Phase II. Hence, there can be at most $|E_i| + |I_k|$ iterations where $|I_k|$ is number of initial conditions and $|E_i|$ is number of exploits. Computing contributions of initial condition and exploits takes $O(nM^n)$ time

Table 7: Attributes of randomly generated attack graphs

AG	$\|I_k\|$	$\|E_i\|$	$Avg\|Edges\|$
A	21	32	129
B	98	305	1382
C	512	984	4982
D	945	1326	6814
E	2122	4023	21221

since there are at most $O(M^n)$ attack paths with maximum length n. Selecting an exploit or initial condition takes $O(|E_i| + |I_k|)$ time. Therefore, each iteration in Phase II takes $O(nM^n)$ time. Computational complexity of Phase II becomes $O((|E_i| + |I_k|)nM^n)$ time. We know that attackers do not typically use attack paths whose length is more than 4 [15, 18]. Thus worst-case computational complexity of Phase I becomes $O(M^3)$ and Phase II becomes $O((|E_i|+|I_k|)M^4)$. Basically attack path length threshold is helpful to avoid state explosion while generating attack paths.

For complexity analysis, we use different sizes of IoT networks while conducting experiments. All attack graphs and parameter values such as success probability and cost of removal are generated randomly. However, we try to imitate attack graphs of real IoT systems to obtain meaningful results. Cost of removal α is assigned from the range [5,10] for initial conditions and from the range [2,8] for exploits. Success probability of exploits $Eprob(E_i)$ is assigned from the range [0.3,1.0]. M is 10 and likelihood threshold of attack path is 0.01. M is high since we try to simulate densely connected IoT devices with various vulnerabilities. Also the budget is unbounded and *COBANOT* always eliminates all attack paths. Therefore simulations show that worst-case performance of *COBANOT*. We run each simulation for 50 times and report the average of these runs. During the simulations, firstly, exploits and initial conditions are generated according to above values. Then initial conditions are assigned to exploits and exploits are randomly connected with a maximum number of M connections.

The simulation environment and *COBANOT* were implemented in Java. Experiments are conducted single-threaded on a computer with 3.4GHz CPU and 8GB RAM to show run-time performance of *COBANOT*. The operating system was Microsoft Windows 7 Ultimate with Service Pack 1. Table 7 shows different attack graph configurations with different number of initial conditions ($|I_k|$), different number of exploits ($|E_i|$) and different number of edges ($|Edges|$). When attack graphs are constructed, number of edges and extracted attack paths are changed according to the topology of the graph. Therefore, the parameter $Avg|Edges|$ shows the average number of edges for our randomly generated attack graphs.

In Fig. 3, the average number of attack paths (denoted as $Avg|A_j|$) with different path length threshold n for randomly generated attack graphs is shown. As expected, the size of attack graph has a substantial effect on the number of attack paths. It can also be seen from the graph that path length threshold n can reduce the number of attack paths ($|A_j|$) drastically. $|A_j|$ increases significantly when n is changed from 5 to 10 or from 10 to 15. Likelihood threshold becomes active when the length of attack paths exceeds 15. Consequently, this dramatic growth turns into a manageable one. By setting likelihood and path length thresholds, only simple and probable attack paths which are generally preferred by the attackers can be found and eliminated. These findings indicate the importance of both path length and likelihood threshold parameters.

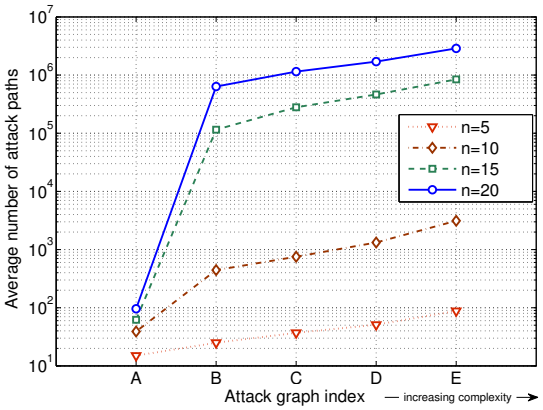

Figure 3: Average number of attack paths for randomly generated attack graphs with different path length threshold n.

