

COST-AWARE NETWORK HARDENING USING COMPACT ATTACK GRAPHS

by

Beytullah Yiğit

B.S., Computer Engineering, Boğaziçi University, 2012

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2014

ACKNOWLEDGEMENTS

I would like to thank to my thesis supervisor Prof. Fatih Alagöz for his support and guidance during my thesis. I also would like to thank to M. Ufuk Çağlayan and Berk Canberk who agreed to participate in my thesis committee.

I am really grateful to Gürkan Gür for all his continuous support, invaluable feedback and contributions. He spent a lot of time to review and improve my studies. I would like to express my deep and sincere gratitude to him. This thesis would certainly not come to an end without him. I would like to thank Levent Altay for his precious comments and support.

I thank to Meltem Ertuna for giving me the strength to live.

I would like to dedicate this thesis to my parents, Kenan and Esme, to thank them for their unconditional love, trust, appreciation and support.

This work was supported by ITEA2 ADAX Project No. 10030 and TÜBİTAK TEYDEB Project No. 9130016.

ABSTRACT

COST-AWARE NETWORK HARDENING USING COMPACT ATTACK GRAPHS

Attack graphs provide analytical support to prevent multistep network attacks by showing all possible sequences of vulnerabilities and their interactions. Since attack graphs generally consist of a very large number of nodes, it is computationally challenging to analyze them for hardening a network against attacks. In this thesis, we propose a greedy heuristic method to find a cost-effective solution to protect a network using compact attack graphs. First, we extract all possible attack paths which reach predetermined critical resources embedded in the network. The exploit or initial condition which contributes the most to attack paths with least cost is selected to be removed. This process continues iteratively and a security analyst can stop it when the total cost exceeds the allocated budget. The experimental results show that our algorithm scales almost linearly with the size of the networks and it can be applied to large-scale graphs with a very large number of nodes. They also show that the algorithm finds nearly minimum cost solution compared to optimal solution. In addition to providing a network-hardening solution, our proposal measures the security level of the network in every step to demonstrate how vulnerable the network is against threats. This accompanying feature is beneficial for network security assessment and situation awareness.

ÖZET

KOMPAKT SALDIRI ÇİZGELERİ KULLANILARAK MALİYET ODAKLI AĞ GÜVENLİĞİ

Saldırı çizgeleri, güvenlik açıkları ve bunların etkileşimlerinin olası olduğu tüm dizileri göstererek aşamalı ağ saldırılarını önlemek için analitik destek sağlamaktadır. Saldırı çizgeleri genellikle çok fazla sayıda düğümden oluşmaktadır. Bu düğümleri analiz etmek ve aşamalı ağ saldırılarına karşı korunma çözümleri bulmak zordur. Bu tezde, kompakt saldırı çizgeleri kullanarak, ağları korumak için düşük maliyetli bir çözüm bulan açgözlü, sezgisel bir yöntem öneriyoruz. İlk olarak, ağda önceden belirlenmiş kritik kaynaklara ulaşan olası tüm saldırı yolları çıkartılır. Minimum maliyetle saldırı yollarına en fazla katkı sağlayan güvenlik açığı yada başlama koşulu devre dışı bırakılmak için seçilir. Bu işlem tekrarlanarak devam eder ve toplam maliyet ayrılan bütçeyi aştığında bir güvenlik analisti onu durdurabilir. Simülasyon sonuçları, önerilen yöntemin çizge büyüklüğü ile hemen hemen doğrusal karmaşıklığa sahip ve çok geniş ölçekli çizgelere uygulanabilir olduğunu göstermektedir. Simülasyon sonuçları ayrıca, yöntemin optimum çözüme yaklaşık düşük maliyetli bir çözüm bulduğunu göstermektedir. Buna ek olarak, önerilen yöntem ağın tehditlere karşı ne kadar savunmasız olduğunu göstermek için her adımda ağın güvenlik seviyesini ölçer. Bu özellik, ağ güvenlik değerlendirmesi ve mevcut durum farkındalığı için faydalıdır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF SYMBOLS	xi
LIST OF ACRONYMS/ABBREVIATIONS	xii
1. INTRODUCTION	1
2. RELATED WORK	3
3. BUDGET CONTROLLED NETWORK HARDENING ALGORITHM (<i>BC-NHA</i>)	12
4. SAMPLE ATTACK GRAPHS FOR <i>BCNHA</i> OPERATION	20
4.1. Example Attack Graph I	20
4.2. Example Attack Graph II	24
5. COMPLEXITY ANALYSIS AND EXPERIMENTAL RESULTS	28
5.1. Complexity Analysis of <i>BCNHA</i>	28
5.2. Protection Cost Analysis of <i>BCNHA</i>	32
6. CONCLUSIONS	34
APPENDIX A: SAMPLE NETWORK 2 (AG2)	36
APPENDIX B: SIMULATION DATA	38
B.1. Complexity Analysis Data	38
B.2. Cost Efficiency Data	38
REFERENCES	41

LIST OF FIGURES

Figure 2.1.	The network model of AG1.	5
Figure 2.2.	An attack graph (AG1) with four exploit types, 11 exploits and seven initial conditions.	6
Figure 3.1.	The flowchart of <i>BCNHA</i>	13
Figure 4.1.	Second attack graph (AG2) with eight exploit types, 12 exploits and 11 initial conditions.	25
Figure 5.1.	Average number of attack paths for randomly generated attack graphs with different path length threshold n	30
Figure 5.2.	Average CPU time in milliseconds for Phase I of <i>BCNHA</i> in randomly generated attack graphs with different path length threshold n	31
Figure 5.3.	Average CPU time in milliseconds for Phase II of <i>BCNHA</i> in randomly generated attack graphs with different path length threshold n	31
Figure 5.4.	Average total protection costs for <i>BCNHA</i> , optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.	33
Figure 5.5.	Average CPU time in milliseconds for <i>BCNHA</i> , optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.	33

Figure A.1. The network model of AG2 used as second example attack graph of this thesis.	37
---	----

LIST OF TABLES

Table 2.1.	Explanations of exploits.	5
Table 3.1.	Analysis parameters.	14
Table 3.2.	<i>CVSS</i> attributes used to calculate base metric exploitability scores.	16
Table 3.3.	Temporal exploitability score values and descriptions.	16
Table 4.1.	Success probability and cost value for each exploit type in AG1. . .	21
Table 4.2.	Attack paths in AG1 and their success probabilities.	21
Table 4.3.	Contribution and effective cost β values for AG1.	22
Table 4.4.	Success probability and cost value for each exploit type in AG2. . .	24
Table 4.5.	Attack paths in AG2 and their success probabilities.	26
Table 4.6.	Contribution and effective cost β values for AG2.	27
Table 5.1.	Attributes of randomly generated attack graphs.	29
Table 5.2.	Features of randomly generated attack graphs for cost comparison.	32
Table A.1.	System characteristics of AG2 used as second example attack graph.	37
Table B.1.	Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of <i>BCNHA</i> for randomly generated attack graphs with path length threshold $n = 5$	39

Table B.2.	Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of <i>BCNHA</i> for randomly generated attack graphs with path length threshold $n = 10$	39
Table B.3.	Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of <i>BCNHA</i> for randomly generated attack graphs with path length threshold $n = 15$	39
Table B.4.	Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of <i>BCNHA</i> for randomly generated attack graphs with path length threshold $n = 20$	40
Table B.5.	Average protection cost of <i>BCNHA</i> , optimal solution and optimal solution with only initial conditions for randomly generated attack graphs.	40
Table B.6.	Average CPU time in millisecond for <i>BCNHA</i> , optimal solution and optimal solution with only initial conditions for randomly generated attack graphs.	40

LIST OF SYMBOLS

A_j	An attack path
$ A_j $	Number of attack paths
$Aprob(A_j)$	Likelihood of an attack path
$Avg. A_j $	Average number of attack paths
$Avg. Edges $	Average number of edges
$Con(E_i)$	Contribution of an exploit
$Con(I_k)$	Contribution of an initial condition
$Con(x)$	Contribution of x where x can be an exploit or an initial condition
E_i	An exploit
$ E_i $	Number of exploits
$Eprob(E_i)$	Likelihood of an exploit
I_k	An initial condition
$ I_k $	Number of initial conditions
M	The maximum number of exploits which point to a condition in the attack graph
n	The path length threshold
α	Cost of removal for exploits and initial conditions
β	Effective cost
γ	Remaining budget
θ	Exploit Diversity Constant
φ	For an exploit E_i , φ shows number of exploits whose type is equal to E_i and exploited before E_i in an attack path

LIST OF ACRONYMS/ABBREVIATIONS

<i>AC</i>	Access Complexity
<i>AG</i>	Attack Graph
<i>Au</i>	Authentication Metric
<i>AV</i>	Access Vector
<i>BCNHA</i>	Budget Controlled Network Hardening Algorithm
<i>CVE</i>	Common Vulnerabilities and Exposures
<i>CVSS</i>	Common Vulnerability Scoring System
<i>DNF</i>	Disjunctive Normal Form
<i>F</i>	Functional
<i>H</i>	High
<i>IDS</i>	Intrusion Detection System
<i>ND</i>	Not Defined
<i>NVD</i>	National Vulnerability Database
<i>P</i>	Proof-of-concept
<i>ROBDD</i>	Reduced Ordered Binary Decision Diagram
<i>SM</i>	Security Metric
<i>U</i>	Unproven

1. INTRODUCTION

From communications to finance, computer networks have become an indispensable part of our daily lives. To provide robust, trusted and dependable services, security is a vital requirement for all types of information and communication systems. Therefore, vulnerabilities of networks must be identified to avoid disruption or failure of network-based services. The criticality of these infrastructures has led to the huge body of research focusing on cybersecurity. Accordingly, analysis of security vulnerabilities and how to avoid them are important topics.

In practice, there are various vulnerability scanners such as Nessus [1] and Snort [2] which detect vulnerabilities of each element of an investigated network in isolation. Since such tools do not give any information about interaction of vulnerabilities, attackers can combine different vulnerabilities to breach a seemingly well-guarded network and then can disrupt it. Therefore, the information of not only isolated vulnerabilities, but also the interactions among them are essential to effectively protect the network from multistep attacks which are combination of correlated vulnerabilities exploited one after other to reach intended target(s).

Attack graphs provide crucial information to prevent the multistep attacks since they show all possible sequences of vulnerabilities and their relationships [3, 4]. In other words, they reveal all potential threats against networks by showing all possible attack paths. An attack graph can be a useful tool to extract recommendation in order to harden network nodes against inherent vulnerabilities [5–18]. It can also be instrumental to reckon how secure a network is [19–22]. However, as the number of hosts and vulnerabilities increase, an attack graph can become huge and very complex for a security analyst to find which vulnerabilities should be removed. According to Verizon’s 2012 Data Breach Investigations Report, “97% of the breaches could be prevented by taking small measures since most of them were launched by script kiddies who do not have much skills and resources” [23]. Hence, automation of recommendation extraction from attack graphs and their user accessibility have a practical importance.

While eliminating vulnerabilities to prevent multistep attacks, one should also consider the cost of removal since patches for different remedies usually incur different costs. Therefore, a critical question is how we can minimize the cost of protection while removing all attack paths or reducing their success probabilities since cost of protection is generally constrained by a limited budget [24]. Moreover, removing all identified vulnerabilities can be infeasible due to lack of known patches (zero-day attacks), resource limitations, and operational constraints [25]. On the theoretical side, finding optimal minimum cost solution is an intractable problem [5].

In this thesis, we present a cost-aware network hardening solution with limited budget by using compact attack graphs. Firstly, all attack paths to critical resources are extracted from the attack graph and their success probabilities are calculated. The security analyst can set length and likelihood threshold to exclude attack paths whose lengths exceed the length threshold or success probability is lower than the likelihood threshold. Our algorithm uses *Common Vulnerability Scoring System (CVSS)* [26, 27] exploitability scores while calculating success probabilities of attack paths. Then, security of the network is measured with a novel method by considering success probabilities of attack paths. This accompanying feature is beneficial for network security assessment and situation awareness. Subsequently, the most critical vulnerability is selected according to cost and its contribution to the attack paths. This selection process continues iteratively until the network is completely secure or allocated budget is consumed.

