

2014

DrAGON: A Framework for Computing Preferred Defense Policies from Logical Attack Graphs

Swapnanjan Chatterjee
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Chatterjee, Swapnanjan, "DrAGON: A Framework for Computing Preferred Defense Policies from Logical Attack Graphs" (2014).
Graduate Theses and Dissertations. 13954.
<https://lib.dr.iastate.edu/etd/13954>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**DrAGON: A Framework for Computing Preferred Defense Policies from Logical
Attack Graphs**

by

Swapnanjan Chatterjee

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Samik Basu, Major Professor

Johnny S. Wong

Andrew S. Miner

Iowa State University

Ames, Iowa

2014

Copyright © Swapnanjan Chatterjee, 2014. All rights reserved.

DEDICATION

I would like to dedicate this thesis to my parents Subrata Kumar Chatterjee and Kalpana Chatterjee, without their support and blessings I would not have been able to complete this work. I would also like to dedicate this thesis to my brother Sayan Chatterjee for always believing in me and taking my side when things got tough, and finally my girlfriend Shivani Choudhary for being there always and supporting me through thick and thin. I would also like to thank all my friends and family both in India and USA for their loving guidance and assistance during the writing of this work.

TABLE OF CONTENTS

LIST OF TABLES	v
LIST OF FIGURES	vi
ACKNOWLEDGEMENTS	vii
ABSTRACT	ix
CHAPTER 1. Introduction	1
1.1 Logical Attack Graphs	4
1.2 Defense Policy Computation as SAT-Solving	6
1.3 Defense Policy Properties	7
1.3.1 Problem Statement	8
1.4 Contributions	8
1.5 Organization	10
CHAPTER 2. Background	11
2.1 Attack Graphs	11
2.1.1 Logical Attack Graphs	12
2.2 Defense Policy Identification	16
2.2.1 Formalization of the SAT Problem	17
2.2.2 CVSS Impact Metrics: An Overview	19
2.3 Scope of this Thesis	21
CHAPTER 3. A Qualitative Analysis of Attack Graph Exploitability and Impacts	22
3.1 Advantages of Qualitative Analysis	22

3.2	Logical Attack Graphs and Our Work	23
3.3	Integration of CVSS Impact Metrics in Analysis	24
3.4	Conditional Preference Statements	27
3.4.1	CP-nets and TCP-nets	27
3.4.2	CI-nets	30
3.4.3	Computing Next-Preferred Reasoning based on Dominance Testing	31
3.4.4	Preferences Over Defense Policies	35
CHAPTER 4. Implementation and Tool Development		46
4.1	Architecture Overview	46
4.2	The Helper Modules of <i>DrAGON</i>	50
4.3	Input Languages	50
4.4	Output Visualizations	51
CHAPTER 5. Experiments and Results		56
5.1	Bottom-up Analyzer	56
5.2	Top-down Analyzer	57
CHAPTER 6. Conclusion		61
6.1	Summary	61
6.2	Future Work	62
6.2.1	Extending Our Work to Include Cyclic Graphs	62
6.2.2	Allowing Partial Ordering over the Policy Search Space	62
6.2.3	Improving the Output Visualizations	63
6.2.4	Designing a Hybrid Algorithm Using the Bottom-up and Top-down Analyzers	63
BIBLIOGRAPHY		64

LIST OF TABLES

Table 3.1	An example of impact values	36
Table 4.1	An example valuation of the CVSS impacts for each of the nodes . . .	49
Table 4.2	Input language keywords & description	52
Table 4.3	Delimiters and their usage in the input attack graph file	53

LIST OF FIGURES

Figure 1.1	A simple logical attack graph	3
Figure 1.2	Example [Ou et al.(2006)]	5
Figure 1.3	An example logical attack graph	5
Figure 2.1	Simple Attack Tree Describing Ways to Burgle a House [AmenazaLtd. (2003)]	14
Figure 3.1	An example dependency logical attack graph	25
Figure 3.2	“My Top-Secret Network” CP-net & TCP-net	29
Figure 3.3	CI-net preference statements, Induced preference graph depicting the most preferred statement with respect to given preferences, and the Complete ordering of the variables [Oster et al. (2013)]	32
Figure 3.4	Attack graph described in Example 3.4.5	43
Figure 3.5	Bottom-up policy generation/validation order	43
Figure 3.6	Top-down policy generation/validation order	44
Figure 4.1	Architectural overview and data flow in <i>DrAGON</i>	48
Figure 4.2	Example input attack graph file	51
Figure 4.3	Corresponding logical dependency attack graph	51
Figure 4.4	Different layouts visualizing the same attack graph using yEd	54
Figure 4.5	A graphical visualization of the attack graph using JUNG	55
Figure 5.1	Graphs representing iterations vs time (ms) for bottom-up analyzer	58
Figure 5.2	Graphs representing iterations vs time (ms) for top-down analyzer	59

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Samik Basu for his guidance, patience and support throughout this research and the writing of this thesis. Also, my research partner Dr. Ganesh Ram Santhanam. Their insights and words of encouragement have often inspired me and renewed my hopes of completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Johnny S. Wong and Dr. Andrew S. Miner. I would like to thank my lab-mates Dr. Zachary Oster, Dr. Tanmoy Sarkar, Debasis Mandal and my dear friend and room-mate Guruprasad Sivagurunatha Krishnan for helping and supporting me in all stages of my graduate career.

I want to thank all my wonderful professors Dr. Samik Basu, Dr. David Weiss, Dr. David Fernández-Baca, Dr. Ying Cai, Dr. Douglas Jacobson, Dr. Shashi Gadia and Dr. Chris Harding for teaching some of the best and interesting computer science courses in the most effective manner.

Moreover, I would like to thank my teaching instructors Dr. Andrew S. Miner, and Dr. Simanta Mitra for their wonderful support. Their advise has not only helped me do my job effectively but also I have learnt a lot about effective teaching. It is because of them I enjoyed my teaching duties throughout my Master's career.

Thanks to Abigail Andrews, Darlene Brace, Maria-Nera Davis, and Laurel Tweed for always being so helpful, approachable, and friendly. I would like to thank my parents, my family and friends in India for their patience and support to help me complete my Master's study. Special thanks to my close friends, Anirban Karak without whom I would not even have appeared for my GRE, much less get admitted to Iowa State University; Guruprasad for being the best room-mate and confidant, who inspired me to learn new algorithms and coding techniques. My

first room-mates in life Saurabh and Vignaraj for helping me settle into this unknown land far away from home. I am grateful to my girlfriend Shivani Choudhary for all those delicious meals, while I was cramped away coding and for her extreme patience in dealing with my incessant complaints, nagging, and mood swings brought on by the pressure of my work. She has been extremely supportive during the writing of this work and has patiently helped me through the final stretch. I am not a big believer in God, but thanks to any super-power at large that helped me accomplish this goal, and I am eternally grateful to my mom for praying regularly to the deities and making sure I was always in their good books.

This work is supported in parts by US National Science Foundation grants CCF1143734 and CCF1116836.

ABSTRACT

Attack graphs provide formalism for modelling the vulnerabilities using a compact representation scheme. Two of the most popular attack graph representations are *scenario attack graphs*, and *logical attack graphs*. In logical attack graphs, the host machines present in the network are represented as *exploit nodes*, while the configurations (IDS rules, firewall policies etc.) running on them are represented as *fact nodes*. The actual user privileges that are possible on each of these hosts are represented as *privilege nodes*.

Existing work provides methods to analyze logical attack graphs and compute attack paths of varying costs. In this thesis we develop a framework for analyzing the attack graph from a defender perspective. Given an acyclic logical dependency attack graph we compute defense policies that cover all known exploits that can be used by the attacker and also are preferred with respect to minimizing the impacts. In contrast to previous work on analysis of logical attack graphs where quantitative costs are assigned to the vulnerabilities (exploits), our framework allows attack graph analysis using descriptions of vulnerabilities on a qualitative scale. We develop two algorithms for computing preferred defense policies that are optimal with respect to defender preferences. Our research to the best of our knowledge is the first fully qualitative approach to analyzing these logical attack graphs and formulating defense policies based on the preferences and priorities of the defender.

We provide a prototype implementation of our framework that allows logical attack graphs to be input using a simple text file (custom language), or using a GUI tool in *graphical markup language (GML)* format. Our implementation uses the NVD (National Vulnerability Database) as the source of CVSS impact metrics for vulnerabilities in the attack graph. Our framework generates a preferred order of defense policies using an existing preference reasoner. Preliminary experiments on various attack graphs show the correctness and efficiency of our approach.

CHAPTER 1. Introduction

The internet being a global system of interconnected networks, each having their own topologies and dependencies, is a tough challenge to manage. From a security standpoint, system administrators have the tough task of enforcing security protocols to stop breaches within their network without causing problems to general users of that network. In order to enforce such policies they need to be well aware of existing vulnerabilities in their network, which can be exploited by attackers with malicious intent. One of the worst known cyber-attack occurred in 2004, when a Sandia National Laboratories employee, Shawn Carpenter, discovered a series of large “cyber raids” [Graham (2005)]. The Federal Bureau of Investigation (FBI) named the attacks, “Titan Rain”, and found that several sensitive computer networks were infiltrated by hackers such as those at Lockheed Martin and Sandia (owned by Lockheed), but also at the likes of NASA. The possibility that the hackers did not just make off with classified data and military intel, but could also have left behind backdoors and “zombify” machines, enabling easier attacks in the future.

In order to avoid such incidences it is imperative to know about existing vulnerabilities in the network. The lesser the vulnerabilities, the more difficult it is for attackers to penetrate the internal network. However, sometimes vulnerabilities cannot be avoided due to multitude of reasons like: shared dependencies, lack of structured access policies in the organization etc. In such situations knowledge about vulnerabilities by the system administrator is vital. They can use that knowledge to safeguard against breaches by enforcing defense policies which blocks out external attacks. Thus it is crucial to detect and respond to the attacks in a timely manner. Intrusion detection systems (IDS) and firewalls help system administrators to identify attacks. However, IDSs are very noisy and most of the times flag normal traffic as attacks, making its use an issue to keep track of actual attacks [Kabiri and Ghorbani (2005), Hassan et al.

(2013)]. Once attacks have taken place, it is very difficult and sometimes impossible to detect the amount of losses, let alone recovering the lost data [Smith (2004), Cashell et al. (2004)].

In this context, it is important to supplement detection and response capabilities of Intrusion Detection and Response systems with a framework that will allow system administrators to effectively and correctly (a) identify the *cause* and *impact* of an attack and (b) deploy response(s).

Attack graphs [Phillips and Swiler (1998)] provide such a supplementary framework. They capture the inter-relations between nodes or capabilities of a network and formulate the sequences or sets of events that can potentially compromise the network. The attack graphs can be broadly classified as Network-based attack graph, which models the topological relationship of the network nodes [Noel et al. (2005)]; and Logical attack graph, which models the capability relation of the network [Ou et al. (2006)]. For instance, the Figures 1.1, 1.3 are two examples of how logical attack graphs are represented.

Attack graphs allow security analysts to assess vulnerability of critical network resources and to understand how impacts on individual services contribute to overall vulnerability. Attack graphs have traditionally been created manually by security red teams [Noel et al. (2005)]. However, significant progress has been made towards generating attack graphs based on network models and attacker exploits, notably [Ramakrishnan and Sekar (2000); Ritchey and Ammann (2000); Sheyner et al. (2002); Baldwin (1994); Zerkle and Levitt (1996); Phillips and Swiler (1998); Dawkins et al. (2002); Ammann et al. (2002); Jajodia et al. (2005); Cuppens et al. (2002); Ning et al. (2004); Noel et al. (2005)]. Most of the attack graphs generated are potentially huge rendering analyses difficult.

Ou et al. (2006) introduced the notion of scalability and a standardized input mechanism for generating attack graphs. One of the pioneering works by Sheyner et al. (2002) is the first attack-graph tool based on formal logical techniques, popularly known as model-checking. However, it still encountered the problem of growth explosion for moderate sized networks. For example, a network of only 10 hosts with 5 vulnerabilities per host takes about 15 minutes to generate and results in a graph consisting of 10 million edges [Ou et al. (2006)]. Therefore, the method adopted for the tool developed for our research was based upon these acute observations

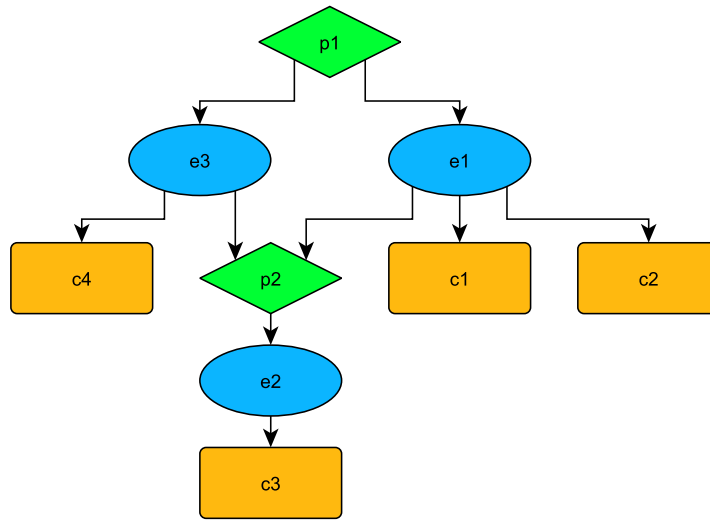


Figure 1.1 A simple logical attack graph

and representing the attack graphs as a logical AND/OR tree made the most sense. Figure 1.1 displays an example of a simple logical attack graph. In the figure, the nodes labelled ‘ p_1 ’ and ‘ p_2 ’ are called as *privilege nodes*, while nodes ‘ e_1 ’, ‘ e_2 ’ and ‘ e_3 ’ are called *exploit nodes*. The nodes labelled, ‘ c_1 ’, ‘ c_2 ’, ‘ c_3 ’, and ‘ c_4 ’ are referred to as *fact/configuration nodes*. In essence, the exploit nodes represent the host machines in the network that can be attacked, while the configuration nodes are sets of configurations (firewall rules, IDS configurations etc.) that those hosts have been configured with. The privilege nodes, represent the privileges (application programs; user privileges; ftp, rsh privileges etc.) that those host machines are capable of running. In the representation of logical attack graphs, the privilege nodes are treated as *OR* nodes which can be satisfied by any of the children nodes being satisfied (exploited) and exploit nodes are treated as *AND* nodes which can only be satisfied when all of its children are satisfied. The fact nodes (leaf nodes in logical attack graph) are treated as artifacts. Our aim is to help the system administrator by providing a prioritized list of defense policies that will be validated against the attack graph. Thus, it is essential, that the representation of the attack graph be simple to understand and interpret but contain all information about the network.

1.1 Logical Attack Graphs

Logical attack graphs directly illustrate logical dependencies among attack goals and configuration information. A logical attack graph always has size polynomial to the network being analyzed [Ou et al. (2006)]. In this representation, a node in the graph is a logical statement which does not encode the entire state of the network, but only some aspect of it. The edges in the graph specify the causality relations between network configurations and an attacker's potential privileges. In Sheyner's *scenario attack graphs* every node is a collection of boolean variables encoding the entire network state at an attack stage. Thus, even if the number of variables is polynomial in the size of the network, the possible number of states is exponential. However, in logical attack graphs the nodes are not boolean variables but a logical statement, while the edges capture the causality relations between network configurations and an attacker's potential privileges. Thus, not only ensuring that the graphs are always polynomial in size, but the edges clearly outline the casualty relations with respect to attacker's potential privileges. In case of network security analysis, it is vital to consider both multi-stage and multi-host attacks. In logical attack graphs, the propositional formula for each node, captures the state of the attacker and the edges clearly outline the casualty relations with respect to attacker's potential privileges (inter host dependencies). Logical attack graphs not only require less time to generate, but also it is also easier to embed any information either explicit or implicit about the attack graph in them. Moreover, inferences that are to be drawn about the network can be performed easily by the network and system administrators. We use logical attack graphs for our work for all the same advantages.

