Diversifying Network Services under Cost Constraints for Better Resilience against Unknown Attacks

Daniel Borbor[1], Lingyu Wang[1], Sushil Jajodia[2], and Anoop Singhal[3]

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

Abstract. Diversity as a security mechanism has received revived interest recently due to its potential for improving the resilience of software and networks against unknown attacks. Recent work show diversity can be modeled and quantified as a security metric at the network level. However, such an effort does not directly provide a solution for improving the network diversity, and existing network hardening approaches are largely limited to handling previously known vulnerabilities by disabling existing services. In this paper, we take the first step towards an automated approach to diversifying network services under various cost constraints in order to improve the network's resilience against unknown attacks. Specifically, we provide a model of network services and formulate the diversification requirements as an optimization problem. We devise optimization and heuristic algorithms for efficiently diversifying relatively large networks under different cost constraints. We also evaluate our approach through simulations.

1 Introduction

Many critical infrastructures, governmental and military organizations, and enterprises have become increasingly dependent on networked computer systems today. Such mission critical computer networks must be protected against not only known attacks, but also potential zero day attacks exploiting unknown vulnerabilities. However, while traditional solutions, such as firewalls, vulnerability scanners, and IDSs, are relatively successful in dealing with known attacks, they are less effective against zero day attacks.

To this end, diversity has previously been considered for a security mechanism for hardening software systems against unknown vulnerabilities, and it has received a revived interest recently due to its potential for improving networks' resilience against known attacks. In particular, a recent work shows diversity can be modeled and quantified as a security metric at the network level [22]. However, the work does not directly provide a systematic solution for improving the network diversity under given cost constraints, which can be a challenging task for large and complex networks. On the other hand, existing efforts on network hardening (a detailed review of related work will be given later in Section 2) are largely limited to handling previously known vulnerabilities by disabling existing services.

In this paper, we propose an automated approach to diversifying network services under various cost constraints in order to improve the network's resilience against unknown attacks. Specifically, we devise a model of network services and their different instances by extending the resource graph model; such a model allows us to formulate the diversification requirements and cost constraints as an optimization problem; we apply optimization techniques to solve the formulated problems, and design heuristic algorithms to more efficiently handle larger networks. We evaluate our approach through simulations in order to study the effect of optimization parameters on accuracy and running time, and the effectiveness of optimization for different types of networks. In summary, the main contribution of this paper is twofold:

- To the best of our knowledge, this is the first effort on formulating the problem of network service diversification for improving the resilience of networks, which enables the application of existing optimization techniques and also provides a practical application for existing diversity metrics [22].
- As evidenced by the simulation results, the optimization and heuristic algorithms provide a relatively accurate and efficient solution for diversifying network services while considering various cost constraints. By focusing on zero day attacks, our work provides a complementary solution to existing network hardening approaches that focus on fixing known vulnerabilities.

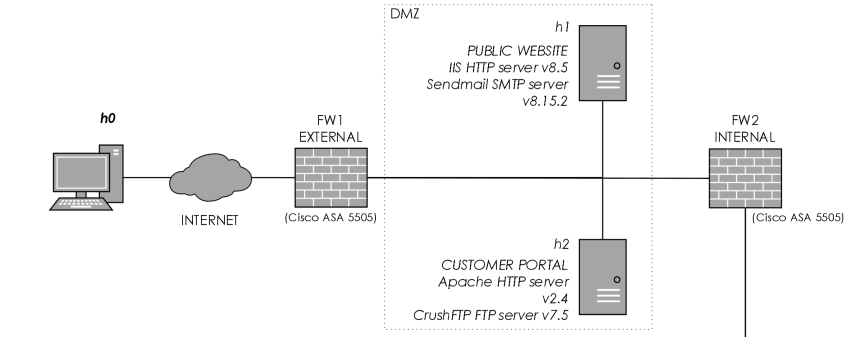
The remainder of this paper is organized as follows: The rest of this section first builds the motivation through a running example. Section 2 reviews related work. In Section 3, we present the model and formulate the optimization problem, and in Section 4 we discuss the methodology and show case studies.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

1.1 Motivating Example

We present a motivating example to demonstrate that diversifying network services can be a tedious and error-prone task if done manually, even if the considered network is of a small size. Figure 1 shows a hypothetical network, which is roughly based on the virtual penetration lab described in [15]. Despite its relatively small scale, it mimics a typical enterprise network, e.g., with DMZ, Web server behind firewall accessible from public Internet, and a private management network protected by another firewall.

Specifically, the network consists of four hosts running one or more services allowing accesses from other hosts. We assume the two firewalls and other host-based mechanisms (e.g., personal firewalls or iptables) together enforce the connectivity described inside the connectivity table shown in the figure. We consider attackers on external hosts (represented as $h0$) attempting to compromise the database server ($h4$), and we assume the network is secured against known vulnerabilities (we exclude exploits and conditions that involve the firewalls).