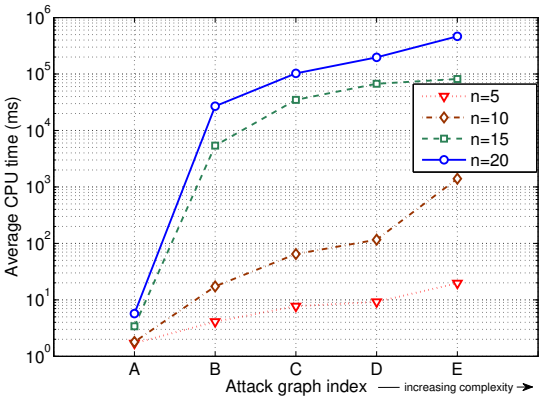

Figure 4: Average CPU time in milliseconds for Phase I of *COBANOT* in randomly generated attack graphs with different path length threshold n.

Fig. 4 and 5 show average CPU time in milliseconds for Phase I and Phase II of *COBANOT* with different path length threshold n for randomly generated attack graphs. As n and network size increase, Phase I needs more time to find all attack paths. Similarly, Phase II requires more time for each iteration. Also, as the number of attack paths which are found by Phase I increases, Phase II needs more time to protect the network with minimum cost. Moreover, larger $|A_j|$ generally induces more

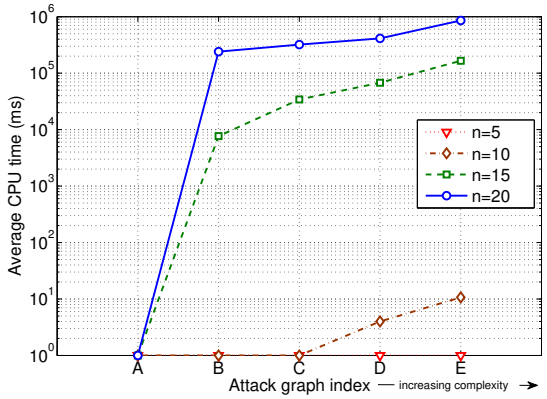

Figure 5: Average CPU time in milliseconds for Phase II of *COBANOT* in randomly generated attack graphs with different path length threshold n.

Table 8: Features of randomly generated attack graphs for cost comparison

AG	$\|I_k\|$	$\|E_i\|$	$Avg\|Edges\|$	$Avg\|A_j\|$
F	8	12	54.7	4.9
G	10	15	64.7	8.3
H	12	18	76.8	18.9
I	15	20	99.9	23.6
J	17	23	119.7	26.4

iterations in Phase II. These results are consistent with our computational complexity analysis and show that average run-time complexity of *COBANOT* is much better than the computational worst-case complexity. Furthermore, when n is five, it finds minimum cost solution for all randomly generated graphs in less than one minute.

The results also indicate that *COBANOT* scales approximately linearly with the size of the graphs although CPU time can vary according to the characteristic of the attack graph and selected thresholds. In less than 25 minutes, it finds a cost-effective solution for an attack graph with tens of thousands of nodes by eliminating nearly three million attack paths during the experiments. These results show that *COBANOT* can be used in large IoT networks encompassing a huge number of devices with prevalent vulnerabilities.

5.2. Protection Cost Analysis of COBANOT

We investigate the protection cost of *COBANOT* with respect to optimal protection cost and optimal protection cost by using only initial conditions in this section. In the literature, there are many works which consider only initial conditions for protection [15, 30, 31, 32, 33]. Table 8 shows some small and random attack graph configurations. We choose small graphs which are different from rather complex graphs used before since finding optimal solution is a NP-hard problem and it takes too much time as shown

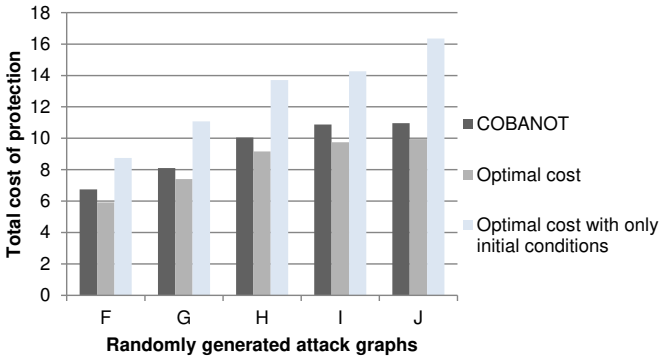

Figure 6: Average total protection costs for *COBANOT*, optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.

