Journal of Computer Security 0 (2018) 1
IOS Press

Surviving Unpatchable Vulnerabilities through Heterogeneous Network Hardening Options

Daniel Borbor [a,*], Lingyu Wang [a] and Sushil Jajodia [b] Anoop Singhal [c]

[a] *Concordia Institute for Information Systems Engineering, Concordia University, Quebec, Canada*
E-mails: d_borbor@ciise.concordia.ca, wang@ciise.concordia.ca
[b] *Center for Secure Information Systems, George Mason University, MD, USA*
E-mail: jajodia@gmu.edu
[c] *Computer Security Division, National Institute of Standards and Technology, VA, USA*
E-mail: anoop.singhal@nist.gov

Abstract. The administrators of a mission critical network usually have to worry about non-traditional threats, e.g., how to live with known, but unpatchable vulnerabilities, and how to improve the network's resilience against potentially unknown vulnerabilities. To this end, network hardening is a well-known preventive security solution that aims to improve network security by taking proactive actions, namely, hardening options. However, most existing network hardening approaches rely on a single hardening option, such as disabling unnecessary services, which becomes less effective when it comes to dealing with unknown and unpatchable vulnerabilities. There lacks a heterogeneous approach that can combine different hardening options in an optimal way to deal with both unknown and unpatchable vulnerabilities. In this paper, we propose such an approach by unifying multiple hardening options, such as service diversification, firewall rule modification, adding, removing, and relocating network resources, and access control, all under the same model. We then apply security metrics designed for evaluating network resilience against unknown and unpatchable vulnerabilities, and consequently derive optimal solutions to maximize security under given cost constraints. Finally, we study the effectiveness of our solution against unpatchable vulnerabilities through simulations.

Keywords: Network Hardening, Heterogeneous Hardening, Unpatchable Vulnerabilities, Security Metrics, Diversity

1. Introduction

Today's computing networks are playing the role of nerve systems in many mission critical infrastructures, such as cloud data centers and industry control systems. However, the scale and severity of security breaches in such networks have continued to grow at an ever-increasing pace, which is evidenced by many high-profile security incidents, such as the recent large-scale DDoS attacks caused by the Mirai Botnet on the Dyn DNS, and the cyber-physical attack on the Ukrainian power grid in 2015 [1]. The so-called zero-day attacks, which exploit either previously unknown or known, but unpatched vulnerabilities, are usually behind such security incidents, e.g., Stuxnet employs four different zero day vulnerabilities to target SCADA [2]. Therefore, administrators of a mission critical network usually

*Corresponding author. E-mail: d_borbor@ciise.concordia.ca.

need to worry about not only patching known vulnerabilities and deploying traditional defense mechanisms (e.g., firewalls, IDSs, and IPSs), but also non-traditional security threats, e.g., how to live with known, but unpatchable vulnerabilities, and how to improve the network's resilience against potentially unknown vulnerabilities.

In fact, it is known that cybercriminals frequently leverage vulnerabilities that are not publicly known. On the other hand, even for known vulnerabilities, patching is not always a viable option. For example, a patch may not be readily available at the time of the attack (e.g. the remote exploit vulnerability CVE-2016-4502 [3]), or the system may have reached their end-of-support with no more patch available (e.g. the Atom-Bombing windows vulnerability [4]); patching a vulnerability may cause unacceptable service disruptions on a regular basis; even worse, patching a vulnerability may sometimes reintroduce other security vulnerabilities that have previously been fixed (e.g., Apache MINA SSHD 2.0.14 introduces an SSL regression previously fixed in 2.0.13 [5]).

Consequently, security professionals need to block the exploitation of such vulnerabilities through other means, such as adding, removing, or relocating services, as well as modifying firewall rules, service diversification, or access control. A critical question is *How to optimally combine such options in order to both improve the security and lower the cost?* To this end, network hardening is a well-known preventive security solution that aims to improve network security by taking proactive actions, namely, hardening options. However, most existing network hardening approaches rely on a single hardening option, such as disabling unnecessary services [6, 7] or service diversification [8] (a detailed review of related work will be given later in Section 5). Such a solution becomes less effective when it comes to dealing with unknown and unpatchable vulnerabilities. There lacks a heterogeneous approach that can combine different hardening options in an optimal way to deal with such vulnerabilities.

In this paper, we develop such an approach to optimally combine heterogeneous hardening options in order to increase a network's resilience against both unknown and unpatchable vulnerabilities under various cost constraints. Specifically, we first devise a unified model for heterogeneous hardening options. We also design a cost model and discuss how hardening cost may be estimated in a realistic fashion. We then formulate network hardening as an optimization problem and develop optimization and heuristic algorithms to derive optimal solutions under given cost constraints. We evaluate our approach through simulations in order to study the effect of optimization parameters on accuracy and running time, as well as the effectiveness of hardening against unpatchable vulnerabilities and for different types of networks. In summary, the main contribution of this paper is the following.

- To the best of our knowledge, this is the first effort on network hardening that covers a spectrum of heterogeneous hardening options, including service diversification, adding, removing, and relocating resources, as well as firewall and access control rule modification.
- In contrast to previous works, which typically assume ad-hoc hardening cost assignments, we provide a refined cost model and cost estimation criteria that take into account real world variables in calculating hardening costs.
- As evidenced by the simulation results, our optimization and heuristic algorithms are efficient and effective, and hence they provide a practical solution for network administrators to improve their networks' resilience against unknown and unpatchable vulnerabilities.
- Finally, by focusing on unknown and unpatchable vulnerabilities, our work provides a more encompassing complementary solution to existing network hardening approaches that focus on fixing known vulnerabilities.

The preliminary version of this paper has previously appeared in [9]. In this paper, we have substantially improved and extended the previous version. The most significant extensions are as follows. First, in addition to the hardening options already covered in the previous version, i.e., service diversification and firewall rule modification, we have further introduced three new hardening options in this paper, i.e., adding new resources, removing existing resources, and relocating resources between given locations in the network (detailed in Sections 2.1). Integrating those new options into the existing model allows us to further improve the capability of surviving unpatchable vulnerabilities. Second, in addition to the cost model proposed in the preliminary version, we have further provided realistic methods for estimating the costs of all the hardening options we propose (detailed in Section 2.3). Third, we have provided an additional analysis on the steps taken to instantiate our metric for given networks (Section 2.4). Fourth, a new heuristic algorithm for efficiently computing the hardening metric in special cases is provided (Section 3.3). Finally, we have conducted a series of new simulations to demonstrate how our solution performs in the presence of the newly added hardening options (Section 4).

The remainder of this paper is organized as follows: The rest of this section first builds the motivation through a running example. In Section 2, we present the model and formulate the optimization problem, and in Section 3 we discuss the methodology and show case studies. Section 4 shows simulation results. Section 5 reviews related work and Section 6 concludes the paper.

1.1. Motivating Example

We first consider a concrete example to demonstrate why deriving an optimal solution with heterogeneous hardening options can be a tedious and error-prone task if done manually and would therefore benefit from a systematic and automated approach, even if the considered network is of a small size. Figure 1 shows a hypothetical network for a typical cloud data center [10] based on the OpenStack architecture [11]. Despite its relatively small scale, it mimics a typical cloud network: The client layer connects the cloud network to the internet through a router (CRS 7600); a firewall (ASA v1000) separates the outside network from the inner one; there is a security/authentication layer (authentication server, Neutron server, etc.) as well as a virtual machine (VM) and application layer (web and application servers); finally, a storage layer is separated and protected by another firewall (ASA 5500) and an MDS 9000 multilayer switch [10].

We make the following assumptions about the network. We assume the two firewalls and other host-based security mechanisms (e.g., personal firewalls or iptables) together enforce the connectivity described inside the connectivity table shown in the figure. External users (including attackers) are represented with host $h0$, and the most critical asset is assumed to be the Xen database server ($h4$), which may be accessed through the three-tier architecture [12] involving hosts $h1$, $h2$, and $h3$. We assume the network is free of any known vulnerabilities, except for an unpatchable vulnerability on the application server running SecurityCenter 5.5 (which cannot be changed due to functionality requirements), and another one on the database server running MySQL 5.7 which may be changed to MSQL 2012 or PostgreSQL 9. For simplicity, we exclude exploits and conditions that involve firewalls in this example.

To measure the network's resilience against zero-day attacks, we apply the *h safety metric* [9]. This metric counts how many distinct services must be compromised using unknown vulnerabilities before an attacker may compromise the critical asset (i.e., the number of distinct services along the shortest path) while also taking into consideration the potentially uneven distribution of distinct services along the shortest path [13, 14] (e.g., a path consisting of three *http* and one *Xen* would be considered slightly "shorter", or less secure, than a path consisting of two *http* and two *Xen* services, although both paths have the same number of resource instances and resource types).

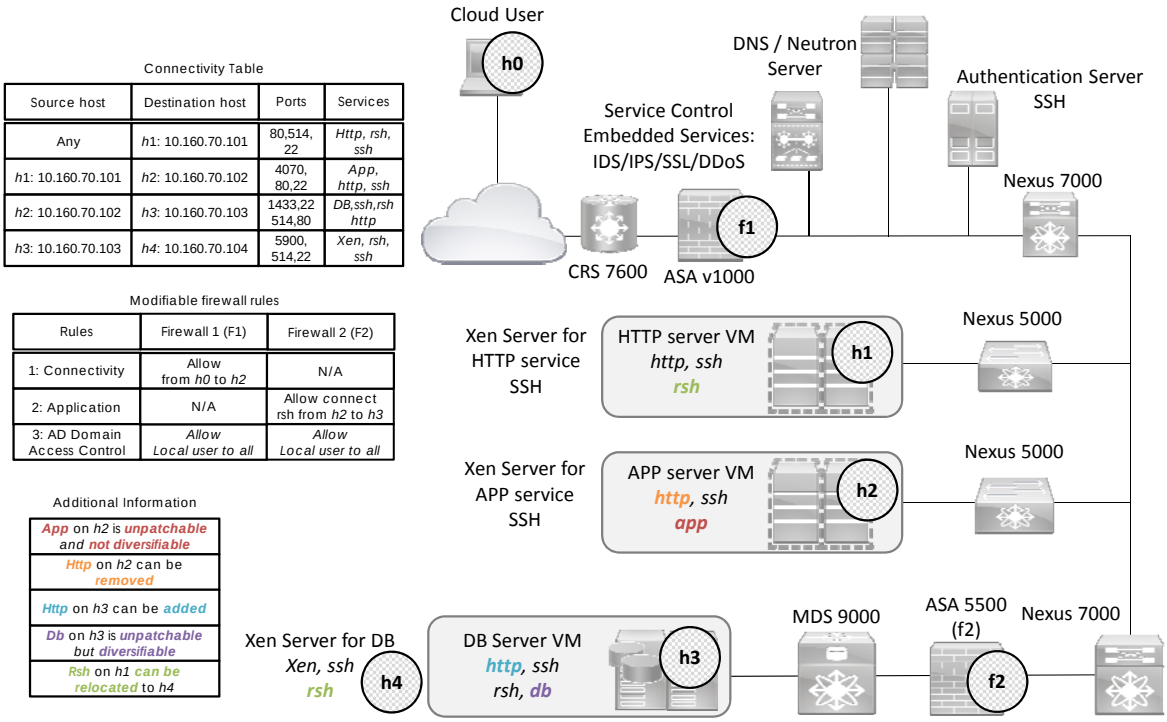

Fig. 1. An Example Network.

For hardening options, we consider the following options: *i*) adding new resources, *ii*) removing existing resources, *iii*) relocating existing resources from one host to another, *iv*) changing the firewall or access rules, and *v*) changing the service types through service diversification. More specifically,

- We assume the administrator may enable or disable firewall rules on both the firewall ASA v1000 ($f1$) and on the firewall ASA 5500 ($f2$).
- On $f1$ the administrator has a rule that allows the connection from the cloud user ($h0$) to the app VM ($h2$);
- The administrator also has the option to allow local user access to the web server VM ($h1$) and $h2$.
- The firewall $f2$ has a rule where he allows the rsh connection on the database server VM ($h3$) from $h2$, as well as local user access to $h3$ and the Xen server ($h4$).
- The administrator has the possibility to remove the $http$ service on $h2$ if he/she would want to stop web access to the application server VM; he/she also has the option to add the $http$ service on $h3$ to administer the database via a web interface;
- The firewall $f2$ has a rule to allow the rsh connection on the database server VM ($h3$) from $h2$, as well as local user access to $h3$ and the Xen server ($h4$).
- The administrator also has the option to relocate the rsh service from $h1$ to $h4$ to execute shell commands on the Xen Server holding the database VM.