To measure the network's resilience against unknown zero day attacks, we consider the *k-zero day safety metric* [18] (which will be referred to as $k0d$ from now on for simplicity), which basically counts how many distinct zero day vulnerabilities must exist and be exploited before an attacker may reach the goal. For simplicity, although the attacker may follow many paths to compromise $h4$, here we only consider the Web



Connectivity Table

Source host	Destination host	Ports	Services
Any	h1: 10.160.70.101	25, 80	smtp, http
h1: 10.160.70.101	h2: 10.160.70.102	21, 80	ftp, http
h2: 10.160.70.102	h3: 10.160.70.103	80	http
h3: 10.160.70.103	h4: 10.160.70.104	22, 80	ssh, http

Fig. 1. Example network.

servers as the initial targets. We can observe that there must exist at least two distinct zero-day vulnerabilities, one for the Apache servers ($h2$, $h3$, and $h4$) and one for the IIS server[1] ($h1$), and the attacker must exploit both in order to compromise $h4$. Finally, we assume the administrator has the option of replacing those Web servers with either an NGINX 1.9 or a Litespeed 5.0.14 Web server and each replacement will incur a given installation/maintenance cost (we will discuss the cost model in more details later in Section 3). Based on above assumptions, we may consider different use cases as follows.

- *Scenario 1:* The administrator aims to render the network as resilient as possible to zero-day attacks (which means to maximize the aforementioned $k0d$ metric).
- *Scenario 2:* He/she aims at the same goal as in above Scenario 1, but under the constraint that the overall diversification cost must be less than a given budget.
- *Scenario 3:* He/she aims at the same goal as in above Scenario 2, but under an additional constraint that at most two Web servers may be replaced.
- *Scenario 4:* He/she aims at the same goal as in above Scenario 3, but under an additional constraint that replacing the IIS web server on the DMZ ($h4$) should be given a higher priority.

Clearly, many more use cases may exist in practice than those listed above, and the solution may not always be straightforward even for such a small network. For example, while the administrator can easily increase the $k0d$ metric value to 4 under Scenario 1 (by having four different Web servers), the optimal solution in other scenarios will critically depend on the specific cost constraints and given budgets. Considering that the attacker may also follow other paths to attack (e.g., starting with SMTP, instead of Web, on h1), the problem becomes even more complicated. This shows the need for an automated approach, which will be the subject matter of the remainder of this paper.

[1] If different software are considered likely to share common vulnerabilities, a similarity-sensitive diversity metric may be needed [22].

2 Related Work

In general, the security of networks may be qualitatively modeled using attack trees [7, 8, 16] or attack graphs [2, 17]. A majority of existing quantitative models of network security focus on known attacks [21, 1], while few works have tackled zero day attacks [19, 18, 22] which are usually considered unmeasurable due to the uncertainties involved [13].

Early works on network hardening typically rely on qualitative models while improving the security of a network [17, 20]. 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 [1, 24] 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. [11]. The authors employed genetic algorithms to solve the problem of choosing the best set of security hardening options while reducing costs. Dewri et a. [7] 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 consider the damage of compromising any node in the cost model in order to determine the most cost-effective hardening solution. Later on, in [8] and in [23], the authors extrapolate the network hardening optimization problem as vulnerability analysis with cost/benefit assessment, and risk assessment respectively. In [14] Poolsappasit et al. extend Dewri's model to also take into account dynamic conditions (conditions that may change or emerge while the model is running) by using Bayesian attack graphs in order 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 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 [3], and similar ideas have been employed for preventing security attacks, such as the N-Variant system [5], and the behavioral distance approach [9]. In addition to design diversity and generated diversity, recent work employ opportunistic diversity which already exists among different software systems. For example, the practicality of employing OS diversity for intrusion tolerance is evaluated in [10]. More recently, the authors in [22] 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, which is the topic of this paper.

3 Model

We first introduce the extended resource graph model to capture network services and their relationships, then we present the diversity control and cost model, followed by problem formulation.

3.1 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 resource (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). For this purpose, we will extend the concept of *resource graph* introduced in [22], which is syntactically equivalent to attack graphs, but models network resources instead of known vulnerabilities as in the latter.

Specifically, we will define an *extended resource graph* by introducing 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 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.