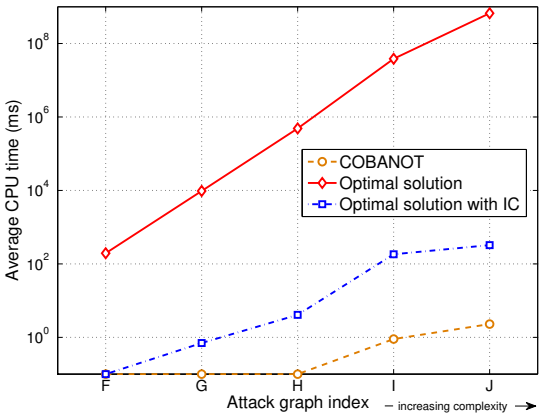

Figure 7: Average CPU time in milliseconds for *COBANOT*, optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.

later. Cost of removal values are the same with the settings in Section 5.1.

In Fig. 6, average protection cost for randomly generated attack graphs is shown for *COBANOT*, optimal solution and optimal solution by using initial conditions. Fig. 7 shows average required time for this protection. Apparently, the required time for optimal solution grows exponentially as the number of exploits and initial conditions rise. Because $2^{|E_i|} + |I_k|$ iterations are required to find it. The optimal solution by using initial conditions has the same tendency. However, it is only affected by the number of initial conditions ($2^{|I_k|}$). Therefore, both optimal solution and optimal solution by using only initial conditions are unfeasible to use in large attack graphs. Average protection cost of *COBANOT* is generally about 10% higher than the optimal solution which is tolerable when considering the time gains (reduction). Additionally, the performance of *COBANOT* is better than the optimal solution by using only initial conditions which shows that considering exploits while protecting network offers a huge advantage.

6. Related Work

There are various works focusing on attack graphs and network hardening in the literature. In [9], initial security conditions (initial conditions) are removed in an attack graph for network hardening. Their proposed solution traverses attack graph with a backward algorithm. In [30] Man *et al.* calculate criticality degree for each node in the attack graph. The nodes with higher degrees are selected for removal. However, [9] and [30] do not consider the cost factor while selecting conditions or exploits to remove.

Islam *et al.* propose a heuristic approach for minimum cost network hardening [31]. They aim to select initial conditions which are disabled to protect the network. But they do not consider success probabilities of attack paths in their cost function. If two initial conditions have the same cost of removal, the one which participates in more attack paths is selected. However, initial condition which participates in attack paths with higher success probability is a more suitable candidate for removal. Therefore, our approach *COBANOT* considers success probabilities of attack paths while selecting the best option.

In [15], all n-valid attack paths which are loop-free with distance n are extracted from an attack graph. Then the security metric of the network is calculated according to the path length and the number of different exploits in all paths. However, this security metric does not provide much information about the success probability of an attack since long attack paths can be easier to breach. Considering this fact, we calculate network security metric according to success probabilities of attack paths so that a security analyst can assess potential threats more easily. In [15], initial conditions are removed according to the cost and the number of participated attack paths.

Jun-chun *et al.* use genetic algorithms to find minimum cost network hardening solution by transforming it into a non-constrained optimization problem with penalty in [32]. They use bidirectional-based search strategy to explore the relationship of network vulnerabilities. In this way, the efficiency of attack graph generation is improved. The proposed method guarantees the network security with the least cost. In [33], attack graphs are converted to *Reduced Ordered Binary Decision Diagram (ROBDD)*. Then Chen *et al.* find the minimum cost solution from this new representation which does not require to list every possible solution. Hence, the algorithmic complexity is reduced. The minimum cost solution is found by a recursive algorithm which is a depth-first search of *ROBDD*.