Based on above assumptions, the administrator needs to carefully analyze which of those options, if any, will help to make the network more resilient against zero-day attacks. The administrator must also consider the causal relationships between network resources (e.g., some resources can only be reached as a result of compromising other resources), the dependency between hardening options, and finally the different costs associated with different types of hardening options (e.g., the cost can vary significantly between adding, removing or relocating a resource, and also for diversifying services and changing firewall or access rules). Specifically,

- How would adding a new resource, removing an existing one, or relocating a resource from one host to another, impact the overall network resilience against unknown and unpatchable vulnerabilities?
- How would using different instances of a service (diversifying) help with the network resilience?
- How would enabling or disabling predefined firewall rules help hardening the network?
- How would the fact that some services are not patchable (whether or not they can be diversified) impact the efforts to secure the network?
- How to make sure that the options that are chosen respect predefined costs?

Clearly, even with such a small scale network, to answer those questions through manual efforts or experiences would obviously be a tedious and error-prone task and thus demands a systematic and automated approach, which is the subject matter of this paper.

2. Model

We first define our model to capture network services and their relationships; we then present the heterogeneous hardening control, our cost model and discussions on how to estimate costs, followed by an analysis of the metrics that we will be using, and the optimization problem formulation.

2.1. Extended Resource Graph

The first challenge is to model different resources, such as services (e.g., Web servers) that can be remotely accessed over the network, different instances of each service (e.g., Apache and IIS), and the causal relationships existing among resources (e.g., a host is only reachable after an attacker gains a privilege to another host). This challenge applies to both unpatchable and unknown vulnerabilities. An additional challenge is how to model the addition of new resources into the network, the removal of existing ones, and the relocation of predefined resources within the network, as well as considering any potential dependency among different options. Finally, there is also the added complexity of considering predefined firewall rules which may affect initially satisfied conditions.

To address these challenges, we adopt the concept of *Extended Resource Graph* [8, 9], which is syntactically equivalent to attack graphs, but models network services instead of known vulnerabilities [13, 14]. This graph introduces the notion of *Service Instance* to indicate which instance (e.g., Apache) of a particular service (e.g., Web server) is being used on a host. Like the original extended resource graph, we only consider services that can be remotely accessed. The extended resource graph of the running example is shown in Figure 2 and detailed below.

In Figure 2, each pair shown in a rectangle is a security-related condition. If the condition is a privilege, it is represented as $\langle privilege, host \rangle$; if it is connectivity, it is represented as $\langle source, destination \rangle$.

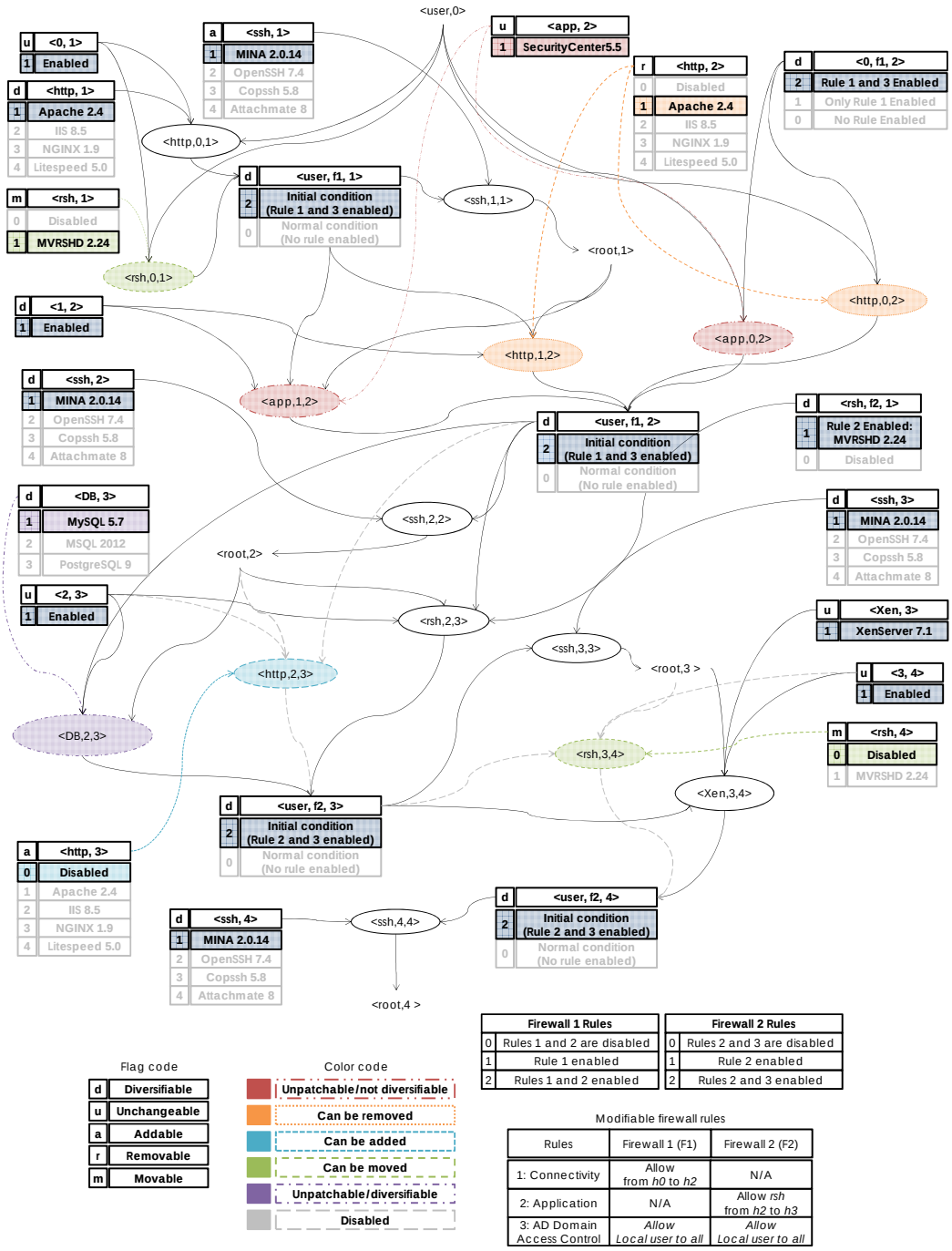

Fig. 2. The extended resource graph of our running example.

If a firewall affects a security-related condition, it is represented as $\langle privilege, firewall, host \rangle$ or as

$\langle source, firewall, destination \rangle$. Each one of the rows below the rectangle indicate different hardening options available for that condition. The option currently in use is indicated by the highlighted integer and other potential instances are in a lighter text (e.g., 0 means disabled; in the case of service diversification for the *http* service, 1 means Apache, and 2 means IIS, etc.). For the conditions modifiable by a firewall rule, the rows below the rectangle indicate the firewall rules that affect it.

Each exploit node (oval) is a tuple that consists of a service running on a destination host, the source host, and the destination host (e.g., the tuple $\langle http, 1, 2 \rangle$ indicates a potential zero-day vulnerability in the *http* service on host 2, which is exploitable from host 1). If the exploit is diversifiable, it is represented by the texture linked to a condition with *d* flag (e.g., $\langle DB, 2, 3 \rangle$); if it is unpatchable, it is represented by the texture linked to a condition with *u* flag (e.g., $\langle app, 1, 2 \rangle$); if an exploit is a result of a service that can be added to a host, then it is represented by the texture linked to a condition with *a* flag (e.g., $\langle http, 2, 3 \rangle$); if it is a result of a service that can be removed from a host, it is represented by the texture linked to a condition with *r* flag (e.g., $\langle http, 1, 2 \rangle$); if the exploit comes from a service that can be relocated from one host to another, then it is represented by the texture linked to a condition with *m* flag (e.g., $\langle rsh, 3, 4 \rangle$). These different types of exploits will contribute to the calculation of the security metric value as detailed later. The self-explanatory edges point from preconditions to an exploit (e.g., from $\langle 0, 1 \rangle$ and $\langle http, 1 \rangle$ to $\langle http, 0, 1 \rangle$), and from the exploit to its post-conditions (e.g., from $\langle http, 0, 1 \rangle$ to $\langle user, 1 \rangle$).

We make three design choices here. First, we associate the service instance concept as a property (label) of a condition (e.g., $\langle http, 1 \rangle$), instead of an exploit (as in our previous work [8]). The reason is an administrator only has control over initial conditions [15]. This label can then be inherited by the corresponding exploits. We will use this label to specify which service instance of a particular service is currently chosen. Second, as with the service instance, we add an additional condition property, called a service flag, as a label, to specify if that condition or service is unchangeable (*u*), diversifiable (*d*), or if this condition/service can be relocated (*m*) to a different host, can be added (*a*) if it was previously not present on the network, or if it can be removed (*r*) completely from the network. Finally, while some conditions indicate the involved firewall rules, the actual label values that they will take will depend on the number of predefined modifiable rules in the firewall itself. Therefore, for each firewall, instead of modeling service instances, we model the number of modifiable firewall rules that can be enabled.

The service flag has an important implication in our extended resource graph when considering the removal of existing services, the addition of services not initially present, and the relocation of services from one host to another. This flag is used to validate if a condition (and by extension, an exploit) is to be considered when optimizing the security of a network. For example, if a service that is currently enabled on the network, is removed (*r* service flag), then this service and the exploits associated to it, will not form part of the topology used to calculate the hardening metric. We further discuss this in Section 3.3. Additionaly, because our extended resource graph models services that may not be present (due to the addition, removal, or relocation of said services), it will contain a small increase in the number of additional conditions and exploits. We believe this is acceptable if we consider that an administrator typically only deals with a limited number of possibilities to add, remove, or relocate resources. For the case of firewall rules, this would help to avoid the need for introducing new conditions and exploits into the extended service graph when firewall rules are to be disabled and hence we may work with a fixed structure of the extended service graph. Definitions 1 to 4 formally introduce these concepts.

Definition 1 (Service Pool and Service Instance)**.** *Denote S the set of all services and Z the set of integers, for each service $s \in S$, the function $sp(.) : S \rightarrow \mathsf{Z}$ gives the service pool of s which represents all available instances of that service.*

Definition 2 (Service Flag)**.** *Denote S the set of all services and $\mathsf{D} = \{d, u, r, a, m\}$ the set of flag values, for each service $s \in S$, the function $d(.) : S \rightarrow \mathsf{D}$ gives the service flag of s.*

Definition 3 (Firewall Rule Pool and Firewall Rule)**.** *Denote F the set of all firewalls and Z the set of integers, for each firewall $f \in F$, the function $r(.) : F \rightarrow \mathsf{Z}$ gives the firewall rule pool of f which represents all modifiable firewall rules of that firewall.*

Definition 4 (Extended Resource Graph)**.** *Given a network composed of*

- *a set of hosts H,*
- *a set of services S, with the service mapping $serv(.) : H \rightarrow 2^S$, and service flag $d(.) : s \rightarrow \mathsf{d}$,*
- *the collection of service pools $SP = \{sp(s) \mid s \in S\}$,*
- *the collection of firewall rules $FR = \{r(f) \mid f \in F\}$,*
- *a set of firewalls F, with the rule mapping $r(.) : F \rightarrow \mid FR \mid$,*
- *and the labeling function $v(.) = v_f(.) \cup v_c(.)$ where $v_f(.) : f \rightarrow F$ and $v_c(.) : C \rightarrow SP$.*

Let E be the set of exploits $\{\langle s, h_s, h_d\rangle \mid h_s \in H, h_d \in H, s \in serv(h_d)\}$, $R_r \subseteq C \times E$ and $R_i \subseteq E \times C$ be the collection of pre and post-conditions in C, $R_F \subseteq F \times C$ be the relationship between firewall rules and conditions, $R_B \subseteq C \times \mathsf{D}$ be the relationship between conditions with their service flag, We call the labeled directed graph, $\langle G(E \cup C, R_r \cup R_i \cup R_F \cup R_B), v\rangle$ the extended resource graph.

2.2. Heterogeneous Hardening Control

We employ the notion of *heterogeneous hardening control* as a model to account for all hardening options in a network where we represent each initial condition as an optimization variable. We formulate the heterogeneous hardening control vectors using those variables as follows. The number of optimization variables present in a network will depend on the number of initial conditions that are affected by one or more hardening options (many exploits may share the same service instance, and hence the optimization variable). Since we only consider remotely accessible services in the extended resource graph model, we would expect in practice the number of optimization variables to grow linearly in the size of the network (i.e., the number of hosts). We will further evaluate and discuss the scalability of our solution in Section 4.