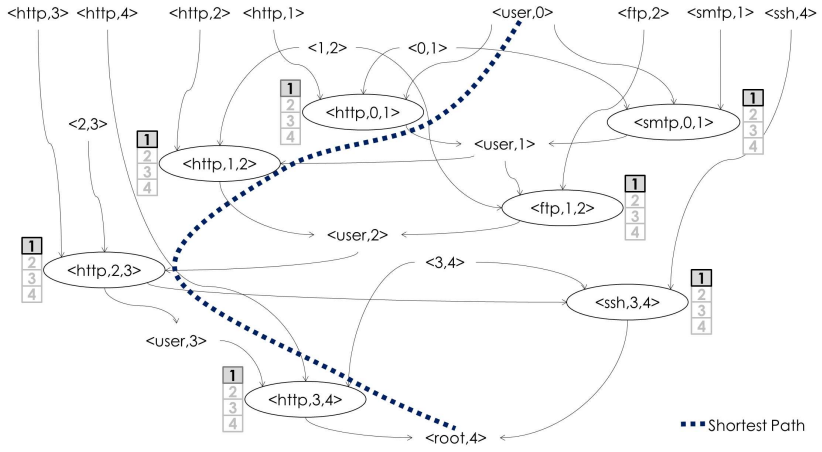


Fig. 2. The example network's resource graph

In figure 2, each pair shown in plaintext is a security-related condition (e.g., connectivity $\langle source, destination\rangle$ or privilege $\langle privilege, host\rangle$). 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). The small one-column table beside each exploit indicates the current service instance using a highlighted integer (e.g., 1 means Apache and 2 means IIS) and other potential instances in lighter text. The self-explanatory edges point from pre-conditions to an exploit (e.g., from $\langle 1, 2\rangle$ and $\langle http, 2\rangle$ to $\langle http, 1, 2\rangle$), and from the exploit to its post-conditions (e.g., from $\langle http, 1, 2\rangle$ to $\langle user, 2\rangle$).

A design choice here is whether to associate the service instance concept with a condition indicating the service (e.g., $\langle http, 2\rangle$) or the corresponding exploits (e.g., $\langle http, 1, 2\rangle$). While it is more straightforward to have the service instance defined as the property of a condition, which can then be inherited by the corresponding exploits, we have opted to define this property as a label for the exploit nodes in the graph, be-

cause this will make it easier to check the number of distinct services along a path, as we will see later. One complication then is that we must ensure all exploits with the same service and destination host (e.g., $\langle http, 1, 2\rangle$ and $\langle http, 3, 2\rangle$) to be associated with the same service instance.

Definitions 1 and 2 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 represent all available instances of that service.*

Definition 2 (Extended Resource Graph). *Given a network composed of*

- *a set of hosts H,*
- *a set of services S, with the service mapping $serv(.) : H \rightarrow 2^S$,*
- *the collection of service pools $SP = \{sp(s) \mid s \in S\}$,*
- *and the labelling function $v(.) : E \rightarrow SP$, which satisfies $\forall h_s \in S \ \forall h'_s \in S, v(\langle s, h_s, h_d\rangle) = v(\langle s, h'_s, h_d\rangle)$ (meaning all exploits with common service and destination host must be associated with the same service instance, as explained earlier).*

Let E be the set of zero day exploits $\{\langle s, h_s, h_d\rangle \mid h_s \in H, h_d \in H, s \in serv(h_d)\}$, and $R_r \subseteq C \times E$ and $R_i \subseteq E \times C$ be the collection of pre and post-conditions in C. We call the labeled directed graph, $\langle G(E \cup C, R_r \cup R_i), v\rangle$ the extended resource graph.

3.2 Diversity control and cost model

We employ the notion of *diversity control* as a model for diversifying one or more services in the resource graph. Since we represent the service instance using integers, it will be straightforward to regard each pair of service and destination host on which the service is running as an optimization variable, and formulate diversity control vectors using those variables as follows. We note that the number of optimization variables present in a network will depend on the number of conditions indicating services, instead of the number of exploits (since 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 network (i.e., the number of hosts).

Definition 3 (Optimization Variable and Diversity Control). *Given an extended resource graph $\langle G, v\rangle$, $\forall e \in E$, $v(e)$ is an optimization variable. A diversity control vector is the integer valued vector $\boldsymbol{V} = (v(e_1), v(e_2), ..., v(e_{|E|})$.*

Changing the value of an optimization variable has an associated *diversification cost* and the collection of such costs is given in a *diversity cost matrix* in a self-explanatory manner. We assume the values of cost are assigned by security experts or network administrators. Like in most existing works (e.g., [7]), we believe an administrator can estimate the diversification 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. We define the diversity cost, diversity cost matrix, and the total diversity cost.

Definition 4 (Diversification Cost and Diversity Cost Matrix). *Given $s \in S$ and $sp(s)$, the cost to diversify a service by changing its service instance to another inside the service pool is called the diversification cost. The collection of all the costs constraints q associated with changing services in S are given as a diversity cost matrix DCM in which the element at i^{th} row and j^{th} column indicates the diversification cost of changing the i^{th} service instance to be the j^{th} service instance. Let $v_s(e_i)$ be the service associated with the optimization variable $v(e_i)$ and $\boldsymbol{V}_0$ the initial service instance values for each of the exploits in the network. The total diversification cost, Q_d, given by the diversity vector $\boldsymbol{V}$ is obtained by*

$$Q_d = \sum_{i=1}^{|E|} DCM_{v_s(e_i)}(\boldsymbol{V}_0(i), \boldsymbol{V}(i))$$

We note that the above definition of diversification cost between each pair of service instances has some advantages. For example, 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. Also, the concept can be used to specify many different types of cost constraints, which we will examine in the coming section. For example, an administrator who wants to restrict the total cost to diversify all servers running the $http$ service can do so by simply formulating the cost as the addition of all the optimization variables corresponding to $http$.