Although, [15, 30, 31, 32, 33] focus only on initial conditions, it is possible to remove an exploit by patching it to protect the network against known attacks [12]. In [29, 11, 34], minimal critical attack set is found to harden the network since finding minimum critical attack sets leading to optimum solution is NP-hard. The minimal critical attack set is a minimal set of exploits in the attack graph whose removal eliminates all the attack paths which reach target assets. Due to complexity, these approaches are heuristic based and only exploits are considered for removal. In [29] and [11], model checking techniques are used to generate and analyze attack graphs. These techniques produce attack graphs with a very large number of states which also include many redundant states. Thus, they suffer from scalability issue while finding minimal critical attack set. In our work, both initial conditions and exploits are considered jointly to protect a network with minimum cost.

Keramati *et al.* introduce a framework for minimum cost network hardening in [12]. Initially, all attack paths are extracted from attack graphs. The priority of an attack path is calculated by predefined rules and then attack paths with highest priority are selected for removal. While calculating the priority, CVSS exploitability and impact metrics are used directly by summation to find easiness and impact of an attack path. The drawback of this approach is that some attack paths are not removed, which makes the network insecure. Additionally, it suffers from exponential complexity since it uses the method in [9] to extract the critical set.

Homer *et al.* use Boolean Satisfiability Solving (SAT) to harden the network by changing its configurations [35]. Firstly, an attack graph which is generated by MulVAL [13] is converted to a first-order logic formula which is in conjunction with a security policy. For usability, the security analyst can mark some configuration as unchangeable similar to our work. The authors use UnSAT core elimination and MinCostSAT separately. UnSAT core elimination does not need discrete cost values and any conflict is asked to the user with possible options.

In [36], security analysts determine all possible countermeasures to protect a network. A genetic algorithm is used to find the minimum cut-set in the attack graph according to determined countermeasures. However, security analysts cannot determine all the countermeasures since there can be millions of them in an large attack graph. Hence, our algorithm automatically extracts all possible countermeasures from attack graph. In [37], attack graphs are used to select countermeasures according to stateful return on investmest metric. As a result, an optimal set of countermeasures are found and enforced over the network.

Overall, works in [9, 11, 12, 15, 29, 30, 31, 32, 33, 34, 35, 36, 37] have a similar drawback compared to our approach. Usually, it is difficult to secure a network completely due to resource limitations. Since our proposal works iteratively, it can be stopped when the allocated budget is depleted. Therefore, our scheme lets the security analyst balance security vs. cost according to context-based requirements. This is critical for practical network hardening in network security.

There are also some works considering the budget aspect in the literature. In [38] and [39], network hardening solutions with budget consideration are proposed. However, these solutions require that a set of possible defenses with specified cost and expected benefit is given by a security analyst. In [38], a heuristic algorithm which con-

siders benefit and cost together is utilized to find the optimal solution. The authors also offer a method to generate attack graph of a network and calculate its security risk. In [39], minimum cost solution is transformed to a binary knapsack problem which has exponential complexity. They choose hardening measures according to security metrics and cost. The proposed method guarantees optimal solution and utilize dynamic programming to achieve this.

Sawilla *et al.* try to minimize connectivity in *AND/OR* directed dependency attack graph by using partial cuts in [4]. AssetRank [40] is applied to produce a rank metric for each asset in the network. A k-connected graph is used to find critical attack paths. A k-connected graph is a graph where k is the smallest number of vertices whose removal disrupts the graph. Then they try to eliminate critical attack paths according to cost and rank metric with budget consideration. However, their solution does not provide any security metric to a security analyst to show the effect of network hardening or its current state. Moreover, usage of AssetRank to find critical paths is a costly operation in terms of complexity and there is no analysis provided regarding that aspect in their work.

In [41], Homer *et al.* propose a security metric which shows likelihoods of an attack to a given target. While calculating this metric, authors use *Access Complexity* submetric of *CVSS* to assign probabilities to exploits by using a simple mapping similar to our temporal score approach. In [20], base and temporal metric scores are used together to calculate likelihood of an exploit. Houmb *et al.* use base and temporal metrics for frequency estimation [21]. They also compute impact of an exploit by using base and environmental metrics.