The rest of the thesis is organized as follows. The next chapter introduces essential background for attack graphs and reviews related work on attack graphs and network hardening. Our novel network hardening method, *Budget Controlled Network Hardening Algorithm (BCNHA)*, is described in Chapter 3 while Chapter 4 provides two example attack graphs to illustrate *BCNHA* operation. In Chapter 5, experimental results are presented to examine the time complexity and minimum cost efficiency of *BCNHA*. Finally, Chapter 6 discusses future research directions and concludes the thesis.

2. RELATED WORK

In this chapter, we give background information about attack graphs before utilizing them for network hardening. We also review related work on network hardening using attack graphs. Multi-stage and multi-host attacks must be considered to analyze the security of a network effectively since an attacker can penetrate the network by hopping from one machine to another. Attack graphs are used enumerating all possible attack sequences to show multi-stage and multi-host attacks. Therefore, it can be used for intrusion alert correlation, as part of an *Intrusion Detection System (IDS)*, penetration testing and so on [28–31].

Domain knowledge and vulnerabilities are necessary information to generate an attack graph for a network. In the past, security experts called Red Teams used to produce attack graphs manually. However, this process is error-prone and impracticable for large networks. As a first formal method, Sheyner *et al.* use model checking to construct attack graphs of a network [12]. Only state-based attack graphs can be generated by these earlier works for just small network with size of tens [12, 14, 16, 32–37]. Some of these earlier methods use symbolic model checker (logic-based) approaches [12, 14, 34, 37], while others use graph-based approaches [16, 32, 36]. These methods can use forward algorithm logic by starting from attacker node [34, 37] or backward algorithm logic by starting from the target nodes [12, 14].

Attack graphs show possible attacks against a computer network in detailed settings. There are different types of attack graphs [38] but two of them are generally used in network protection: *state-based attack graphs* and *compact or exploit-based attack graphs*. In the former, nodes represent a state of network and show each step of attacks explicitly [14]. However, state-based attack graphs suffer from scalability issues since they grow exponentially with increasing number of hosts and vulnerabilities.

Since state-based attack graphs suffer from scalability issues, a compact representation of attack graph was proposed under monotonicity assumption [13, 35, 39–41].

Monotonic logic assumes that an attacker never relinquishes any obtained capabilities. This type has polynomial complexity in terms of total number of vulnerabilities and security conditions [11]. Hence compact attack graphs can be generated for large enterprise networks with size of thousands. Considering the advantage it offers in terms of complexity, we utilize compact attack graphs for security modeling and analysis in our work. Lots of different attack graph types [38] have also been proposed like multiple prerequisite attack graph [41], coordinated attack graph [42], host-compromised attack graph [43] and host-based network attack graph [44].

Compact attack graphs are directed graphs which consist of two node types: *exploits* and *security conditions* [8]. An exploit is a vulnerability from which an attacker can benefit to obtain higher privileges. Security conditions show required preconditions or post conditions for exploits. There are no direct edges between exploits and between security conditions [5]. However, exploits and security conditions are connected with two types of directed edges. First type of edges points to an exploit from a security condition. It represents a require relation such that execution of the exploit requires that the security condition must be satisfied. Second type of edges point to a security condition from an exploit. This type represents an imply relation such that execution of the exploit will satisfy the security condition. Attack graphs are formally described in Definition 1.1 [5]. We shall also assume this notion of attack graphs in our work.

Definition 1.1. Given a set of exploits E , a set of security conditions S , a require relation $R_r \subseteq S \times E$, and an imply relation $R_i \subseteq E \times S$, an attack graph AG is the directed graph $AG(E \cup S, R_r \cup R_i)$ ($E \cup S$ represents the vertex set and $R_r \cup R_i$ represents the edge set) [5].

A sample network is shown in Figure 2.1 [12,14]. There are three hosts in this network: one attacker, *host 0*, and two victim machines, *host 1* and *host 2*. Table 2.1 shows exploit types and explains how they work [12]. Attack graph (AG1) of that sample network is shown in Figure 2.2. For the identifiers, numbers in parenthesis specify related hosts. For example, *root(2)* indicates that attackers have root privilege on *host 2* and execution of *ftp.rhosts* vulnerability from *host 0* to *host 1* is shown as

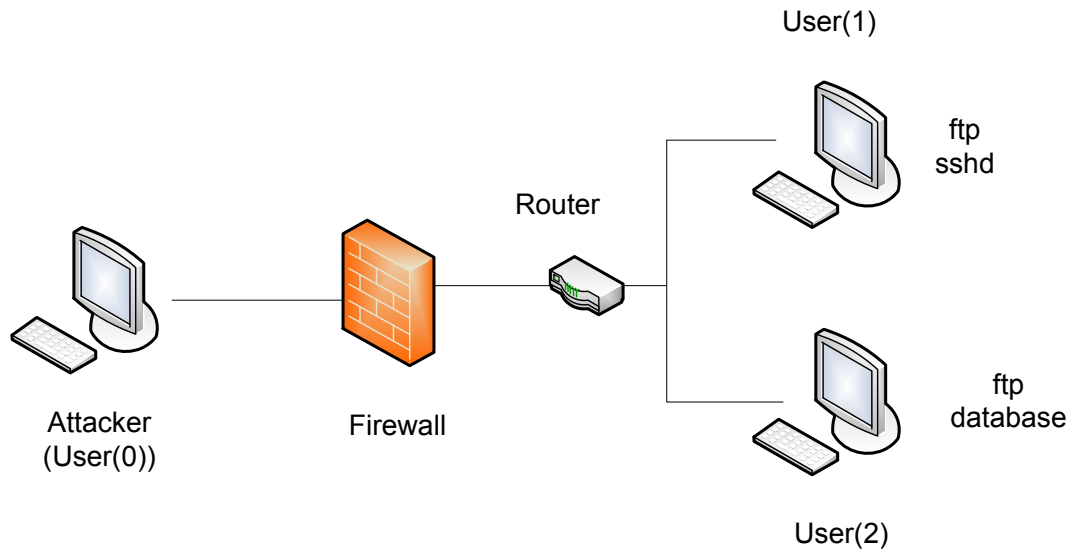


Figure 2.1. The network model of AG1.

$ftp_rhosts(0,1)$. In that figure, rectangles represent security conditions while rounded boxes represent exploits.

Table 2.1. Explanations of exploits.

Exploit Type	Explanation
<i>sshd buffer overflow (sshd_bof)</i>	This attack instantly gives a remote user a root shell on the target machine.
<i>ftp .rhosts (ftp_rhosts)</i>	Using ftp vulnerability, the intruder creates an <i>.rhosts</i> file in the ftp home directory, creating a remote login trust relationship between his machine and the target machine.
<i>remote login (rsh)</i>	Using existing remote login trust relationship between two machines, the intruder logs in from one machine to another, getting a user shell without supplying a password.
<i>local buffer overflow (local_bof)</i>	If the intruder has acquired a user shell on the target machine, the next step is to exploit a buffer overflow vulnerability on a <i>setuid root</i> file to gain root access.

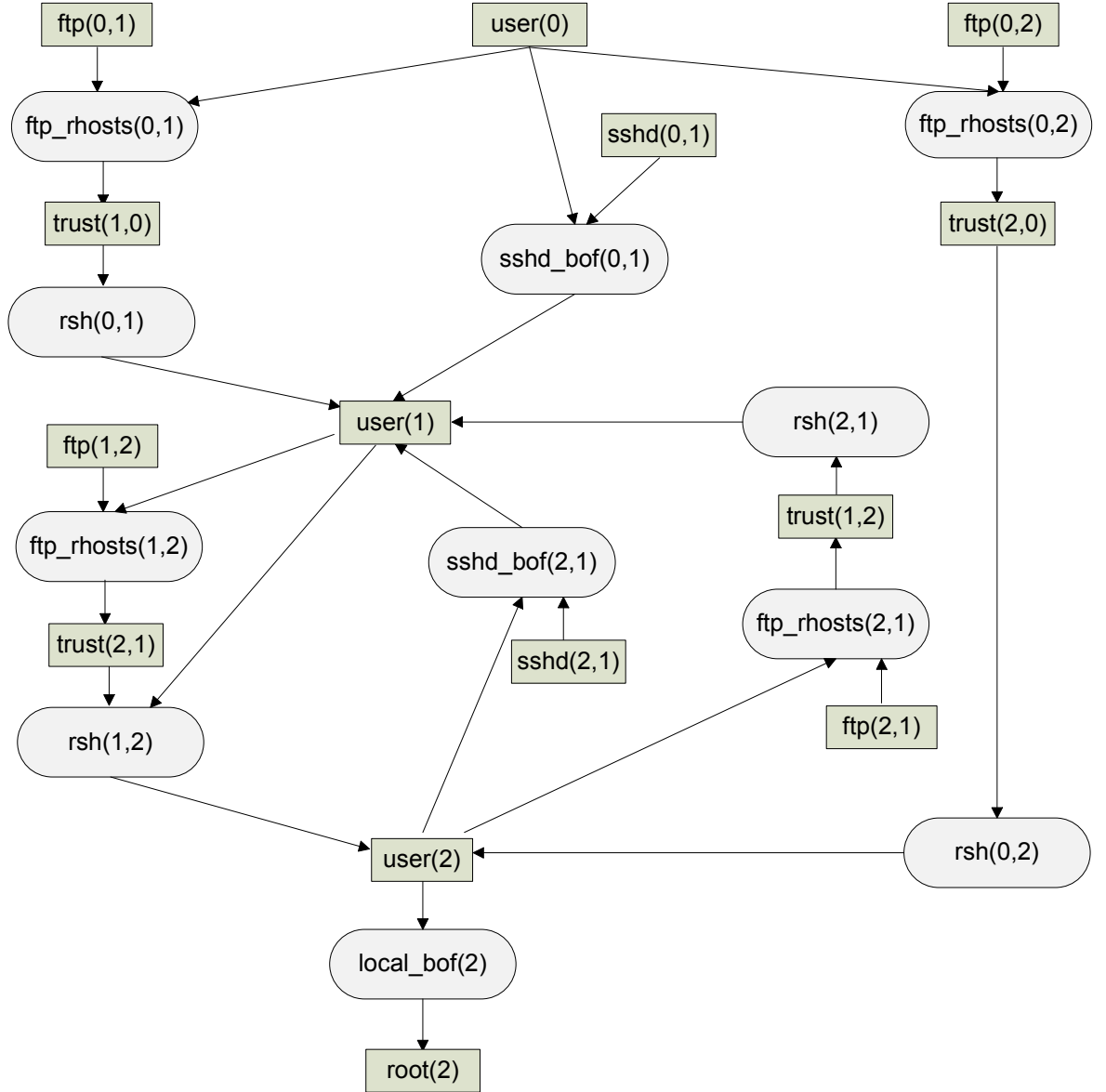


Figure 2.2. An attack graph (AG1) with four exploit types, 11 exploits and seven initial conditions.

Security conditions have two different types: *initial conditions* and *intermediate conditions*. An initial condition is not a post-condition of any exploits but a necessary precondition for them. Accessibility rules of hosts or network configuration can be examples of initial conditions like $user(0)$ and $ftp(0,1)$ in Figure 2.2. Intermediate conditions are both preconditions and post conditions of some exploits. Condition $trust(1,0)$ in Figure 2.2 represents a trust relationship between the attacker $user(0)$ and $host\ 1$ which is $user(1)$, and is an example of intermediate condition.