Definition 5 (Network Hardening and Hardening Option)**.** *Given an extended resource graph $\langle G, v\rangle$, the collection of any instance of service or rule in $SP \cup F$, is called a hardening option, $\forall sp(s) \in SP$ and $\forall f \in F$. The process of optimally selecting these options to maximize security with respect to given cost constraints is called network hardening.*

Definition 6 (Optimization Variable and Heterogeneous Hardening Control)**.** *Given an extended resource graph $\langle G, v\rangle$, $\forall c \in C$ and $\forall f \in F$, $v(c)$ and $v(f)$ are optimization variables. A hardening control vector is the integer valued vector $\vec{V} = (v(c_1), v(c_2), ..., v(c_{|C|}) \cup (v(f_1), v(f_2), ..., v(f_{|F|})$*

Changing the value of an optimization variable has an associated *hardening cost* and the collection of such costs is given in a *hardening cost matrix* in a self-explanatory manner. Like in most existing works (e.g., [6, 7, 16]), we believe an administrator can estimate the hardening costs based on monetary, temporal, and scalability criteria like *i*) installation cost, *ii*) operation cost, *iii*) training cost, *iv*) system

downtime cost and, *v*) incompatibility cost. Taking this criteria as a point of reference, subsection 2.3 provides a guideline on how our hardening costs are estimated. We define the hardening cost, hardening cost matrix, and the total hardening cost as follows.

Definition 7 (Hardening Cost)**.** *Given $s \in S$ and $sp(s)$, and given $f \in F$ and $r(f)$, the cost to change from one specific hardening option to another is defined as the hardening cost.*

Definition 8 (Hardening Cost Matrix)**.** *The hardening cost matrix (HCM) is defined as a matrix in which the i^{th} row and i^{th} column both represent the i^{th} hardening option. The collection of all hardening costs for all hardening options are given as a hardening cost matrix HCM. For the different hardening options, the element at i^{th} row and j^{th} column is the given cost of changing the i^{th} hardening option to the j^{th} hardening option.*

Definition 9 (Total Hardening Cost)**.** *Let $v_s(c_i)$ be the service associated with the optimization variable $v(c_i)$ and $\vec{V}_{c0}$ the initial service instance values for each of the conditions in the network. Let $v_f(f_i)$ be the firewall associated with the optimization variable $v(f_i)$ and $\vec{V}_{f0}$ the initial firewall rule set values for each of the firewalls in the network. The total hardening cost, Q_h, given by the heterogeneous hardening vector $\vec{V}$ is obtained by*

$$Q_h = \sum_{i=1}^{|C|} HCM_{v_s(c_i)}(\vec{V}_{c0}(i), \vec{V}_c(i)) + \sum_{i=1}^{|F|} HCM_{v_f(f_i)}(\vec{V}_{f0}(i), \vec{V}_f(i))$$

The above definition of hardening cost between each pair of service instances has three advantages. First, in practice we can easily imagine cases where the cost is not symmetric, i.e., changing one service instance to another (e.g., from Apache to IIS) carries a cost that is not necessarily the same as the cost of changing it back (from IIS to Apache). Our approach of using a collection of two-dimensional matrices allows us to account for cases like this. Additionally, by considering instance 0, it provides us the advantage to model disabling (or removing) a service as a special case of service diversification if the hardening option allows it. Second, our cost model concept can be used to specify many different types of cost constraints which can be added to the base formula as will be discussed in the next section. For example, an administrator might have configured service groups to group related services together (e.g., SIP, RTP, and RTSP) and a change in one service might also affect the others. In other words, the way our costs are calculated can be derived as a function of the status of other services or conditions. Finally, another advantage of our definition is the inclusion of negative costs. While at a first glance this concept may not seem self-evident, the inclusion of negative cost values can be interpreted as an incentive to opt for a specific option. For example, an administrator may want to phase out the use of *rsh* in favor of a more secure protocol like *ssh*. This can be easily represented by negative cost values within our two-dimensional matrix which effectively subtracts costs from the total hardening cost.

2.3. Cost Estimation

Our main assumption for the values of these cost is that they are assigned by security experts or network administrators. While our cost model does consider that individual hardening costs can depend on factors such as downtime costs or the status of other services, we believe a baseline cost can be first estimated to better inform and justify the hardening costs. Therefore, we make use of Gartner's

2003's *Total Cost of Ownership* (TCO) analysis report [17] and Emerson-Ponemon Institute's 2016's analysis report on the cost of data center outages [18] to establish a more realistic cost estimate which a company might incur when selecting one or more hardening options. Based on Gartner's report, a company's costs can be divided into two main categories: base costs and ongoing costs. The base costs are mostly associated with planning costs that include, but are not limited to, server/software acquisition and installation costs. The ongoing costs are the costs of keeping a server, or a service, up and running. The ongoing costs are further divided into direct and indirect costs which include operational costs and downtime costs, respectively. A more detail list of different costs and how these may be associated with different hardening options is given in Table 1.

Next, we apply the TCO's ongoing costs as a reference point for estimating the hardening costs. It can be observed that direct costs (e.g., support costs, changes in upgrade costs or production control costs), as well as indirect costs (e.g., downtime cost), are costs that need to be considered when implementing a network hardening solution. Additionally, since the ongoing costs alone will incur on average around 85% of the total costs of ownership, it is reasonable estimation to use the ongoing costs as the baseline reference for hardening costs. Furthermore, since our hardening options (diversifying, adding, removing, and relocating services, as well as modifying firewall rules) mainly involve existing service instances (inside the service pool), we only consider ongoing costs for hardening and assume the base costs (acquisition and installation) are already applied before the hardening process. As seen in Table 1, because the indirect costs make up at least 50% of the total ongoing costs, we can further narrow down the base of the hardening costs as being based on the indirect costs, in particular the system downtime cost. In Emerson-Ponemon's 2016's [18] report on the downtime costs of a data center, the impact that downtime costs can have on a network is highlighted. Based on their industry benchmarks and insights, our hardening costs can be estimated by system administrators in making decisions about network hardening. Although the hardening cost can be defined based on system downtimes in more rigorous ways, we will adopt the simple estimation method given in [19] as follows:

$$\bar{q}_{hr(dt)} = \bar{\bar{E}}_{q(hr)} \times \breve{E}_{af} + \bar{R}_{hr} \times \breve{R}_{af}$$

Where

- $\bar{q}_{hr(dt)}$ is the estimated average cost of one hour of downtime,
- $\bar{\bar{E}}_{q(hr)}$ is the estimated average employee costs per hour (i.e., the total salaries and benefits of employees per week divided by the average number of working hours, or the total revenue per week divided by average number of open hours).
- $\breve{E}_{af}$ is the estimated fraction of employees affected by the downtime,
- $\bar{R}_{hr}$ is the estimated average revenue per hour, and
- $\breve{R}_{af}$ is the estimated fraction revenue affected by the downtime.

Because the *Fraction Employees Affected by Outage* and the *Fraction Revenue Affected by Outage* are not values readily available, an educated guesses based on about plausible range should be considered.

To better illustrate this, we take as an example the reported 2015 revenue for Amazon. This revenue was reported at approximately $107 billion [20] with approximately 250,000 employees for that same year [21]. From this information, the approximate revenue per hour (considering that Amazon is a 24/7 business) is about $12M. Assuming an average annual salary of an employee being around $100,000 then we can have approximate yearly expenditure of $25B on salaries or approximately $471M per

Gartner's TCO's base costs			D	A	R	M	F_c	F_s	F_a
Planning costs (Approx. 15% of TCO)	Acquisition costs	Cost of Hardware							
		Cost of OS	x			x			
		Cost of Application	x	x	x	x			
	Installation costs	Hardware setup							
		OS installation	x			x			
		Application installation	x	x	x	x			
Gartner's TCO's ongoing costs			D	A	R	M	F_c	F_s	F_a
Indirect costs (Approx. 50% of TCO)	Downtime costs	Planned downtime	x	x	x	x	x	x	x
		Unplanned downtime	x	x	x	x	x	x	x
	End-user costs	Casual learning							
		Peer and self support							
Direct costs (Approx. 35% of TCO)	Operational costs	Communication fees	x	x	x	x	x	x	x
		Leased asset fees	x	x		x			
		IS commodity expenditures							
		Insurance							
	Support costs	Help desk					x	x	x
		Request and problem management					x	x	x
		Casual learning	x	x	x	x	x	x	x
		Training	x	x	x	x	x	x	x
		Operating costs	x	x	x	x	x	x	x
	Changes in upgrade costs	Change planning	x	x	x	x	x	x	x
		Asset management	x	x	x	x	x	x	x
		Product evaluation and testing	x	x	x	x	x	x	x
		Product procurement and implementation	x	x	x	x	x	x	x
		User administration	x	x	x	x	x	x	x
	Security management and failure control costs	Security and virus protection							
		LAN/WAN troubleshooting/repair	x				x	x	x
		Disaster planning and recovery							
		Hardware maintenance fees							
	Monitoring costs	Event management					x	x	x
		Performance management	x	x	x	x			
		Physical site management	x	x	x	x			
	Production control costs	Application management	x	x	x	x	x	x	x
		Storage management	x	x	x	x	x	x	x
		Traffic management	x	x	x	x	x	x	x

Table 1

The association between Gartner's TCO costs and hardening options. D: Diversifying services; A: Adding a new service; R: Removing an existing service; M: Relocating a service; F_c: Connectivity based firewall rule; F_s: Service based firewall rule; F_a: Access control based firewall rule.

week for all staff. If we consider that an Amazon employee works on average 50 hours per week, then the average expenditure per salary per hour is around $9.4M per hour. We assume that if an outage for the ftp services affects 84% of the revenue, that would equate to a loss of around $10M. If it affects 85% of the employees, then that would equate to approximately $8M. Thus, the total revenue loss for an outage would be valued at approximately $\bar{q}_{hr(dt)} = \$9.4M \times 0.85 + \$12M \times 0.84 = \$18M$. This value can be used as a base monetary reference to define the costs to diversify the ftp service.

The above discussions only provide a starting point for both network administrators and security to estimate hardening costs, and those can certainly be refined, e.g., by considering outage prevention mechanisms which may reduce the downtime.

2.4. Hardening Metric

The security metric used in this paper is an extension of the k-zero-day safety metric [22]. Specifically, our metric is based on the minimum number of distinct resources, excluding those with unpatchable vulnerabilities, on the shortest attack path in the extended resource graph, with the extension for considering the uneven distribution of services along that path [13, 14], as well as the unpatchable services. It is formally defined below.

Definition 10 (*h*-Safety Metric)**.** *Given an extended resource graph $\langle G(E \cup C, R_r \cup R_i \cup R_F \cup R_B), v\rangle$ and a critical asset $c_g \in C$; let t be the total number of services, and let p_j be the relative frequency of each resource. For each $c \in C$ and $q \in seq(c)$ (attack path), denote $R(q)$ for $\{s : s \in R$, r appears in q, r is not unpatchable$\}$, we define the network's h-safety metric (where min(.) returns the minimum value in a set) $h = min_{q \in seq(c_g)} r(R(q))$; where $r(R(q))$ is the attack path's effective richness of the services, defined as $r(G) = \frac{1}{\prod_1^n p_i^{p_i}}$ [13]*

In Figure 3, we can see that while both paths have three unique exploits, their associated *h* metric will be different due to the difference in the relative frequency of each exploit. Since *h* is the minimum value between the two, the *h* value between these two paths would be equal to h_2.

2.5. Problem Formulation

As demonstrated in our discussions about the motivating example in Section 1.1, hardening a network with multiple options demands a systematic and automated approach. For any data center or cloud network, to manually conduct the network hardening task is not feasible, as demonstrated earlier by the small network in our motivating example. Applying the *h*-safety metric (or simply the *h* metric) defined in previous section to Figure 3, it is not straightforward to see how changing the number of unique exploits and their relative frequency will affect the overall value of the *h* metric. On the other hand, it is clear that by changing the service instances, modifying firewall rules, or adding, removing or relocating services, the network's *h* metric value can likely be improved. This motivates for a systematic and automated solution. Consequently, we consider the concrete network hardening problem of maximizing the *h* metric value by optimally changing the hardening options, while respecting the available budget in terms of given cost constraints. In the following, we formally define this as an optimization problem.