3.3 Problem formulation

As demonstrated in Section 1.1, the $k0d$ metric is defined as the minimum number of distinct resources on a single path in the resource graph [18]. For example, a closer look at figure 2 shows that the $k0d$ value for our example network is 1. That is, an attacker needs only one zero-day vulnerability (in $http$ service instance 1) to compromise this network. The dashed line in figure 2 depicts the shortest path that provides this metric value.

The $k0d$ value can be increased by changing the service instances as long as we respect the available budget of cost. For example, consider a total budget of 78 units, and assume the costs to diversify the $http$ service from service instance 1 to 2, 3 or 4 be 78, 12, and 34 units, respectively. We can see that changing $\langle http, 2, 3\rangle$ from instance 1 to 2 would respect the budget, as well as increasing the $k0d$ value of the network to be 2. We may also see that this is not the optimal solution, since we could also replace $\langle http, 2, 3\rangle$ and $\langle http, 3, 4\rangle$ with instances 3 and 4, respectively, increasing $k0d$ to 3 and still respecting the budget. In the following, we formally formulate this as an optimization problem.

Problem 1 ($k0d$ Optimization Problem). Given an extended resource graph $\langle G, v\rangle$, find a diversity control vector $\boldsymbol{V}$ which maximizes $min(k0d(\langle G(\boldsymbol{V}), v\rangle))$ subject to the constraint $Q \leq B$, where B is the availble budget and Q is the total diversification cost as given in Definition 4.

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., the tool in [12] has seen many real applications to enterprise networks) may be easily extended to generate the extended resource graphs we need as inputs. Additionally, our problem formulation assumes a very general 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 explained in the following section.

4 Methodology

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

4.1 Genetic Algorithm Optimization

The genetic algorithm (GA) is a simple and robust search method and optimization technique inspired in the mechanisms of natural selection. We employ GA for our automated optimization approach because it requires little information to search effectively in a large search space in contrast to other optimization methods (e.g., the mixed integer programming [4]). It also provides a simple way to encode candidate solutions to the problem [6]. While inspired by [7], we focus on service diversification and not on disabling services.

The extended resource graph is the input to our automated optimization algorithm where the function to be optimized (fitness function) is $k0d$ defined on the resource graph (later we will discuss cases where directly evaluating $k0d$ is computationally infeasible). One important point to consider when optimizing the $k0d$ function on the extended resource graph is that, for each generation of the GA, the graph's labels will dynamically change. This in turn will change the value of $k0d$, since the shortest path may have changed with each successive generation of the GA. Our optimization tool takes this into consideration. We also note one limitation here is that the optimization does not provide a priority if there are more than one shortest path that provide the optimized $k0d$ since the optimization only aims at maximizing the minimum $k0d$.

The constraints are defined as a set of inequalities in the form of $q \leq b$, 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 diversity cost Q_d) or specific constraints to address certain requirements or priorities while diversifying services (e.g., the cost to diversify $http$ services should be less than 80% of the cost to diversify ssh). 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 particular network hardening problem, the GA will be dealing with integer variables representing the selection of the service instances. Because $v(e)$ 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. The initial service

instance for each of the services is given by the extended resource graph while the final diversity control vector $\boldsymbol{V}$ is obtained after running the GA.

The population size that we defined for our tool was 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). In the following, we discuss several test cases to demonstrate how the optimization works under different types of constraints. For all the test cases, we have used the following algorithm parameters: population size = 100, number of generations = 150, crossover probability = 0.8, and mutation probability = 0.2.

Test case A: $Q_d \leq 124$ units with individual constraints per service. We start with the simple case of one overall budget constraint ($Q_d \leq 124$). The solution provided by the GA is $\boldsymbol{V} = [3, 2, 1, 4, 1, 1, 1]$ (represented by label column a in figure 3). The associated costs for $\boldsymbol{V}(1)$, $\boldsymbol{V}(2)$, and $\boldsymbol{V}(4)$ are 12, 78, and 34, respectively, and the test network's $k0d$ metric becomes 4 while keeping Q_d within the budget ($Q_d \leq 124$).