The example network in Figure 1.2 is borrowed from the MuLVAL paper [Ou et al. (2006)]. Suppose the following potential attack paths are discovered after analyzing the configuration. An attacker first compromises webServer by remotely exploiting vulnerability CVE-2002-0392 to get local access on the server. Since webServer is allowed to access fileServer through the NFS protocol, he can then try to modify data on the file server. There are two ways to achieve this. If there are vulnerabilities in the NFS service daemons, he can try to exploit them and get local access on the machine; or if the NFS export table is not set up appropriately, he can

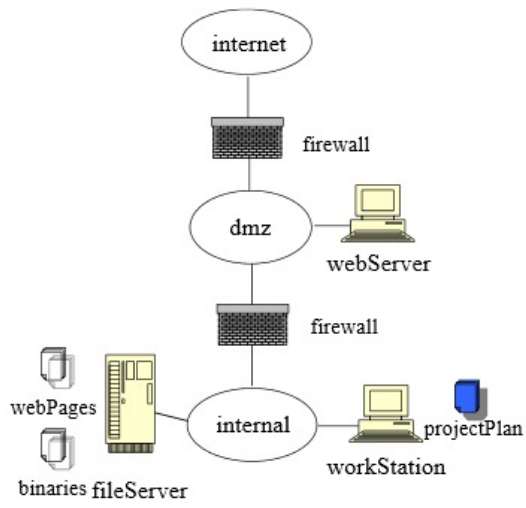


Figure 1.2 Example [Ou et al.(2006)]

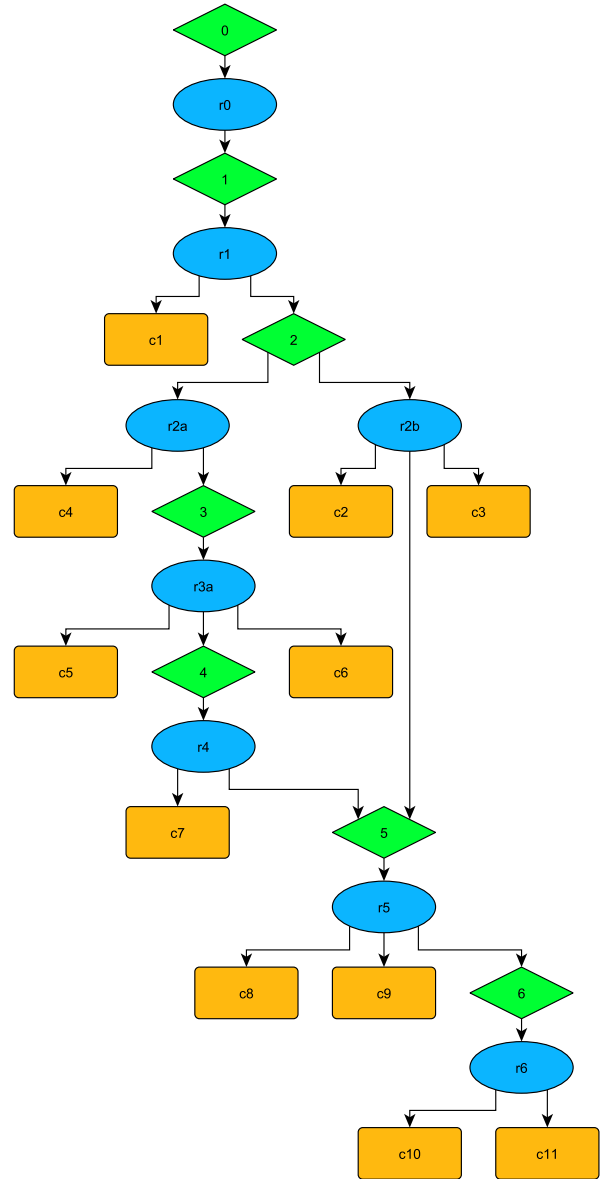


Figure 1.3 An example logical attack graph

modify files on the server through the NFS protocol by using programs like NFS Shell. Once he can modify files on the file server, the attacker can install a Trojan-horse program in the executable binaries on fileServer that are mounted by machine workStation. The attacker can now wait for an innocent user on workStation to execute it and obtain control on the machine [Ou et al. (2006)]. The logical attack graph corresponding to the above scenarios is illustrated in Figure 1.3.

1.2 Defense Policy Computation as SAT-Solving

The purpose of generating and analyzing these attack graphs is to aid the system administrators design defense policies to prevent attacks. If these graphs are too big and cumbersome, it is of no practical use to the analysts even if they are accurate and complete. It was this observation that led to the concept of *attack graph distillation* by Huang et al. (2011). Their work is an extension of an earlier work by Homer and Ou (2009) which addressed the issue of balancing security and usability with certain trade-offs. Consider the simple example of Figure 1.1, here the system administrator is aware of existing exploits (vulnerabilities), namely, ‘ e_1 ’, ‘ e_2 ’ and ‘ e_3 ’. However, despite these existing vulnerabilities in the network, the administrator will try to defend the goal privilege (here, ‘ p_1 ’) from being attained by the malicious attacker. In order to accomplish this, the defender will try to either modify or remove certain configurations (‘ c_1 ’, ‘ c_2 ’, ‘ c_3 ’, and/or ‘ c_4 ’) on the hosts (exploits ‘ e_1 ’, ‘ e_2 ’ and ‘ e_3 ’). The defender may also, simply place monitors on certain privileges (here ‘ p_2 ’) using some IDS engine, to prevent/log certain privilege usage.

In this thesis, we base our work along the lines of Homer and Ou (2009) and Huang et al. (2011). Although, we will not restrict ourselves to only using SAT-solvers for forming and verifying valid defense policies against possible attacks. As our approach is not quantitative but more qualitative, we focus on the preferences of the defense policies based on recommendations from the system administrator.

1.3 Defense Policy Properties

From the point of view of a system administrator, defense policies are a set of tasks that could prevent an attacker from breaching the network. With regards to our research we will define a defense policy as consisting of certain privileges and configurations present in the AND/OR Logical Attack Graph. Previous works for assessing the security risks in an enterprise have mostly been from a quantitative standpoint. They have either quantified the security risks in an organization by basing their work on the Common Vulnerability Scoring System (CVSS) alone [Ou et al. (2006)], or used certain probabilistic approaches [Homer and Ou (2009)] to analyze the vulnerabilities in the enterprise and perform comparisons.

We try to address the issue of defense policy enforcement, by providing a prioritized list of defense policies based on certain inputs from the system administrator. Again, let us try to understand the issue from our example attack graph in Figure 1.1. Here's the system administrator's main objective is to ensure that the privilege 'p1' never be achieved by any attacker inspite of existing exploits (vulnerabilities) 'e1', 'e2' and 'e3'. Now, there are several ways of achieving this. The most naive solution would be to ensure, none of the configurations, namely, 'c1', 'c2', 'c3', and 'c4' are ever placed. Note, this is similar to removing exploits 'e1', 'e2' and 'e3' from the network all together. However, this is not a viable solution to the problem, because in real life these exploits are host machines, being used for actual purposes by a lot of genuine users. Removing them all together, might ensure no possibility of attacks, but it also means, shutting down all normal functionality within the network. This is certainly not possible.

Consider alternative solutions, where the administrator can remove any or a combinations of the configurations (c_1 , c_2 , c_3 or c_4). For example, removing c_4 and c_3 ; c_1 , c_2 , and c_3 ; or just removing c_3 . Note, the choice which is a better defense policy is entirely upon the administrator, based on his/her experience about the network topology and traffic. If the chances of an attack are very less, placing a monitor for the execution of privilege 'p2' may be sufficient. While all the policies are correct, some might make the system more restrictive, and others might just render it useless. Therefore, it is clear that there are a wide variety of

choices in terms of defense policy for a certain attack. Now, in case of an enterprise network, consisting of hundreds or thousands of hosts, these choices are too many for the administrator to decide manually. This shows the usefulness of our tool, where the system administrator is provided with a prioritized list of defense policies, based on certain generic inputs. Now, the defender may have different priorities for different properties of the system like: availability, confidentiality, and integrity. These properties are clearly defined by the industrial benchmark of Common Vulnerability Scoring System (CVSS) [Mell et al. (2007), Schiffman et al. (2004)]. CVSS provides certain measurable (quantitative) metrics for communicating the characteristics and impacts of IT vulnerabilities between different organizations.

1.3.1 Problem Statement

The question(s) we ask ourselves again and again, during the course of thesis is, “Can the network be visualized in a logical manner that depicts the privileges possible to attain from present configurations on existing vulnerabilities (exploits) in the system. Is there a way to help formulate defense policies that would help system administrators block attacks? If so, can these policies be prioritized, based on certain preferences input by the administrators?”

1.4 Contributions

The following are the contributions of this thesis:

1. Integration of Logical Attack graphs and CVSS impact metrics. To the best of our knowledge, this work is the first of its kind, which analyses attack graphs by bringing together the concept of logical attack graph based on Huang et al. (2011) and the CVSS impact metrics (Confidentiality Impact (C), Integrity Impact (I) and Availability Impact (A)). The defense policies generated are analyzed in a prioritized order based on certain recommendations of the system administrator and these policies are then compared against the logical attack graph to validate them.
2. Preferential treatment of system privileges for obtaining priorities over defense policies. The preferences input by the system administrator are over the privileges in the logical

attack graphs. As we are interested in a qualitative analysis, we rely on the inputs of the system administrator to create a list of preferences from high priority to low. Based on the priority we help analyse the defense policies in the same order against the attack graph of the network. This is efficient both in terms of scalability and practicality. In a practical approach, one would want to safeguard the high priority assets first, then their medium priority ones and finally the low priority ones. The CVSS impact metrics also work in a similar manner. However, we ensure that in no way, the attacker even if he/she gets hold of the low priority assets can escalate to higher privileges. In essence we perform a holistic check on all possible paths to higher priority privileges before we analyze the lower priority privileges.

3. Realization of a framework for obtaining prioritized defense policies (DrAGON¹). To the best of our knowledge, no such tool exists till date which analyses an attack graph and provides a prioritized list of defense policies based on CVSS impact metrics. Here are a couple of highlights of the tools features,
 - a. Experimental evaluation presenting the feasibility of the approach. We have run conclusive tests both from synthetic examples and from previously published sources. We have run the tests based on a wide variety of parameters to verify the correctness of our approach and have observed results that align with established principles and expected results.
 - b. GUI for easy-to-understand justification of defense policy prioritization. For most of the previous work on attack graphs, the input for the actual graphs requires some really complicated syntax and semantics that varies based on the approach. Some are based on formal methods, where the semantics can be cumbersome for the average lay-man user. Our work bridges the gap, by allowing the user to input the attack graph either, in our specified format for parsing, or using *graphical modeling language*(GML) for input, or to directly use their own graph editor tool(we recommend using Y-ed graph editing tool for this purpose as it provides a diverse list of

¹Defense by pReferred policies for Attack Graph of the Network

options to layout the attack graphs) and create their own *gml* file from scratch.

1.5 Organization

This thesis is organized up into six chapters. In chapter 1, we introduced the reader to the background of our work and to certain concepts and definitions. Chapter 2 provides insights into the background of our work, and provides in-depth explanations of certain concepts and ideas required to understand our work. It also provides certain examples to help understand the underlying principles and concepts. Finally it compares our work to existing work and clearly outlines the contributions of this thesis. In Chapter 3 we discuss about our concept of qualitatively analyzing logical attack graphs and figuring out exploitable nature of the graphs. We also delve into the impact analysis based on CVSS impact metrics and corresponding defense criterions. Chapter 4 explains in detail, our implementation of qualitative analyses of attack graphs and a preferential ordering of the defense policies. It explains how our tool works and how to use it. We will discuss its architecture, modularization, input languages/formats, outputs and visualization of the output. The experiments we conducted using our tool and the observed results and how to interpret these results is explained in Chapter 5. Finally, in Chapter 6, we will list the inferences we can draw from our experiments and conclude our arguments regarding our work. We will also discuss the scope of future work based on our work and extensions to the tool we developed.

CHAPTER 2. Background

In this chapter we have discussed the background to our work involving and looked into some of the past and existing work done in the field of attack graph. These have helped us in building the foundation for our research. In the present study we have offered solutions to the shortcomings of the research done in the past. Attack graphs are a way to visualize and understand a network, thereby addressing the ongoing and evolving challenge of tracking changes to inherent vulnerabilities, variable nature of attacks and tools used over time. Thus attack graphs are a valuable tool for network administrators in patching vulnerabilities and blocking attacks.

2.1 Attack Graphs

Attack graphs allow security analysts to assess true vulnerability of critical network resources and to understand how impacts on individual services contribute to overall vulnerability. Attack graphs have come a long way since their manual creation by the security red teams [Noel et al. (2005)], by making use of network models and attacker exploits [Ramakrishnan and Sekar (2000); Ritchey and Ammann (2000); Sheyner et al. (2002); Baldwin (1994); Zerkle and Levitt (1996); Phillips and Swiler (1998); Dawkins et al. (2002); Ammann et al. (2002); Jajodia et al. (2005); Cuppens et al. (2002); Ning et al. (2004); Noel et al. (2005)]. Most of the attack graphs generated are potentially huge rendering their analysis difficult. A new graph representation was presented by Kunz and Pradhan (1994), to identify implications in a multi-level combinational circuit. While the emphasis provided in Kunz and Pradhan (1994) was on how to extract implications, it had the underlying concept of search algorithms for recursive learning which addressed the issue of exponential growth, as encountered by most of

the previous works cited. This gave new theoretical and practical insight into many problems [Stoffel et al. (1995)]. The issue of exponential growth can be solved by AND/OR tree which was born as an underlying principle in recursive learning and it *cannot* be related to any of the above.

In the field of security, all the work done before Ou et al. (2006) had not only failed to provide an account of the scalability of the graph generating process, but also lacked logical formalism in their representations of attack graphs. Although the work done by Sheyner et al. (2002) in developing the attack-graph tool based on formal logical techniques, popularly known as model-checking was first of its kind, it still could not address the issue of exponential explosion for moderate sized networks.

2.1.1 Logical Attack Graphs

In case of network security analysis, it is vital to consider both multi-stage and multi-host attacks. Attackers generally do not stop at the first machine they can break into, but try to escalate up the hierarchy. As is evident, this means there is quite a lot of complicated routes that the attacker can take. The defender's work is extremely challenging as not all of these paths can be cut off, as some might involve bringing down the network in part or full. Although, that would stop the attack, it would also cause immense harm to the regular activities of the real users and the organization. Logical attack graphs directly illustrate logical dependencies among attack goals and configuration information. A logical attack graph always has size polynomial to the network being analyzed [Ou et al. (2006)].

2.1.1.1 Definition

A logical attack graph is a graph where every node in the graph is a logical statement, which does not encode the entire state of the network, but only some aspect of it. The edges in the graph specify the causality relations between network configurations and an attackers potential privileges. Logical attack graph illustrates causes of the attacks.

The mathematical definition of Logical Attack graphs is as follows:

Definition 2.1.1. $(N_p, N_e, N_c, E, \mathcal{L}, \mathcal{G})$ is a logical attack graph, where N_p , N_e and N_c are three sets of disjoint nodes in the graph, $E \subseteq (N_p \times N_e) \cup (N_e \times (N_p \cup N_c))$, \mathcal{L} is a mapping from a node to its label, and $\mathcal{G} \in N_p$ is the attackers goal.

N_p, N_e and N_c are the sets of privilege nodes, exploit nodes and fact(leaf) nodes, respectively. The labeling function maps a privilege node to the privilege that can be run on the host machine(exploit), and maps a configuration node to the configuration in place on the host machine(exploit). For example, the attack graph illustrated in Figure 1.1, $N_p = \{p_1, p_2\}$; $N_e = \{e_1, e_2, e_3\}$; $N_c = \{c_1, c_2, c_3, c_4\}$. The labelling function \mathcal{L} , maps each node to the labels of each node. For the same example, $\mathcal{L}(N_p) = (p_1, p_2)$; $\mathcal{L}(N_e) = (e_1, e_2, e_3)$; $\mathcal{L}(N_c) = (c_1, c_2, c_3, c_4)$. Formally, the semantics of a logical attack graph is defined as follows.

Property 1. *For every exploit node e , let \mathbf{P} be the set of e 's parent node(s) and \mathbf{C} be the set of e 's child nodes then, $(\wedge \mathcal{L}(\mathbf{C})) \Rightarrow \mathcal{L}(\mathbf{P})$ is an instantiation of interaction rule $\mathcal{L}(e)$ [Ou et al. (2006)].*

Here, \wedge is the conjunction operator. For example, the exploit node ' e_2 ' is an application of the interaction rule shown in Figure 1.1. In the example, $e_2 : (c_1 \wedge c_2 \wedge p_2) \Rightarrow p_1$. This in literal terms means, if the attacker can gain control over the configurations c_1 and c_2 on host e_2 and already has gained privilege p_2 , then the attacker will get access to privilege p_1 as well..