Problem 1 (*h*-Optimization Problem)**.** *Given an extended resource graph $\langle G, v\rangle$, find a heterogeneous hardening control vector $\vec{V}$ which maximizes $min(h(\langle G(\vec{V}), v\rangle))$ subject to the constraint $Q \leqslant B$, where B is the available budget and Q is the total hardening cost as given in Definition 9.*

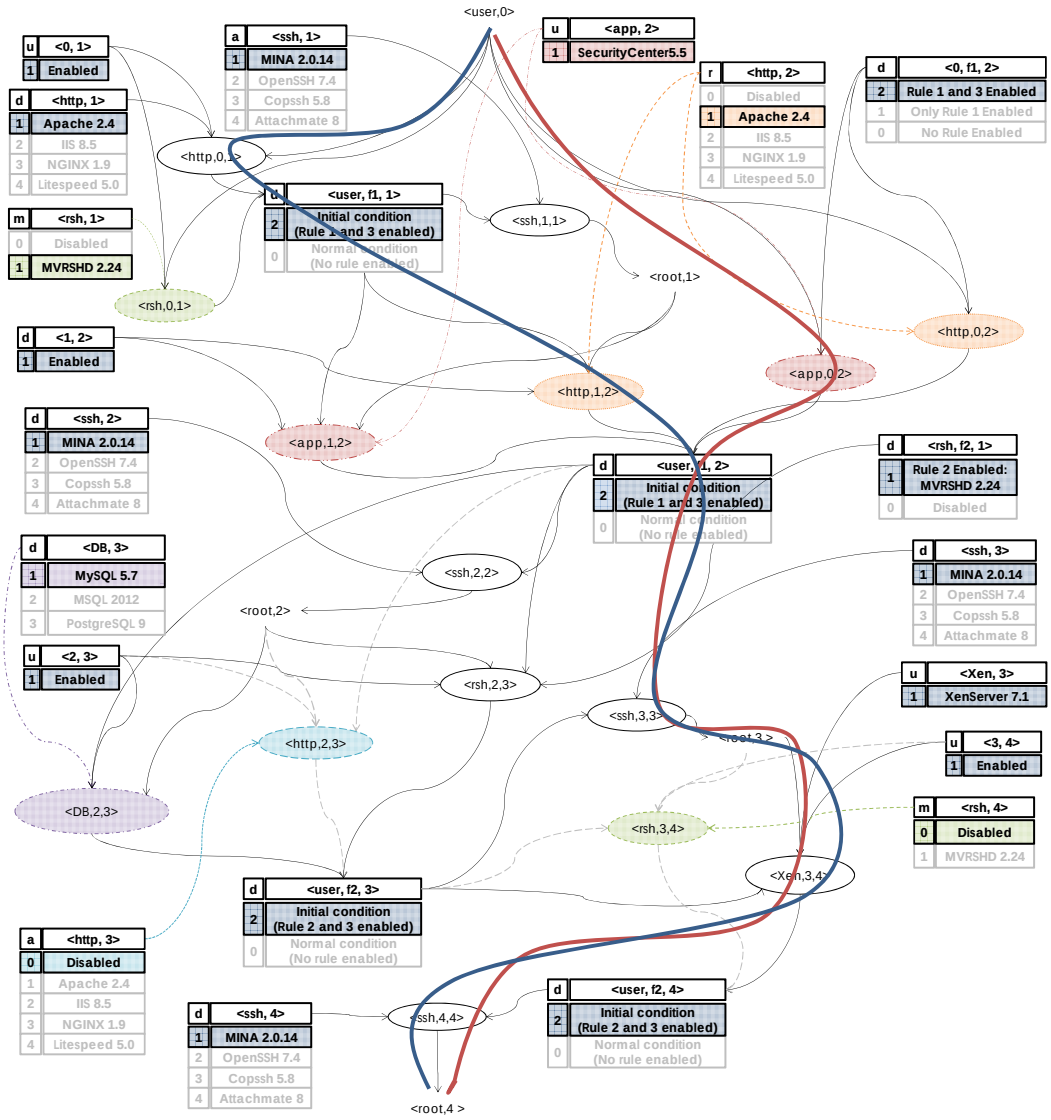

Fig. 3. *h* metric example.

Since our problem formulation is based on an extended version of the resource graph, which is syntactically equivalent to attack graphs, many existing tools developed for the latter (e.g., MULVAL [23] or CAULDRON [24]) may be easily extended to generate extended resource graphs. Additionally, our problem formulation assumes a very general model of budget B and cost Q, which allows us to account for different types of budgets and cost constraints that an administrator might encounter in practice, as will be demonstrated in the following section.

Finally, the problem is NP-hard since the sub-problem of finding the shortest paths (within the objective function) in resource graphs is already intractable by the well know results in attack graphs. Specifically, due to the common syntax between resource graphs and attack graphs, calculating $min(h(\langle G(\vec{V}), v\rangle))$ is equivalent to finding the shortest path in attack graphs, which is known to be NP-complete [22, 31]. By fixing all optimization variables (i.e., assuming only one hardening option for each variable), the problem given in Definition 1 can be reduced to simply finding $min(h(\langle G(\vec{V}), v\rangle))$, which shows the problem to be NP-hard.

3. Methodology

This section details the optimization and heuristic algorithms used for solving the formulated hardening problem and describes a few case studies.

3.1. Optimization Algorithm

Our first task is to select an optimization algorithm that is suitable for solving the hardening problem. Generally, there exist mainly two types of optimization algorithms: Conventional methods or exact algorithms and meta-heuristic approaches [25]. Exact (gradient-based) algorithms, such as Lagrangian relaxation and branch and bound, consider all the solution spaces to give a global solution [26]. However, it is well known that most of these methods require to satisfy mathematical properties like convexity or differentiability [27], which are not applicable to our problem. The problem we want to solve includes different if-then-else constructs to account for the different hardening techniques used (as well as the cost constraints), and thus, an algorithm that allows to insert this construct is necessary. Meta-heuristic approaches, such as genetic algorithm, particle swarm optimization, imperialist competitive algorithm, etc., consider some parts of the solution space to reach a global optimum or near-solution optima, which provides an advantage when dealing with discrete variable spaces [26], which closely match the requirement of our hardening problem. They provide a simple and robust search method and optimization technique. Because the problem we want to solve uses variables that are defined as discrete, a meta-heuristic approach is needed.

In particular, the genetic algorithm (GA) provides a simple and clever way to encode candidate solutions to the problem [28]. One of the main advantages is that we do not have to worry about explicit mathematical definitions (which allow for a quick implementation). For our automated optimization approach, we chose GA, which is popular among the different evolutionary algorithms due to certain characteristics: It requires little information to search effectively in a large search space in contrast to other optimization methods (e.g., the mixed integer programming [26]); and that it uses both crossover and mutation operators which makes its population more diverse and thus more immune to be trapped in some local optima. While our work was inspired by [6], our main difference and contribution is that we focus on multi-option hardening and not just on disabling services.

The extended resource graph is the input to our automated optimization algorithm where the function to be optimized (fitness function) is the h metric of the extended resource graph. There are two important points to consider when optimizing the h metric function on the extended resource graph for each generation of the GA: *i*) the graph's service instance labels for the chosen hardening option will dynamically change. *ii*) the actual shape of the graph will dynamically change due to the firewall rules and the *service flag* labels which account for the added, removed, and relocated services. This in turn will change the value of h, since the shortest path may have changed with each successive generation of GA and the change in the hardening options (as well with the adding, removal, and relocation of services) will enable or disable certain conditions, vulnerabilities, and paths. Our optimization tool takes this into consideration. Additionally, if there are more than one shortest path that provides the optimized h, our optimization tool gives priority to the paths by considering the uneven distribution and relative frequency of resources in that path, thus addressing one of the limitations that was present in [8] where no priority was provided.

The constraints are defined as a set of inequalities in the form of $q \leqslant b$, where q represents one or more constraint conditions and b represents one or more budgets. These constraint conditions can be overall constraints (e.g., the total hardening cost Q_h) or specific constraints to address certain requirements or priorities while implementing the heterogeneous hardening options (e.g., the cost to diversify $http$ services should be less than 80% of the cost to diversify ssh; if $http$ is added into $h2$, ssh and app also incur a cost; the relocation of rsh could mean a negative cost (credit) and thus incentivizing its relocation; etc.) Those constraints are specified using the diversity control matrix.

The number of independent variables used by the GA (genes) are the optimization variables given by the extended resource graph. For our network hardening problem, the GA will be dealing with integer variables representing the selection of a hardening option. Because $v(.)$ (optimization variable) is defined as an integer, the optimization variables need to be given a minimum value and a maximum value. This range is determined by the number of instances provided in the service pool of each service and firewall rule pool of each firewall. The initial service instance for each of the services and the initial set of firewall rules are given by the extended resource graph while the final heterogeneous hardening control vector $\vec{V}$ is obtained after running the GA.

3.2. Use Cases

In the following, we demonstrate potential use cases of our method with varying cost constraints and hardening options. For these use cases, the population size defined for our tool is set to be at least the value of optimization variables (more details will be provided in the coming section). This way we ensure the individuals in each population span the search space. We ensure the population diversity by testing with different settings in genetic operations (like crossover and mutation). For all the use cases, we have used the following algorithm parameters: population size = 100, number of generations = 150, crossover probability = 0.8, and mutation probability = 0.2.

Use Case A: $Q_h \leqslant$ \$500k with firewall rule constraints and the possibility to add, remove, relocate services. We start with the simple case of one overall budget constraint (refer to Figure 4). There are 13 different services-based optimization variables and two firewall-based optimization variables. If we allow the firewall rules to be modified, and if we consider that some services can be added ($http$ on $h3$), removed ($http$ on $h2$) or relocated (rsh from $h1$ to $h4$), we can see some interesting results. While the algorithm does not enable $< http, 3 >$, it does disable $< http, 2 >$ and relocates rsh to $h4$ (that is, it disables $< rsh, 1 >$ and enables $< rsh, 4 >$). It is worth noting that the cost function that governs the

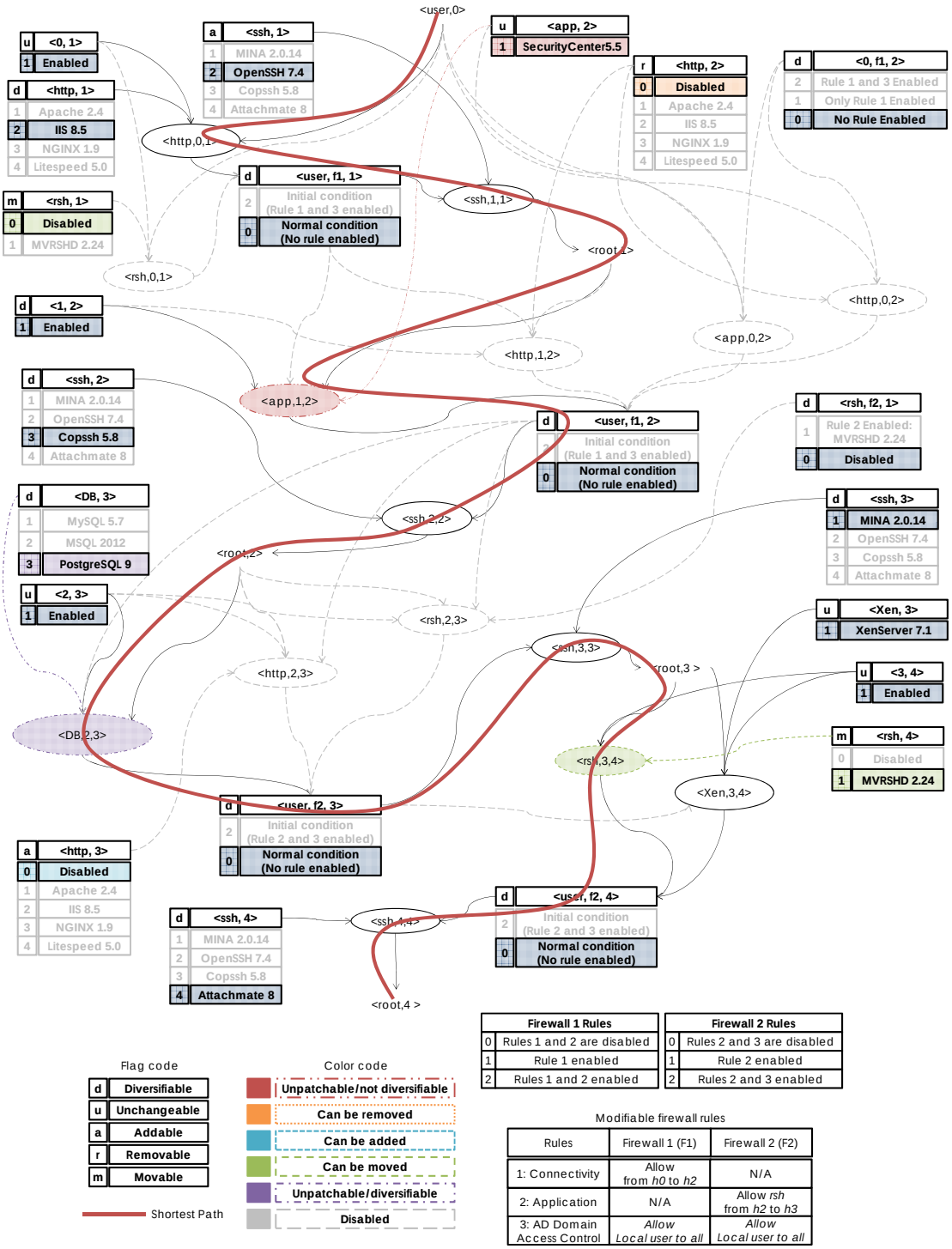

Fig. 4. Use Case A: Effect of modifiable hardening options and budget constraints.

relocation of *rsh* is such that it reduces the overall spending (this can be interpreted as an incentive to relocate the service).