Security of the IoT systems has attracted significant attention in recent years [42, 43, 44, 45]. There are many attack types against IoT systems like sinkhole, wormhole, and sybil attacks. These attacks are possible since IoT devices have lots of security vulnerabilities and hardening these devices is challenging. Attack graphs are instrumental to support security hardening of IoT networks. To this end, Ge *et al.* propose a framework to model and assess the security of IoT systems via attack graphs [46]. This work is the first model which uses attack graph to model and assess security of an IoT system. The proposed framework finds the possible attack paths, measures security metrics by using CVSS and assesses defense options. However, cost of the defense and budget limitations are not considered. The framework mainly calculates success probability, impact and cost of attack paths. After that, it decides on a defense solution according to these metrics. In another work, Ge *et al.* propose a proactive defense mechanism for IoT by using Software-Defined Networking (SDN) [47]. The defense approach is similar to their previous work while SDN is used to reconfigure the network topology to remove non-patchable vulnerabilities. In other words, they use SDN to disable initial conditions easily. Wang *et al.* propose a vulnerability assessment method for industrial IoT by using attack graphs [23]. Firstly, attack graph is generated for the IoT system and attack paths are extracted. Then, vulnerability degree of the attack paths according to *CVSS* is calculated and priority order of the attack paths is determined.

Overall, fueled by the critical need of secure networks, numerous network hardening methods using attack graphs have been proposed in the literature. However, these solutions generally focus on and address only one aspect of the problem. Some of them work only with initial conditions to protect the network, and thus violate minimum cost requirement, or just calculate the security metric and do not offer a solution to harden the network. Moreover, most of them do not consider the budget constraints and do not pose as a viable solution in a real-world situation. Although budget constraint is considered in some works such as [38, 39, 4], the presented solutions lack fundamental network hardening features such as a security metric. Furthermore, their methods have significant drawbacks regarding applicability in real networks as described above.

7. Discussion

COBANOT requires that security experts provide the cost estimations for initial conditions and exploits. Although this estimation process may emerge as the biggest challenge for *COBANOT*, this is due to the context-dependent nature of the problem and apparent in all such works in the literature. The cost of patching an exploit or disabling an initial condition will vary substantially from one network to another. Moreover, IoT corresponds to a multitude of use cases and services ranging from infotainment to critical infrastructure protection. Thus there is no one-size-fits-all cost function applicable to all IoT systems [35]. Hence, security experts are expected to estimate the costs based on native requirements and policies. In the simulation and case study, cost values are assigned randomly in a suitable range under these constraints.

Our security metric is calculated by adding likelihoods of attack paths and it can be between 0 to ∞. It does not show success probability of the attacker as in [41] which has high complexity. Since our algorithm calculates security metric in every iteration, it requires low-complexity and quick calculation. Accordingly it does not consider overlaps among attack paths, since it aims to show that how secure the network is and how changes in the network affect the system.

Another challenge may be the zero day vulnerabilities since they do not have patches or *CVSS* scores. For that case, the security analyst can mark a zero day vulnerability as unpatchable as we suggest in Section 2. Moreover, if *CVSS* score of an vulnerability does not exist or is not defined, the security analyst can assign them probabilities according to specific security requirements. For instance, 1.0 value can be assigned if the expert wants to consider worst-case scenario, e.g. in an IoT environment operating in a critical industrial control network.

In *COBANOT*, a network security manager can tune three parameters: path length, likelihood threshold and exploit diversity constant θ. Considering the sensitivity of the algorithm to these parameters, experiments show the impact of the threshold parameters. On the other hand, exploit diversity constant θ is used in likelihood calculation. If all attack paths will be removed, θ does not affect the selected hardening option but the order of precautions since minimum cost hardening does not depend on likelihoods. However, if the budget is limited, then the order of precautions is changed.

The experimental results also convey the effect of budget parameter on the algorithm's operation. The budget does not affect Phase I of the algorithm as expected. If it is limited, some attack paths may not be removed which needs less iterations in Phase II and thus less CPU time. If it exceeds total protection cost, time spent in Phase II does not change. This behavior is consistent with the expectations stemming from the mechanics of the algorithm.