2.1.1.2 Origin of Logical Attack Graphs

Logical Attack Graphs closely resemble Attack Trees which are conceptual diagrams showing how an asset, or target, might be attacked. In the field of information technology, attack trees have been used to describe threats on computer systems and possible attacks to realize those threats. However, their use is not restricted to the analysis of conventional information systems. They are widely used in the fields of defense and aerospace for the analysis of threats against tamper resistant electronics systems (e.g., avionics on military aircraft). Attack trees are increasingly being applied to computer control systems (especially relating to the electric power grid). They have also been used to understand threats to physical systems. Some of the earliest descriptions of attack trees are by Schneier (1999). Figure 2.1 is borrowed from

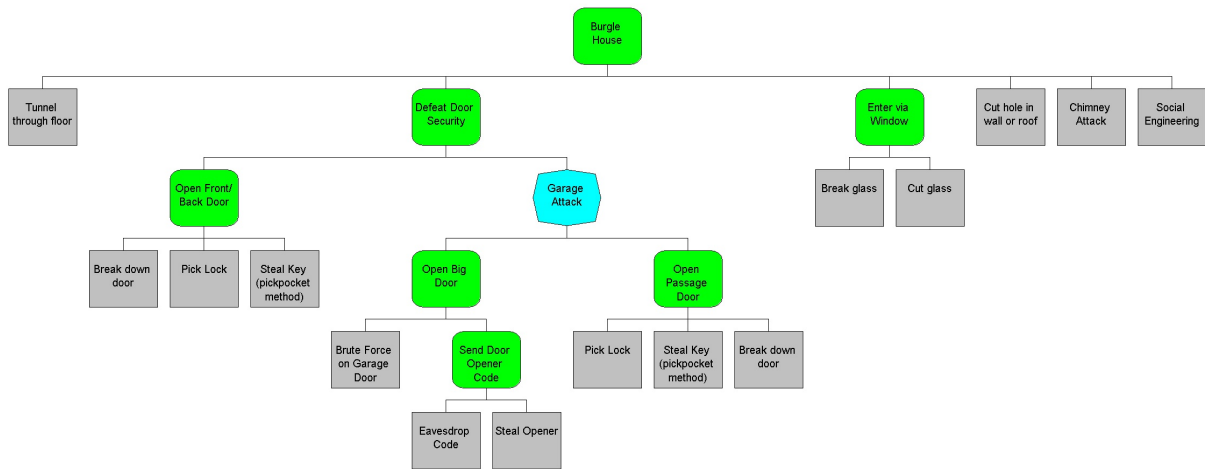


Figure 2.1 Simple Attack Tree Describing Ways to Burgle a House [AmenazaLtd. (2003)]

AmenazaLtd. (2003) which is an attack tree that illustrates (in a very simple manner) the different ways in which a residence, with an attached garage, can be burglarized. The green colored nodes are called *OR* nodes and can be satisfied by any one of its immediate child nodes. The blue colored node(s) are called *AND* nodes and they are satisfied if and only if all of its immediate children are true (satisfied). The topmost node is the *root* node and represents the ultimate goal of the attacker. All the grey colored boxes are sub-tasks called as Leaf nodes which must be performed to satisfy the task nodes (AND/OR nodes).

Attack trees laid the foundation for some later works. We have seen that the dependencies and interactions between multiple hosts and components in any moderately sized network can be very large. Thus, it is important to generate automatic tools for analyzing the configurations in an enterprise and spotting potential security vulnerabilities [Ou et al. (2006)].

The earliest work in producing full attack graphs is the Kuang system [Baldwin (1994)], and its extension to a network environment, the NetKuang system [Zerkle and Levitt (1996)]. In these systems, a backward goal-based search scans UNIX systems for poor configurations. The output is a combination of operations that lead to compromise.

Lippmann and Ingols (2005) presented a fine overview of various attack graph tools used in the past and stated that, “although research has made significant progress in the past few

years, no system has analyzed networks with more than 20 hosts, and computation for most approaches scales poorly and would be impractical for networks with more than a few hundred hosts.” This is quite a remarkable observation as most of the attack graph generation tools not only suffer from scalability problems but also perform an ad-hoc method of inputting and displaying this information.

Ammann et al. (2002) address the exponential nature of the attack graphs common to Phillips and Swiler (1998), Swiler et al. (2001), and Sheyner et al. (2002). They present an implicit, scalable attack graph representation that avoids the exponential blow-up of explicit attack graphs. Attack graphs are encoded as dependencies among exploits and security conditions, under the assumption of monotonicity. Here, monotonicity means, that the attackers ability to perform any action is not affected by whatever actions or choices he/she takes. The authors treat vulnerabilities, intruder access privileges, and network connectivity as atomic boolean attributes, and actions as atomic transformations that, given a set of pre-conditions on the attributes, establish a set of post-conditions.

Dawkins et al. (2002) have specified another language for modeling exploits and used this language to provide a hierarchical view of attack trees. The hierarchy helps in presenting information to the user in a more manageable way. However, their procedure results in a graph representing all possible compromises, and not just those of interest to a specific intruder goal.

Most of the works discussed here have been mentioned in Sheyner’s thesis [Sheyner et al. (2002)]. The work by Sheyner et al. (2002) is the first of its kind in this regard and is based on formal methods. In it the state of the network is formally modelled as a collection of boolean variables, representing configuration parameters and attacker’s privileges. The actions performed by the attacker are modeled as state-transition relations. The security property of the network is specified as a temporal formula, which is modeled and then verified by a model checker. Unlike traditional model checkers, which provide just one counter-example if the formula is not satisfied, Sheyner’s tool outputs all possible counter examples in the form of a *scenario graph*.

However, Sheyner’s tool was far from perfect based on observations made by Ou et al. (2006). They found that for a network of only 10 hosts with 5 vulnerabilities per host, Sheyner’s tool

takes about 15 minutes to generate and results in a graph containing 10 million edges, which pretty much renders Sheyner’s tool useless. Ou et al. (2006) in their work addressed this issue by introducing logical attack graphs by performing subtle changes in the way the information about the graph is stored. In their representation, a node in the graph is a logical statement. This logical statement does not encode the entire state of the network, but only some aspect of it. In some sense it can be viewed as one boolean variable in the nodes of Sheyners graph. The edges in the graph specify the causality relations between network configurations and an attackers potential privileges. According to Ou et al. (2006), if Sheyner’s attack graph illustrates snapshots of attack steps, or “how the attack can happen”, their attack graph illustrates causes of the attacks, or “why the attack can happen”. Sheyner’s work is referred to as scenario attack graphs and the work done by Ou et al. (2006) is referred to as logical attack graphs.

2.2 Defense Policy Identification

We have seen different approaches to generate attack graphs so far. The main aim behind generating these attack graphs is to allow the system administrators to understand the existing vulnerabilities and to help in blocking the attacks. From the example scenario we talked about in Section 1.3, we saw that the system administrators have a lot of different ways of blocking the same attack. Now, these different defense strategies may involve certain different trade-offs as far as normal working of the network is concerned. These defense measures can impact different parameters of the network viz: confidentiality, integrity, and availability.

Previous work for assessing the security risks in an enterprise have either quantified the security risks in an organization by basing their work on the Common Vulnerability Scoring System (CVSS) alone, or used certain probabilistic approaches to analyze the vulnerabilities in the enterprise and perform comparisons. Ou et al. (2006) in their book, “Quantitative Security Risk Assessment of Enterprise Networks”, presented a methodology for quantitative security risk analysis based on the model of attack graphs and the Common Vulnerability Scoring System (CVSS). They helped answer questions like: “are we more secure than yesterday” or “how does the security of one network configuration compare with another one”. Homer and Ou (2009) takes yet again a quantitative approach to analyzing security risk in an enterprise network, but

they applied probabilistic reasoning to produce an aggregation that has clear semantics and sound computation. They also address the issue of shared dependencies between attack paths and cycles in their aggregation analysis.

Our work defines a defense policy as consisting of certain privileges and/or configurations present in the AND/OR Logical Attack Graph. We will also base our approach more from a qualitative stand-point integrating the CVSS impact metrics and allowing priorities over these metrics.

2.2.1 Formalization of the SAT Problem

The logical relationships represented in a dependency attack graph can be analyzed in two directions. One way to do this is by *forward deduction*, where analysis is from attacker’s present state to the final goal state. This approach has been used to encode the effect of changing configurations on attackability [Homer and Ou (2009)]. The second approach is by *backward induction*, which is from the goal state to the initial attacker location [Huang et al. (2011)]. This approach has also been used to transform an attack graph into disjunctive normal form (DNF)[Wang et al. (2006)] that captures the exact configuration conditions for a privilege(s) to be obtainable. In our work we will perform backward induction similar to Huang et al. (2011), where the backward induction generates a conjunctive normal form (CNF) that encodes the requirement relations between attack steps. However, unlike them we will not assign cost metrics to the literals in the CNF formula and perform a quantitative analysis. We will take a qualitative standpoint and prioritize the defense policies based on the SAT formula and CVSS [CVSS (2007)] impact metrics.

Let us revisit our first basic example of a logical attack graph as illustrated in Figure 1.1 and review the logical structure of the graph and provide examples of the transformation process. An OR node (the green colored diamond-shaped node) represents the privilege an attacker can gain after exploiting any one of children AND nodes, e.g. p_1 can be achieved through one of e_1 , and e_3 , corresponding to the OR logic. An AND node (the blue colored oval-shaped node) is used to simulate the process of exploiting a vulnerability on a host, only after all its children are obtained, e.g. e_3 can be exploited if and only if all of its children are obtained viz: p_2 , and

c4. It is worth noting that for this type of exploits, the logic constraint indicates that in order to set e_1 to be “True”, its child privilege p_2 should be set to “True” as well. For example, consider the possible attack paths for reaching goal privilege p_1 through exploits e_1 , e_2 , and e_3 could be traced as follows:

$$e_2 \rightarrow p_2 \rightarrow e_3 \rightarrow p_1; e_2 \rightarrow p_2 \rightarrow e_1 \rightarrow p_1$$

The basic logic constraint encode based on the original attack graph depicted in Figure 1.1, are defined as follows. For p_1 , we have

$$p_1 ::= p_1 \Rightarrow e_1 \vee e_3$$

Or, equivalently,

$$p_1 ::= \neg p_1 \vee e_1 \vee e_3$$

For an AND node like e_1 ,

$$e_1 ::= c_1 \wedge c_2 \wedge p'_2 \Rightarrow p_1$$

where, p'_2 means the privilege node p_2 is not monitored

Or, equivalently,

$$e_1 ::= (c_1 \wedge c_2 \wedge p'_2) \vee \neg p_1$$

For our work, we consider that a privilege node can be represented as is p_i or as un-monitored (p'_i). The un-monitored privilege node (p'_i) means the same privilege exists in the attack graph, but currently there are no monitors on it. Negating an un-monitored privilege means explicitly placing a monitor on the privilege p_i . Based on this transformation, each of the OR nodes and AND nodes will be represented in one or more disjunctive clauses. Now, a valid defense policy for protecting the goal privilege node, defined by the symbol $Def()$ will be negation of conjunctions all exploit nodes (AND nodes), along with the negation of the goal privilege node in the attack graph. Mathematically, it can be represented as,

$$Def(p_1) ::= (e_1 \wedge e_2 \wedge e_3) \wedge \neg p_1$$

A valid defense policy is one where, the system administrator considers that all the exploits are present, yet the goal privilege is safeguarded from an attack. It is worth pointing out that

for our work, we do not consider an exploit to be negated even if its parent privilege(s) is negated, e.g. in Figure 1.1, e_2 is not considered negated, even if p_2 is negated according to the system administrator, when formulating a defense policy. This is because, e_2 can still be exploited by an attacker by using any other exploitable configurations (known/unknown) that have not been modeled in the attack graph. However, if e_2 is considered to be negated from the defender perspective, p_2 will be automatically negated. Thus, our work ensures the correctness of the defense policies enforced.

2.2.2 CVSS Impact Metrics: An Overview

The National Vulnerability Database (NVD) is a product of the National Institute of Standards and Technology (NIST) Computer Security Division and is sponsored by the Department of Homeland Security's National Cyber Security Division. It is the U.S. government content repository for the Security Content Automation Protocol (SCAP). They use a standardized method for analyzing vulnerabilities in a network based on certain quantitative valuations. This guideline of scores (called metrics) is based on expert assessment and is termed as Common Vulnerability Scoring System (CVSS), maintained by the Forum of Incident Response and Security Teams (FIRST). CVSS helps organizations prioritize and coordinate a joint response to security vulnerabilities by communicating the base, temporal and environmental properties of a vulnerability [CVSS (2007); Mell et al. (2007); Schiffman et al. (2004)]. CVSS defines the following terms:

- Vulnerability: a bug, flaw, weakness, or exposure of an application, system, device, or service that could lead to a failure of confidentiality, integrity, or availability.
- Threat: the likelihood or frequency of a harmful event occurring.
- Risk: the relative impact that an exploited vulnerability would have to a user's environment.

Research by the National Infrastructure Advisory Council in 2003/2004 led to the launch of CVSSv1 in 2004. Work on CVSSv2 (the current version) began in 2005 and launched in

2007. Work on version 3 began in 2012, and is expected to be released in 2014. CVSSv2 has 6 base metrics for analyzing the vulnerability status of a network. The base metric group captures the characteristics of a vulnerability that are constant with time and across user environments. The Access Vector, Access Complexity, and Authentication metrics capture how the vulnerability is accessed and whether or not extra conditions are required to exploit it. The three impact metrics measure how a vulnerability, if exploited, will directly affect an IT asset, where the impacts are independently defined as the degree of loss of confidentiality, integrity, and availability. For example, a vulnerability could cause a partial loss of integrity and availability, but no loss of confidentiality.

Here we're concerned with the 3 CVSS Impact metrics:

1. **Confidentiality Impact (C):** This metric measures the impact on confidentiality of a successfully exploited vulnerability. Confidentiality refers to limiting information access and disclosure to only authorized users, as well as preventing access by, or disclosure to, unauthorized ones. There are 3 levels associated with it:
 - **None (N):** There is no impact to the confidentiality of the system.
 - **Partial (P):** There is considerable informational disclosure. Access to some system files is possible, but the attacker does not have control over what is obtained, or the scope of the loss is constrained. An example is a vulnerability that divulges only certain tables in a database.
 - **Complete (C):** There is total information disclosure, resulting in all system files being revealed. The attacker is able to read all of the system's data (memory, files, etc.)
2. **Integrity Impact (I):** This metric measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and guaranteed veracity of information. There are 3 levels associated with it:
 - **None (N):** There is no impact to the integrity of the system.

- **Partial (P):** Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited. For example, system or application files may be overwritten or modified, but either the attacker has no control over which files are affected or the attacker can modify files within only a limited context or scope.
 - **Complete (C):** There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised. The attacker is able to modify any files on the target system.
3. **Availability Impact (A):** This metric measures the impact to availability of a successfully exploited vulnerability. Availability refers to the accessibility of information resources. Attacks that consume network bandwidth, processor cycles, or disk space all impact the availability of a system. There are 3 levels associated with it:
- **None (N):** There is no impact to the availability of the system.
 - **Partial (P):** There is reduced performance or interruptions in resource availability. An example is a network-based flood attack that permits a limited number of successful connections to an Internet service.
 - **Complete (C):** There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.

2.3 Scope of this Thesis

Our work is based on identifying and validating defense policies in a prioritized manner against the input logical attack graph. We do not approach the problem from a purely quantitative view-point like many of the previous works [Homer and Ou (2009); Homer et al. (2009); Homer et al. (2013)]. Even though we integrate our defense policy validator with the CVSS impact metrics, we allow for the system administrator to prioritize the policies based on the impact metrics. To the best of our knowledge, no such tool exists till date which analyses an attack graph and provides a prioritized list of defense policies based on CVSS impact metrics.

CHAPTER 3. A Qualitative Analysis of Attack Graph Exploitability and Impacts

In this chapter we will discuss about our work in detail. We will try to understand, why we are interested in a qualitative analysis of the attack graph over a quantitative analysis. We will also showcase how our work differs from previous works. We will see how our work integrates the network metric standards of CVSS, and finally, we will look at the advantages of such a qualitative analysis.

We have seen already, that the National Vulnerability Database (NVD) was set up to help standardize the vulnerability measures across different organizations based on the CVSS metrics. We have also seen by now, that having a very detailed and complete analysis of a network might be overwhelming and not very useful to the system administrators. Thus, there is a need for balancing usability with completeness when it comes to analysis of network security. Our work tries to accommodate all of these ideals.

3.1 Advantages of Qualitative Analysis

We are concerned about a qualitative analysis in our work, than a quantitative analysis. There are many reasons as to why we are interested in such an analysis. The biggest reason we felt for adopting such a strategy, is that qualitative analysis allows for a much more in-depth analysis of a phenomenon than a quantitative analysis. Also, we found, that there has been a lot of quantitative analysis of attack graphs that have been previously performed [Swiler et al. (1998); Wang et al. (2007); Frigault et al. (2008); Wang et al. (2008); Huang et al. (2011); Keramati and Akbari (2013)]. Although, many of them helped address a lot of missing features of quantifying the vulnerabilities and the overall security of the network, none could actually

factor in specific details about a network. Our work, to the best of our knowledge allows inputs from the system administrator about basing priorities over the CVSS impact metrics, allowing for a much more in-depth analysis of the dependency logical attack graph. Moreover, NVD and FIRST (Forum of Incident Response and Security Teams) themselves recommend explicitly that the metrics be handled qualitatively to preserve information. Since, we do not rigidly depend on the cost metrics put forward by the CVSS [Mell et al. (2007); CVSS (2007); Schiffman et al. (2004)] as maintained by the NVD, we allow for natural extension to any other metric that might be more accurate in future. As the metrics we used do not have numeric domain, we do not have to make unjust approximations about our analysis. Hence, our defense policy validator is not only always correct, but it can analyze in much more detail inputs fed in by the system administrator about the network administrator along with the attack graph.