The solution provided by the GA is a *h* metric of 8. This total hardening cost satisfies both the overall budget constraints. We can see that the hardening options enforced by the firewall rules and the relocation of services in our optimization tool can affect the optimization. Nevertheless, additional budget constraints might not allow achieving the maximum possible *h* value.

h metric value based on hardening options applied					
Hardening Option	Use Case A's *h* value	One unpatchable vuln.	Two unpatchable vuln.	Three unpatchable vuln.	Use case B's *h* value
No hardening option applied	4	4	4	4	4
Only Diversity Applied	4	4	4	4	4
Only Firewall Rules Apllied	2.828	2.828	2.828	2.828	2.828
Only Adding, Removing, Relocating Services	4	4	4	4	4
All hardening options applied	8	8	7.583	6.295	4.332

Table 2

h metric value for different hardening options for when no unpatchable vulnerabilities are present (Use Case A), up to 4 unpatchable vulnerabilities (Use Case B).

Use Case B: $Q_h \leqslant$ \$500k with a critical service with an unpatched vulnerability. While Use Case A shows how enabling or disabling predefined firewall rules can affect the *h* metric optimization, when considering the effects of unpatchable vulnerabilities the *h* metric value will change. This use case models such a scenario by assigning a restriction for the *ssh* services not to be diversified or disabled.

In Figure 5, we can see that the *ssh* service is highlighted to represent the fact that it cannot be patched. The solution provided by the GA is *h*=4.332. While the increase is less than when the *ssh* service can be diversified, we can still have an increase in the *h* metric even with unpatchable vulnerabilities on the network.

It is interesting to note, based on the results shown on Table 2, that not all hardening options can help increase the network's resilience against zero-day attacks. Applying one set of options might have no effect at all and some options can even reduce the resilience of the network, as seen in values of the *h* metric when only firewall rules are applied. It is worth noting that our proposed solution is capable of increasing the resilience of a network even in the presence of unpatchable vulnerabilities.

As seen from the above use cases, our model and problem formulation makes it relatively straightforward to apply any standard optimization techniques, such as the GA, to optimize the *h* metric through combining different network hardening options while dealing with unpatchable vulnerabilities and respecting given cost constraints.

3.3. Heuristic Algorithm

All the test cases described above rely on two main assumptions *i*) that the firewall rules and relocation of service will enable or disable conditions and exploits *ii*) that all the attack paths are readily available. We will design an algorithm specifically for the special cases where conditions and exploits are removed from the extended resource graph (as a result from disabling or relocating services). As to the second case, due to the well-known complexity that resource graphs have inherited from attack graphs due to

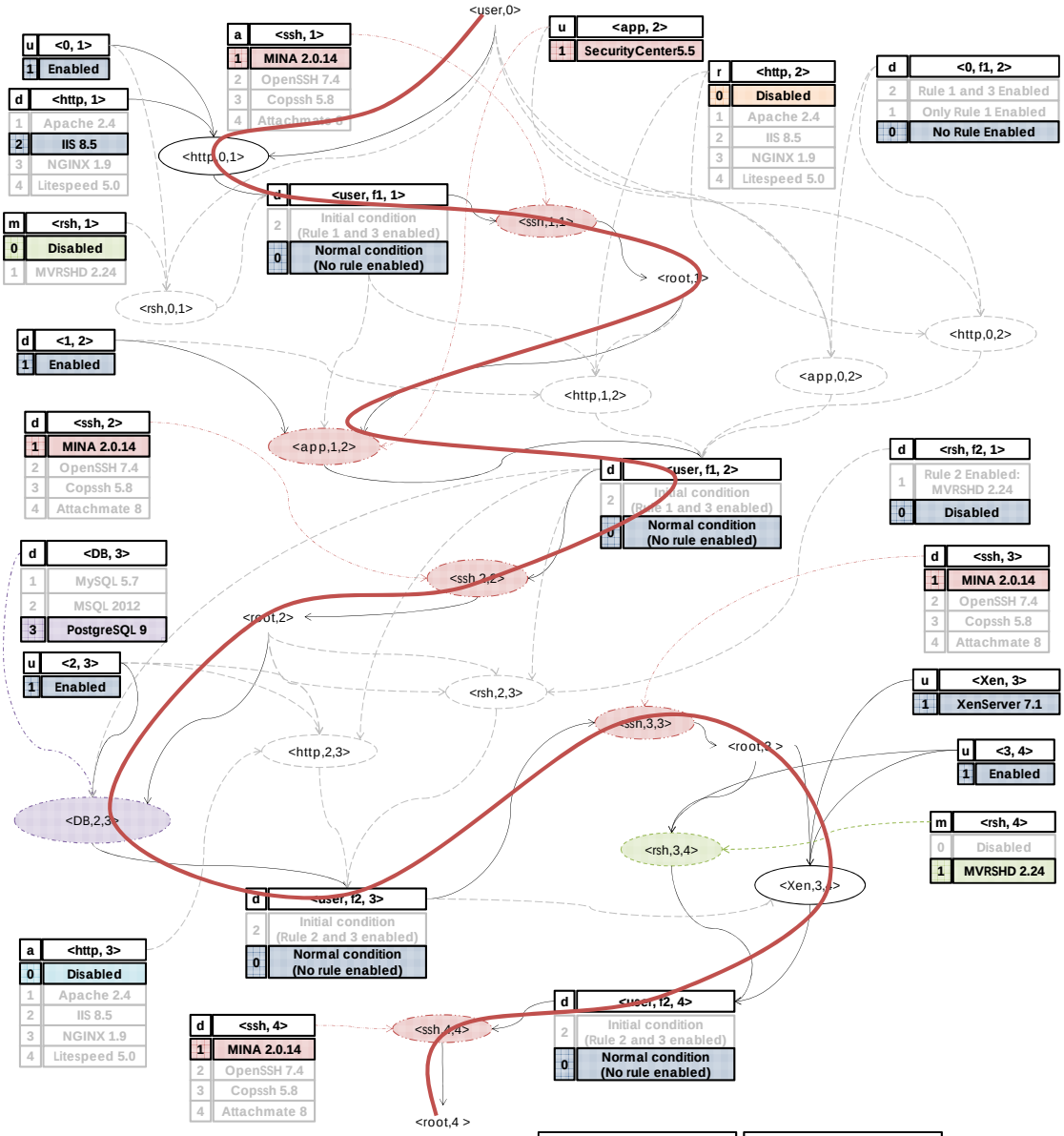

Fig. 5. Use Case B: Effect of having an unpatchable vulnerability in the network.

their common syntax [13, 14], it is usually computationally infeasible to enumerate all the available attack paths in a resource graph for large networks. Therefore, we present a modified version of the heuristic algorithm [8, 9] to reduce the search complexity when calculating and optimizing the *h* metric by only storing the *m*-shortest paths at each step, and which is shown in Figure 7.

Procedure *Processed_Topological_Sort*
Input: Extended resource graph $\langle G, v\rangle$, hardening control vector H
Output: $vlist_p$
Method:
1. **Let** *vlist* be any topological sort of G
2. **While** all $c_f \in C_I$ and $(\forall f \in F)((f,c) \in R_F \Rightarrow c_f$ unprocessed)
3. **If** $f \in F$ is enabled
4. **Let** $\tau(f) \leftarrow c_f$
5. **Mark** c_f as processed
6. **While** all $c_s, c_d \in C_I$ such that $(c_s, c_d) \in R_B \Rightarrow m$ unprocessed
7. **If** $c_s \in C_I$ is enabled and $d(c_d) \in C_I$ is enabled
8. **Let** $\tau(r) \leftarrow c_s, c_d$
9. **Mark** c_s, c_d as processed
10. **Let** $\tau(z) = \tau(f) \cup \tau(r)$
11. **If** $\tau(z)$ is empty
12. $vlist_p = vlist$
13. **Return** $vlist_p$
14. **Else**
15. **While** $e \in E$ (e is not processed) $(\forall c \in \tau(z))\ (c,e) \in R_r$
16. **Let** $\alpha(e) = a_1 \cup a_2 \ldots \cup e : a_i \in \sigma(c_i),\ c_i \in \tau(z)\ 1 \leqslant i \leqslant n$
17. **Let** $\tau(x) \leftarrow e$
18. **Let** $\tau(w) \leftarrow \alpha(e)$
19. **Let** $\alpha(c) = a_1 \cup a_2 \ldots \cup c : a_i \in \sigma(e_i),\ e_i \in \tau(x)\ 1 \leqslant i \leqslant n$
20. **Let** $\tau(y) \leftarrow \alpha(c)$
21. **Let** $\tau(k) = \tau(w) \cup \tau(x) \cup \tau(y) \cup \tau(z)$
22. **Let** $vlist_p = remove(vlist, \tau(k))$
23. **Return** $vlist_p$

Fig. 6. Algorithm for eliminating infeasible conditions and paths

The algorithm in Figure 6, which has linear complexity ($O(N)$), is the one we use to check for the topological changes that the extended resource graph has whenever conditions (and exploits) are removed or relocated. This algorithm starts by topologically sorting the graph (line 1). It then proceeds to go through each one of the firewall rule. If the firewall rule is enabled, it checks which are the conditions that are affected by that firewall rule and stores them on a list, $\tau(f)$, and marks the conditions as processed (lines 2-5). It then goes through each one of the conditions that have a relocation service flag (that is, if the condition can me relocated from one host to another). If a condition with an *m* service flag is enabled on both the source and the destination host, the algorithm stores it on a list, $\tau(r)$, and marks the source and destination conditions as processed (lines 6-9). Both lists are combined into one list, $\tau(z)$ (line 10). If $\tau(z)$ is empty, the algorithm ends by returning the initial topological sort, *vlist* (line 11-13). Otherwise, using the list $\tau(z)$, the algorithm then proceeds to check which are the associated exploits and stores them on a list, $\tau(x)$, as well as the attack paths, $\alpha(e)$ and $\alpha(c)$, and stores them on two other lists, $\tau(w)$ and $\tau(y)$ (lines 14-20). The algorithm then proceeds to combine the lists $\tau(w)$, $\tau(x)$, $\tau(y)$, and $\tau(z)$ into one list, $\tau(k)$ (line 21), which will be used to remove the conditions, the exploits and attack paths from *vlist* using the function *remove*() (line 22), after which, the algorithm return the processed topological sort, $vlist_p$ (line 23).

The algorithm on Figure 7 is similar to the previous one and thus also has an $O(N)$ complexity. This algorithm starts by finding the processed topological sort of the graph, $vlist_p$ (line 1), and proceeds to

Procedure *Heuristic_m-shortest*
Input: Extended resource graph $\langle G, v\rangle$, critical asset c_g, number of paths m, hardening control vector H, processed topological sort, $vlist_p$
Output: $\sigma(c_g)$
Method:
1. **Let** $vlist_p$= Processed_Topological_Sort($\langle G, v\rangle$,H)
2. **While** all $vlist_p$ elements are unprocessed
3. **If** $c \in C_I$ and c is unprocessed
4. **Let** $\sigma(c) = c$
5. **Mark** c as processed
6. **Else if** $e \in E$ (e is not processed) and $(\forall c \in C)((c,e) \in R_r \Rightarrow c$ is processed)
7. **Let** $\{c \in C : (c,e) \in R_r\} = \{c_1, c_2, \ldots, c_n\}$
8. **Let** $\alpha(e) = a_1 \cup a_2 \ldots \cup e : a_i \in \sigma(c_i),\ 1 \leqslant i \leqslant n$
9. **Let** $\alpha'(ov(e)) = a'_1 \cup a'_2 \ldots \cup e : a'_i \vdash a_i,\ 1 \leqslant i \leqslant n$
10. **If** $n > m$
11. **Let** $\sigma(e) = ShortestM(\langle \alpha(e), \mid HeurER(\alpha'[ov(e)]) \mid \rangle, m))$
12. **Else**
13. $\sigma(e) = a_1 \cup a_2 \ldots \cup e : a_i \in \sigma(c_i),\ 1 \leqslant i \leqslant m$
14. **Mark** e as processed
15. **Else** (c s.t. $(e,c) \in R_i$ and c is unprocessed)
16. **If** $(\forall e' \in E)((e',c) \in R_i \Rightarrow e'$ is processed)
17. **Let** $\alpha(c) = \bigcup_{e' \text{ s.t. } (e',c) \in R_i} \sigma(e')$
18. **Let** $\alpha'(c) = \bigcup_{e' \text{ s.t. } (e',c) \in R_i} \sigma(ov(e'))$
19. **If** $length(\alpha(c)) > m$
20. **Let** $\sigma(c) = ShortestM(\langle \alpha(c), \mid HeurER(\alpha'[ov(c)]) \mid \rangle, m))$
21. **Else**
22. **Let** $\sigma(c) = \bigcup_{e' \text{ s.t. } (e',c) \in R_i} \sigma(e')$
23. **Mark** c as processed
24. **Return** $\sigma(c_g)$