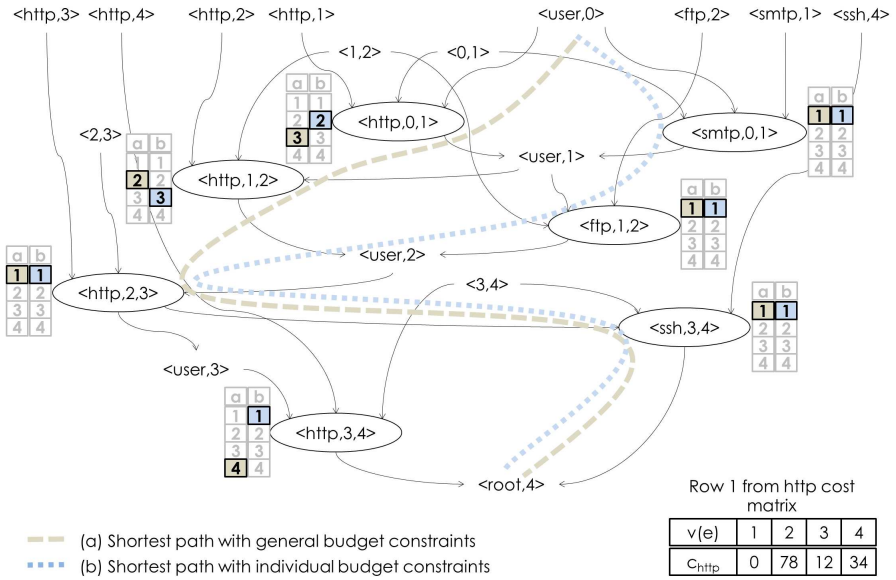


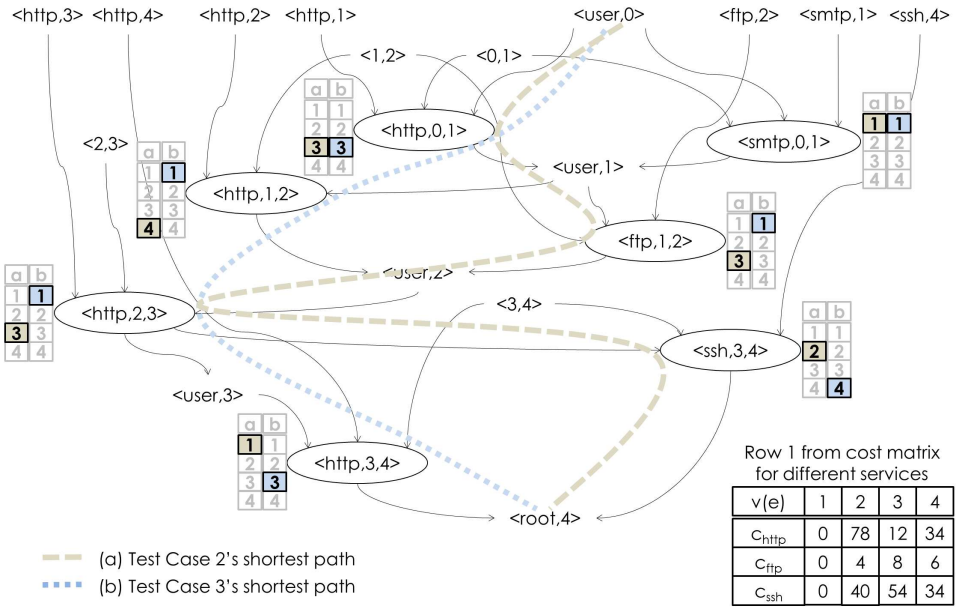
Fig. 3. Test case A: general and individual budget constraints.

On the other hand, if we assign individual budgets per services, while maintaining the overall budget $Q_d \leq 124$, the optimization results will be quite different. In this case, assume the budget to diversify the $http$ services cannot exceed 100 units ($q_{http} \leq 100$); for ftp, it cannot exceed 3 units ($q_{ftp} \leq 3$); for ssh, it cannot exceed 39 units ($q_{ssh} \leq 39$); finally, for $smtp$, it cannot exceed 50 units ($q_{smtp} \leq 50$). The solution provided by the GA is a $\boldsymbol{V}$ vector where $\boldsymbol{V}(1) = 2$ and $\boldsymbol{V}(2) = 3$, with a cost of 78 and 12 units, respectively. The value of the $k0d$ metric rises to 3 with $Q_d = 90$. This total diversification cost satisfies both the overal budget constraint and each of the individual constraints per service.

From this test case, we can see that even with the minimum requeried budget to maximize the $k0d$ metric, additional budget constraints might not allow to achieve the maximum $k0d$ possible. We can see the result of running the GA for this test case in label column b in figure 3.

Test case B: $Q_d \leq 124$ units while $q_{http} + q_{ssh} \leq 100$. While test case 1 shows how individual cost constraints can affect the $k0d$ metric optimization, in practice not all services may be of concern and some may have negligible cost. This test case models such a scenario by assigning a combined budget restriction for only the $http$ and ssh services, i.e., the cost incurred by diversifying these two services should not exceed 100 units.