3.2 Logical Attack Graphs and Our Work

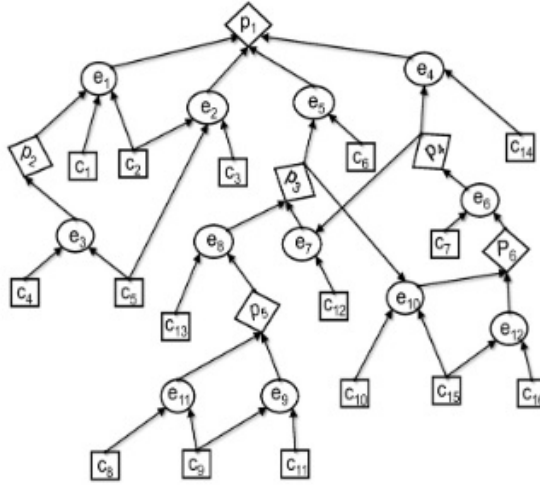
The choice of logical attack graphs based on Ou et al. (2006) work, was a deliberate attempt to keeping our attack graphs both scalable and usable. Sheyner’s contribution [Sheyner et al. (2002); Sheyner (2004)] to this field is considered as one of the most impactful contributions yet based on a couple of reasons. Not only was his tool based on formal logical methods but, his tool also performed a complete analysis of all possible attack scenarios based on the input attack graph. However, it suffered in terms of scalability of the generated attack graph. Based on all previous works on this topic, we found Huang et al. (2011) approach to be the most effective. Hence, we base our work around the same concept of logical attack graph as explained by them. Although we will deal with a slightly different representation for our work. In their work, Huang et al. (2011) represented dependency in their attack graph with edges from the leaves towards the goal privilege (which is the root). However, we shall represent our attack graph with edges from the root node (goal privilege) and outwards to the leaves (fact/configuration nodes). This difference in representation stems from the fact that, while Huang et al. (2011) were concerned with the steps taken by an *attacker* to gain the goal privilege, our main objective is to help the *defender* (system/network administrator) stop any attacker from gaining the goal privilege. This difference can be understood from the example in figures 3.1. Observe that in Figure

3.1(a), the edges are inward towards the goal privilege (p_1), whereas, in Figure 3.1(b), the edges are directed from the goal privilege (p_1) outwards to the leaf nodes (fact/configuration nodes). This simple variation in the two representations arises from the fundamental difference, that while in Figure 3.1(a), the graph is from an attacker’s point of view, in Figure 3.1(b), we consider the graph from the defender’s perspective. It is worth mentioning here, that unlike the example in Figure 3.1(b), we will not deal with attack graphs having cycles but work with *acyclic logical attack graphs* only. We intend to extend our work to accommodate cycles in the future.

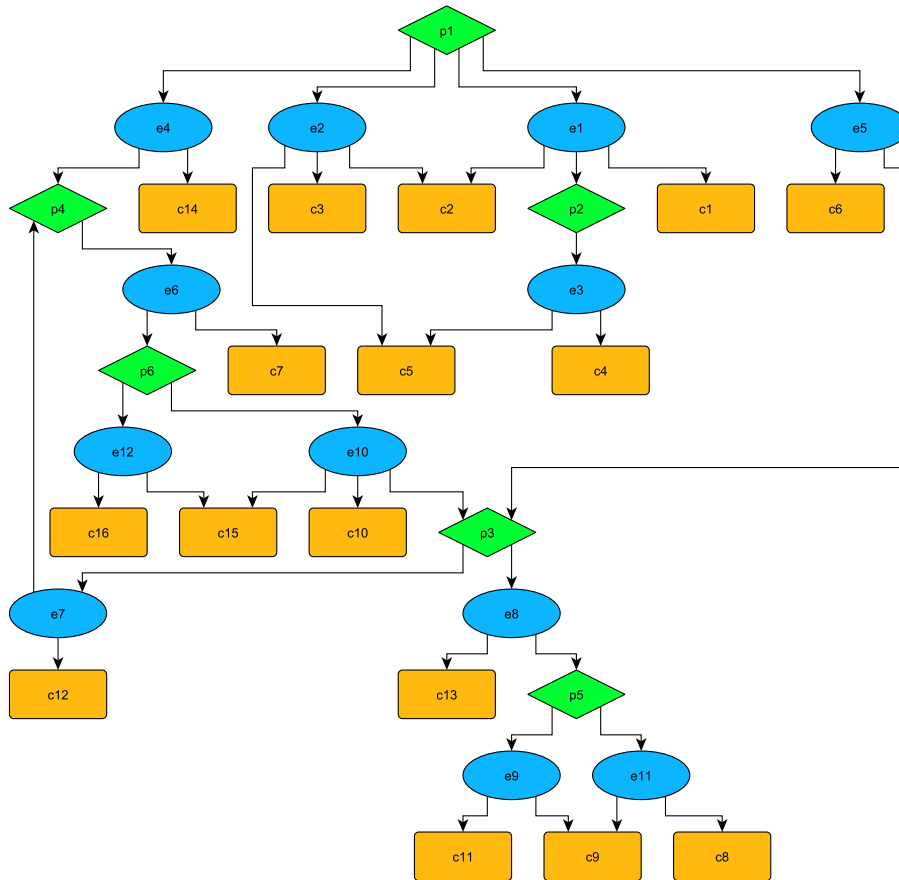
Huang et al. (2011) in their work, provided a tool for analyzing attack graphs quantitatively. Their contribution is the concept of attack graph surface and a way to compute the critical attack graph surface iteratively through SAT solving. SAT solving had been previously applied in network configuration management [Narain et al. (2008)] and MinCostSAT had been applied to attack graph for context-aware security management [Homer and Ou (2009)]. Wang et al. (2006) in their work had a boolean formula based approach to identify minimum-cost network hardening options from attack graphs. However, only Huang et al. (2011) focused on the critical problems in an attack graph and identified a critical attack surface. The idea of attack surface [Howard et al. (2005), Manadhata and Wing (2011), Manadhata et al. (2006)] was proposed in software engineering as a metric to indicate the exposed resources to potential abuses. Similarly, a critical attack graph surface refers to a portion of an attack graph that demonstrates the most critical security exposure in a network.

3.3 Integration of CVSS Impact Metrics in Analysis

There have been a wide variety of approaches to quantify the security of attack graphs [Swiler et al. (1998), Wang et al. (2007), Frigault et al. (2008), Wang et al. (2008), Huang et al. (2011), Keramati and Akbari (2013)]. However, most of these quantitative approaches involve some level of probabilistic analysis. Now, the problem with adopting any of the metrics discussed in these works is that they have not been tested out rigorously and not applied to different network conditions. However, the CVSS metrics have been used by a wide variety of organizations, with different network topologies, with varying threats and vulnerabilities. The



(a) Attacker perspective [Huang et al. (2011)]



(b) Defender perspective

Figure 3.1 An example dependency logical attack graph

CVSS metrics are publicly maintained and available in the National Vulnerability Database (NVD). We base our work on the CVSS metrics, because of these very reasons. However, we do realize that CVSS also has its limitations for assessing the impact metrics, but, since we do not work explicitly with the cost metrics set by CVSS and concentrate more on the qualitative nature of analyzing impacts, our research is somewhat independent of its limitations. This allows adapting our study on some new and better metrics that might be available in the future.

In Chapter 2 we discussed about the 3 CVSS impact metrics, when solving the problem of validating the defense policies and prioritizing them. For the benefit of the reader, we reiterate the 3 CVSS impact metrics again, they are: *confidentiality impact*, *integrity impact*, and *availability impact*. Now, each of these impacts have 3 impact levels, *low*, *medium*, and *high*, in the order of better to worse conditions. Confidentiality impact measures how well, data is protected from unauthorized access. Integrity impact measures the trustworthiness and veracity of the information. Availability impact measures the accessibility of the information resource. Now, the system administrators(s) might have a preference over any of the impact metrics, e.g, confidentiality might be most important, followed by availability, and then integrity in an enterprise. Another situation might be that, medium confidentiality impact is preferred over a high availability impact. In essence, there may be a complete or partial ordering over these impact metrics and levels. We accommodate such preferences while putting forward defense policies in a prioritized order to the system administrator. We use a tool called *iPref-R* [Santhanam et al. (2010); Oster et al. (2013)] which is a reasoner for qualitative preferences expressed in languages such as CP-nets, TCP-nets and CI-nets. *i-PrefR* is founded on decision theory, formal methods and software engineering. It helps generate preferences of partial or complete order over the input literals (here, CVSS impact metrics and sub-levels). The following section provides an overview of the foundations and working of this *preference reasoner*.

3.4 Conditional Preference Statements

3.4.1 CP-nets and TCP-nets

Extracting preference information from users is generally an arduous process, and human decision analysts have developed sophisticated techniques to help elicit this information [Howard and Matheson (1984)]. Methods for extracting, representing and reasoning about preferences are important in AI (Artificial Intelligence) applications. Even for analyzing the attack graphs, we need a structured method to analyze the preferences among the various defense policies that can be enforced to stop a certain attack. However, in most cases just like in our work, users may not be able to provide much more than qualitative rankings of fairly circumscribed outcomes. This idea gave birth to the novel graphical representation, *CP-nets*, that can be used for specifying preference relations in a relatively compact, intuitive, and structured manner using conditional *ceteris paribus* (all else being equal) preference statements [Boutilier et al. (2004)]. The semantics of CP-net allows variables to have arbitrary finite domains. It is important to note that the *ceteris paribus* component of these definitions ensures that the statements one makes are relatively weak. In particular, they do not imply a stance on specific value tradeoffs.

Consider two variables A and B that are preferentially independent, so that the preferences for values of A and B can be assessed separately; for instance, suppose $a_1 \succ a_2$ and $b_1 \succ b_2$. Clearly, a_1b_1 is the most preferred outcome and a_2b_2 is the least; but if feasibility constraints make a_1b_1 impossible, we must be satisfied with one of a_1b_2 or a_2b_1 . However, we cannot tell which is most preferred using these separate assessments. Brafman et al. (2006) enhanced the expressive power of CP-nets by introducing information about importance relations, obtaining a preference-representation structure which they called TCP-nets (for *Tradeoff-enhanced CP-nets*). By capturing information about both conditional preferential independence and conditional relative importance, TCP-nets provide a richer framework for representing user preferences, allowing stronger conclusions to be drawn, yet remaining committed to the use of only very intuitive, qualitative information. They modeled a preference relation as a strict partial order. A strict partial order is a binary relation over outcomes that is anti-reflexive, anti-symmetric and transitive. Given two outcomes o, o' , the representation $o \succ o'$ is used to

denote that o is strictly preferred to o' . The TCP-net (for *Tradeoff-enhanced CP-nets*) model is an extension of CP-nets [Boutilier et al. (2004)] that encodes conditional relative importance statements, as well as the conditional preference statements supported in CP-nets. The primary usage of the TCP-net graphical structure is in consistency analysis of the provided preference statements, and in classification of complexity and developing efficient algorithms for various reasoning tasks over these statements.

In the following example borrowed from Brafman et al. (2006) we will try to understand the inherent advantages of TCP-nets over CP-nets. Again for simplicity of presentation, the variables are boolean in nature, but they can have arbitrary finite domains.

Example 3.4.1 (My Top-Secret Network). *Figure 3.2(a) presents a CP-net that consists of three variables C, I, and A, standing for the confidentiality impact, integrity impact, and availability impact, respectively. I run a top-secret research lab working with top-secret stuff and I prefer low impact to high impact as an impact level for both confidentiality and integrity impacts, while my preference for the availability impact (low/high) is conditioned on the impact levels of confidentiality and availability: If both confidentiality and integrity impacts are low, a high availability impact will make the network break down completely, therefore, low availability impact is preferable. Otherwise, if the confidentiality and the integrity impacts are different, a low availability impact will probably render the existing users vulnerable to attacks and allow the attack to spread easier, or develop inconsistencies in the data effectively puzzling the genuine users, therefore, a high availability impact is preferable. However, if both the confidentiality and integrity impacts are high, then the availability should be low (to prevent exfiltration of data), hence, availability impact should be high. The solid lines in Figure 3.2(c) show the preference relation induced directly by the information captured by this CP-net; The top and the bottom elements are the worst and the best outcomes, respectively, and the arrows are directed from less preferred to more preferred outcomes. Figure 3.2(b) depicts a TCP-net that extends this CP-net by adding an i -arc from C to I, i.e., having low confidentiality impact is (unconditionally) more important than having low integrity impact. This induces additional relations among outcomes, captured by the dashed lines in Figure 3.2(c).*

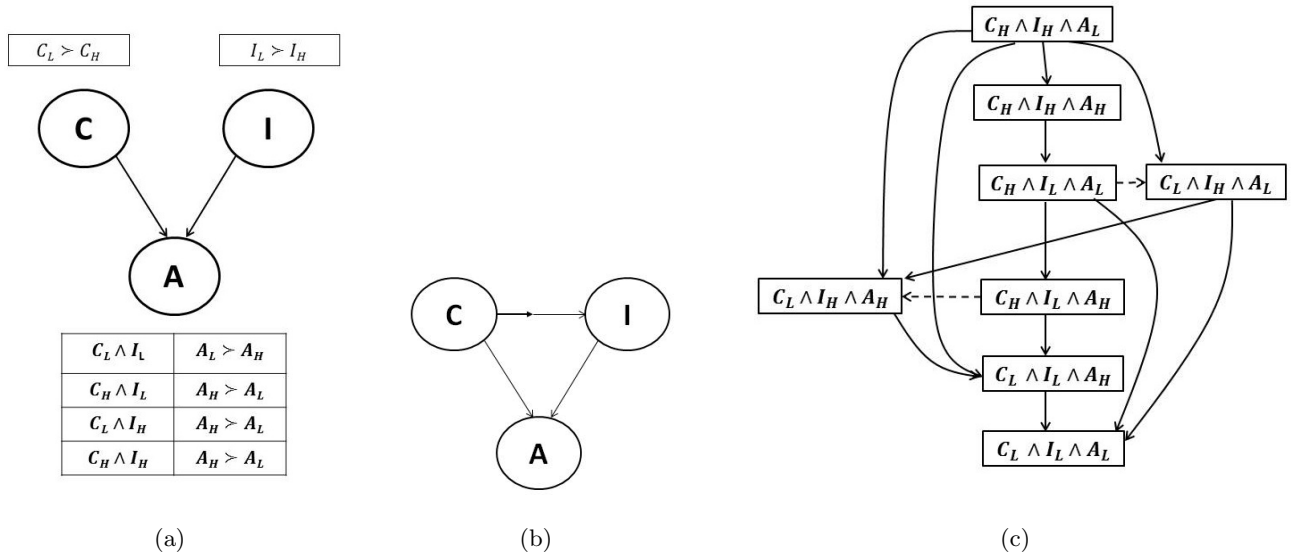


Figure 3.2 “My Top-Secret Network” CP-net & TCP-net

According to the preference, it seems that I will always have low confidentiality and low availability impact. However, while my preferences are clear, various constraints may make some outcomes, including the most preferred one, infeasible. For instance, I may not have a low confidentiality impact in my network (because my system administrator went to a bar and lost a document with all root passwords and reset-passwords in it after getting drunk), in which case the most preferred feasible alternative is a high confidentiality impact, low integrity impact, and a high availability impact. Alternatively, suppose that the only options for me are to have a low confidentiality impact and high integrity impact by installing state of the art hardware based firewall device or high confidentiality impact and low integrity impact by upgrading to new enterprise database software. I do not have enough money to install the firewall device and upgrade to a new enterprise database software, so I will have to compromise, and either upgrade to a newer enterprise database solution and deal with high confidentiality impact but maintain a low integrity impact, or to install the hardware equipment and allow data integrity to be hampered but maintain a low confidential impact. In this case, the fact that I prefer low confidentiality impact than low integrity impact determines higher desirability for the hardware based firewall device than the upgrade to the enterprise database software. Now, if my assistant

has to run the network security lab in case I'm kidnapped and held at gun-point (for top-secret information), having this information will help him/her to choose among the available options a decision which I would have concurred. Essentially, the same concept goes into picking the most preferred defense policies with respect to a given attack graph for blocking attacks.

3.4.2 CI-nets

Both CP-nets and TCP-nets help represent ordinal preferences over sets of alternatives and not over set of goods which are monotonic. *Conditional importance networks* (CI-nets) includes questions of the form, “if I have a set A of goods, and I do not have any of the goods from some other set B , then I prefer the set of goods C over the set of goods D ” [Bouveret et al. (2009)]. So far we have seen CP-nets [Boutilier et al. (2004)] allow eliciting and representing ordinal preferences over combinatorial domains where preference relations on the domain of each variable are conditioned by the values of the variables it depends on. TCP-nets [Brafman et al. (2006)] extend CP-nets by allowing statements of conditional importance between single variables, and conditional preference theories [Wilson (2004)] which is an extension TCP-nets. CI-nets are very similar to TCP-nets [Brafman et al. (2006)], but, CI-nets can compare sets of objects of arbitrary size, while TCP-nets can only express importance statements between single objects, *ceteris paribus* (all else being equal). However, CI-nets do not express preferences between values of variables, as TCP-nets do, since monotonicity makes them redundant. The following are the definitions of *conditional importance statement* and *CI-nets* by Bouveret et al. (2009).