Fig. 7. A Heuristic algorithm for calculating m-shortest paths

go through each one of the nodes on the resource graph (initial conditions, exploits, and privileges) looking for the collection of attack paths, as set of exploits $\sigma()$, that reach that particular node. The main loop cycles through each unprocessed node. If a node is an initial condition, the algorithm assumes that the node itself is the only path to it and it marks it as processed (lines 6-8). For each exploit e, all its preconditions are placed in a set (line 10). The collection of attack paths $\alpha(e)$ is constructed from the attack paths of those preconditions (lines 10 and 11). In a similar way, $\sigma'(ov(e))$ is constructed with the function $ov()$ which, aside of using the exploits includes value of element of the diversity control vector that supervises that exploit. If there are more than m paths to that node, the algorithm will use the function $HeurER$ to first look for unique combinations of service and service instance in $\alpha'(ov(e))$ and calculate the effective richness (it calculates the h metric). Then, the algorithm creates a dictionary structure where the key is a path from $\alpha(e)$ and the value is the h metric value given by each one of the respective paths in $\alpha'(ov(e))$. The function $ShortestM()$ selects the top m keys whose values are the smallest and returns the m paths with the minimum value of the h metric (line 13). If there are less than m paths, it will return all the paths (line 15). After this, it marks the node as processed (line 16). The process is similar when going through each one of the intermediate conditions (lines 17-24). Finally, the algorithm returns the collection of m paths that can reach the critical asset c_g. It is worth noting that the algorithm does not make any distinction in if a path has a higher priority over another when they share the same h value.

4. Simulations

In this section, we show simulation results. All simulations are performed using a computer equipped with a 3.0 GHz CPU and 8GB RAM in the Python 2.7.10 environment under Ubuntu 12.04 LTS and MATLAB 2015a's GA toolbox. To generate many resource graphs for simulations, we first construct a small number of seed graphs based on realistic networks and then generate larger graphs from those seed graphs by injecting new hosts and assigning resources in a random but realistic fashion (e.g., the number of pre-conditions of each exploit is varied within a small range since real world exploits usually have a constant number of pre-conditions). For the different hardening options that are implemented through firewall rules, we randomly select 10% of the initial conditions. Additionally, to analyze the effect of unpatchable vulnerabilities, our graphs include randomly assigned unpatchable services. The resource graphs are used as the input for the optimization toolbox where the objective function is to maximize the minimum h value subject to budget constraints.

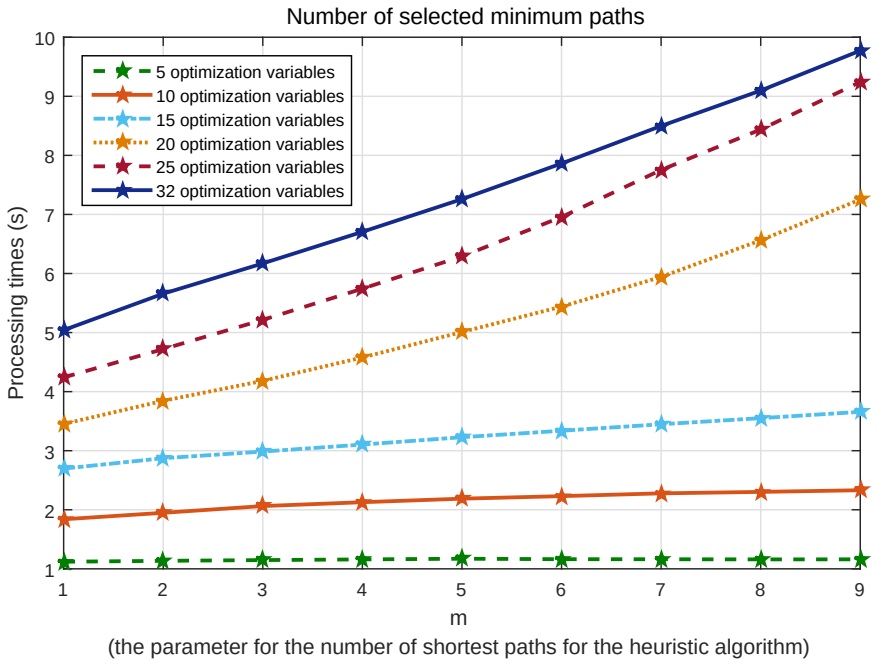

Fig. 8. The processing time.

To determine the genetic operators, we used the hill climbing algorithm. Our simulations showed that, using the GA with a crossover probability of 80%, a mutation rate of 20%, and setting the number of generations to 70 will be sufficient. Additionally, our experiences also show that, because our largest resource graph had a heterogeneous hardening control vector of fewer than 100 variables, we could set the population size equal to 200; nevertheless, we believe that when dealing with a bigger number of optimization variables, the population size that is at least twice number of variables.

The complexity of our proposed solution will depend on the objective function, the population size, and the length of hardening control vector. Since the optimization problem is NP-hard, we will rely on the heuristic algorithm presented in Section 3.3. Figure 8 shows that the processing time increases

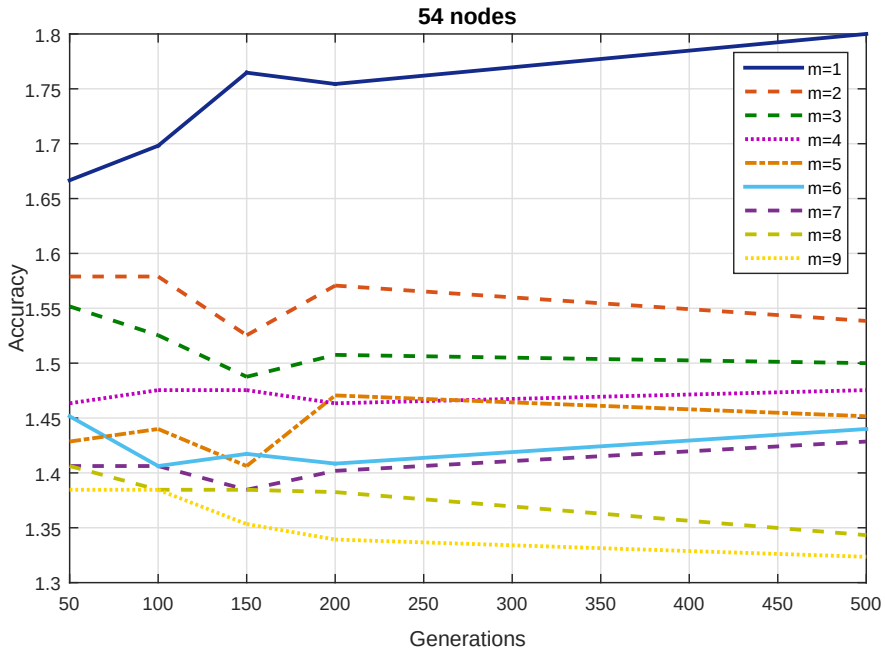

Fig. 9. The accuracy vs. *m* (the parameter of the heuristic algorithm).

almost linearly as we increase the number of optimization variables or the parameter *m* of the heuristic algorithm. The results show that the algorithm is relatively scalable with a linear processing time.

The accuracy of the results presented in Figure 8 is evaluated through simulations. This is addressed through the simulations depicted in Figure 9. Here the accuracy refers to the approximation ratio between the result obtained for the *h* metric using our heuristic algorithm and that of simply enumerating and searching all the paths while assuming all services and service instances are different ($\frac{d_{Heuristic}}{d_{BruteForce}}$). The heterogeneous hardening control vector provided by the GA is used to calculate the accuracy. A ratio close to 1 indicates that our algorithm can provide a solution that is closer to the one provided by enumerating all paths (brute force). From the results, we can see that when *m* is greater or equal to 4 the approximation ratio reaches an acceptable level. For the following simulations, we have settled with an *m* value of 9.

We also consider the ratio between the difference in the *h* metric before and after optimization, ($\frac{d_{Optimized}-d_{NotOptimized}}{d_{NotOptimized}}$), which will be called the *gain* of the *h* metric (or simply the gain). The gain provides us with an idea on how much room there is to improve the security with respect to given cost constraints using our method. Figure 10 shows that the gain will increase linearly as we increase the number of firewall-based hardening options. These results confirm that firewall-based hardening options can positively affect our effort to provide better resilience for networks against zero-day attacks. Additionally, the figure shows that the number of unpatchable vulnerabilities that are present in the network will significantly reduce the gain that can be achieved through other hardening techniques. Since it is not probable to find a large number of different unpatchable vulnerabilities all at the same time within a network, we only consider up to three unpatchable vulnerabilities.

In Figure 11, we analyze the average gain in the optimized results for different sizes of graphs. In this figure, we can see that we have a good enough gain for graphs with a relatively high number of nodes.

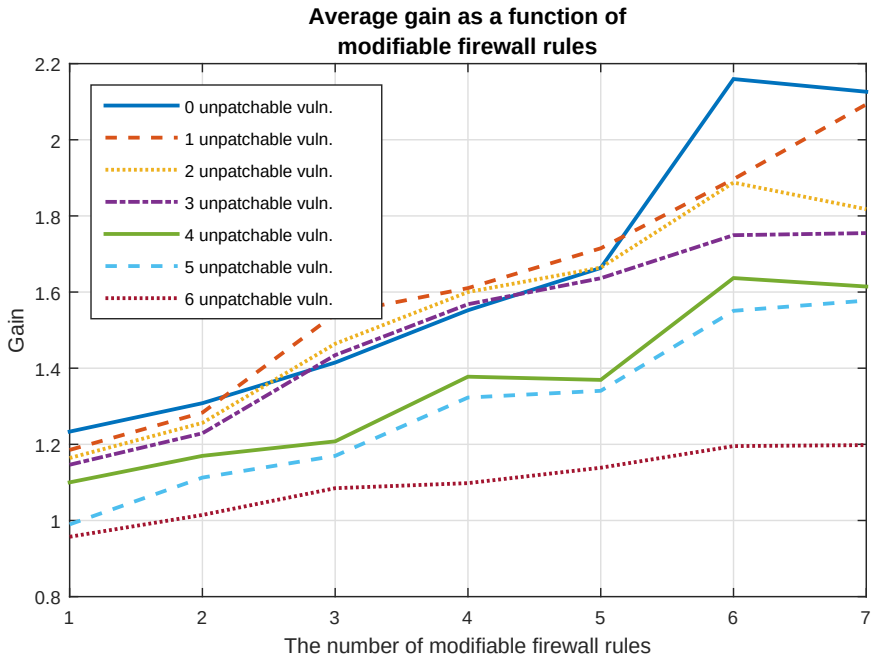

Fig. 10. The average gain based on the number of modifiable firewall rules.

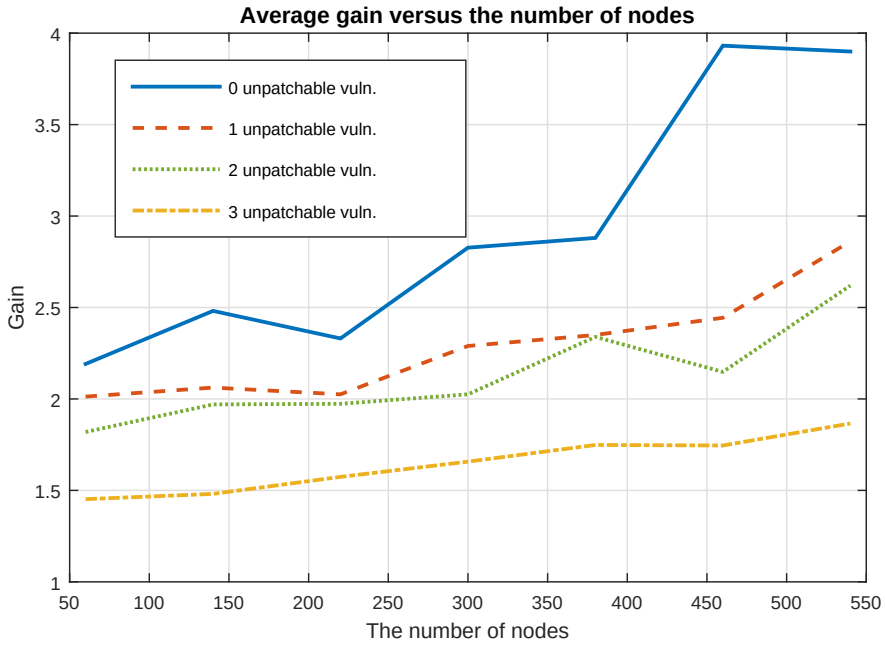

Fig. 11. The average gain vs the number of nodes.