The solution provided by the GA is $\boldsymbol{V} = [3, 4, 3, 1, 1, 3, 2]$ (lable column a in figure 4). Since $\boldsymbol{V}(1)$ to $\boldsymbol{V}(3)$ deal with the $http$ service, we can see that the total incurred cost for $http$ is q_{http} =12+34+12=58 units. Because $\boldsymbol{V}(6)$ and $\boldsymbol{V}(7)$ are the only optimization variables that deal with the ftp and ssh services respectively, we can see that $q_{ftp} = 8$, and $q_{ssh} = 40$. The value of the $k0d$ metric rises from 1 to 3 by incurring a total cost of $Q_d = 106$ units. The combined $http/ssh$ budget constraint of 100 units is also satisfied since $q_{http} + q_{ssh} = 98$ units.



v(e)	1	2	3	4
c_{http}	0	78	12	34
c_{ftp}	0	4	8	6
c_{ssh}	0	40	54	34

Fig. 4. Test case B and test case C.

Test case C: $Q_d \leq 124$ units while $q_{http} \leq 0.8 \cdot q_{ssh}$. This final case deals with scenarios where some services might have a higher priority over others. The constraint in this test case is that the total incurred cost while diversifying the $http$ service should not exceed 80% of what is incurred by diversifying the ssh service.

The solution provided by the GA is $\boldsymbol{V}$ =[3,1,3,1,1,1,4] (see column b in figure 4). Here $\boldsymbol{V}(1)$ and $\boldsymbol{V}(3)$ have changed from service instance 1 to 3, while $\boldsymbol{V}(7)$ have

changed from service instance 1 to 4. The incurred cost for the $http$ service is $q_{http} =$ 12+12=24 units and for the ssh service is $q_{ssh} = 34$ units. While the value of the $k0d$ metric only rises from 1 to 2, the budget constraints are satisfied.

As seen from the above test cases, our model and problem formulation makes it relatively easy to apply any standard optimization techniques, such as the GA, to optimize the $k0d$ metric through diversity while dealing with different budget constraints.

4.2 Heuristic Algorithm

All the test cases described above rely on the assumption that all the attack paths are readily available. However, this is not always the case in practice. Due to the well known complexity that resource graphs have inherited from attack graphs due to their common syntax [22], it is usually computationally infeasible to enumerate all the available attack paths in a resource graph for large networks. Therefore, we design a heuristic algorithm to reduce the search complexity when calculating and optimizing the $k0d$ metric by only storing the m-shortest paths at each step, as depicted in figure 5 and detailed below.

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

Fig. 5. A Heuristic algorithm for calculating m-shortests paths

The algorithm starts by topologically sorting the graph (line 1) and proceeds to go through each one of the nodes on the resource graph 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 conditions, 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 of 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 $Unique$ to first look for unique combinations of service and service instance in $\alpha'(ov(e))$. Then, the algorithm creates a dictionary structure where the key is a path from $\alpha(e)$ and the value is the number of unique service/service instance combinations 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 number of distinct combination of services and service instances (line 13). If there are less than m paths, it will return all of 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 goal condition c_g. It is worth noting that the algorithm does not make any distinction in whether or not a particular path has a higher priority over another when they share the same number of unique service/service instance combinations.

5 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 a large number of resource graphs for simulations, we first construct a small number of seed graphs based on real 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). The resource graphs were used as the input for the optimization toolbox where the objective function is to maximize the minimum $k0d$ value subject to budget constraints. In all the simulations, we employ the heuristic algorithm described in section 4.2.

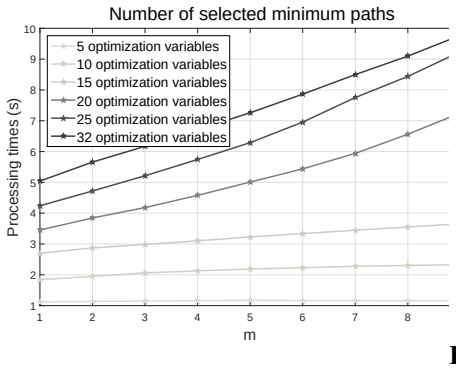


Fig. 6. Processing time.

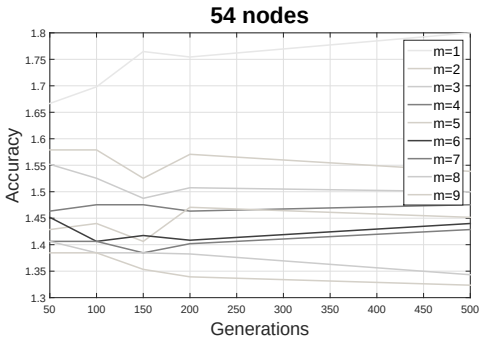


Fig. 7. Accuracy vs m (parameter of the heuristic algorithm).