Definition 3.4.1 (Conditional importance statement). A conditional importance statement on \mathcal{V} is a quadruple $\gamma = (\mathcal{S}^+, \mathcal{S}^-, \mathcal{S}_1, \mathcal{S}_2)$ of pairwise disjoint subsets of \mathcal{V} , written as $\mathcal{S}^+, \mathcal{S}^- : \mathcal{S}_1 \triangleright \mathcal{S}_2$.

The informal reading is: if I have all the items in \mathcal{S}^+ and none of those in \mathcal{S}^- , I prefer obtaining all items in \mathcal{S}_1 to obtaining all those in \mathcal{S}_2 , *ceteris paribus*. \mathcal{S}^+ and \mathcal{S}^- are called the *positive precondition* and the *negative precondition* of γ , respectively. \mathcal{S}_1 and \mathcal{S}_2 are called the *compared sets* of γ . Here, \mathcal{V} is a set of attributes describing the alternatives.

Definition 3.4.2 (CI-net). A CI-net on \mathcal{V} is a set \mathcal{N} of conditional importance statements on \mathcal{V} .

CI-statements are similar to importance statements in TCP-nets [Brafman et al. (2006)], up to a very important difference: CI-nets can compare sets of objects of arbitrary size, while TCP-nets can only express importance statements between single objects, *ceteris paribus*. On the other hand, CI-nets do not express preferences between values of variables, as TCP-nets do, since monotonicity makes them redundant.

CI-nets are very compact in nature which stems from the *ceteris paribus* interpretation of preference statements; a single CI-statement may express up to an exponential number of comparisons. While CP-nets and TCP-nets have a two-tier language where preferences are expressed on properties that the set of objects have, CI-nets express preferences directly at the object level. CI-nets also address the monotonicity problem, which was concerned neither by CP-nets nor TCP-nets. A formalism based on CI-nets to represent and reason over countermeasures that the administrator may hold when devising a strategy to thwart the attackers goals was presented by Santhanam et al. (2013). Santhanam et al. (2013) presented a method to find a strategy that is guaranteed to thwart any attack on the system, which is also most preferred.

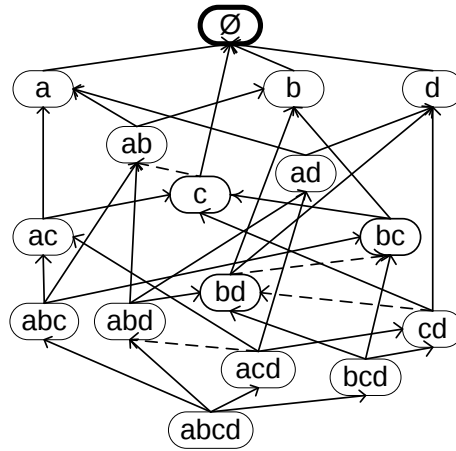
3.4.3 Computing Next-Preferred Reasoning based on Dominance Testing

Dominance testing is the problem of determining whether an outcome is preferred over another. It is of fundamental importance in many applications. Santhanam et al. (2010) provided an encoding of TCP-nets in the form of a Kripke structure for CTL and showed how to compute dominance using NuSMV, a model checker for CTL. The results of their experiments proved the feasibility of the approach. They reduced dominance testing to reachability analysis in a graph of outcomes. Specifically, they formalized the *ceteris paribus* semantics of preferences in terms of a direct and succinct representation of preference semantics using Kripke structures [Clarke et al. (2000)] that encoded preferences over outcomes as reachability within a graph of outcomes. In essence they reduced dominance testing to satisfiability of corresponding

a = Name
 b = Address
 c = Bank Routing Number
 d = Bank Account Number

-
- P1. $\{d\}, \{\} : \{b\} \succ \{c\}$
 - P2. $\{b\}, \{a\} : \{c\} \succ \{d\}$
 - P3. $\{\}, \{d\} : \{a, b\} \succ \{c\}$

(a) CI-net preference statements



(b) Induced preference graph

- \emptyset

- a
- b
- d

- ab
- ad
- c

- ac
- bc
- abc
- bd

- abd
- cd

- acd
- bcd

- abcd

(c) Complete Ordering of the variables

Figure 3.3 CI-net preference statements, Induced preference graph depicting the most preferred statement with respect to given preferences, and the Complete ordering of the variables [Oster et al. (2013)]

temporal formulas in the model and provided a translation from TCP-nets to the Kripke model specification language of a widely used model checker NuSMV [Cimatti et al. (2002)].

In a more recent work by Oster et al. (2013), a qualitative preference formalism based on CI-nets for representing and reasoning with client preferences over the relative sensitivity of sets of credentials was presented. They developed a model checking-based approach for analyzing the preference graph, efficiently verifying whether one set of credentials is more sensitive than another (dominance testing). Moreover, they identified the least (minimum) sensitive set of information that may be disclosed by the client to get access to the desired service. This laid the foundation for computing the next preferred statements in the i-PrefR tool [Santhanam et al. (2010); Oster et al. (2013)] which we shall use to compute the preferred order of defense policies qualitatively. The technique is based on iterative verification and refinement of the preference graph for computing a sequence of credential sets, ensuring that a credential set with higher sensitivity is never returned before one with lower sensitivity.

The following example borrowed from Oster et al. (2013) illustrates the main idea behind computing the preferred statements.

Example 3.4.2. *Consider a client who is interested in obtaining some financial quote (e.g., auto and/or home insurance, mortgage, etc.) using an online service. Suppose that there are multiple servers that provide the required service, and each server’s access control policy requires a combination of several credentials from the client before granting access to the service. We consider four such credentials: **the client’s name, residential address, bank account number, and bank routing number.***

The client has some qualitative preferences over the relative importance of his credentials based on their sensitivity. The rationale behind these preferences is that the client would like to make it impossible (or at least difficult) for a third party to perform any financial transaction maliciously posing as the client. Therefore, from the client’s perspective, the objective is to choose the server that provides the desired financial service by requiring the least sensitive set of client credentials. Consider the following qualitative preferences specified by the client:

P1. *If bank account number is disclosed, then I would rather give my address than bank*

routing number *to the server*.

P2. *If I have to disclose my address but not my name, then I would prefer to give my bank routing number rather than my bank account number.*

P3. *If I don't need to disclose my bank account number, I will give my name and address instead of my bank routing number.*

Based on these preferences, the client can identify successively more sensitive sets of credentials (starting from the empty set) and verify whether a set of credentials is sufficient to satisfy the access control policy of any server providing the desired service. Any server that accepts this least sensitive acceptable set of credentials may be selected to provide the service to the client.

Given two choices (of sets of credentials), deciding the preference of one choice over the other is referred to as dominance testing. Dominance testing is known to be PSPACE-complete [Boutilier et al. (2004); Goldsmith et al. (2008)]. The objective now is to identify a most preferred set γ of credentials that the client has to disclose in order to meet the server's requirements. The corresponding CI-net statements (figure 3.3(a)) and the induced preference graph based on the preferences (figure 3.3(b)) are depicted in Figure 3.3. The computed ordering of all the variables is depicted in Figure 3.3(c). Essentially, the method involves two processes:

1. *Decide:* Automatically decide the preference of a set of credentials over another, where preferences are specified using CI-nets.
2. *Order:* Use the above decision process to automatically identify the preference ordering of sets of credentials, starting from the most preferred sets and ending in the least preferred ones.

The ordering of the preference statement variables based on the CI-net preference statements is depicted in Figure 3.3. This example provides a clear picture as to how the preference

logic in i-PrefR generates preferred statements based on user inputs, which are obviously qualitative in nature. We use the same framework in our work for computing preferences over the different possible defense policies to prevent/stop an attack based on inputs from the system administrator with respect to a particular attack graph.

3.4.4 Preferences Over Defense Policies

The main aim of our work is to provide a qualitative analysis of the network based on the input logical attack graph and certain preferences over the different impact parameters as defined in CVSS. Based on these inputs and preferences, our goal is to provide a prioritized list of valid defense policies. The prioritization of the defense policies is a two-step process. In step 1, the preference reasoner, i.e., i-PrefR [Santhanam et al. (2010); Oster et al. (2013)] is fed with the qualitative preference statements based on certain preferences of the user. The preference reasoner then returns a prioritized list of CVSS impact valuations. Each of these elements in the list consists of a set containing encoded values corresponding to the levels of the different impact parameters (*Confidentiality impact, Integrity impact, and Availability impact*). Each set is sent back from the preference reasoner one at a time in order based on preference (highest to lowest) which is fed into our analysis engine for step 2. In step 2, our analyzer generates two sets based on the set from the preference reasoner. The first set consists of a set of *exact nodes* ($E_{=}$), which is essentially all the *privilege* and *fact* nodes that have the exact same corresponding impact valuations as the returned set from the preference reasoner. The second is *set of betters* ($E_{>}$) that consists of all *privilege* and *fact* nodes that have impact valuations lower than the returned set from the preference reasoner. For the following example, consider the input attack graph illustrated in Figure 1.1 and that each impact parameter has 3 levels: *high, medium, and low*.

Example 3.4.3. *Suppose, the impact valuations for the privilege and fact nodes present in the attack graph, in Figure 1.1, are as defined in Table 3.4.4¹. Now, suppose based on the inputs provided by the user, the preference reasoner returns the following set of impact valuations **{medium, low, medium}**, corresponding to **confidentiality, integrity and availability***

¹ p_1 is the goal privilege and has no impact valuations

impacts respectively. The analyzer then generates the following 2 sets: $E_{=} = \{c_3\}$; $E_{>} = \{p_2, c_1\}$.

Node name	Confidentiality Impact	Integrity Impact	Availability Impact
p_2	medium	low	low
c_1	low	low	low
c_2	low	medium	low
c_3	medium	low	medium
c_4	high	medium	high

Table 3.1 An example of impact values

Now that the set of $E_{=}$ and $E_{>}$ are generated, the analyzer calls our *bottom-up* and *top-down* methods successively to analyze in the corresponding manner. In both the approaches (*bottom-up* and *top-down*), the first policy that is generated is the super-policy that results from negating the union of the $E_{=}$ and $E_{>}$ sets conjuncted over the literals. In essence, the first policy generated in our example is, $\neg(c_3 \vee p_2 \vee c_1)$. The algorithms for the *bottom-up* and *top-down* analysis are presented in Algorithms 1, and 2 respectively.

3.4.4.1 Bottom Up Analyzer

Generating Super Policy

Proceeding further with our example in 3.4.3, and the input valuations as per Table 3.4.4, we reached the point, where the *bottom-up analyzer* was called. As stated earlier, the initial policy generated by the bottom-up analyzer is the policy $\neg c_3 \wedge \neg p_2 \wedge \neg c_1$. Now, the verifier returns *true* and *check* is set to *true* in line 3, as this is a valid defense policy. This is because, by negating fact nodes c_1 and c_3 and the privilege p_2 , essentially all the possible exploits viz: e_1 , e_2 , and e_3 are negated, thus, not allowing any attacker to exploit these vulnerabilities and gain goal privilege p_1 .

Generating Strict Subsets of Super Policy

The bottom-up analyzer then generates all possible valid combinations of the super-policy, and stores them in *allPossibleCombinations*, according to line 4 of the Algorithm 1. Here, the *allPossibleCombinations* will be: $\{(\neg c_3 \wedge \neg p_2), (\neg c_3 \wedge \neg c_1), \neg c_3\}$. Line 5 then prints the policy

Algorithm 1 Bottom-Up analyzer

```

1: procedure BOTTOM-UP(policy, nodes)    ▷ nodes is the list of nodes in the attack graph
2:   if verify(policy, nodes) = True then
3:     check ← True
4:     Generate allPossibleCombinations of policy in decreasing order of size of policies
5:     Print policy, “is a valid policy”                                     ▷ policy =  $E_{=} \cup E_{>}$ 
6:   else
7:     Quit and print “No valid policies exist”
8:   end if
9:   while  $|markedPolicies| < |allPossibleCombinations|$  do
10:    policyCombinations ← generateCombinations(policy,  $|policy| - 1$ )
11:    if policyCombinations  $\subseteq$  markedPolicies then
12:      policy ←  $\{p | p \in allPossibleCombinations \text{ and } p \notin markedCombinations\}$ 
13:    else
14:      policy ←  $\{p | p \in policyCombinations \text{ and } p \notin markedCombinations\}$ 
15:    end if
16:    check ← verify(policy, nodes)
17:    if check = True then
18:      Print policy, “is a valid policy”
19:    else
20:      Print policy, “is an invalid policy”
21:      for ( $i=1$ ;  $i < |policy|$ ;  $i++$ ) do
22:        policyCombinations ← generateCombinations(policy,  $i$ )
23:        markedPolicies ← markedPolicies  $\cup$  policyCombinations
24:      end for
25:    end if
26:    markedPolicies ← markedPolicies  $\cup$  policy
27:  end while
28: end procedure

```

and a message stating that it is valid policy. The algorithm in *bottom-up analyzer* then tries to find sub-policies of this super-policy that can still prevent attacks effectively and enters the while loop at line 9 of Algorithm 1. The next policy combinations that are generated in line 10 are: $\{(\neg c_3 \wedge \neg p_2), (\neg c_3 \wedge \neg c_1)\}$ by the *bottom-up analyzer*. The policy to be picked first here is, $(\neg c_3 \wedge \neg p_2)$ according to line 14 of the Algorithm 1, which is again a valid policy and the verifier returns *true* in line 16, as none of the exploits in the attack graph can be exploited by the attacker. Line 17 evaluates to *true*, so, the policy is printed along with a message stating it is valid in line 18. Finally in line 26, the policy is added to the list of marked policies and the next iteration of the while loop begins.

Exploring the Subset Policies Space

In the next iteration the next policy combinations generated is, $\{\neg c_3\}$, since it is the only possibility. Now, $\neg c_3$ is a valid policy (verifier returns *true*) along with a message stating that a monitor (IDS rule, firewall policy etc.) needs to be placed on the privilege p_2 to prevent exploits e_1 and e_3 from being attacked without any knowledge of the system administrator and the policy is again displayed along with a message stating it is valid according to lines 17, and 18 respectively. The policy is then added to the list of marked policies in line 26 and the next iteration of the while loop begins. In the next iteration, the policy combinations generated is an *empty set* ($\{\}$), as there can be no policy smaller than $\neg c_3 \wedge \neg p'_2$. So, the *if* statement in line 11 becomes *true*, and the policy chosen according to line 12 is, $(\neg c_3 \wedge \neg c_1)$, which is the first policy present in *allPossibleCombinations* but not in the list of already marked policies. Now, $(\neg c_3 \wedge \neg c_1) \wedge \neg p'_2$ is also a valid policy as long as a monitor is placed on the *privilege node* p_2 to prevent exploit e_3 from being compromised, because exploits e_1 , and e_2 are already protected by negating c_1 and c_3 respectively. There are no more policies generated for this input impact valuation ($\{medium, low, medium\}$), sent from the preference reasoner. It is worth noting that $(\neg p_2 \wedge \neg c_1)$, $\neg p_2$, and $\neg c_1$ are not considered defense policies for the input valuation $\{medium, low, medium\}$, since, they do not match the exact specifications for it. This is because, in the context of finding the preferred policy, the assumption is that the analyzer is going to have preference values in descending order (from better to worse). As a result for each iteration policies which are formed from the combinations of the elements in $E_{>}$ (set of

better valuations) should have been already explored. In essence, the elements in the $E_{>}$ set cannot on their own (or combination of them) form a defense policy (even if it is valid); they have to be combined with at least an $E_{=}$ set (set of exact valuations) element.

Based on the characteristics of the algorithm for BOTTOM-UP, we present the following proposition:

Proposition 3.4.1. *For a given preferred valuation ϑ , algorithm BOTTOM-UP returns a policy P such that $P \not\models Def(p_i)$, where p_i is the goal privilege and the impact valuations for policy P is ϑ , then $\nexists P'$ such that $P' \subset P$ and the impact valuations for P' is ϑ and $P' \models Def(p)$.*

Proof: Suppose there exists a policy P such that $P \not\models Def(p_i)$, where p_i is the goal privilege and $Def(p_i)$ is defined as in the presence of all exploits can one defend privilege p_i and the impact valuations for policy P is ϑ . Suppose there also exists a policy P' such that $P' \subset P$ and the impact valuations for P' is ϑ and $P' \models Def(p)$. However, in SAT solving, if a statement containing conjunctions over a set of literals does not satisfy a condition, none of the combinations of the subset of that conjunction can ever satisfy the same condition. Hence, if $P \not\models Def(p)$ then $\nexists P'$ such that $P' \subset P$ and the impact valuations for P' is ϑ and $P' \models Def(p)$.