In this paper, attack graph generation is not studied and we assume that attack graph of the IoT network is generated a priori and given to the proposed scheme. Realistic large-scale attack graph generation is an important research topic. Accordingly, there are lots of works in the literature which work on efficient attack graph generation, update and visualization such as [5, 48]. However, new concepts like social engineering must be taken into account while generating attack graphs to better cope with a wide spectrum of attacks since according to the ENISA Threat Landscape report [49], around 90% of the attacks with malware involve social engineering. This topic can be a promising future research direction to extend COBANOT.

Another such emerging research challenge is the nation state attackers. In our solution, nation state attackers are not considered since they have much more budget, time and arsenal than ordinary cybercriminals. In that regard, they exhibit unique attack characteristics which may call for optimized custom approaches. For instance, the employed temporal exploitability score mapping can be optimistic in the case of nation state attacks. Hence, that mapping in *COBANOT* can be changed by security analyst to cope with different threat agents. Insider attacks are not considered either since *COBANOT* benefits from forward algorithm while extracting attack paths. Hence, the attacker in our model is an arbitrary entity in the Internet performing a remote attack. However, the security analyst can still utilize *COBANOT* to tackle with such attacks by disabling forward algorithm pruning at the expense of having a higher number of attack paths extracted during the process.

Finally, *COBANOT*'s approach does not recommend or encourage to leave some vulnerabilities unpatched. Please note that all vulnerabilities exposed to the public should be fixed in the long run. *COBANOT* provides an effective framework to prioritize hardening options based on maximum remedial impact with the resources available. If the security metric *SM* shows that the attack surface/risk is still too large, the security analyst should think hard to get additional resources to fix these issues since leaving the system vulnerable in the Internet might induce much higher costs than fixing the problem itself.

8. Conclusion

In this paper, we have presented a network hardening solution with cost minimization by using compact attack graphs for IoT systems. The proposed solution *COBANOT* provides heuristic-based cost-controlled network hardening by considering exploits and initial conditions for potential removal. Furthermore, we consider budget limitation while protecting the system. Therefore, a feasible hardening option under the given budget constraint is discovered while trying to maximize the security level of the IoT system. With the help of the path length and likelihood thresholds, less likely and complicated attack paths are eliminated. Hence, the allocated budget is used efficiently. We further extract success probabilities of exploits from *CVSS* metrics and propose a novel method to measure security metric of a system to represent how secure the analyzed system is. The extensive simulations show that *COBANOT* has a reasonable run-time complexity even for large-scale IoT networks. Besides, it is shown that the protection cost of *COBANOT* is minimal with respect to the optimal solution and much better than the optimal solution which uses only initial conditions for removal.

In addition to the points discussed in Section 7, the impact of attack paths on integrity, confidentiality and availability of IoT assets can be considered while calculating the security metric as future work. In that way, not only the likelihood of an attack path but also its impact on the network assets will be taken into account. Moreover, base and temporal exploitability scores of *CVSS* can be merged to calculate the success probabilities of exploits considering their inherent and time-dependent features jointly.

Acknowledgments

This work was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under grant number 117E165.

[1] R. Deraison, Nessus scanner, `http://www.nessus.org`, [Accessed March 2018].

[2] M. Roesch, Snort - lightweight intrusion detection for networks, in: Proc. of the 13th USENIX Conf. on System Administration, 1999, pp. 229–238.

[3] R. P. Lippmann, K. W. Ingols, An annotated review of past papers on attack graphs, Tech. rep., MIT Lincoln Lab (2005).

[4] R. Sawilla, D. Skillicorn, Partial cuts in attack graphs for cost effective network defence, in: IEEE Conf. on Technologies for Homeland Security (HST), 2012, 2012, pp. 291–297.

[5] J. B. Hong, D. S. Kim, Towards scalable security analysis using multi-layered security models, Journal of Network and Computer Applications 75 (2016) 156 – 168.