Figure 6 shows that the processing time increases 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. On the other hand, the accuracy of the results is also an important issue to be considered. Here the accuracy refers to the approximation ratio between the result obtained using the heuristic algorithm and that of the brute force algorithm (i.e., simply enumerating and searching all the paths while assuming all services and service instances are different). For the simulations depicted in figure 7, we settled for 50 iterations per graph per m-paths. The diversity control vector provided by the GA is used to calculate the accuracy. 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 6 and 100 generations.

Our simulations also showed that (detailed simulation results are omitted here due to page limitations), when no budget constraints are in effect, using the GA with a crossover probability of 80%, a mutation rate of 20%, and setting the number of generations to 50 will be sufficient to obtain good results. However, this is no longer the case when dealing with budget constraints. We have noticed that, by decreasing the crossover probability (and consequently increasing the mutation rate), we can reach a viable solution with less generations. We have therefore settled with a crossover probability of 40% which provides us with a fast (with less generations) way to converge to viable solutions. Additionally, our experiences also show that, when dealing with a diversity control vector (also known as a chromosome in the GA) of less than 100 variables (genes in the GA), the population size could be equal to the amount of variables in the diversity control vector; when dealing with a bigger number, the population size should be at least twice the amount of variables.

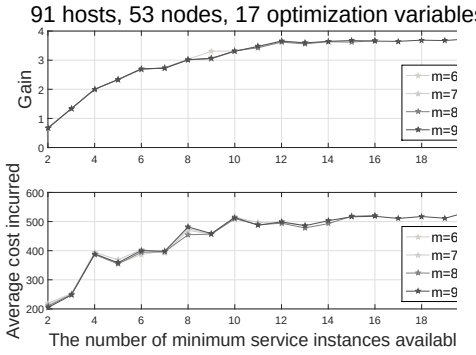


Fig. 8. The effect of the number of available service instances.

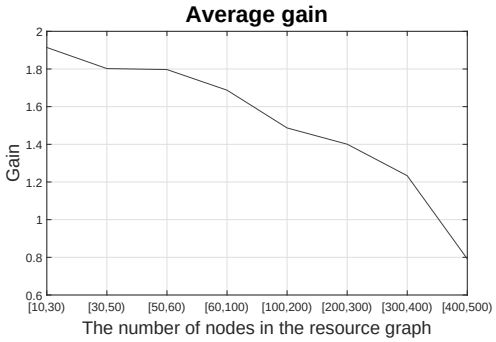


Fig. 9. The average gain vs the number of nodes.

Figure 8 shows the results when the diversity control vector has different numbers of sevice instances to take from (i.e., different sizes of the service pools). In this simulation, we have picked graphs with a relative high difference in the length of the shortest path before and after all services are diversified using the algorithm (the maximum $k0d$ value is 16 and the minimum 3). We can see an increasing gain in the $k0d$ value after optimization, when more service instances are available. However, this trend begins to stall after a certain number (13). From this observation it can be inferred that the number of available service instances will affect the difference between the maximum $k0d$

value possible and the minimum $k0d$, but such an effect also depends on the size of the network (or the extended resource graph) and increasing the number of available service instances does not always help.

In figure 9, 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 amount of nodes. As expected, as we increase the size of the graphs, the gain will decrease if we keep the same optimization parameters. For those simulations, we have used a population size of 300, 50 generations, and a crossover fraction of 50%. It is interesting to note that the decrease in gain is very close to being linear.

Figure 10 and figure 11 show the optimization results on different shapes of resource graphs. 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 services are diversified. There is a relative linear increase in the gain as we increase the relative distance in the shortest path. 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 $k0d$ for graphs with different depths, as shown in figure 10.

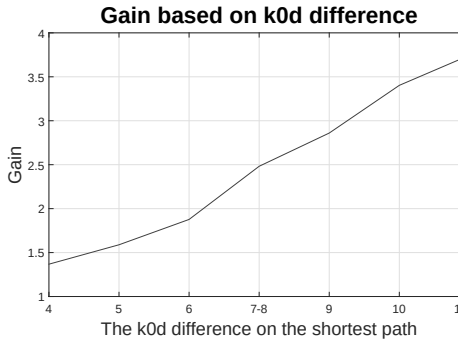


Fig. 10. Average gain based on relative distance of shortest path.

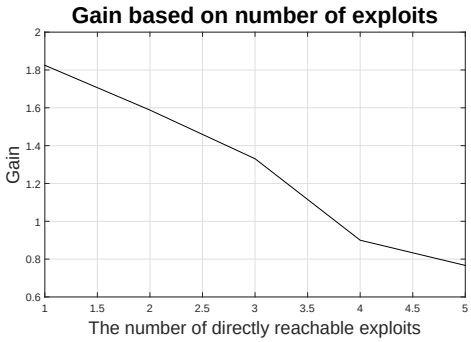


Fig. 11. Average gain based on directly reachable vulnerabilities.