■

Let us try to understand the above proposition with the help of an example. Suppose, one of the possible policies for the example we saw earlier was $(\neg c_1 \wedge \neg c_2)$, but, this is an invalid policy based on the structure of the attack graph. Hence, sub-policies of this policy, i.e. $\neg c_1$, and $\neg c_2$ are invalid policies also (which is true).

3.4.4.2 Top Down Analyzer

Generating Super Policy

When *top-down analyzer* is called, the first policy generated is the super-policy which is the union of $E_{=}$ and $E_{>}$ sets, which is, $(\neg c_3 \wedge \neg p_2 \wedge \neg c_1)$. Now, the verifier returns *true* and *check* is set to *true* in line 3 of Algorithm 2, as this is a valid defense policy, for the same reasons stated earlier.

Generating Strict Subsets of Super Policy

The next task performed by the top-down analyzer is to generate all possible valid combinations of the super-policy in increasing order of size (opposite of *bottom-up* approach), and stores them in *allPossibleCombinations*, according to line 4. Here, the *allPossibleCombinations* will be: $\{\neg c_3 \wedge \neg p'_2, (\neg c_3 \wedge \neg p_2), (\neg c_3 \wedge \neg c_1)\}$. Line 5 then prints the policy and a message stating that it is valid policy.

Exploring the Subset Policies Space

A lot of sub-policies of this super-policy is possible that can still prevent attacks, so the algorithm enters the while loop at line 9 of Algorithm 2. The next set of policy combinations that is generated in line 10, is of size 1, i.e., $\{\neg c_3 \wedge \neg p'_2\}$, and the verifier returns *true* along with a message stating that a monitor (IDS rule, firewall policy etc.) needs to be placed on the privilege p_2 to prevent exploits e_1 and e_3 from being attacked without any knowledge of the system administrator and the policy is again displayed along with a message stating it is valid according to lines 18, and 19 respectively. The policy is then added to the list of marked policies in line 24 and the next iteration of the while loop begins. As there doesn't exist any other policy of size 1, the Algorithm 2 enters the *if-else* block. As the *if* condition is satisfied, the control goes to line 13 and the policy now becomes, $(\neg c_3 \wedge \neg p_2)$, again a valid policy and the verifier returns *true* in line 17, as none of the exploits in the attack graph can be exploited by the attacker. Line 18 evaluates to *true*, so, the policy is printed along with a message stating it is valid in line 19. Finally in line 24, the policy is added to the list of marked policies and the next iteration of the while loop begins. In the next iteration, the exact same events occur, in which the policy chosen is, $(\neg c_3 \wedge \neg c_1) \wedge \neg p'_2$, which is also a valid policy as long as a monitor is placed on the *privilege node* p_2 to prevent exploit e_3 from being compromised, because exploits e_1 , and e_2 are already protected by negating c_1 and c_3 respectively. There are no more iterations following this policy, since the cardinality of the sets *allPossibleCombinations* and *markedCombinations* are equal. Hence, no more policies generated for the input impact valuation $\{medium, low, medium\}$, sent from the preference reasoner.

Based on the characteristics of the algorithm for TOP-DOWN, we present the following proposition:

Proposition 3.4.2. *For a given preferred valuation ϑ , algorithm TOP-DOWN returns a policy P such that $P \models Def(p_i)$, where p_i is the goal privilege and the impact valuations for policy P is ϑ , then $\nexists P'$ such that $|P'| < |P|$ and the impact valuations for P' is ϑ and $P' \models Def(p)$.*

Proof: Suppose there exists a policy P such that $P \models Def(p)$ and there also exists a policy P' such that $|P'| < |P|$ and the impact valuations for P' is ϑ and $P' \models Def(p_i)$, then P' then *allPossibleCombinations* (all possible combinations of policies of the super-policy) has a policy combination P_c such that $|P_c| < |P|$ and $P_c \models Def(p_i)$. However, according to line 4 of the Algorithm 2, the *allPossibleCombinations* is constant because it stores all possible combinations of the super policy. Thus, the policy $|P| \geq |P_c|$, so, $\nexists P_c$ such that $|P_c| < |P|$ and $P_c \models Def(p_i)$. Therefore, $\nexists P'$ such that $|P'| < |P|$ and the impact valuations for P' is ϑ and $P' \models Def(p)$. ■

In the above proposition note that, that cardinality of P' is never strictly less than P ($|P'| < |P|$). This observation stems from the fact that, the top-down analyzer computes, all possible combinations of the super-policy in increasing order of policy size. It then picks the first policy (the smallest policy) from that set. Since, it explores all policies from smallest to biggest, there cannot exist a smaller valid policy than the first explored valid policy by the analyzer.

Example 3.4.4. *Consider the following scenario. In iteration i , the preference reasoner provides as input the preferred values: {low, low, low}, corresponding to confidentiality, integrity, and availability impacts respectively for the same attack graph as illustrated in Figure 1.1 and the impact values for each node same as described in Table 3.4.4. The set of $E_=" and $E_>$ for this input would yield the following, $E_=" = \{c_1\}$, $E_> = \{\}$. The first policy is, $\neg c_1$, for both bottom-up and top-down analyzers, which is not a valid policy as it negates only exploit e_1 . However, exploits e_2 and e_3 can be exploited in a variety of fashions by an attacker. The top-down and bottom-up analyzer fails to find any policy that matches with the preferred value and informs the preference reasoner to provide the next preferred value.$*

Algorithm 2 Top-down analyzer

```

1: procedure TOP-DOWN(policy, nodes)    ▷ nodes is the list of nodes in the attack graph
2:   if verify(policy, nodes) = True then
3:     check ← True
4:     Generate allPossibleCombinations of policy in increasing order of size of policies
5:     Print policy, “is a valid policy”    ▷ policy =  $E_{=} \cup E_{>}$ 
6:   else
7:     print “No valid policies exist” and Quit
8:   end if
9:   pass = ( $|policy| - 1$ )
10:  while  $|markedPolicies| < |allPossibleCombinations|$  do
11:    policyCombinations ← generateCombinations(policy,  $|policy| - pass$ )
12:    if policyCombinations  $\subseteq$  markedPolicies then
13:      policy ←  $\{p | p \in allPossibleCombinations \text{ and } p \notin markedCombinations\}$ 
14:    else
15:      policy ←  $\{p | p \in policyCombinations \text{ and } p \notin markedCombinations\}$ 
16:    end if
17:    check ← verify(policy, nodes)
18:    if check = True then
19:      Print policy, “is a valid policy”
20:    else
21:      Print policy, “is an invalid policy”
22:      pass – –
23:    end if
24:    markedPolicies ← markedPolicies  $\cup$  policy
25:  end while
26: end procedure

```

Example 3.4.5. For this example consider the attack graph illustrated in Figure 3.4, the order of policies generated and verified as valid/invalid by both the bottom-up analyzer (Figure 3.5) and top-down analyzer (Figure 3.6) for a particular preferred valuation of confidentiality, integrity, and availability impacts against the same set of impact values for the nodes. The green boxes represent valid policies while the red boxes represent invalid policies. It is worth noting that the bottom-up analyzer, on encountering an invalid policy automatically marks all its sub policies as invalid also, and never explores them. This is justified because, if a policy is invalid, none of the sub policies formed by combinations over the literals in that policy can be a valid defense policy according to the same reasons stated previously in Section 3.4.4.

Our research focuses on 3 aspects: a simple view of logical attack graphs, integrating the

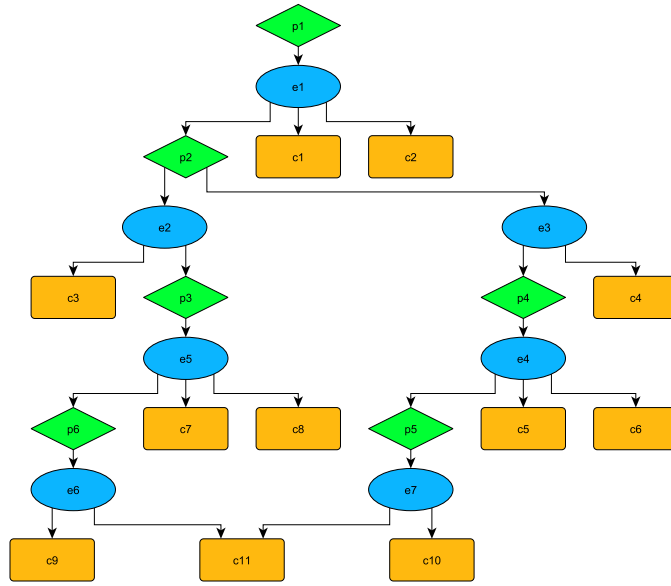


Figure 3.4 Attack graph described in Example 3.4.5

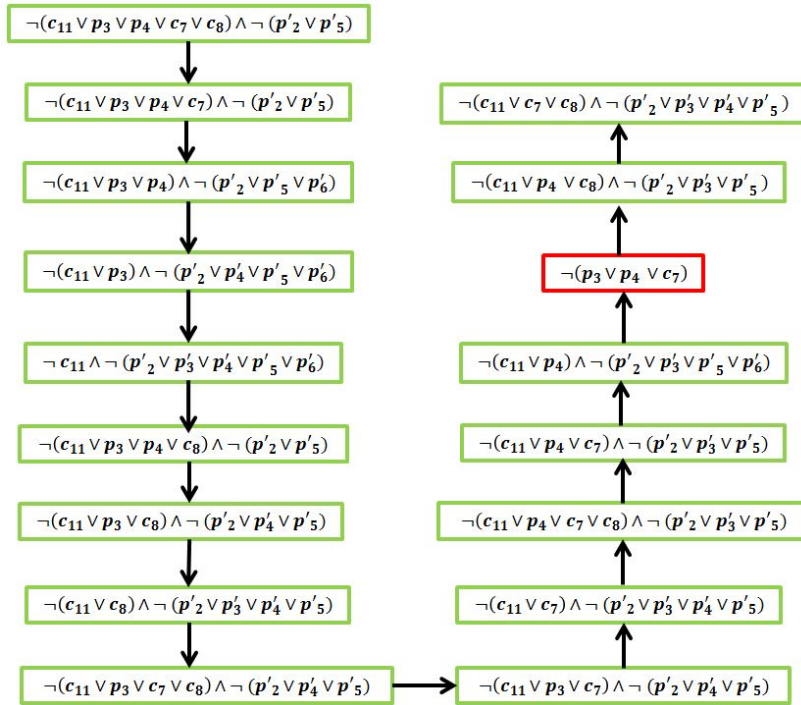


Figure 3.5 Bottom-up policy generation/validation order

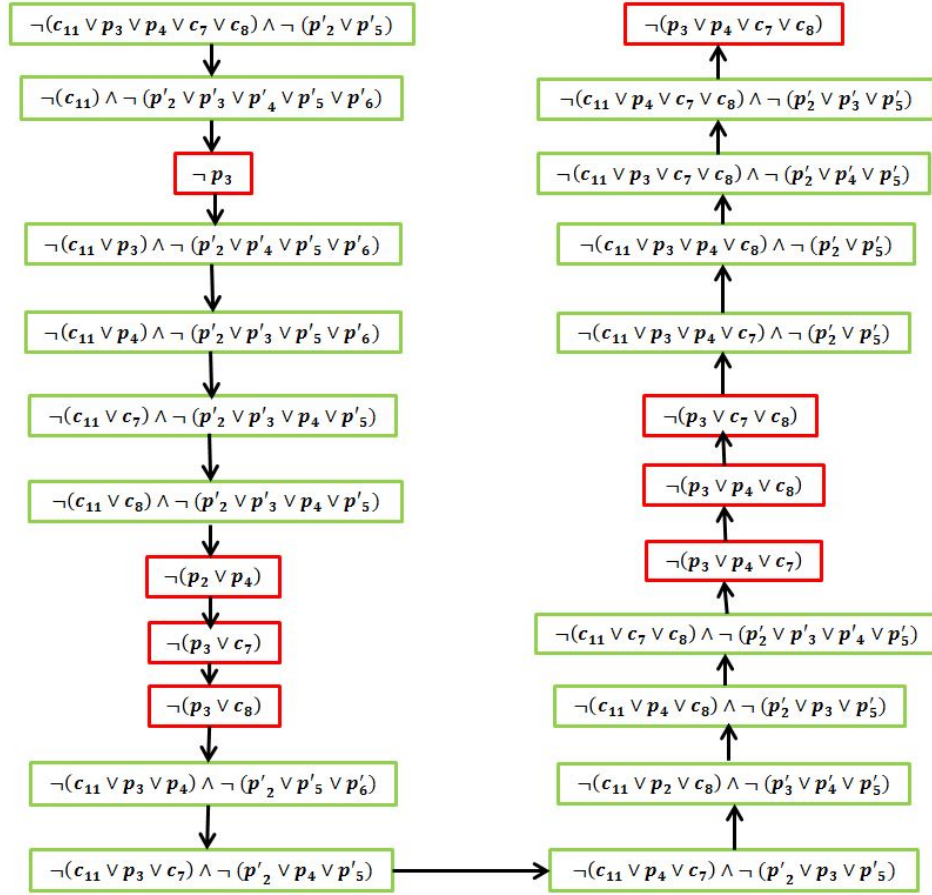


Figure 3.6 Top-down policy generation/validation order

CVSS impact metrics with analysis of defense policies, and providing preferences over these defense policies based on qualitative inputs. We have designed two analysis engines, namely, the *bottom-up analyzer* and the *top-down analyzer*. There is no conclusive evidence to suggest, which analyzer is more effective or best, but our endeavour has been to equip the system administrator with as much analysis power as possible. While the *bottom-up analyzer* fares well for most attack graphs, however, if the rate of invalid policies is very less, or if the possibility of combinations is very high and the system administrator is interested in smaller sized defense policies, then the *top-down analyzer* may be more effective. We formulate preferences over the user inputs using preference reasoner discussed in section [3.4.3](#)

CHAPTER 4. Implementation and Tool Development

We have realized dependency logical attack graphs (without cycles) and defence policy validation and prioritization based on the attack graph through the tool we developed for our research called, *DrAGON*¹. Figure 4.1 presents an overview of the various modules used in *DrAGON*. For generating a prioritized defense policy list to the user (system administrator) we have used, i-PrefR (preference reasoning tool) [Santhanam et al. (2010); Oster et al. (2013)]. i-PrefR is a reasoner for qualitative preferences expressed in languages such as CP-nets [Boutilier et al. (2004)], TCP-nets [Brafman et al. (2006)] and CI-nets [Bouveret et al. (2009)], and was founded on decision theory, formal methods and software engineering. The user of our tool need not necessarily be aware of how this preference reasoner functions. We have implemented our tool, *DrAGON*, to work with i-PrefR in the back-end based on preference inputs from the system administrators. Here we briefly discuss the foundations and theory behind the i-PrefR reasoning tool.

4.1 Architecture Overview

Our tool, *DrAGON*, is completely modular, Figure 4.1, illustrates the architectural overview of the main analysis engine. Besides the modules, displayed in figure 4.1, there are other modules, namely, the “generate” module, that contains, the AttackGraphGenerator, CVSS-FileGenerator, GMLGenerator, GraphFileGenerator, and ImpactMetricsAnalyzer. Before we discuss what the *generate* module does, we explain how the rest of our tool works and analysis is performed.

The AttackGraphMain takes in as input, the filenames, and location of the input attack graph file, the CVSS impact assignment file for each of the privilege and fact/configuration

¹Defense by pReferred policies for Attack Graph of the Network

nodes in the attack graph, and send it to the parser (the input and output semantics and syntax is explained elaborately in the later sections of this chapter). The TreeParser and CVSSParser then parses these input files respectively and send it to the TreeValidator to validate the feasibility of the input attack graph. For instance, if there are more than one node with the same name in the attack graph, or if any of the fact/configuration nodes have children according to the input, then the validator returns false and stops further processing. Once the validator, has successfully validated that, the input file is correct, the logical dependency attack graph (acyclic) is setup, and each of the nodes are assigned the appropriate CVSS impact metrics. This attack graph, along with the preferred output from the i-PrefR is fed to the ImpactAnalyzer. Based on the input from the i-prefR, the ImpactAnalyzer generates a set of *exact* node matched to the input impact values and a set of nodes that have valuation, *better* than the valuation got from the preference reasoner. These two sets are then sent to the PolicyGenerator, which generates policies, either in top-down or bottom-up fashion and sends it back to the ImpactAnalyzer. The ImpactAnalyzer sends this candidate policy, along with attack graph to the PolicyVerifier, which returns either “yes”, or “no”, indicating a valid and an invalid defense policy. This process, continues, until, no more defense policies can be found for that input preferred CVSS impact valuation. After which the next preferred value is fed in from the preference reasoner.