[6] B. Yigit, G. Gür, F. Alagöz, Cost-aware network hardening with limited budget using compact attack graphs, in: Military Communications Conference (MILCOM), 2014 IEEE, IEEE, 2014, pp. 152–157.

[7] L. D. Bodin, L. A. Gordon, M. P. Loeb, Evaluating information security investments using the analytic hierarchy process, Communications of the ACM 48 (2) (2005) 78–83.

[8] L. Wang, T. Islam, T. Long, A. Singhal, S. Jajodia, An attack graph-based probabilistic security metric, in: Proc. of the 22nd Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2008, pp. 283–296.

[9] L. Wang, S. Noel, S. Jajodia, Minimum-cost network hardening using attack graphs, Computer Communications 29 (18) (2006) 3812 – 3824.

[10] P. Mell, K. Scarfone, S. Romanosky, Common vulnerability scoring system, IEEE Security and Privacy 4 (6) (2006) 85–89.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, Automated generation and analysis of attack graphs, in: Proc. of IEEE Symposium on Security and Privacy, 2002, pp. 273–284.

[12] M. Keramati, H. Asgharian, A. Akbari, Cost-aware network immunization framework for intrusion prevention, in: Proc. of the IEEE Int. Conf. on Computer Applications and Industrial Electronics (ICCAIE'11), 2011, pp. 639–644.

[13] X. Ou, S. Govindavajhala, A. W. Appel, Mulval: A logic-based network security analyzer, in: Proc. of the 14th Conf. on USENIX Security Symposium, SSYM'05, 2005, pp. 8–33.

[14] K. Ingols, R. Lippmann, K. Piwowarski, Practical attack graph generation for network defense, in: Proc. of 22nd Annual Computer Security Applications Conf. (ACSAC06), 2006, pp. 121–130.

[15] F. Chen, D. Liu, Y. Zhang, J. Su, A scalable approach to analyzing network security using compact attack graphs, Journal of Networks 5 (5) (2010) 543–550.

[16] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, C. Xu, Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things, in: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, ACM, 2015, p. 5.

[17] Common Vulnerability Scoring System (CVSS), `http://www.first.org/cvss/`, [Accessed March 2018].

[18] Z. Liu, S. Li, J. He, D. Xie, Z. Deng, Complex network security analysis based on attack graph model, in: Proc. of 2012 Second Int. Conf. on Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2012, pp. 183–186.

[19] L. Wang, A. Singhal, S. Jajodia, Toward measuring network security using attack graphs, in: Proc. of the 2007 ACM Workshop on Quality of Protection, 2007, pp. 49–54.

[20] N. Ghosh, S. Ghosh, An approach for security assessment of network configurations using attack graph, in: First Int. Conf. on Networks and Communications, NETCOM '09, 2009, pp. 283–288.

[21] S. H. Houmb, V. N. Franqueira, E. A. Engum, Quantifying security risk level from cvss estimates of frequency and impact, Journal of Systems and Software 83 (9) (2010) 1622 – 1634.

[22] M. Ge, J. B. Hong, W. Guttmann, D. S. Kim, A framework for automating security analysis of the internet of things, Journal of Network and Computer Applications 83 (2017) 12 – 27.

[23] H. Wang, Z. Chen, J. Zhao, X. Di, D. Liu, A vulnerability assessment method in industrial internet of things based on attack graph and maximum flow, IEEE Access.

[24] National Vulnerability Database. (NVD), `http://nvd.nist.gov`, [Accessed March 2018].

[25] Q. Hui, W. Kun, Real-time network attack intention recognition algorithm, International Journal of Security and Its Applications 10 (4) (2016) 51–61.

[26] M. Özçelik, N. Chalabianloo, G. Gür, Software-defined edge defense against IoT-based DDoS, in: 2017 IEEE International Conference on Computer and Information Technology (CIT), 2017, pp. 308–313.

[27] Shodan, `https://www.shodan.io/`, [Accessed March 2018].

[28] C. Feng, S. Jin-Shu, A flexible approach to measuring network security using attack graphs, in: 2008 Int. Symposium on Electronic Commerce and Security, 2008, pp. 426–431.