Finally, in figure 11, 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 in almost a linear way. That is, there will less room for diversification if the network is more exposed.

6 Conclusions

In this paper, we have formulated service diversity as an optimization problem and proposed an automated diversity-based network hardening approach against zero-day attacks. This automated approach used a heuristic algorithm that helped to manage the complexity of computing the $k0d$ value as well as limiting the time for optimization to an acceptable level. We have shown some sample cost constraints while our model and problem formulation would allow for other practical scenarios to be specified and optimized. We have tested the scalability and accuracy of the proposed algorithms through

simulation results, and we have also discussed how the gain in the $k0d$ value will be affected by the number of available service instances in the service pools and 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 focuses on diversifying services, a natural future step is to integrate this approach with other network hardening options, such as addition or removal of services, or relocating hosts or services (e.g., firewalls).
- This study has relied on a simplified model by assuming all service instances to be completely different from each another and all service instances are equally likely to be exploited. A possible future research direction would be to model the degree of difference (or similarity) between the different types of service instances.
- We have assumed an abstract cost model in this paper and an important direction is to elaborate the model from different aspects of potential cost for diversifying network resources.
- We will also consider other optimization algorithms, in addition to GA, to compare and potentially use them in hybrid optimization schemes, when searching for more efficient and effective solutions to our problem.
- This study relied on a static network configuration. A future research direction would be to consider a dynamic network model by using a probabilistic approach for the attack likelihood (e.g., using Bayesian networks).

Acknowledgements. This work was partially supported by the National Institute of Standards and Technology grant 60NANB15D091, by the National Science Foundation grant IIP-1266147, and by the Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035.

References

1. Massimiliano Albanese, Sushil Jajodia, and Steven 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*, pages 1–12. IEEE, 2012.
2. Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.
3. Algirdas Avizienis and Liming Chen. On the implementation of n-version programming for software fault tolerance during execution. In *Proc. IEEE COMPSAC*, volume 77, pages 149–155, 1977.
4. H Md Azamathulla, Fu-Chun Wu, Aminuddin Ab Ghani, Sandeep M Narulkar, Nor Azazi Zakaria, and Chun Kiat Chang. Comparison between genetic algorithm and linear programming approach for real time operation. *Journal of Hydro-environment Research*, 2(3):172–181, 2008.
5. Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. N-variant systems: a secretless framework for security through diversity. In *Usenix Security*, volume 6, pages 105–120, 2006.
6. Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2):311–338, 2000.

7. Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell 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*, pages 204–213. ACM, 2007.
8. Rinku Dewri, Indrajit Ray, Nayot Poolsappasit, and Darrell Whitley. Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, 11(3):167–188, 2012.
9. Debin Gao, Michael K Reiter, and Dawn Song. Behavioral distance measurement using hidden markov models. In *Recent Advances in Intrusion Detection*, pages 19–40. Springer, 2006.
10. Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 383–394. IEEE, 2011.
11. Mukul Gupta, Jackie Rees, Alok Chaturvedi, and Jie Chi. Matching information security vulnerabilities to organizational security profiles: a genetic algorithm approach. *Decision Support Systems*, 41(3):592–603, 2006.
12. S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publisher, 2003.
13. John McHugh. Quality of protection: measuring the unmeasurable? In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 1–2. ACM, 2006.
14. Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *Dependable and Secure Computing, IEEE Transactions on*, 9(1):61–74, 2012.
15. Penetration testing virtual labs. https://www.offensive-security.com/offensive-security-solutions/virtual-penetration-testing-labs/, Sep, 2015.
16. Indrajit Ray and Nayot Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *ESORICS 2005*, pages 231–246. Springer, 2005.
17. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.
18. Lingyu Wang, Sushil Jajodia, Anoop Singhal, Pengsu Cheng, and Steven 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):30–44, 2014.
19. Lingyu Wang, Sushil Jajodia, Anoop Singhal, and Steven Noel. k-zero day safety: Measuring the security risk of networks against unknown attacks. In *ESORICS 2010*, pages 573–587. Springer, 2010.
20. Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006.
21. Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring the overall security of network configurations using attack graphs. In *Data and Applications Security XXI*, pages 98–112. Springer, 2007.
22. Lingyu Wang, Mengyuan Zhang, Sushil Jajodia, Anoop Singhal, and Massimiliano Albanese. Modeling network diversity for evaluating the robustness of networks against zero-day attacks. In *ESORICS 2014*, pages 494–511. Springer, 2014.
23. Shuzhen Wang, Zonghua Zhang, and Youki Kadobayashi. Exploring attack graph for cost-benefit security hardening: A probabilistic approach. *Computers & security*, 32:158–169, 2013.
24. Beytullah Yigit, Gurkan Gur, and Fatih Alagoz. Cost-aware network hardening with limited budget using compact attack graphs. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 152–157. IEEE, 2014.