Now, let us understand what is meant by top-down and bottom-up approach of policy generation. When the ImpactAnalyzer is fed the attack graph, and a preferred CVSS impact valuation, the ImpactAnalyzer generates a set of exact nodes matching the preferred valuation, and a set which has values better than the preferred valuation. This can be understood from the example in the Table 4.1. The table depicts the various levels of the CVSS impact metrics for the privilege and configuration nodes in Figure 1.1. Here, very-low impact = 1, low impact = 2, low-medium impact = 3, high-medium impact = 4, high impact = 5, and very-high impact = 6. For all the other nodes, the default value of their impact metrics = 7. This ensures, they are not featured in the better set by any chance. Now, suppose the valuation returned by i-PrefR is, *confidentiality impact* = 3 (low-medium), *integrity impact* = 3 (low-medium), and *availability impact* = 3 (low-medium). Then, the ImpactAnalyzer would generate $E_- = \{c_1\}$,

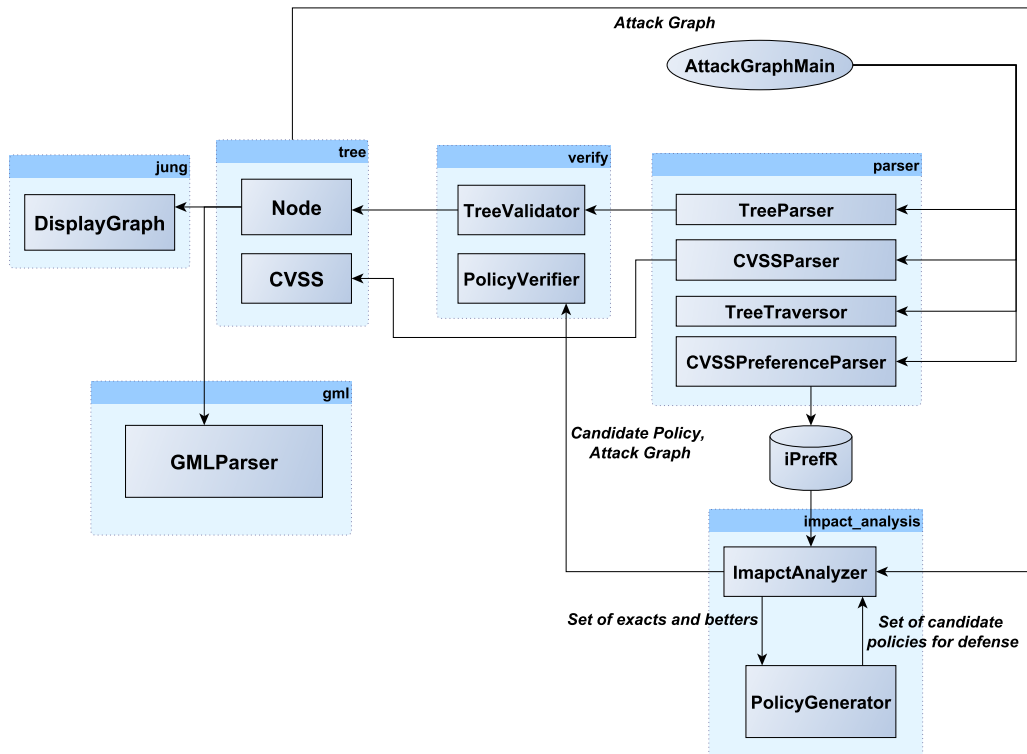


Figure 4.1 Architectural overview and data flow in *DrAGON*

Node	Confidentiality Impact(C)	Integrity Impact(I)	Availability Impact(A)
p_2	3	1	3
c_1	3	3	3
c_2	6	6	6
c_3	2	3	2
c_4	1	1	1

Table 4.1 An example valuation of the CVSS impacts for each of the nodes

and $E_{>} = \{p_2, c_3, c_4\}$.

The ImpactAnalyzer then sends these 2 sets to the PolicyGenerator who return the first policy as, $(\neg c_1 \wedge \neg p_2 \wedge \neg c_3 \wedge \neg c_4)$. This obviously is a valid policy. So, the PolicyGenerator will try to find an even smaller sub-set of this policy, say $(\neg c_1 \wedge \neg p_2 \wedge \neg c_3)$, which is again a valid policy. Again, the PolicyGenerator will try to find a smaller sub-set of this policy that can be a valid defense policy, so it will find, $(\neg c_1 \wedge \neg c_3) \wedge \neg p'_2$. This is valid because, we negated c_1 , and c_3 , thus negating the exploits e_2 , and e_1 all together. However, e_3 can still be exploited, so, we instruct the system administrator to place a monitor on privilege p_2 for keeping exploit e_3 under check. A monitor essentially is an IDS rule or firewall policy that would prevent attackers from exploiting e_3 easily and unnoticed. This is a partial ordering of the defense policy and one of the advantages of our work, because we are performing a qualitative analysis. Such a measure could never be achieved by any quantitative analysis approach. Further atomic policies are not possible, so the PolicyGenerator again goes back to the last subset found, and tries to find defense policies, that can be valid, one such policy possible is, $(\neg c_1 \wedge \neg c_3 \wedge \neg c_4)$. It is worth noting here, that $(\neg c_1 \wedge \neg p_2 \wedge \neg c_4)$ is an invalid policy, since, e_2 can still be exploited, which could lead to p_2 actually being achieved by the attacker inspite of considering it negated by the system administrator. So, our tool, ensures, it verifies the truth about the assumptions made by the system administrator and helps him/her out, even if they have made a wrong assumption. This is another remarkable achievement of our work.

4.2 The Helper Modules of *DrAGON*

The tool developed during the course of this research is extremely modular. Each part is able to function entirely independent of the other parts, thus allowing for easy extensions in the future. In this section, we discuss about the modules, which do not explicitly help in our analysis and defense policy formation and validation. However, they are useful, as they either provide a seamless interface for the user, or help create log files, that can help document all the test results.

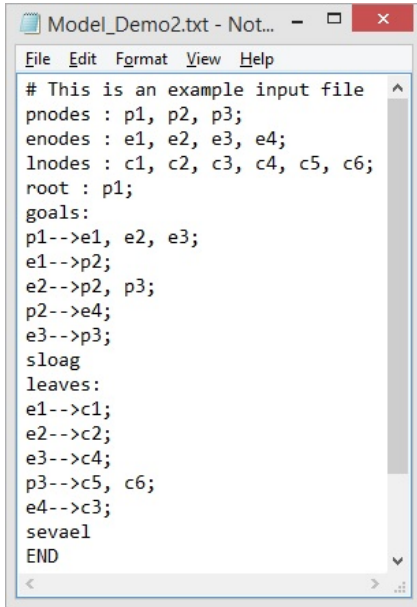
The *generate* package contains the following classes, AttackGraphGenerator, CVSSFileGenerator, GMLGenerator, GraphFileGenerator, and ImpactMetricsAnalyzer. The AttackGraphGenerator creates attack graph files according to the semantics and syntax discussed in section 4.2. The CVSSFileGenerator helps create files with CVSS input valuations for the nodes in the corresponding attack graph. This is essential for conducting thorough testing of our tool, where random test scenarios for each attack graph model can be created. The generated files would then act as inputs for analysis.

The GMLGenerator creates GML (Graphical Markup Language) files for any attack graph. These can then be viewed in any graph editor. We recommend using Y-ed Graph Editor [yWorks (2013)]. yEd is freely available and runs on all major platforms: Windows, Unix/Linux, and Mac OS X. It offers various visual layouts for the graphs, at the touch of a button. We recommend, installing y-Ed along with our tool on the system and associate “.gml” file extensions to yEd Graph Editor.

The ImpactMetricsAnalyzer takes in the attack graph and a set of impact values from the preference reasoner to conduct analysis. It first calls the ImpactAnalyzer for performing the analysis and then writes the output to a log file.

4.3 Input Languages

For inputting attack graphs, we have developed a very simple and easy to use language. The semantics and syntax of this input language can be understood from Figures 4.2 and 4.3. Let us first understand the syntax of the input attack graph file. As we can see in Figure



```

# This is an example input file
pnodes : p1, p2, p3;
enodes : e1, e2, e3, e4;
lnodes : c1, c2, c3, c4, c5, c6;
root : p1;
goals:
p1-->e1, e2, e3;
e1-->p2;
e2-->p2, p3;
p2-->e4;
e3-->p3;
sloag
leaves:
e1-->c1;
e2-->c2;
e3-->c4;
p3-->c5, c6;
e4-->c3;
sevae1
END

```

Figure 4.2 Example input attack graph file

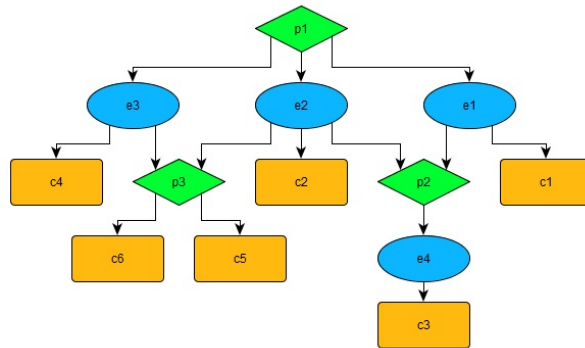


Figure 4.3 Corresponding logical dependency attack graph

4.2, the keywords are: *pnodes*, *enodes*, *lnodes*, *root*, *goals-sloag* block, *leaves-sevae1* block, and *END*. The “#” (hash symbol) is used to comment a line. The Table 4.3 explains the meaning and usage of the keywords used for the input attack graph file. Table 4.3 contains the list of delimiters and their usage.

4.4 Output Visualizations

Our tool, *DrAGON*, allows the user to visualize the attack graph for correctness after input. For this purpose we recommend that the user have yEd graph editor installed on their system [yWorks (2013)]. The example attack graph in Figure 1.1, can be viewed in different layouts in yEd as depicted in Figure 4.4. Also, yEd allows the user to create, edit and re-use, new and existing *gml* input files for generating the attack graphs. Our tool also has a built-in viewer using JUNG (Java Universal Network/Graph Framework) [Madadhain et al. (2005), Bernstein (2010)]. The GUI interface using JUNG is a very primitive view and does not allow the user the ability to visualize different layouts or edit the graph. The same attack graph is illustrated in Figure 4.5. It is obvious to the naked eye that this visualization is not very pretty and does not allow the user to change layouts like yEd. So, we recommend the usage of yEd along with

Keyword	Description
pnodes	Used to describe all the privilege nodes present in the attack graph. The keyword <i>pnodes</i> is followed by a “:” followed by the names of all the privilege nodes separated by commas (,).
enodes	Used to describe all the exploit nodes present in the attack graph. The keyword <i>enodes</i> is followed by a “:” followed by the names of all the exploit nodes separated by commas (,).
lnodes	Used to describe all the fact/configuration nodes present in the attack graph. The keyword <i>lnodes</i> is followed by a “:” followed by the names of all the exploit nodes separated by commas (,).
root	Used to specify the name of the root node (typically a privilege node). The keyword <i>root</i> , is followed by a “:” followed by the name of the root privilege node of the attack graph. There can be only 1 root node for any attack graph.
goals-sloag block	Used to specify the parent(s) of each privilege and exploit node. Any parent node (singular) is written on the left of the arrow ($- \rightarrow$) symbol and all its corresponding children (privilege and exploit) nodes are written on the right side of the arrow ($- \rightarrow$) symbol. Each child node is separated by comma (,). The keyword <i>goals</i> followed by a colon (:) signifies the start of the block, and the keyword <i>sloag</i> is used to indicate the end of the block.
leaves-sevael block	Used to specify the parent(s) of each fact node. Any parent node (singular) is written on the left of the arrow ($- \rightarrow$) symbol and all its corresponding children (fact/configuration) nodes are written on the right side of the arrow ($- \rightarrow$) symbol. Each child node name is separated by comma (,). The keyword <i>leaves</i> followed by a colon (:) signifies the start of the block, and the keyword <i>sevael</i> is used to indicate the end of the block.
END	Represents the end of the file.

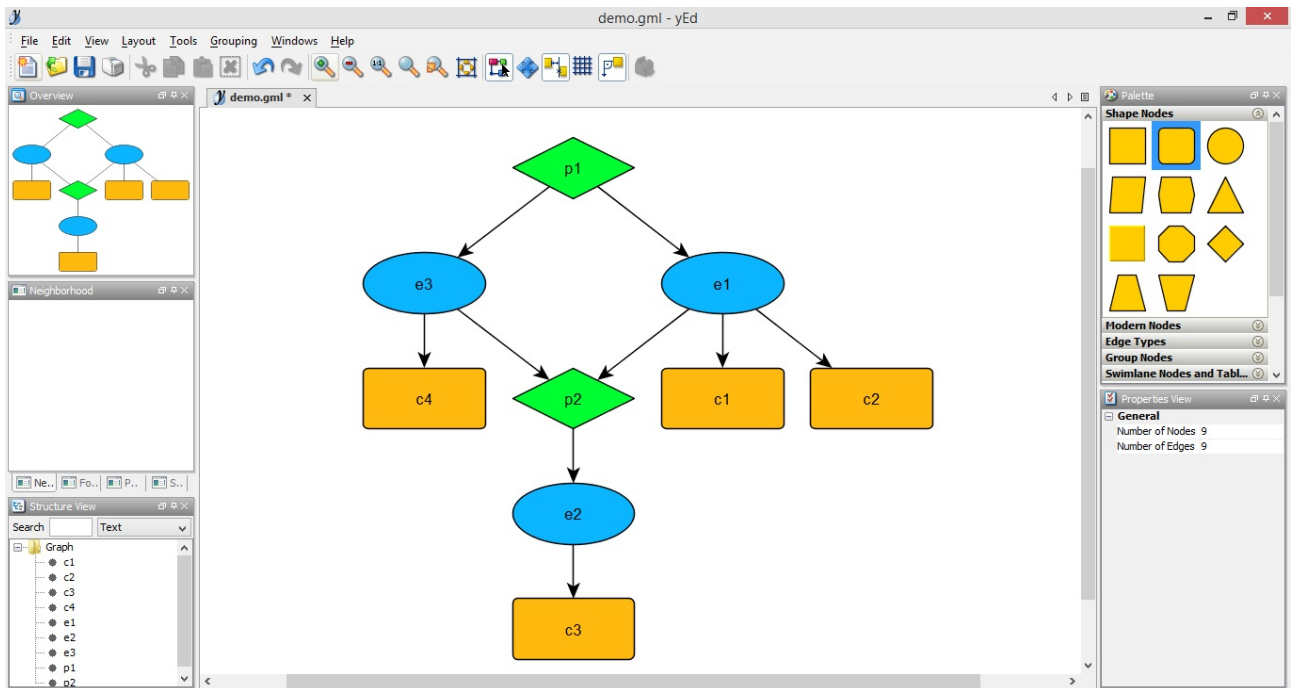
Table 4.2 Input language keywords & description

Delimiter	Usage
Comma (,)	The comma (,) symbol is used to separate the names of the nodes in any line.
Colon (:)	The colon (:) follows immediately after the usage of a keyword. The colon (:) delimiter separates the keyword from the input parameters.
Semi-colon (;)	The semi-colon (;) indicates the end of a line of input for the input file.
Hash (#)	The hash (#) comments that line and marks it as not needed to parse for creating the attack graph.

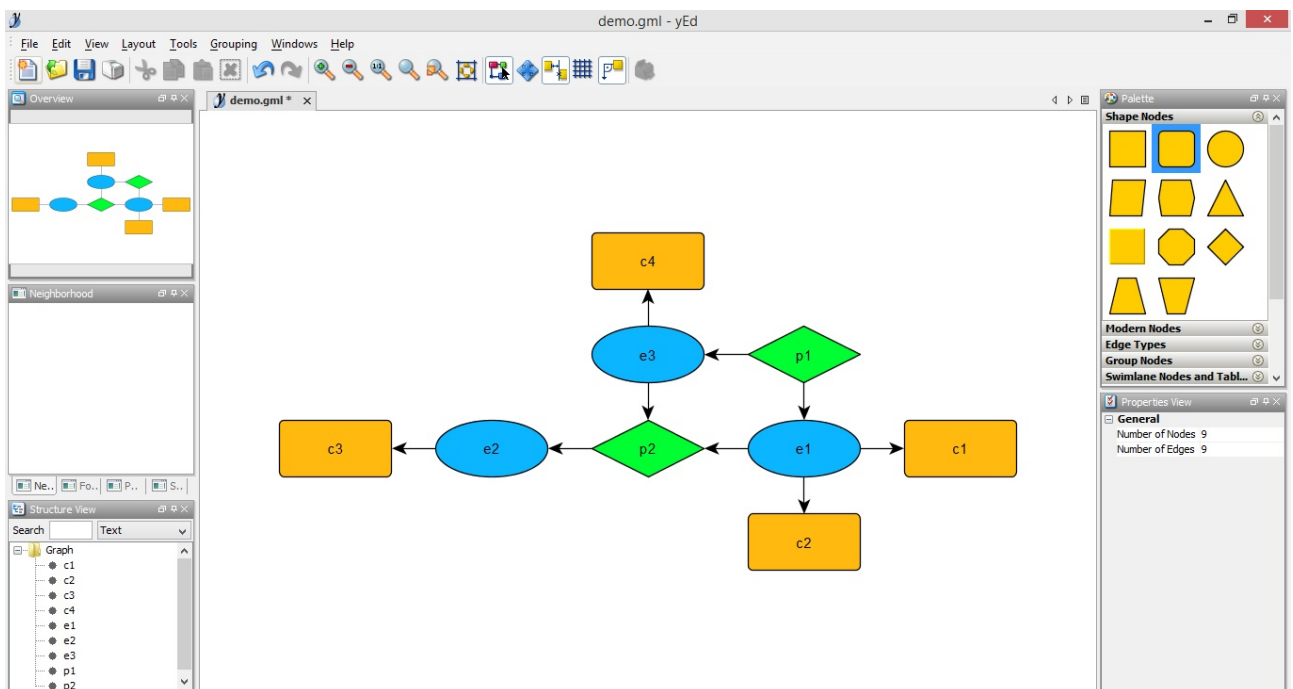
Table 4.3 Delimiters and their usage in the input attack graph file

our tool for best results.