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

There are two ways to eliminate an attack path. First one is to disable one of the security conditions in the attack path. However, intermediate conditions are typically not controlled directly by the security analyst [5]. Hence initial conditions are left as the only option to disable security conditions. Patching one of the exploits in the attack path is the second option. However, we have to keep in mind that both initial conditions and exploits cannot be fixed in all cases. Like zero-day attacks, patches are not available for all known vulnerabilities. Furthermore, disabling an initial condition may seriously affect service availability of the system. As an example, disabling $ftp(0,2)$ by stopping ftp service can be unacceptable. Hence, our approach *BCNHA* allows marking of designated exploits and initial conditions such that they are ignored when selection for network hardening is performed. In other words, security analyst can mark exploits and initial conditions which cannot be removed from the system due to some constraint or reason.

There are various works focusing on attack graphs and network hardening in the literature. In [5], initial security conditions (initial conditions) are removed in an attack graph for network hardening. Their proposed solution traverses attack graph with a backward algorithm. Then initial conditions are viewed as Boolean variables and represented as a logic proposition. The logic proposition is converted to *disjunctive normal form (DNF)* and conjunctive clauses of *DNF* have all potential solutions. However, *DNF* conversion may have exponential explosion in terms of solution space. In [6], Yoda *et al.* calculate criticality degree for each node in the attack graph. The nodes with higher degrees are selected for removal. However, [5] and [6] do not consider the cost factor while selecting conditions or exploits to remove.

Islam *et al.* propose a heuristic approach for minimum cost network hardening [7]. They aim to select initial conditions which are disabled to protect the network. But they do not consider success probabilities of attack paths in their cost function. If two initial conditions have the same cost of removal, the one which participates in more attack paths is selected. However, initial condition which participates in attack paths with higher success probability is a more suitable candidate for removal. Therefore, our approach *BCNHA* considers success probabilities of attack paths while selecting the best option.

In [8], all n -valid attack paths which are attack paths with distance n and contain no loops are extracted from an attack graph. Then the security metric of the network is calculated according to the path length and the number of different exploits in all paths. However, this security metric does not provide much information about the success probability of an attack since long attack paths can be easier to breach. Considering this fact, we calculate network security metric according to success probabilities of attack paths so that a security analyst can assess potential threats more easily. In [8], initial conditions are removed according to the cost and the number of participated attack paths.

Jun-chun *et al.* use genetic algorithms to find minimum cost network hardening solution by transforming it into a non-constrained optimization problem with penalty in

[9]. They use bidirectional-based search strategy to explore the relationship of network vulnerabilities. In this way, the efficiency of attack graph generation is improved. The proposed method guarantees the network security with the least cost. Attack graphs are converted to *Reduced Ordered Binary Decision Diagram (ROBDD)* in [10]. Then Chen *et al.* find the minimum cost solution from this new representation which does not require to list every possible solution. Hence, complexity of the algorithm is reduced. The minimum cost solution is found by a recursive algorithm which is a depth-first search of *ROBDD*.

Although, [5–10] focus only on initial conditions, it is possible to remove an exploit by patching it to protect network against known attacks [11]. In [12–14], minimal critical attack set is found to harden the network since finding minimum critical attack sets which leads to optimum solution is NP hard. The minimal critical attack set is a minimal set of exploits in the attack graph whose removal eliminates all the attack paths which reach target assets. Due to complexity, these approaches are heuristic based and only exploits are considered for removal. In [12, 14], model checking techniques are used to generate and analyze attack graphs. These techniques produce attack graphs with a very large number of states which also include many redundant states. Thus, they suffer from scalability while finding minimal critical attack set. In our work, both initial conditions and exploits are considered jointly to protect network with minimum cost.

Keramati *et al.* introduce a framework for minimum cost network hardening in [11]. Initially, all attack paths are extracted from attack graphs. The priority of an attack path is calculated by predefined rules and attack paths with highest priority are selected for removal. The drawback of this approach is that some attack paths are not removed, which makes the network insecure. Additionally, it suffers from exponential complexity since it uses the method in [5] to extract the critical set.

In [15], security analysts determine all possible countermeasures to protect network. Genetic algorithm is used to find minimum cut-set in attack graph according to determined countermeasures. However, security analysts cannot determine all the

countermeasures since there can be millions of them in an large attack graph. Hence, our algorithm extracts all possible countermeasures from attack graph.

There are also some works considering the budget aspect in the literature. In [16] and [17], network hardening solutions with budget consideration are proposed. However, these solutions require that a set of possible defenses with specified cost and expected benefit is given by a security analyst. In [16], a heuristic algorithm which considers benefit and cost together is utilized to find optimal solution. Also Phillips *et al.* offer a method to generate attack graph of a network and calculate security risk of the network. Minimum cost solution is transformed to a binary knapsack problem in [17] which has exponential complexity. They choose hardening measures according to security metrics and cost. The proposed method guarantees optimal solution and utilize dynamic programming to achieve this.

Sawilla *et al.* try to minimize connectivity in *AND/OR* directed dependency attack graph by using partial cuts [18]. AssetRank [45] is applied to produce a rank metric for each asset in the network. A k -connected graph is used to find critical attack paths. A k -connected graph is a graph where k is the smallest number of vertices whose removal disrupts the graph. Then they try to eliminate critical attack paths according to cost and rank metric with budget consideration. However, they do not consider likelihoods of attack paths when determining critical attack paths. Additionally, their solution does not provide any security metric to security analyst to show effect of network hardening or current state of it. Moreover, usage of AssetRank to find critical paths is a costly operation in terms of complexity and there is no complexity data provided about that aspect in the work.

Overall, [5–15] have a similar drawback compared to our approach. Usually, it is difficult to completely secure a network because of resource limitations. Since our proposal works iteratively, it can be stopped when allocated budget is depleted. Therefore, our scheme lets the security analyst balance security vs. cost tradeoff according to context-based requirements. This is critical for practical network hardening capabilities in network security. In [16–18], although budget constraint is considered, the

presented solutions do not provide fundamental network hardening features such as security metric. Furthermore, their methods have drawbacks regarding their usage in real networks as described above.

3. BUDGET CONTROLLED NETWORK HARDENING ALGORITHM (*BCNHA*)

By traversing on attack graph, we can find all attack paths which end in given critical assets of a network. If all these attack paths are removed, the network assets become secure against attacks. There are plenty of options while eliminating all attack paths since different exploits and initial conditions can be selected to achieve this goal. However, we need to consider the cost factor of elimination to harden network with minimum cost. Some exploits or initial conditions may also be unsuitable for removal as discussed in Chapter 2. On the other hand, cost consideration induces budget constraint since cost of making network completely secure can be unacceptable. However, likelihood of attacks against critical resources of networks can be reduced even if it cannot be totally eliminated due to budget limitations. Hence, the security metric of network is measured to determine how secure the network is and then a security analyst can decide how much protection needs to be performed.

In this thesis, we propose a network hardening solution, namely *Budget Controlled Network Hardening Algorithm (BCNHA)*, with minimal cost by considering budget and cost factor in an iterative way. The flowchart of *BCNHA* is shown in Figure 3.1. Accordingly, Table 3.1 shows parameters used in our algorithm and analysis. Our proposal consists of two phases. In first phase, all attack paths are extracted from the attack graph with a backward algorithm. Meanwhile, success probabilities of all attack paths are computed. Then security metric SM is calculated. In second phase, the contributions of each exploit and initial condition to attack paths are calculated. According to cost of removal α and the contribution to attack paths, an exploit or an initial condition is selected for removal. Finally, attack paths which include selected exploit or initial condition are removed and security metric is re-calculated. This second phase continues until network is “completely” secure which means that there is no attack path for an intruder to breach the critical assets, or the available budget is depleted.

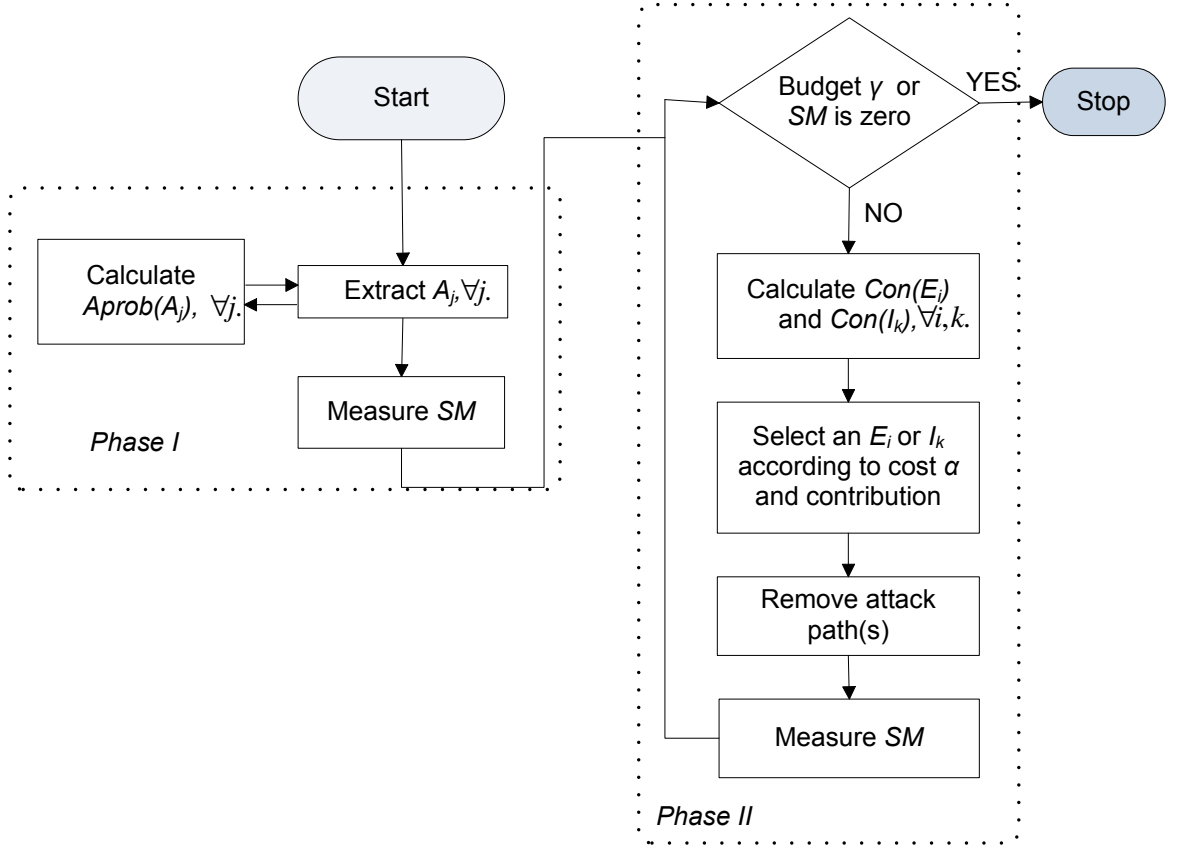


Figure 3.1. The flowchart of *BCNHA*.

To find all attack paths which terminate at given critical resources, a backward algorithm is used in the first phase of *BCNHA*. Thus, exploits which cannot reach the target are never explored. We also benefit from forward algorithm logic and discard attack paths which do not start from the attacker node. Therefore, the complexity of second phase is reduced since only necessary and useful attack paths are considered for elimination. Attack graphs can also have cyclic paths. However, intruders do not usually choose cyclic paths as noted in [8]. In backward algorithm, while extracting one attack path, if an exploit which was already included by that attack path is visited again, extraction of the path is stopped to avoid a cyclic path.

Similar to cyclic paths, complex attack paths would not be preferred by attackers. Typically, attackers are not expected to use attack paths which take more than four steps [8, 46]. In *BCNHA*, while extracting attack paths, a security analyst can assign length and success probability thresholds for attack paths. During the extraction of attack paths with backward algorithm, if the length of an attack path exceeds given

Table 3.1. Analysis parameters.