[29] S. Jha, O. Sheyner, J. Wing, Two formal analysis of attack graphs, in: Proc. of the 15th IEEE Workshop on Computer Security Foundations, 2002, pp. 49–63.

[30] D. Man, Y. Wu, Y. Wu, A method based on global attack graph for network hardening, in: Proc. of the 4th Int. Conf. on Wireless Communications, Networking and Mobile Computing (WiCOM'08), 2008, pp. 1–4. `doi:10.1109/wicom.2008.1086`.

[31] T. Islam, L. Wang, A heuristic approach to minimum-cost network hardening using attack graph, in: Proc. of the New Technologies, Mobility and Security (NTMS'08), 2008, pp. 1–5.

[32] M. Jun-chun, W. Yong-jun, S. Ji-yin, C. Shan, A minimum cost of network hardening model based on attack graphs, Procedia Engineering 15 (2011) 3227 – 3233.

[33] F. Chen, L. Wang, J. Su, An efficient approach to minimum-cost network hardening using attack graphs, in: Fourth Int. Conf. on Information Assurance and Security (ISIAS'08), 2008, pp. 209–212.

[34] P. Ammann, D. Wijesekera, S. Kaushik, Scalable, graph-based network vulnerability analysis, in: Proc. of the 9th ACM Conf. on Computer and Communications Security, 2002, pp. 217–224.

[35] J. Homer, X. Ou, Sat-solving approaches to context-aware enterprise network security management, Selected Areas in Communications, IEEE Journal on 27 (3) (2009) 315–322.

[36] M. Alhomidi, M. Reed, Finding the minimum cut set in attack graphs using genetic algorithms, in: Proc. of Int. Conf. on Computer Applications Technology (ICCAT'13), 2013, pp. 1–6.

[37] G. Gonzalez-Granadillo, E. Doynikova, I. Kotenko, J. Garcia-Alfaro, Attack graph-based countermeasure selection using a stateful return on investment metric, in: International Symposium on Foundations and Practice of Security, Springer, 2017, pp. 293–302.

[38] C. Phillips, L. P. Swiler, A graph-based system for network-vulnerability analysis, in: Proc. of the 1998 Workshop on New Security Paradigms, 1998, pp. 71–79.

[39] N. C. Idika, B. H. Marshall, B. K. Bhargava, Maximizing network security given a limited budget, in: The Fifth Richard Tapia Celebration of Diversity in Computing Conf.: Intellect, Initiatives, Insight, and Innovations, 2009, pp. 12–17.

[40] R. E. Sawilla, X. Ou, Identifying critical attack assets in dependency attack graphs, in: Proc. of the 13th European Symposium on Research in Computer Security: Computer Security, 2008, pp. 18–34.

[41] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, A. Singhal, Aggregating vulnerability metrics in enterprise networks using attack graphs, Journal of Computer Security 21 (4) (2013) 561–597.

[42] X. Jia, D. He, Q. Liu, K.-K. R. Choo, An efficient provably-secure certificateless signature scheme for internet-of-things deployment, Ad Hoc Networks.

[43] M. Ammar, G. Russello, B. Crispo, Internet of things: A survey on the security of IoT frameworks, Journal of Information Security and Applications 38 (2018) 8–27.

[44] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications, IEEE Internet of Things Journal 4 (5) (2017) 1125–1142.

[45] J. M. de Fuentes, P. Peris-Lopez, J. E. Tapiador, S. Pastrana, Probabilistic yoking proofs for large scale iot systems, Ad Hoc Networks 32 (2015) 43–52.

[46] M. Ge, J. B. Hong, W. Guttmann, D. S. Kim, A framework for automating security analysis of the internet of things, Journal of Network and Computer Applications 83 (2017) 12–27.

[47] M. Ge, J. B. Hong, S. E. Yusuf, D. S. Kim, Proactive defense mechanisms for the software-defined internet of things with non-patchable vulnerabilities, Future Generation Computer Systems 78 (2018) 568–582.

[48] V. Shandilya, C. B. Simmons, S. Shiva, Use of attack graphs in security systems, Journal of Computer Networks and Communications 2014.

[49] ENISA threat landscape report 2017, `https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2017`, [Accessed March 2018].