As expected, as we increase the number of unpatchable vulnerabilities, the gain will decrease. However, we can also see this decrease is linear. In the case where no unpatchable vulnerabilities are present, we can see that the gain stops to increase after reaching a certain size of the graph, which can be explained as that the number of available service instances is not large enough (in contrast to the increasing size of the graph) to allow to optimize the *h* metric any further.

Figures 12 and 13 show the optimization results of different shapes of resource graphs in terms of depth and degree of exposure, which roughly represents the extent to which the network is protected. While it may be difficult to exactly define the depth of a resource graph, we have relied on the relative distance, i.e., the difference of the shortest path before and after all hardening options have been applied. There is a linear increase in the gain as we increase the relative distance in the shortest path. This is independent of the amount of unpatchable vulnerabilities. While this does not provide an accurate description of the graph's shape, it does provide an idea of how much our algorithm can increase the minimum *h* for graphs with different depths, as shown in Figure 13, we can see the effect of the network's degree of exposure, which is defined as the number of exploits that are directly reachable by the attacker from the external host *h0*. As we increase the degree of exposure, the gain in optimization decreases (circles in the graph). That is, there will less room for hardening if the network is more exposed.

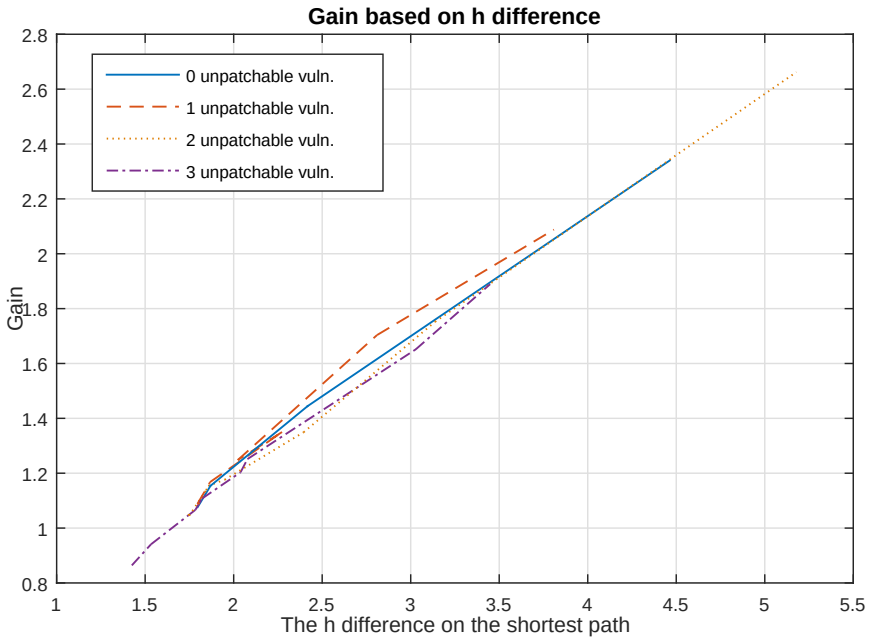

Fig. 12. The *h* difference on the shortest path.

Figures 14 to 16 show the gain is affected by the inclusion, the removal, or the relocation of predefined network services. In Figure 14 we can see that if no services are added, we get the maximum possible gain. We can see that while there is a gain to be obtained when new services are introduced into the network, the rate at which this gain increases decreases as the number of services that can be added increases. This is to be expected since the increase of services also provides an increase in the number of optimization variables. The more optimization variables present on a network, the greater the need

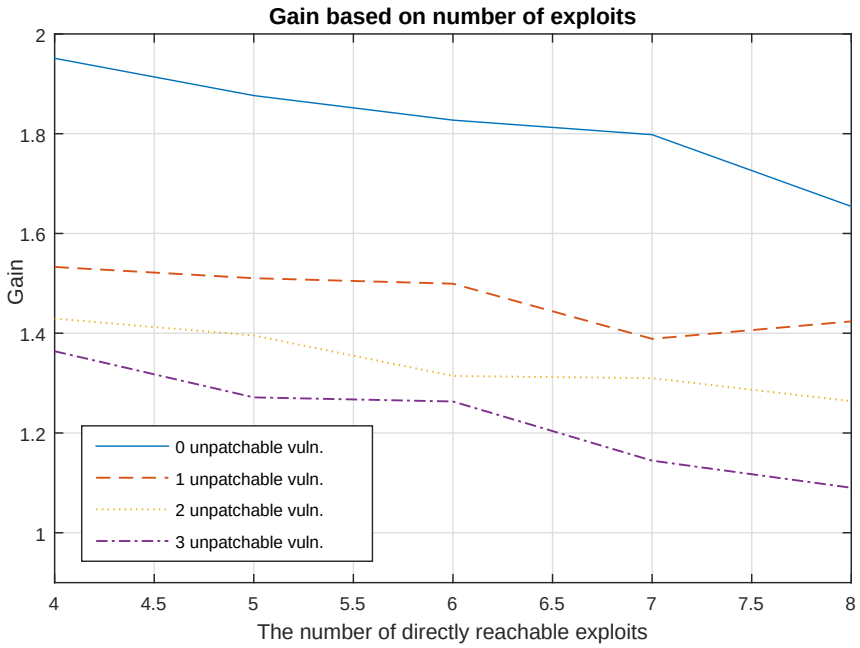

Fig. 13. The number of directly reachable exploits.

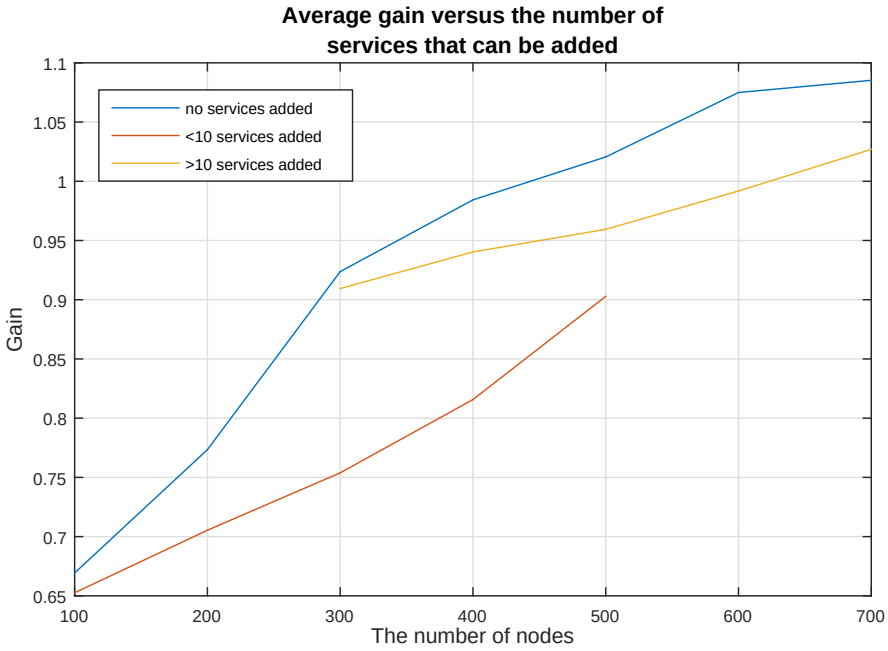

Fig. 14. The average gain based on the number of services that can be added.

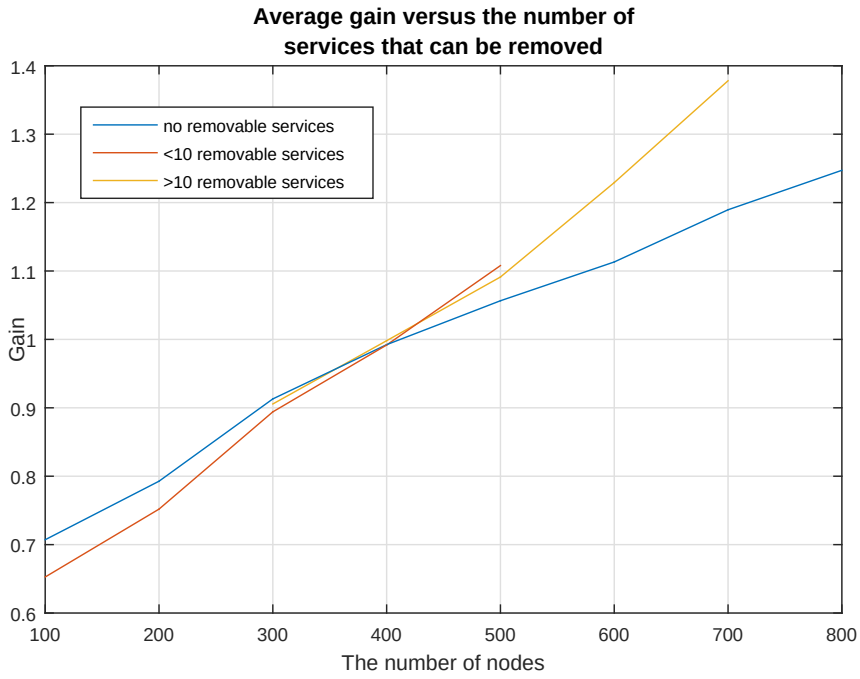

Fig. 15. The average gain based on the number of services that can be removed.

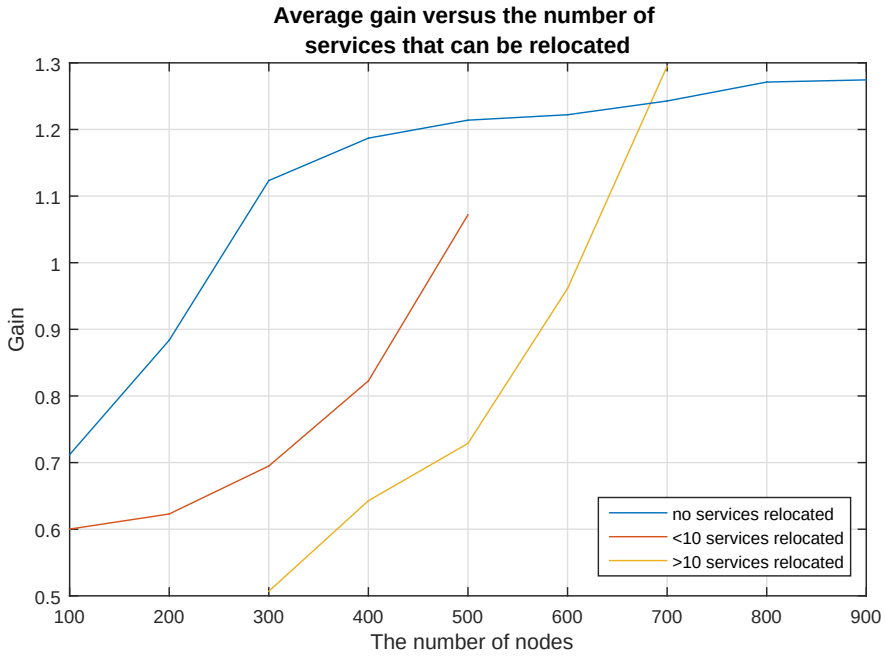

Fig. 16. The average gain based on the number of services that can be moved.

to have a greater amount of unique services to diversify. In Figure 15 we can see a different case. By removing services from the network, we can see the gain increases at a higher rate. This can be explained because we are removing potential zero-day vulnerabilities, and thus rendering our network more secure. Finally, in Figure 16, we can see the effect that moving services has on the network. We can see that, when no services are to be moved, the curve reaches a point where it flattens and where any additional service instances would be needed to continue the growth. What is interesting to note here is that moving services doesn't have that limitation of the gain stopping when we increase the number of nodes, which could allow for further optimization when there are no more instances available to diversify.

5. Related Work

In general, the security of networks may be qualitatively modeled using attack trees [6, 16, 29] or attack graphs [30, 31]. A majority of existing quantitative models of network security focus on known attacks [32, 33], while few works have tackled zero day attacks [13, 14, 22, 34] which are usually considered unmeasurable due to the uncertainties involved [35]. Early works on network hardening typically rely on qualitative models while improving the security of a network [15, 31, 36]. Those works secure a network by breaking all the attack paths that an attacker can follow to compromise an asset, either in the middle of the paths or at the beginning (disabling initial conditions). Also, those works do not consider the implications when dealing with budget constraints nor include cost assignments, and tend to leave that as a separate task for the network administrators. While more recent works [33, 37] generally provide a cost model to deal with budget constraints, one of the first attempts to systematically address this issue is by Gupta et al. [38]. The authors employed genetic algorithms to solve the problem of choosing the best set of security hardening options while reducing costs.