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

length or its likelihood is lower than the likelihood threshold, extraction of that path is halted. Hence, complexity of Phase I is reduced by not traversing for complex and unlikely attacks paths. Additionally, the complexity of Phase II is also reduced since the number of attack paths which passes to Phase II is decreased. Moreover, a security analyst can prevent easier and thus more likely intrusions by considering them in the first place like as suggested in Verizon’s 2012 Data Breach Investigations Report [23].

Each vulnerability has different difficulty level for an attacker to exploit. Therefore, exploits have different likelihoods of successful occurrence which could be estimated by average time or computational complexity required to breach the network [20]. However, this estimation task is not trivial for a non-expert person. Hence,

an alternative approach based on empirical data can be adopted and success probabilities of exploits $Eprob(E_i)$ can be extracted from *Common Vulnerability Scoring System (CVSS)* [27] exploitability scores [11,21,22]. *CVSS* provides two exploitability scores for every known and reported vulnerability such as for every vulnerability in NVD (National Vulnerability Database) [47]. First exploitability score is a subscore of base metric group and it consists of *access vector (AV)*, *access complexity (AC)* and *authentication (Au)* metric. Second one is a subscore of temporal metrics. Both of these exploitability scores can be used separately to find the success probability of an exploit.

Table 3.2 shows *CVSS* attributes used for base metric exploitability scores [27]. Base exploitability score is calculated as:

$$Exploitability\ Score = 20 * AV * AC * Au, \quad (3.1)$$

Base exploitability score range from 1.2 to 10. Hence we can use base exploitability scores as $Eprob(E_i)$ by normalizing it with division by 10. As an example, base exploitability score of *local_bof* is 10.0 since its *AV* is exploitable from *Network (N)* called as remotely exploitable, *AC* is *Low (L)* and *Au* is *None (N)* which means attackers do not need to authenticate. Therefore, $Eprob(E_i)$ of *local_bof* becomes 1.0 if we use base exploitability score to obtain $Eprob(E_i)$ of *local_bof*.

Temporal metrics are changed over time since exploits evolve, are disclosed and automated or patches for them are developed. The temporal exploitability score shows the current state of exploit techniques or code availability. Table 3.3 shows values and their corresponding scores for temporal metric exploitability scores [27]. We can use temporal exploitability scores directly since they are already in the range [0.0,1.0]. As an example, temporal exploitability score of *ftp_rhost* is 0.85 since its value is *Unproven (U)* which means that there is no exploitation code for it or it is a theoretical exploit.

We use success probabilities of exploits $Eprob(E_i)$ to calculate likelihood of attack paths. Easiness of attack path execution is a more appropriate criterion than path

Table 3.2. *CVSS* attributes used to calculate base metric exploitability scores.

CVSS attribute	Value	Score
Access Vector (AV)	Local (L)	0.395
	Adjacent Network (A)	0.646
	Network (N)	1.0
Access Complexity (AC)	High (H)	0.35
	Medium (M)	0.61
	Low (L)	0.71
Authentication (Au)	Multiple (M)	0.45
	Single (S)	0.56
	None (N)	0.704

Table 3.3. Temporal exploitability score values and descriptions.

Value	Description	Score
Unproven (U)	No exploit code is available or the exploit is entirely theoretical.	0.85
Proof-of-concept (P)	Proof of concept exploit code or an attack demonstration that is not practical for most systems is available. The code or technique is not functional in all situations.	0.90
Functional (F)	Functional exploit code is available. The code works in most situations where the vulnerability exists.	0.95
High (H)	The vulnerability can be exploited by automated code and mobile code like a worm or virus.	1.0
Not Defined (ND)	A signal to ignore this score.	1.0

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

attack path.

Exploit diversity in an attack path also affects the likelihood of that attack path since, when an attacker exploits a vulnerability while executing an attack path, it is more likely to have a higher success probability for exploiting the same vulnerability type. Let's consider attack path "*ftp_rhosts(0,1) → rsh(0,1) → rsh(1,2) → local_bof(2)*" for instance. When attacker exploits *rsh(0,1)*, it is likely that *rsh(1,2)* is exploited more easily. In other words, successful exploitation of *rsh(0,1)* increases the success probability of *rsh(1,2)* since attacker has already required knowledge to exploit the *rsh* vulnerability. Exploit diversity constant θ is a parameter to adjust this success probability increase according to system requirement. In *BCNHA*, security analyst can assign values between 0.0 and 1.0 to θ which has default value 0.0. Default value zero implies that success probability of an exploit is not changed according to diversity. As θ increases, exploiting the same vulnerability type while executing an attack path becomes more and more easier. If θ is assigned to 1.0, when attacker exploits a vulnerability in an attack path, success probability of that vulnerability type becomes 1.0 which means that attacker can always exploit an exploit type successfully after a single successful exploitation of that vulnerability type.

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

$$Aprob(A_j) = \prod min(1.0, (Eprob(E_i) + \theta * \varphi * (1 - Eprob(E_i)))) \quad (3.2)$$

where $E_i \in A_j$, *min* is the minimum function and φ is a variable which shows the number of exploits whose type is equal to E_i and exploited before E_i .

At the end of algorithm's first phase, security metric *SM* is calculated to determine security risk of the network. *SM* indicates how vulnerable the network is against threats. Hence, the security analyst can understand security status of the network more clearly and take necessary actions like demanding more budget for defense and hardening. In our proposal, likelihoods of extracted attack paths are added to compute *SM*. In this way, number of attack paths and their likelihoods are combined to create

our security metric. We define security metric SM as:

$$SM = \sum Aprob(A_j), \quad (3.3)$$

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

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

$$Con(E_i) = \sum Aprob(A_{j_m}), \quad (3.4)$$

where $E_i \in A_{j_m}$, and

$$Con(I_k) = \sum Con(E_i), \quad (3.5)$$

where I_k enables E_i .

Cost of removal α represents patching cost for exploits and disabling cost for initial conditions. These costs can originate from diverse actions or security investments such as buying a new firewall, stopping a service or upgrading a server. Hence, it is rather context-dependent and assumed to be estimated by security experts and given

as input into the algorithm. Cost of removal α and contribution $Con(x)$ are unified as the effective cost β . Effective cost simply shows how much security risk is reduced per cost. With higher contribution rate, one unit cost provides more reduction in security risk or security metric SM . Effective cost β is defined as:

$$\beta = \alpha / Con(x), \quad (3.6)$$

where x can be an exploit or an initial condition.

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

$$\min(\beta) \wedge \alpha \leq \gamma \quad (3.7)$$

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

4. SAMPLE ATTACK GRAPHS FOR *BCNHA* OPERATION

In this section, we utilize two network examples, AG1 and AG2, in the literature to elaborate on the operation of *BCNHA*. The first one is a well-known network example in the study of attack graphs [5, 8, 11, 48]. The second network example has been proposed in [21]. Network model and system details for AG2 are given in Appendix A.

4.1. Example Attack Graph I

Attack graph of sample network 1 (AG1) is shown in Figure 2.2. Rectangles represent security conditions while rounded boxes represent exploits in the figure. More details can be found in [12, 14]. There are 11 exploits and seven initial conditions in the attack graph. Table 4.1 shows the success probability and the patching cost for each exploit type. All initial conditions have the same disabling cost, which is 10 units and *user(0)* cannot be disabled since it is an attacker. The budget allocated for network protection is 25 units. Success probability values are extracted from *CVSS* base exploitability scores as discussed in Chapter 3. Budget and cost of removal values are assumed to be given by security experts. However, in this example attack graph, budget and cost of removal values are assigned by us appropriately and randomly in order to illustrate the operation of *BCNHA* more clearly. The unit for cost values can be a money currency such as US dollars or average time as man-hours or some other representation of cost according to system requirements.

Attack path length threshold n and attack path likelihood threshold are set to four and 0.01 for this example graph, respectively. Also diversity constant θ is assigned as 0.5. Table 4.2 shows four attack paths for intruders which try to obtain root privileges on *host 2* and their corresponding likelihoods. The backward algorithm finds nine attack paths. However, four of them, one example being $ftp_rhosts(2,1) \rightarrow rsh(2,1) \rightarrow rsh(1,2) \rightarrow local_bof(2)$, are excluded since they do not start from the attacker node (*user(0)*). Thus, our algorithm combines advantages of both backward and forward

Table 4.1. Success probability and cost value for each exploit type in AG1.

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

Table 4.2. Attack paths in AG1 and their success probabilities.

Attack Paths A_j	Success Probability of A_j $Aprob(A_j)$
ftp_rhosts(0,1) $\rightarrow$ rsh(0,1) $\rightarrow$ rsh(1,2) $\rightarrow$ local_bof(2)	0.576
sshd_bof(0,1) $\rightarrow$ ftp_rhosts(1,2) $\rightarrow$ rsh(1,2) $\rightarrow$ local_bof(2)	0.512
sshd_bof(0,1) $\rightarrow$ rsh(1,2) $\rightarrow$ local_bof(2)	0.640
ftp_rhosts(0,2) $\rightarrow$ rsh(0,2) $\rightarrow$ local_bof(2)	0.640

algorithm. Also $ftp_rhosts(0,1) \rightarrow rsh(0,1) \rightarrow ftp_rhosts(1,2) \rightarrow rsh(1,2) \rightarrow local_bof(2)$ is excluded since its length is five. In this case, attack path likelihood threshold does not help to reduce number of attack paths since likelihood of all attack paths is greater than 0.01.

The success probability of an attack path can be calculated by multiplying success probabilities of all exploits which are included in that attack path. For instance, likelihood of first attack path A_1 is $Aprob(A_1) = 0.8 \times 0.8 \times (0.8 + (1 - 0.8) * 0.5) \times 1.0 = 0.576$. Attack path A_1 includes two *rsh* ($rsh(0,1)$ and $rsh(1,2)$) vulnerabilities. Hence, success probability of second *rsh* vulnerability ($rsh(1,2)$) increases to $(0.8 + 1 * (1 - 0.8) * 0.5) = 0.9$ because of diversity constant θ being equal to 0.5. If there were another *rsh* vulnerability in this attack path, its success probability becomes $(0.8 + 2 * (1 - 0.8) * 0.5) = 1.0$. Therefore, θ is a parameter to measure the necessary number of successful exploitations of an vulnerability type in an attack path for the attacker to have 1.0 success probability for that vulnerability type. For instance, θ being equal to 0.5 implies that attackers can exploit a vulnerability type with 1.0 success probability after exploiting that vulnerability type twice in an attack path.

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

Table 4.3. Contribution and effective cost β values for AG1.

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

In first iteration, effective cost β is calculated for each exploit and initial condition. Then an exploit or initial condition which has minimum effective cost and a cost of removal smaller than remaining budget is selected for removal. Contribution of an exploit is measured by summing likelihood of attack paths which include that exploit. Contribution of an initial condition is measured by summing likelihood of

attack paths which include the exploit(s) enabled by the initial condition. To calculate β , cost of removal α is divided by that contribution. Table 4.3 shows effective cost and contribution values of exploits and initial conditions for each step. If an exploit or an initial condition does not contribute to SM , which means that it is not used by attacker, its effective cost is not calculated. For instance, contribution of $ftp_rhosts(0,2)$ is 0.64 since only A_4 include it. Hence, the effective cost of exploit $ftp_rhosts(0,2)$ is equal to $5/0.64 = 7.8125$. The cost of $ftp_rhosts(0,2)$ is not larger than the budget and it has the minimum effective cost. Hence, $ftp_rhosts(0,2)$ is selected for removal and attack path A_4 are removed. Budget γ reduces to $25 - 5 = 20$ and new SM value is $0.576 + 0.512 + 0.64 = 1.728$. Then SM and γ are not zero and we go on with the second iteration.

In the second iteration, contribution of exploit $sshd_bof(0,1)$ which is included by A_2 and A_3 is $0.512 + 0.640 = 1.152$. Since $sshd(0,1)$ enables only $sshd_bof(0,1)$, it has the same contribution. Hence, the effective cost of $sshd(0,1)$ is equal to $10/1.152 = 8.6805$. The cost of $sshd(0,1)$ is not larger than the budget and it has the minimum effective cost. Hence, $sshd(0,1)$ is selected for removal, and A_2 and A_3 which include $sshd_bof(0,1)$ are removed. Budget γ reduces to $20 - 10 = 10$ and new SM is 0.576 (likelihood of A_1). Since SM and γ are not zero, we go on with the third iteration. In third iteration, $ftp_rhosts(0,1)$ is selected for removal. Consequently, with total cost of 20, we harden network such that there is no attack path for an intruder to breach *host 2* with root privilege. If only initial conditions are taken into account while protecting network as in [5–10], the hardening cost would be 30 instead of 20 whereas using only initial condition provides additional gain of completely securing of network elements. However, these former approaches are not adaptive and do not let the security analyst control the cost of hardening in a flexible way, i.e. cost-aware and limited budget operation focusing on the most critical assets. Therefore, considering exploits for removal as in our technique helps us to converge to minimum cost requirement in a budget-aware manner.

Let us consider what happens if budget is 12. The first iteration is the same except now the budget reduces to $12 - 5 = 7$. Hence, in second iteration, $sshd(0,1)$

Table 4.4. Success probability and cost value for each exploit type in AG2.

<i>Exploits E_i</i>	Success Probability $Eprob(E_i)$	Cost α
IIS_bof	0.90	25
squid_port_scan	0.95	10
ftp_rhosts	0.85	5
netbios_ssn_nullsession	0.95	4
ssh_bof	0.85	12
rsh	1.0	15
LICQ_remote_to_user	1.0	10
local_setuid_bof(3,3)	0.95	25

cannot be chosen this time since the cost of $sshd(0,1)$ is 10. Therefore, $ftp_rhosts(0,1)$ is chosen for removal and A_1 is removed. New SM is $0.512 + 0.64 = 1.152$ and there are two attack paths (A_2 and A_3) to breach. However, there is no more hardening option left since the remaining budget is only 2. In this case, $BCNHA$ finds a favorable budget-limited network hardening solution without securing critical assets completely.

4.2. Example Attack Graph II

We use network configuration #2 (AG2) in [21] as our second network example. This network is shown in Figure 4.1. Network characteristics and other details can be found in [21]. There are 12 exploits and 11 initial conditions in the attack graph. Table 4.4 shows the success probability and the patching cost for each exploit type. Cost of removal for initial conditions is 10 units. $user(0)$ and $IIS(0)$ cannot be disabled. The budget allocated for hardening is 25 units. Success probability values are extracted from CVSS temporal exploitability scores as discussed in Section 3 and cost of removal values are again assigned by us.

Attack path length threshold is set to five for this example graph. Diversity constant θ is assigned to 1.0. Table 4.5 shows three attack paths for intruders which try to obtain root privileges on *host 3* and their corresponding likelihoods. Attack path $IIS_bof(0,0) \rightarrow netbios_ssn_nullsession(0,2) \rightarrow ftp_rhosts(2,1) \rightarrow rsh(2,1) \rightarrow$

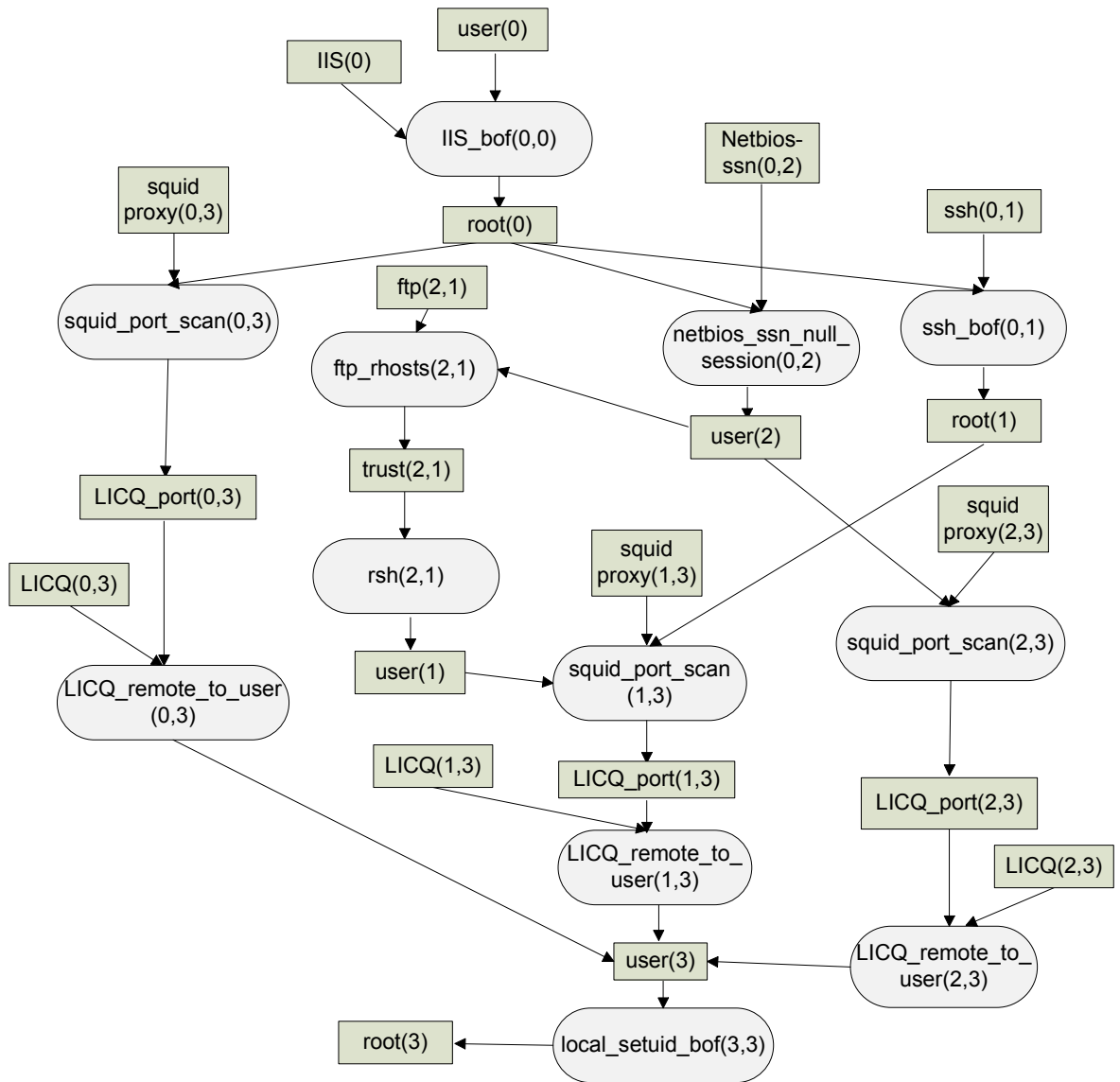


Figure 4.1. Second attack graph (AG2) with eight exploit types, 12 exploits and 11 initial conditions.

Table 4.5. Attack paths in AG2 and their success probabilities.

Attack Paths A_j	$Aprob(A_j)$
IIS_bof(0,0) $\rightarrow$ squid_port_scan(0,3) $\rightarrow$ LICQ_remote_to_user(0,3) $\rightarrow$ local_setuid_bof(3,3)	0.8122500
IIS_bof(0,0) $\rightarrow$ ssh_bof(0,1) $\rightarrow$ squid_port_scan(1,3) $\rightarrow$ LICQ_remote_to_user(1,3) $\rightarrow$ local_setuid_bof(3,3)	0.6904125
IIS_bof(0,0) $\rightarrow$ netbios_ssn_nullsession(0,2) $\rightarrow$ squid_port_scan(2,3) $\rightarrow$ LICQ_remote_to_user(2,3) $\rightarrow$ local_setuid_bof(3,3)	0.7716375

squid_port_scan(1,3) $\rightarrow$ LICQ_remote_to_user(1,3) $\rightarrow$ local_setuid_bof(3,3) is excluded since its length is more than five. In this case, θ does not affect likelihood of attack paths since extracted attack paths do not include any vulnerability type more than once.

In the beginning, SM equals to $0.81225 + 0.6904125 + 0.7716375 = 2.2743$ which corresponds to $Aprob(A_1) + Aprob(A_2) + Aprob(A_3)$. Phase II of *BCNHA* is performed since SM and budget γ is not zero. Table 4.6 shows effective cost and contribution values of exploits and initial conditions for each step. In AG2, we only express selected exploits and conditions which are removed to protect network, for the sake of brevity.

In first iteration, *netbios_ssn_nullsession(0,2)* is patched with cost of four and $Aprob(A_3)$ is removed. Initial conditions *squid proxy(0,3)* and *squid proxy(1,3)* are disabled respectively in second and third iteration with cost of 10. Hence with total cost of 24, we harden network such that there is no attack path for intruder to breach *host 3* with root privilege. If only initial conditions are taken into account while protecting network as in [5–10], the hardening cost would be 30 instead of 24.

As the example attack graphs of the sample networks indicate, *BCNHA* allows the balance between security level gain and the resulting cost to be tuned by the security analyst in a cost and context-aware manner. In other words, this capability provides a more adaptive framework for network security hardening.

Table 4.6. Contribution and effective cost β values for AG2.

Graph Elements	Step 1		Step 2		Step 3	
	$Con(x)$	β	$Con(x)$	β	$Con(x)$	β
IIS_bof(0,0)	2.2743000	10.992	1.5026625	16.637	0.6904125	36.210
squid_port_scan(0,3)	0.8122500	12.311	0.8122500	12.311	0	-
netbios_ssn_null session(0,2)	0.7716375	5.183	0	-	0	-
ssh_bof(0,1)	0.6904125	17.380	0.6904125	17.380	0.6904125	17.380
LICQ_remote_to_ user(0,3)	0.8122500	12.311	0.8122500	12.311	0	-
squid_port_scan(1,3)	0.6904125	14.484	0.6904125	14.484	0.6904125	14.484
squid_port_scan(2,3)	0.7716375	12.959	0	-	0	-
LICQ_remote_to_ user(1,3)	0.6904125	14.484	0.6904125	14.484	0.6904125	14.484
LICQ_remote_to_ user(2,3)	0.7716375	12.959	0	-	0	-
local_setuid_bof(3,3)	2.2743000	10.992	1.5026625	16.637	0.6904125	36.210
squid proxy(1,3)	0.6904125	14.484	0.6904125	14.484	0.6904125	14.484
squid proxy(0,3)	0.8122500	12.311	0.8122500	12.311	0	-
netbios-ssn(0,2)	0.7716375	12.959	0	-	0	-
ssh(0,1)	0.6904125	14.484	0.6904125	14.484	0.6904125	14.484
LICQ(0,3)	0.8122500	12.311	0.8122500	12.311	0	-
LICQ(1,3)	0.6904125	14.484	0.6904125	14.484	0.6904125	14.484
LICQ(2,3)	0.7716375	12.959	0	-	0	-
squid proxy(0,3)	0.7716375	12.959	0	-	0	-
ftp_rhosts(2,1)	0	-	0	-	0	-
rsh(2,1)	0	-	0	-	0	-
ftp(2,1)	0	-	0	-	0	-

5. COMPLEXITY ANALYSIS AND EXPERIMENTAL RESULTS

In this chapter, algorithm and run-time complexity of Phase I and Phase II is analyzed to show performance of *BCNHA*. Extensive simulations are conducted and reported to demonstrate performance of *BCNHA* algorithm. Efficiency of minimum cost solution is investigated by comparing total protection cost of *BCNHA* with optimal minimum cost solution. Simulation details are given in Appendix B.