Dewri et a. [6] build on top of Gupta's work to address the network hardening problem using a more systematic approach. They start by analyzing the problem as a single objective optimization problem and then consider multiple objectives at the same time. Their work considers the damage of compromising any node in the cost model to determine the most cost-effective hardening solution. Later, in [16] and in [39], the authors extrapolate the network hardening optimization problem as vulnerability analysis with cost/benefit assessment, and risk assessment respectively.

In [7] Poolsappasit et al. extend Dewri's model to also consider dynamic conditions (conditions that may change or emerge while the model is running) by using Bayesian attack graphs to consider the likelihood of an attack. Unlike our work, most existing work on network hardening are limited to known vulnerabilities and focus on disabling existing services.

There exist many research works on extending attack trees and attack graphs to security metrics. Most of the current works deal with assigning numeric scores to rank known vulnerabilities (mostly based on the CVSS) [40] to be able to model the impact that they have on a network. This ranking is based on how likely and easily exploitable the known vulnerabilities are. This, however, is not the case for unknown vulnerabilities. Because unknown vulnerabilities cannot be modeled using the same methods used for known vulnerabilities, new metrics needed to be devised for them. The k-zero day safety metric [22, 34] first addressed this limitation in security metrics. The problem with this metric is that it counts how many zero-day vulnerabilities are needed to compromise a critical asset which is not an easy task.

A probabilistic metric is applied to attack graphs to obtain an overall attack likelihood for the network [41]. A Bayesian Network (BN) based security metric applies attack graphs to measure the security level of a network [42]. The metric converts the CVSS scores of vulnerabilities into attack probabilities

and then obtain the overall attack likelihood for reaching critical assets. The National Institute of Standards and Technology (NIST) highlights the importance of using some sort of security metrics on cloud systems and provides detailed frameworks and definitions [43].

There exists a rich literature on employing diversity for security purposes. The idea of using design diversity for tolerating faults has been investigated for a long time, such as the N-version programming approach [44], and similar ideas have been employed for preventing security attacks, such as the N-Variant system [45], and the behavioral distance approach [46]. In addition to design diversity and generated diversity, recent work employs opportunistic diversity which already exists among different software systems. For example, the practicality of employing OS diversity for intrusion tolerance is evaluated in [47].

More recently, the authors in [13, 14] adapted biodiversity metrics to networks and lift the diversity metrics to the network level. While those works on diversity provide motivation and useful models, they do not directly provide a systematic solution for improving diversity. So far, the work done by [8], is one of the first work that has tried to provide a solution for this problem; their limitation, however, is that their metric is too simplistic and does not consider additional hardening metrics, which is the topic of this paper.

6. Conclusion

In this paper, we have provided a heterogeneous approach to network hardening to increase the resilience of a network against both unknown and unpatchable vulnerabilities. By unifying different hardening options within the same model, we derived a more general method than most existing efforts that rely on a single hardening option. Our automated approach employed a heuristic algorithm that helped to manage the complexity of evaluating the security metric as well as limiting the time for optimization to an acceptable level. We have addressed one limitation of our previous work by considering that not all costs are additive but they depend on other conditions. We have further discussed realistic cost estimation methods based on existing works. We have tested the efficiency and accuracy of the proposed algorithms through simulation results, and we have also discussed how the gain in the metric will be affected by the addition, the removal, and the relocation of services, as well as the number of available modifiable firewall rules, unpatchable vulnerabilities, and the different sizes and shapes of the resource graphs.

We discuss several aspects of the proposed automated optimization technique where additional improvements and evaluations can be done.

- While this paper has proven that we can integrate different network hardening options under the same model, a more comprehensive approach could be developed by considering other options which might not immediately fit into this model.
- This study relies on a static network configuration. A future research direction would be to consider a dynamic network model in which both attackers and defenders may cause incremental changes in the network.
- We will evaluate other optimization algorithms in addition to GA to compare and potentially use them in hybrid optimization schemes when searching the most efficient solution for our problem.

Acknowledgements. The authors thank the anonymous reviewers for their valuable comments. Authors with Concordia University were partially supported by the Natural Sciences and Engineering Research

Council of Canada under Discovery Grant N01035. Sushil Jajodia was supported in part by the National Science Foundation under grant IIP-1266147; by the Army Research Office under grants W911NF-13-1-0421 and W911NF-13-1-0317; and by the Office of Naval Research under grants N00014-15-1-2007 and N00014-13-1-0703.

References

[1] D.U. Case, Analysis of the Cyber Attack on the Ukrainian Power Grid (2016).
[2] N. Falliere, L.O. Murchu and E. Chien, W32. stuxnet dossier, *White paper, Symantec Corp., Security Response* **5** (2011).
[3] D. Pauli, Easy remote exploit drops for unpatchable power plant controller, May, 2016.
[4] M. Bani, Duck and cover or how AtomBombing is really unnecessarily alarmism, Nov, 2016.
[5] Apache MINA project, Oct, 2016.
[6] R. Dewri, N. Poolsappasit, I. Ray and D. Whitley, Optimal security hardening using multi-objective optimization on attack tree models of networks, in: *Proceedings of the 14th ACM conference on Computer and communications security*, ACM, 2007, pp. 204–213.
[7] N. Poolsappasit, R. Dewri and I. Ray, Dynamic security risk management using bayesian attack graphs, *Dependable and Secure Computing, IEEE Transactions on* **9**(1) (2012), 61–74.
[8] D. Borbor, L. Wang, S. Jajodia and A. Singhal, Diversifying Network Services Under Cost Constraints for Better Resilience Against Unknown Attacks, in: *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 2016, pp. 295–312.
[9] D. Borbor, L. Wang, S. Jajodia and A. Singhal, Securing Networks Against Unpatchable and Unknown Vulnerabilities Using Heterogeneous Hardening Options, in: *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 2017, pp. 509–528.
[10] K. Bakshi, Cisco cloud computing-Data center strategy, architecture, and solutions, *CISCO White Paper. Retrieved October* **13** (2009), 2010.
[11] T. Fifield, D. Fleming, A. Gentle, L. Hochstein, J. Proulx, E. Toews and J. Topjian, *OpenStack Operations Guide*, " O'Reilly Media, Inc.", 2014.
[12] J. Barr, Building three-tier architectures with security groups, June, 2010.
[13] L. Wang, M. Zhang, S. Jajodia, A. Singhal and M. Albanese, Modeling Network Diversity for Evaluating the Robustness of Networks against Zero-Day Attacks, in: *ESORICS 2014*, Springer, 2014, pp. 494–511.
[14] M. Zhang, L. Wang, S. Jajodia, A. Singhal and M. Albanese, Network Diversity: A Security Metric for Evaluating the Resilience of Networks against Zero-Day Attacks, *IEEE Transactions on Information Forensics and Security (TIFS)* **11**(5) (2016), 1071–1086.
[15] L. Wang, M. Albanese and S. Jajodia, *Network Hardening: An Automated Approach to Improving Network Security*, Springer Publishing Company, Incorporated, 2014. ISBN 331904611X, 9783319046112.
[16] R. Dewri, I. Ray, N. Poolsappasit and D. Whitley, Optimal security hardening on attack tree models of networks: a cost-benefit analysis, *International Journal of Information Security* **11**(3) (2012), 167–188.
[17] L. Mieritz and B. Kirwin, Defining Gartner total cost of ownership, *L. Mieritz, B. Kirwin* (2005).
[18] Cost of Data Center Outages. Data Center Performance Benchmark Series, Jan, 2016.
[19] A. Gunasekaran, *Organizational Advancements through Enterprise Information Systems: Emerging Applications and Developments: Emerging Applications and Developments*, IGI Global, 2009.
[20] Amazon.com, Inc. Revenue and Earnings Per Share (EPS), June, 2017.
[21] Number of Amazon.com employees from 2007 to 2016, June, 2017.
[22] L. Wang, S. Jajodia, A. Singhal, P. Cheng and S. Noel, k-Zero day safety: A network security metric for measuring the risk of unknown vulnerabilities, *Dependable and Secure Computing, IEEE Transactions on* **11**(1) (2014), 30–44.
[23] X. Ou, S. Govindavajhala and A.W. Appel, MulVAL: A Logic-based Network Security Analyzer., in: *USENIX security*, 2005.
[24] S. Jajodia, S. Noel and B. O'Berry, Topological Analysis of Network Attack Vulnerability, in: *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava and A. Lazarevic, eds, Kluwer Academic Publisher, 2003.
[25] P. Festa, A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems, in: *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, IEEE, 2014, pp. 1–20.
[26] H.M. Azamathulla, F.-C. Wu, A. Ab Ghani, S.M. Narulkar, N.A. Zakaria and C.K. Chang, Comparison between genetic algorithm and linear programming approach for real time operation, *Journal of Hydro-environment Research* **2**(3) (2008), 172–181.
[27] D.E. Golberg, Genetic algorithms in search, optimization, and machine learning, *Addion wesley* **1989** (1989).

[28] K. Deb, An efficient constraint handling method for genetic algorithms, *Computer methods in applied mechanics and engineering* **186**(2) (2000), 311–338.
[29] I. Ray and N. Poolsapassit, Using attack trees to identify malicious attacks from authorized insiders, in: *ESORICS 2005*, Springer, 2005, pp. 231–246.
[30] P. Ammann, D. Wijesekera and S. Kaushik, Scalable, graph-based network vulnerability analysis, in: *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ACM, 2002, pp. 217–224.
[31] O. Sheyner, J. Haines, S. Jha, R. Lippmann and J.M. Wing, Automated generation and analysis of attack graphs, in: *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, IEEE, 2002, pp. 273–284.
[32] L. Wang, A. Singhal and S. Jajodia, Measuring the overall security of network configurations using attack graphs, in: *Data and Applications Security XXI*, Springer, 2007, pp. 98–112.
[33] M. Albanese, S. Jajodia and S. Noel, Time-efficient and cost-effective network hardening using attack graphs, in: *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, IEEE, 2012, pp. 1–12.
[34] L. Wang, S. Jajodia, A. Singhal and S. Noel, k-zero day safety: Measuring the security risk of networks against unknown attacks, in: *ESORICS 2010*, Springer, 2010, pp. 573–587.
[35] J. McHugh, Quality of protection: measuring the unmeasurable?, in: *Proceedings of the 2nd ACM workshop on Quality of protection*, ACM, 2006, pp. 1–2.
[36] L. Wang, S. Noel and S. Jajodia, Minimum-cost network hardening using attack graphs, *Computer Communications* **29**(18) (2006), 3812–3824.
[37] B. Yigit, G. Gur and F. Alagoz, Cost-Aware Network Hardening with Limited Budget Using Compact Attack Graphs, in: *Military Communications Conference (MILCOM), 2014 IEEE*, IEEE, 2014, pp. 152–157.
[38] M. Gupta, J. Rees, A. Chaturvedi and J. Chi, Matching information security vulnerabilities to organizational security profiles: a genetic algorithm approach, *Decision Support Systems* **41**(3) (2006), 592–603.
[39] S. Wang, Z. Zhang and Y. Kadobayashi, Exploring attack graph for cost-benefit security hardening: A probabilistic approach, *Computers & security* **32** (2013), 158–169.
[40] P. Mell, K. Scarfone and S. Romanosky, Common vulnerability scoring system, *Security & Privacy, IEEE* **4**(6) (2006), 85–89.
[41] L. Wang, T. Islam, T. Long, A. Singhal and S. Jajodia, An attack graph-based probabilistic security metric, in: *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 2008, pp. 283–296.
[42] M. Frigault, L. Wang, A. Singhal and S. Jajodia, Measuring network security using dynamic bayesian network, in: *Proceedings of the 4th ACM workshop on Quality of protection*, ACM, 2008, pp. 23–30.
[43] NIST Special Publication 500-307: Cloud Computing Service Metrics Description (2015), Accessed September, 2017.
[44] A. Avizienis and L. Chen, On the implementation of N-version programming for software fault tolerance during execution, in: *Proc. IEEE COMPSAC*, Vol. 77, 1977, pp. 149–155.
[45] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong and J. Hiser, N-variant systems: a secretless framework for security through diversity, in: *Usenix Security*, Vol. 6, 2006, pp. 105–120.
[46] D. Gao, M.K. Reiter and D. Song, Behavioral distance measurement using hidden markov models, in: *Recent Advances in Intrusion Detection*, Springer, 2006, pp. 19–40.
[47] M. Garcia, A. Bessani, I. Gashi, N. Neves and R. Obelheiro, OS diversity for intrusion tolerance: Myth or reality?, in: *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, IEEE, 2011, pp. 383–394.