5.1. Complexity Analysis of *BCNHA*

BCNHA is a greedy heuristic-based scheme, since finding optimal solution is an intractable problem for minimum-cost network hardening [5]. Phase I of *BCNHA* extracts at most $O(M^n)$ attack paths in $O(M^{n-1})$ time [8]. Here, M is the maximum number of exploits which point to a condition in the attack graph and n is the path length threshold [8]. An initial condition or an exploit is selected in each iteration of Phase II. Hence, there can be at most $|E_i| + |I_k|$ iterations where $|I_k|$ is number of initial conditions and $|E_i|$ is number of exploits. Computing contributions of initial condition and exploits takes $O(nM^n)$ time since there are at most $O(M^n)$ attack path with maximum length n . Selecting an exploit or initial condition takes $O(|E_i| + |I_k|)$ time. Therefore, each iteration in Phase II takes $O(nM^n)$ time. Computational complexity of Phase II becomes $O((|E_i| + |I_k|)nM^n)$ time. We know that attackers do not typically use attack paths whose length is more than four [8,46]. Thus worst-case computational complexity of Phase I becomes $O(M^3)$ and Phase II becomes $O((|E_i| + |I_k|)M^4)$.

For complexity analysis, different sizes and types of network topology are used while conducting experiments. All attack graphs and parameter values such as success probability and cost of removal are generated randomly. However, we try to imitate real attack graphs to obtain meaningful results. *BCNHA* and the simulation environment are implemented in Java. Experiments are conducted on a computer with 3.4GHz CPU and 8GB RAM to show run-time performance of *BCNHA*. The operating system

Table 5.1. Attributes of randomly generated attack graphs.

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

was Microsoft Windows 7 Ultimate with Service Pack 1. Cost of removal α is assigned between $[5,10]$ for initial conditions and between $[2,8]$ for exploits. Success probability of exploits $Eprob(E_i)$ is assigned between $[0.3,1.0]$. M is 10 and likelihood threshold of attack path is 0.01. Also the budget is unbounded hence *BCNHA* always eliminates all the attack paths. We run each simulation for 50 times and average the results.

Table 5.1 shows different type of attack graphs with different number of initial conditions ($|I_k|$), different number of exploits ($|E_i|$) and different number of edges ($|Edges|$). When attack graphs are constructed, number of edges and extracted attack paths are changed according to the topology of the graph. Therefore, $Avg.|Edges|$ shows average number of edges for randomly generated attack graphs.

In Figure 5.1, average number of attack paths ($Avg.|A_j|$) with different path length threshold n for randomly generated attack graphs is shown. As expected, size of attack graph has a substantial effect on the number of attack paths. It can also be seen from the graph that path length threshold n can reduce the number of attack paths ($|A_j|$) drastically. $|A_j|$ increases significantly when n is changed from 5 to 10 or from 10 to 15. Likelihood threshold becomes active when length of attack paths exceeds 15. Consequently, this dramatic growth turns into a manageable one. By setting likelihood and path length thresholds, only simple and probable attack paths which are generally preferred by attacker can be found and eliminated. These findings indicate importance of both path length and likelihood threshold parameters.

Figure 5.2 and 5.3 show average CPU time in milliseconds for Phase I and Phase

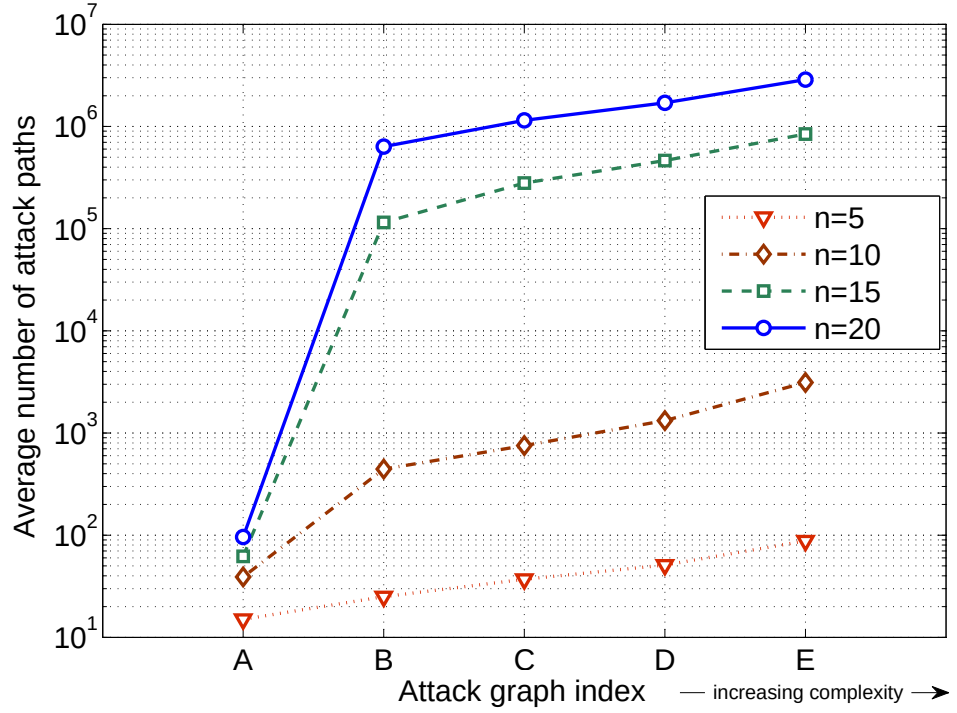


Figure 5.1. Average number of attack paths for randomly generated attack graphs with different path length threshold n .

II of *BCNHA* with different path length threshold n for randomly generated attack graphs. As n and network size increase, Phase I needs more time to find all attack paths. Similarly, Phase II requires more time for each iteration. Also as the number of attack paths which are found by Phase I increases, Phase II needs more time to protect the network with minimum cost. Moreover, larger $|A_j|$ generally induces more iterations in Phase II. These results are consistent with our computational complexity analysis and show that average run-time complexity of *BCNHA* is far better than computational worst-case complexity. Furthermore, when n is five, *BCNHA* finds minimum cost solution for all randomly generated graphs in less than one minute.

The results also indicate that *BCNHA* scales approximately linearly with the size of the graphs although CPU time can vary according to the characteristic of the attack graph and set thresholds. In less than 25 minutes, *BCNHA* found cost-effective solution for an attack graph with tens of thousands node by eliminating nearly three million attack paths during the experiments.

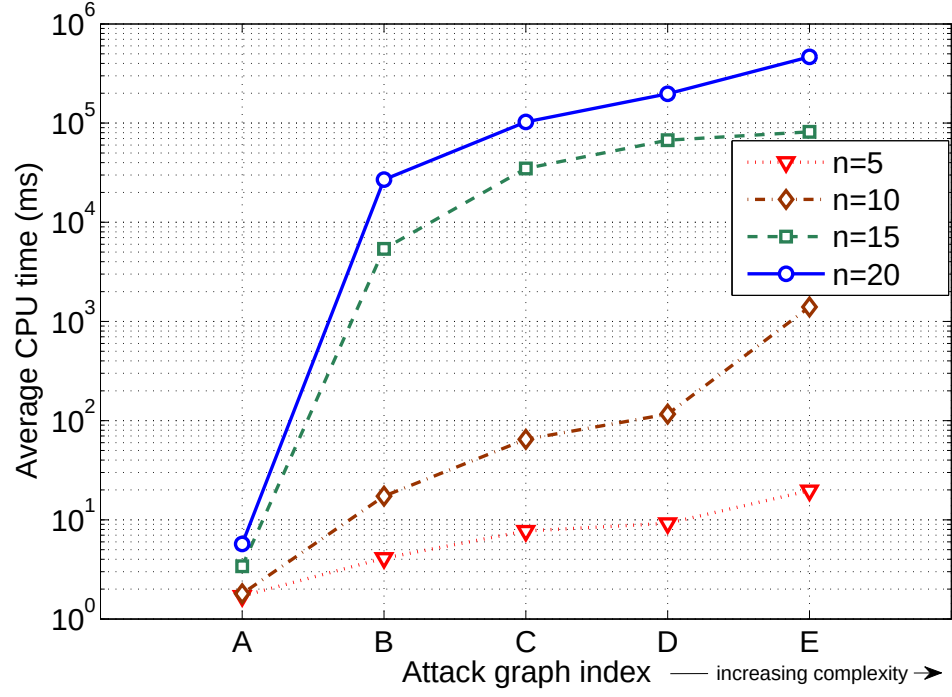


Figure 5.2. Average CPU time in milliseconds for Phase I of *BCNHA* in randomly generated attack graphs with different path length threshold n .

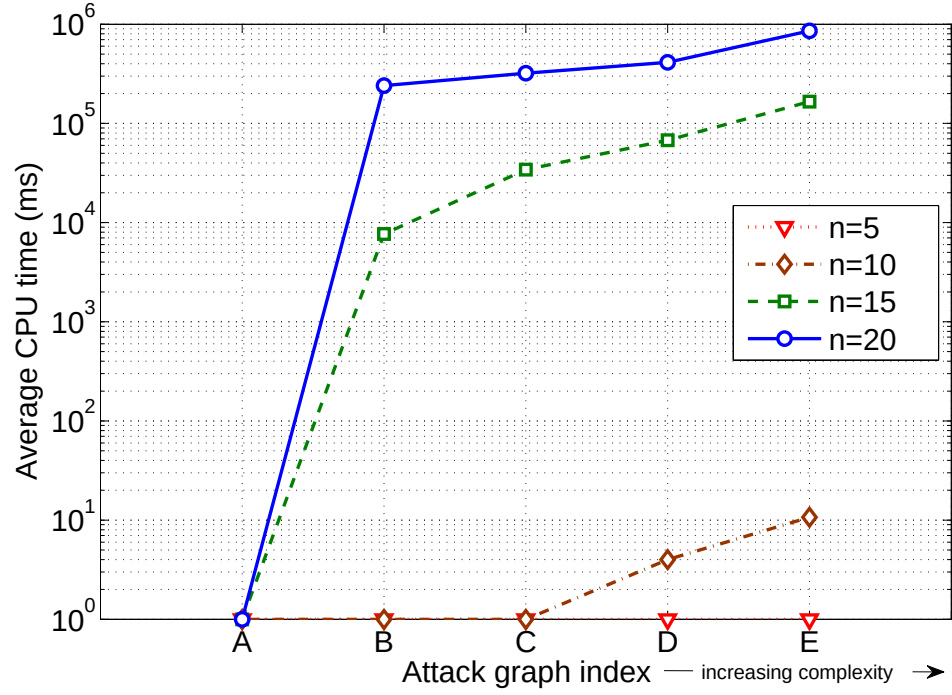


Figure 5.3. Average CPU time in milliseconds for Phase II of *BCNHA* in randomly generated attack graphs with different path length threshold n .

5.2. Protection Cost Analysis of *BCNHA*

In this section, protection cost of *BCNHA* is investigated with respect to optimal protection cost and optimal protection cost by using only initial conditions since there are many works which consider only initial conditions for protection [5–10]. Table 5.2 shows small random attack graph configurations. We choose small graphs since finding optimal solution is a NP-hard problem and it takes too much time as shown later. Cost of removal values are the same with above settings.

Table 5.2. Features of randomly generated attack graphs for cost comparison.

AG	$ I_k $	$ E_i $	$Avg. Edges $	$Avg. A_j $
F	8	12	54.7	4.9
G	10	15	64.7	8.3
H	12	18	76.8	18.9
I	15	20	99.9	23.6

In Figure 5.4, average protection cost for randomly generated attack graphs is shown for *BCNHA*, optimal solution and optimal solution by using initial conditions. Figure 5.5 shows average required time for protection. As shown, required time for optimal solution grows exponentially as number of exploits and initial conditions rises since $2^{|E_i| + |I_k|}$ iteration is required to find it. Optimal solution by using initial conditions has the same tendency however it only affected by number of initial conditions ($2^{|I_k|}$). Therefore, both optimal solution and optimal solution by using only initial condition is unfeasible to use in large attack graphs.

Average protection cost of *BCNHA* is generally nearly 10% higher than optimal solution which is tolerable by considering time reduction. Also performance of *BCNHA* is better than optimal solution by using only initial conditions hence it shown that considering exploits while protecting network offers a huge advantage.

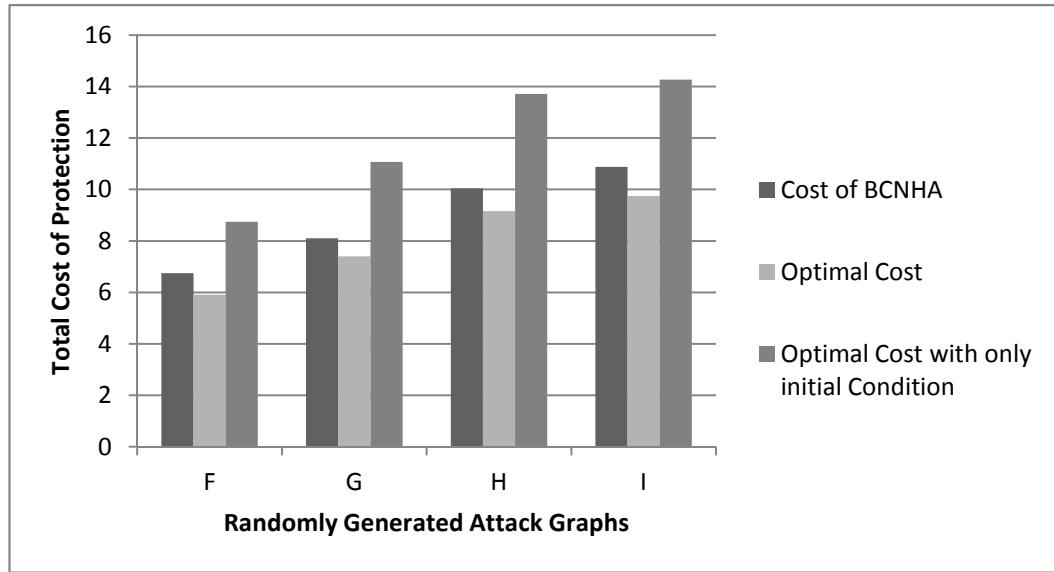


Figure 5.4. Average total protection costs for *BCNHA*, optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.

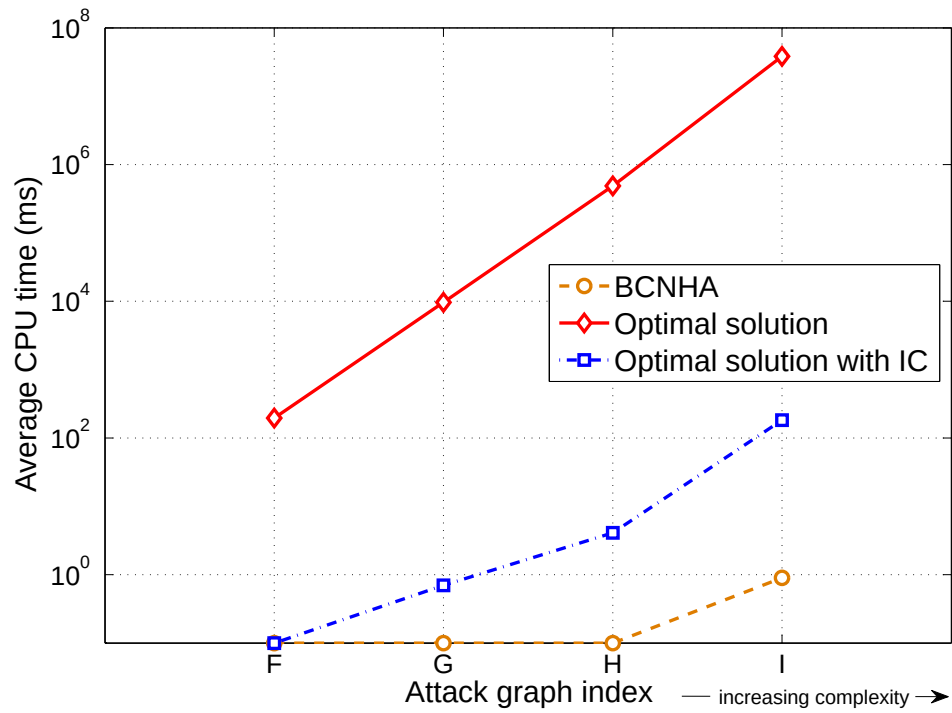


Figure 5.5. Average CPU time in milliseconds for *BCNHA*, optimal solution and optimal solution by using only initial conditions in randomly generated attack graphs.

6. CONCLUSIONS

Fueled by the need of secure network, numerous network hardening methods using attack graphs have been proposed in the literature. However, these solutions are generally focus and solve one side of the problem. Some of them work only with initial conditions to protect the network therefore, violate minimum cost requirement. Some just calculate security metric of the network and do not offer a solution to harden the network. Also most of them does not consider budget constraint and lose chance of usage in a network of the real world.

In this thesis, we have presented a network hardening solution with cost minimization by using compact attack graphs. The proposed solution *BCNHA* provides heuristic-based cost-controlled network hardening by considering both exploits and initial conditions for potential removal. Unlike other solutions, we consider budget limitation while protecting the network. Therefore, a feasible hardening option under the given budget constraint is discovered while trying to maximize the security level of the network. With the help of the path length and likelihood thresholds, less likely and complicated attack paths can also be eliminated hence, allocated budget is used efficiently.

We further extract success probability of an exploit from *CVSS* metrics and propose a novel method to measure security metric of a network to represent how secure the analyzed network is. We combine number of attack paths and their success probability to calculate security metric hence, a security analyst can easily figure out security status of the analyzed network.

The extensive simulations show that *BCNHA* has a reasonable run-time complexity even for large-scale enterprise networks. The simulations demonstrate the importance of path length and likelihood thresholds, and show that performance of *BCNHA* is highly affected by these thresholds. Besides, it is shown that protection cost of *BCNHA* is minimal with respect to optimal solution and much better than optimal

solution which uses only initial condition for removal.

For future research, impact of attack paths on integrity, confidentiality and availability of network assets can be considered while calculating security metric. Therefore, not only likelihood of an attack path but also impact of it on the network assets will be taken into account. Moreover, base and temporal exploitability score of *CVSS* can be merged to calculate success probabilities of exploits. Thus, both inherent and time-dependent features of the exploits are used together to create success probabilities of exploits.

APPENDIX A: SAMPLE NETWORK 2 (AG2)

The network model of AG2 which is used as second example attack graph of this thesis is depicted in Figure A.1. The network includes four hosts, i.e. $User(0)$, $User(1)$, $User(2)$ and $User(3)$. The *Attacker* is someone from outside the internal network. A firewall isolates the internal network from the Internet. There is also an additional firewall within the internal network to reduce connectivity between hosts and increase network security.

System characteristics of AG2 are shown in Table A.1 [21]. In the table, each exploit has a *Common Vulnerabilities and Exposures (CVE)* dictionary [49] ID hence, it is easy to find information such as *CVSS* scores about them. These data are available in Nessus [1], *NVD (National Vulnerability Database)* [47] and in some public sites such as “<http://www.securityfocus.com/archive>”. The table also shows services, vulnerabilities, operating systems of hosts and open ports to exploit the vulnerabilities.

User(3) is the the target machine for attacker and the *MySQL* database running on that host is the critical resource. The objective of attacker is to breach the network and obtain root privilege in User(3). The firewall in the network allows external hosts to connect to *IIS* web service running on port 80 on User(0) and any other connection to other ports are blocked. Connectivity limiting firewall policies and other details can be found in [21].

Table A.1. System characteristics of AG2 used as second example attack graph.

User	Services	Ports	Vulnerabilities	CVE - IDs	Operating System
User(0)	IIS Web Service	80	IIS buffer overflow	CVE-2002-0364	Windows NT 4.0
User(1)	ftp	21	ftp rhost overwrite	CVE-2008-1396	Windows 2000 SP1
	ssh	22	ssh buffer overflow	CVE-1999-1455	
	rsh	514	rsh login	CVE-1999-0180	
User(2)	Netbios-ssn	139	Netbios-ssn nullsession	CVE-2003-0661	Windows XP SP2
	rsh	514	rsh login	CVE-1999-0180	
User(3)	LICQ	5190	LICQ-remote-to-user	CVE-2001-0439	Red Hat Linux 7.0
	Squid Proxy	80	squid-port-scan	CVE-2001-1030	
	Mysql DB	3306	local-setuid-bof	CVE-2006-3368	

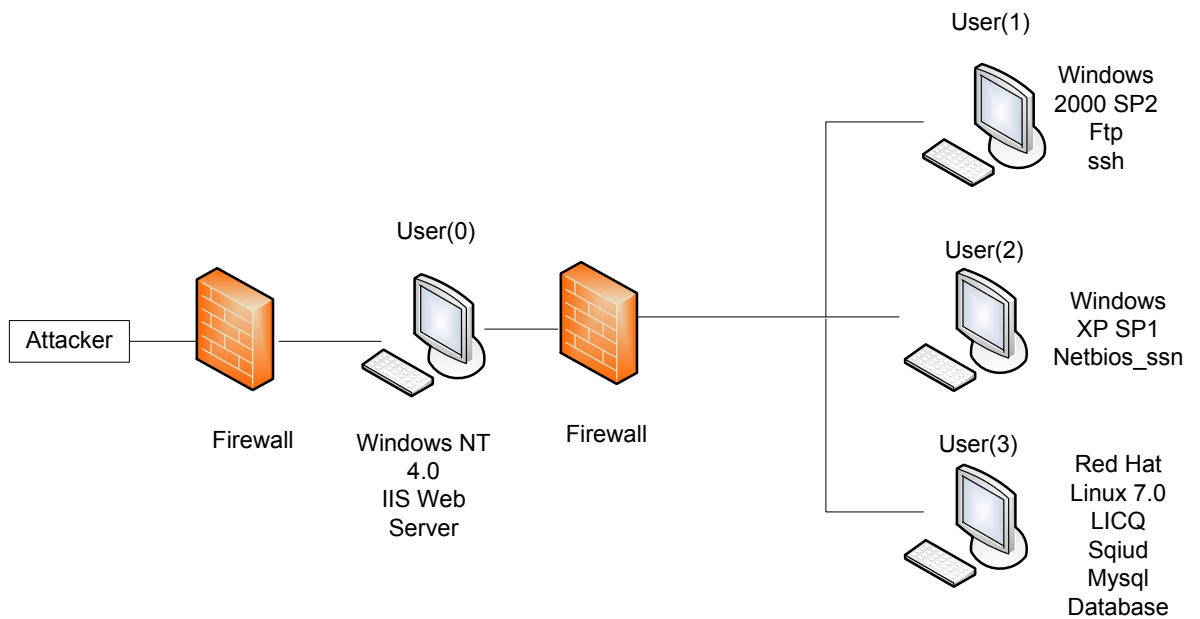


Figure A.1. The network model of AG2 used as second example attack graph of this thesis.

APPENDIX B: SIMULATION DATA

Details of simulation results are given in this chapter.

B.1. Complexity Analysis Data

Experimental results of complexity analysis simulation are shown in Table B.1, B.2, B.3 and B.4 for different path length threshold n . Average number of attack paths and time in milliseconds of Phase I and Phase II are shown in the tables for randomly generated attack graphs.

B.2. Cost Efficiency Data

Experimental results of protection cost simulation are shown in Table B.5 and B.6. Table B.5 shows average protection cost of *BCNHA*, optimal solution and optimal solution with only initial conditions for randomly generated attack graphs. Table B.6 shows average elapsed CPU time in millisecond for *BCNHA*, optimal solution and optimal solution with only initial conditions for randomly generated attack graphs.

Table B.1. Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of *BCNHA* for randomly generated attack graphs with path length threshold $n = 5$.

AG	$Avg. A_j $	Phase I (ms)	Phase II (ms)
A	15	1.7	1
B	25	4.1	1
C	37	7.7	1
D	51	9.2	1
E	88	19.7	1

Table B.2. Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of *BCNHA* for randomly generated attack graphs with path length threshold $n = 10$.

AG	$Avg. A_j $	Phase I (ms)	Phase II (ms)
A	39	1.8	1
B	444	17.3	1
C	753	64.7	1
D	1321	116.5	4
E	3123	1399.7	10.7

Table B.3. Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of *BCNHA* for randomly generated attack graphs with path length threshold $n = 15$.

AG	$Avg. A_j $	Phase I (ms)	Phase II (ms)
A	62	3.4	1
B	115612	5401	7665
C	279371	34840	34240
D	463690	67217	67701
E	843692	81528	165985

Table B.4. Average number of attack paths, average CPU time in milliseconds for Phase I and Phase II of *BCNHA* for randomly generated attack graphs with path length threshold $n = 20$.

AG	$Avg. A_j $	Phase I (ms)	Phase II (ms)
A	96	5.7	1
B	634633	26598	240730
C	1148857	102774	320453
D	1705101	197257	413075
E	2879013	465071	857030

Table B.5. Average protection cost of *BCNHA*, optimal solution and optimal solution with only initial conditions for randomly generated attack graphs.

AG	BCNHA	Optimal Cost	Optimal Cost with IC
F	6.750	5.911	8.743
G	8.105	7.404	11.072
H	10.045	9.159	13.711
I	10.875	9.746	14.266

Table B.6. Average CPU time in millisecond for *BCNHA*, optimal solution and optimal solution with only initial conditions for randomly generated attack graphs.

AG	BCNHA (ms)	Optimal Solution (ms)	Optimal Solution with IC (ms)
F	0.1	195.5	0.1
G	0.1	9654.1	0.7
H	0.1	487027.6	4.1
I	0.9	38319808.5	182.8

REFERENCES

1. Deraison, R., “Nessus Scanner”, 1999, <http://www.nessus.org>, [Accessed July 2014].
2. Roesch, M., “Snort - Lightweight Intrusion Detection for Networks”, *Proceedings of the 13th USENIX Conference on System Administration*, pp. 229–238, 1999.
3. Lippmann, R. P. and K. W. Ingols, “An Annotated Review of Past Papers on Attack Graphs”, Technical report, MIT Lincoln Laboratory, 2005.
4. Sheyner, O. and J. Wing, “Tools for Generating and Analyzing Attack Graphs”, *Formal Methods for Components and Objects*, Vol. 3188, Lecture Notes in Computer Science, pp. 344–371, 2004.
5. Wang, L., S. Noel, and S. Jajodia, “Minimum-Cost Network Hardening Using Attack Graphs”, *Computer Communications*, Vol. 29, No. 18, pp. 3812 – 3824, 2006.
6. Man, D., Y. Wu, and Y. Wu, “A Method Based on Global Attack Graph for Network Hardening”, *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2008*, pp. 1–4, 2008.
7. Islam, T. and L. Wang, “A Heuristic Approach to Minimum-Cost Network Hardening Using Attack Graph”, *Proceedings of the New Technologies, Mobility and Security, 2008 (NTMS’08)*, pp. 1–5, 2008.
8. Chen, F., D. Liu, Y. Zhang, and J. Su, “A Scalable Approach to Analyzing Network Security Using Compact Attack Graphs”, *Journal of Networks*, Vol. 5, No. 5, pp. 543–550, 2010.

9. Jun-chun, M., W. Yong-jun, S. Ji-yin, and C. Shan, “A Minimum Cost of Network Hardening Model Based on Attack Graphs”, *Procedia Engineering*, Vol. 15, No. 0, pp. 3227 – 3233, 2011.
10. Chen, F., L. Wang, and J. Su, “An Efficient Approach to Minimum-Cost Network Hardening Using Attack Graphs”, *Fourth International Conference on Information Assurance and Security, ISIAS 2008*, pp. 209–212, 2008.
11. Keramati, M., H. Asgharian, and A. Akbari, “Cost-Aware Network Immunization Framework for Intrusion Prevention”, *Proceedings of the IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, pp. 639–644, 2011.
12. Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. Wing, “Automated Generation and Analysis of Attack Graphs”, *IEEE Symposium on Security and Privacy, 2002*, pp. 273–284, 2002.
13. Ammann, P., D. Wijesekera, and S. Kaushik, “Scalable, Graph-based Network Vulnerability Analysis”, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02*, pp. 217–224, 2002.
14. Jha, S., O. Sheyner, and J. Wing, “Two Formal Analysis of Attack Graphs”, *Proceedings of the 15th IEEE Workshop on Computer Security Foundations*, pp. 49–63, 2002.
15. Alhomidi, M. and M. Reed, “Finding the Minimum Cut Set in Attack Graphs Using Genetic Algorithms”, *International Conference on Computer Applications Technology (ICCAT), 2013*, pp. 1–6, 2013.
16. Phillips, C. and L. P. Swiler, “A Graph-Based System for Network-Vulnerability Analysis”, *Proceedings of the 1998 Workshop on New Security Paradigms, NSPW '98*, pp. 71–79, 1998.

17. Idika, N. C., B. H. Marshall, and B. K. Bhargava, “Maximizing Network Security Given a Limited Budget”, *The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, TAPIA '09, pp. 12–17, 2009.
18. Sawilla, R. and D. Skillicorn, “Partial Cuts in Attack Graphs for Cost Effective Network Defence”, *IEEE Conference on Technologies for Homeland Security (HST)*, 2012, pp. 291–297, 2012.
19. Idika, N. and B. Bhargava, “Extending Attack Graph-Based Security Metrics and Aggregating Their Application”, *IEEE Transactions on Dependable and Secure Computing*, Vol. 9, No. 1, pp. 75–85, 2012.
20. Wang, L., A. Singhal, and S. Jajodia, “Toward Measuring Network Security Using Attack Graphs”, *Proceedings of the 2007 ACM Workshop on Quality of Protection*, pp. 49–54, 2007.
21. Ghosh, N. and S. Ghosh, “An Approach for Security Assessment of Network Configurations Using Attack Graph”, *First International Conference on Networks and Communications, NETCOM '09*, pp. 283–288, 2009.
22. Houmb, S. H., V. N. Franqueira, and E. A. Engum, “Quantifying Security Risk Level from CVSS Estimates of Frequency and Impact”, *Journal of Systems and Software*, Vol. 83, No. 9, pp. 1622 – 1634, 2010.
23. Khan, M., M. Omer, and J. Copeland, “Decision Centric Identification and Rank ordering of Security Metrics”, *Proceedings of the 2012 IEEE 37th Conference on Local Computer Networks (LCN 2012)*, pp. 208–211, 2012.
24. Bodin, L. D., L. A. Gordon, and M. P. Loeb, “Evaluating Information Security Investments Using the Analytic Hierarchy Process”, *Communications of the ACM*, Vol. 48, No. 2, pp. 78–83, 2005.

25. Wang, L., T. Islam, T. Long, A. Singhal, and S. Jajodia, “An Attack Graph-Based Probabilistic Security Metric”, *Proceedings of the 22Nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pp. 283–296, 2008.
26. Mell, P., K. Scarfone, and S. Romanosky, “Common Vulnerability Scoring System”, *IEEE Security and Privacy*, Vol. 4, No. 6, pp. 85–89, 2006.
27. “Common Vulnerability Scoring System (CVSS)”, <http://www.first.org/cvss/>, [Accessed July 2014].
28. Wang, L., A. Liu, and S. Jajodia, “Using Attack Graphs for Correlating, Hypothesizing, and Predicting Intrusion Alerts.”, *Computer Communications*, Vol. 29, No. 15, pp. 2917–2933, 2006.
29. Gupta, S. and J. Winstead, “Using Attack Graphs to Design Systems”, *IEEE Security and Privacy*, Vol. 5, No. 4, pp. 80–83, 2007.
30. Foo, B., Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. Spafford, “ADEPTS: Adaptive Intrusion Response Using Attack Graphs in an E-Commerce Environment”, *International Conference on Dependable Systems and Networks, 2005. DSN 2005*, pp. 508–517, 2005.
31. Noel, S., E. Robertson, and S. Jajodia, “Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances”, *20th Annual Computer Security Applications Conference, 2004*, pp. 350–359, 2004.
32. Zerkle, D. and K. Levitt, “NetKuang – A Multi-Host Configuration Vulnerability Checker”, *Proceedings of the 6th USENIX Unix Security Symposium*, 1996.
33. Ortalo, R., Y. Deswarte, and M. Kaaniche, “Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security”, *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 633–650, 1999.
34. Ritchey, R. and P. Ammann, “Using Model Checking to Analyze Network Vulner-

- abilities”, *IEEE Symposium on Security and Privacy, 2000*, pp. 156–165, 2000.
35. Ritchey, R., B. O’Berry, and S. Noel, “Representing TCP/IP Connectivity for Topological Analysis of Network Security”, *18th Annual Computer Security Applications Conference, 2002*, pp. 25–31, 2002.
 36. Swiler, L., C. Phillips, D. Ellis, and S. Chakerian, “Computer-Attack Graph Generation Tool”, *DARPA Information Survivability Conference; Exposition II, 2001. DISCEX ’01*, Vol. 2, pp. 307–321, 2001.
 37. Ramakrishnan, C. R. and R. Sekar, “Model-Based Analysis of Configuration Vulnerabilities”, *Journal of Computer Security*, Vol. 10, No. 1-2, pp. 189–209, 2002.
 38. Alhomidi, M. and M. Reed, “Attack Graphs Representations”, *Proceedings of the Computer Science and Electronic Engineering Conference*, pp. 83–88, 2012.
 39. Ou, X., W. F. Boyer, and M. A. McQueen, “A Scalable Approach to Attack Graph Generation”, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS ’06*, pp. 336–345, 2006.
 40. Ou, X., S. Govindavajhala, and A. W. Appel, “MulVAL: A Logic-Based Network Security Analyzer”, *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM’05, pp. 8–33, 2005.
 41. Ingols, K., R. Lippmann, and K. Piwowarski, “Practical Attack Graph Generation for Network Defense”, *22nd Annual Computer Security Applications Conference, 2006. ACSAC ’06*, pp. 121–130, 2006.
 42. Braynov, S. and M. Jadliwala, “Representation and Analysis of Coordinated Attacks”, *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering, FMSE ’03*, pp. 43–51, 2003.
 43. Ammann, P., J. Pamula, R. Ritchey, and J. Street, “A Host-Based Approach to Network Attack Chaining Analysis”, *21st Annual Computer Security Applications*

- Conference*, pp. 72–84, 2005.
44. Zhong, S., D. Yan, and C. Liu, “Automatic Generation of Host-Based Network Attack Graph”, *2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 1, pp. 93–98, 2009.
 45. Sawilla, R. E. and X. Ou, “Identifying Critical Attack Assets in Dependency Attack Graphs”, *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS ’08, pp. 18–34, 2008.
 46. Liu, Z., S. Li, J. He, D. Xie, and Z. Deng, “Complex Network Security Analysis Based on Attack Graph Model”, *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC)*, pp. 183–186, 2012.
 47. “National Vulnerability Database. (NVD)”, <http://nvd.nist.gov>, [Accessed July 2014].
 48. Feng, C. and S. Jin-Shu, “A Flexible Approach to Measuring Network Security Using Attack Graphs”, *2008 International Symposium on Electronic Commerce and Security*, pp. 426–431, 2008.
 49. “Common Vulnerabilities and Exposures Dictionary. (CVE)”, <http://cve.mitre.org>, [Accessed July 2014].