We have described here the functionality and implementation of the framework we have designed during the course of our research. We have implemented our tool and run several conclusive tests to prove both its correctness and effectiveness. The correctness of the algorithms used in the implementation of this framework has been discussed in detail in section 3.4.4. The results that prove that our framework works and performs as expected according to the underlying ideologies of the respective algorithms is described in Chapter 5.



(a) Directed Tree Layout



(b) Classic Orthogonal Layout

Figure 4.4 Different layouts visualizing the same attack graph using yEd

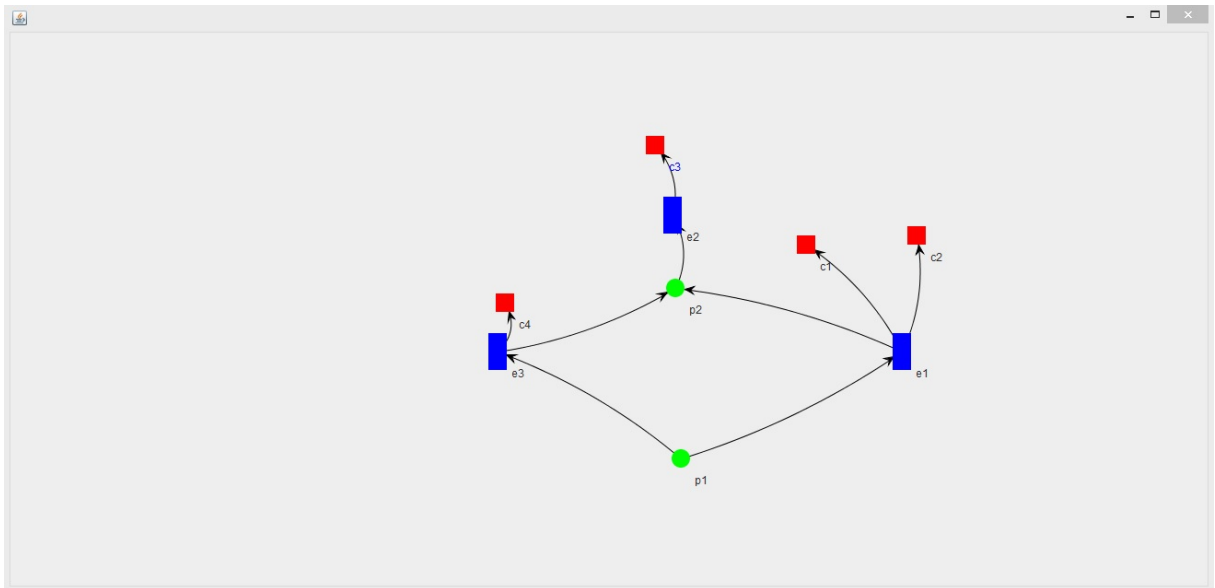


Figure 4.5 A graphical visualization of the attack graph using JUNG

CHAPTER 5. Experiments and Results

In this chapter we will discuss the preliminary results, for generating and validating defense policies using both our developed algorithms *bottom-up* and *top-down analyzers* as described in Algorithms 1 and 2 respectively. These results are based on attack graphs modeled mostly from previous research literatures discussed in Chapter 2, and some which we developed to test the correctness of our tool.

5.1 Bottom-up Analyzer

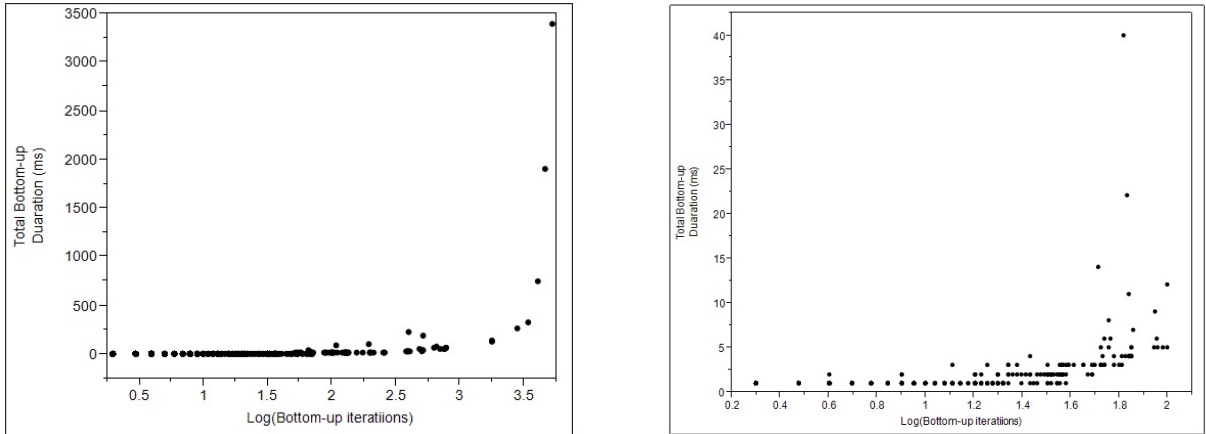
We have already seen how the algorithm for bottom-up analyzer works in section 3.4.4.1. For validating the correctness and understanding the efficiency of the algorithm, we ran a substantial number of tests. We used 12 logical attack graph models, of which 10 were present in previous research papers in the field of attack graph theory, and 2 were developed by us, for validating the correctness of the algorithms. We used 6 different impact levels (*very low, low, low medium, high medium, high, and very high*) for assigning the *confidentiality, integrity* and *availability impacts* for each *privilege* and *fact node*. We randomly generated 100 different assignments of impacts for the nodes present in an attack for all the 12 models. We then ran all 216 ($6 * 6 * 6 = 216$) possible combination of values for impacts and ran our analyzers, giving us a substantial number of output results (259,200) to analyze the correctness and the efficiency of the algorithm, in regard to time taken (in ms). For generating random files with different impact values for the nodes present in any attack graph model, we used the *Random* class of java.

It is important to note, that there exists no other tool/framework, for such analysis that we know of. So, the major concern for us while developing our tool (*DrAGON*) was to ensure that

it performed the task correctly. Although, our major concern was not efficiency in terms of time, the results indicate, that our *bottom-up analyzer* is pretty fast. The maximum iterations that we have encountered amongst, all the 259,200 results that we have analyzed is, 5,276, which took only 3,393 ms ($< 4s$). This is really promising in terms of the possibilities and application of our tool, in real-life scenarios. This particular case arose for an attack graph model with 9 *privilege nodes*, 16 *exploit nodes*, and 20 *fact nodes*. The graphs depicted in Figure 5.1 plots, the total time taken for analyzing a given policy and all its sub policies against the $\log(\text{total number of iterations})$. We chose a log scale instead of a direct scale because, as we get to higher number of iterations, the total time taken increases sharply, also, the number of such high iteration instances are rare. So, to accommodate all the results, we chose a *log* scale. What we see from the graphs is a really encouraging and positive from our research perspective. While we have already proven the correctness of the Algorithm 1 in section 3.4.4.1, from the results displayed here, we see that the time taken for generating sub policies and validating them is also pretty fast. The graph in Figure 5.1(a), contains all the results, while the graph in Figure 5.1(b), depicts a magnified view of iterations for upto 100 iterations. From the Figure 5.1(b), we can infer, that the time taken depends not only on the number of iterations taken, but also on the structure of the attack graph itself and the position of the nodes present in the graph being analyzed.

5.2 Top-down Analyzer

Just like the *bottom-up analyzer*, we have seen the algorithm for *top-down analyzer* in section 3.4.4.2. For validating the correctness and understanding the efficiency of the algorithm, we ran the same number of tests (259,200) on the top-down analyzer as we did for the *bottom-up analyzer*. We used the same 12 logical attack graph models for testing purposes, and the same 6 impact levels (*very low, low, low medium, high medium, high, and very high*) for assigning the *confidentiality, integrity* and *availability impacts* for each *privilege* and *fact node*. We used the same randomly generated 100 different assignments of impacts for the nodes present in an attack for all the 12 models. We then ran all 216 possible combinations of values for impacts and ran our analyzers, giving us 259,200 output results to analyze the correctness and the



(a) Log(Bottom-up iterations) vs Time(ms) (all values) (b) Log(Bottom-up iterations) vs Time(ms) (iterations ≤ 100)

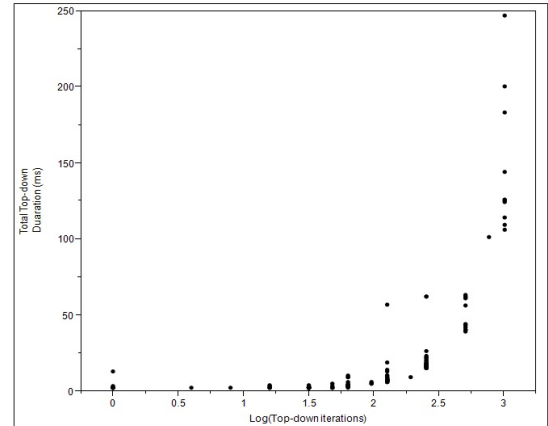
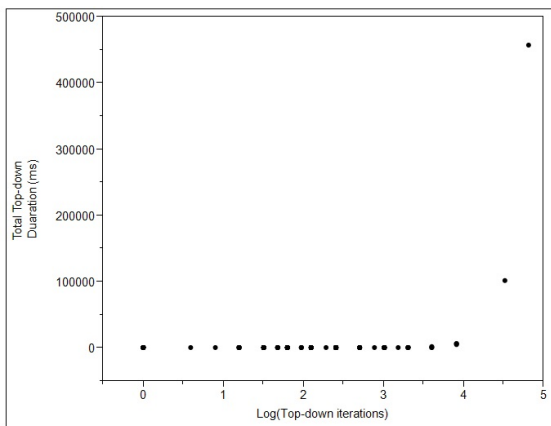
Figure 5.1 Graphs representing iterations vs time (ms) for bottom-up analyzer

efficiency of the algorithm.

From what we saw about the logic used for top-down analysis in section 3.4.4.2, it is clear, that the *top-down analyzer* will definitely, not be more efficient over the *bottom-up analyzer* with respect to time. This is because, like we saw in the Example 3.4.5, the *top-down analyzer* explores all the policies from smaller to larger size. In cases where the policies generated might not be valid for that attack graph, it might be too much overhead to explore all the policies. In most instances, with complicated attack graph structures, it is quite difficult to find valid policies. In such instances, the *bottom-up analyzer* computes faster than the *top-down analyzer*, by neglecting sub policies of invalid policies and saves time. Unfortunately, the *top-down analyzer*, explores all possible policies for any super policy, so, it requires considerable more time. The maximum iterations performed by the *top-down analyzer* is, 65,536 and it took 457,466 ms ($< 500s$). However, the positive thing is, it completed the analysis and explored all the policies. The thing to note while analyzing the results for our top-down analyzer is, our main aim for designing the algorithm was not efficiency, but to give the user a chance to explore all the possible policies (even if they are invalid). Also, the policies generated are from smaller size to larger size, so if the system administrator wants the smallest effective defense

policy first, he/she can make use of the top-down analyzer. Also, if the structure of the attack graph is very complicated yet very small policies can be enforced to stop attacks, then this approach would be very beneficial.

The graphs depicted in Figure 5.2 plots, the total time taken for analyzing a given policy and all its sub policies against the $\log(\text{total number of iterations})$. We chose a log scale instead of a direct scale because, as we get to higher number of iterations, the total time taken increases sharply, also, the number of such high iteration instances are rare. So, to accommodate all the results, we chose a *log* scale. The results are in alignment with our expected results, as the time taken increases rapidly as the number of iterations increases. Also, the number of iterations themselves are more widely spaced than for *bottom-up* approach, as the top-down analyzer tries to explore all possible policies, thus increasing the number of iterations considerably for a valid super-policy. Both these trends can be observed clearly in Figure 5.2(a). Figure 5.2(b) illustrates a magnified view for iterations upto 1024 times.



(a) $\text{Log}(\text{Top-down iterations})$ vs Time(ms) (all values) (b) $\text{Log}(\text{Top-down iterations})$ vs Time(ms) (iterations ≤ 1024)

Figure 5.2 Graphs representing iterations vs time (ms) for top-down analyzer

The results described in this chapter are very much along the expected results during the development of the framework. The *bottom-up analyzer* is quite time efficient, while the *top-down analyzer* is great for policy exploration. Both of these analyzers have their own

advantages. Our main goal is not to base solely on efficiency, but to provide the user with different methods of analysis. The correctness of both these analyzers is a highlight of our research and has been discussed in gamut in section [3.4.4](#).

CHAPTER 6. Conclusion

6.1 Summary

We have presented a practical solution to the problem of finding and verifying policies for complicated networks and developed a novel framework that works with qualitative preferences and can essentially help design defense policies, based on the attack graph. We have also, provided a means of visualizing the attack graph before it is fed into our tool for analysis. We have allowed the use of standard graph storage format, namely GML (Graphical Markup Language) to create, edit and store these attack graphs. We have also provided a custom input language, which is simple to use for storing very large attack graphs easily in a textual format.

We have introduced the concept of representing logical attack graphs from defender perspective and formally defined it. The advantages of using this representation, over existing representation techniques have been discussed elaborately in Chapter 3. Furthermore, we provided a visualizing medium of the attack graph using a graphical user interface, which can be used to create, edit, and store attack graphs of any size and in any layout. Our work is a novel work, in trying to integrate the CVSS impact metrics and logical attack graphs in a qualitative manner. By using the CVSS metrics, we have ensured a standard of comparing results across a variety of networks.

Through this research, we have provided a means to provide preferred policies for defense against attacks in a network. We have formulated 2 unique algorithms, to generate and validate the correctness of the policies that are either enforced or proposed. Since, we provide a preferred order of defense policies; the user is assured of the best policy first, without having to sift through all the possible policies. However, our *top-down analyzer* ensures that the interested user can explore all policies.

6.2 Future Work

The field of attack graphs has seen an increased interest in the last decade. It will continue to be a popularly studied field as humans become increasingly more dependent on technology for information and day-to-day work. Our work, is applicable but not restricted to just the field of attack graphs, but can be applicable to other broader areas like network security and even medical science.

By considering certain cells as sites for attack by viruses, the concept of preferred defense policies can be modeled to provide best treatment procedure for any particular disease based on certain personal traits/history of the patient. Essentially, these traits, or past medical history can be thought of as configurations and the organs, or sites affected by the ailment as the privilege nodes. The challenge in this case would be to document impact values something similar along the lines of CVSS for network vulnerability. However, it can be modeled very easily considering just qualitative aspects like we have done here, without any quantitative estimates for solving the disease. Whether such an analysis is practical and useful would need to be examined more.

6.2.1 Extending Our Work to Include Cyclic Graphs

Our tool is primarily based on acyclic graphs; we would like to extend it to include cyclic graphs as well. The way to include cycles while computing the satisfiability of a policy is by calculating the set of fixed points in order to circumvent the problem of running into cycles.

As part of future work, we plan to study the feasibility of identifying and pre-computing only the necessary information required to guarantee the termination of cyclic attack graphs.

6.2.2 Allowing Partial Ordering over the Policy Search Space

One feature we would like to add to our existing framework is to allow the preference reasoner to return partial impact valuation set. For instance, instead of the preference reasoner returning $\{confidentiality\ impact\ value, integrity\ impact\ value, availability\ impact\ value\}$, it can return just a partial order of these values, like: $\{confidentiality\ impact\ value, availability\ impact$

value}, or *{integrity impact value, availability impact value}*, or even *{integrity impact value}*.

This would be a great feature to include allowing even more flexibility to the user while designing policies and providing preferences. Right now, all three impact values have to be clearly defined; otherwise the search for the sets containing nodes with exact matches and better valuation than the preferred impact value returned by the preference reasoner would break down.

6.2.3 Improving the Output Visualizations

Another aspect of our framework we want to improve would be the visual aspect. While our tool provides a rich graphical interface for visualizing the input attack graph, there currently does not exist a graphical interface to visualize the defense policies. Providing a graphical user interface for visualizing the defense policies (both valid and invalid), would assist the user in understanding, why a policy works or it does not.

6.2.4 Designing a Hybrid Algorithm Using the Bottom-up and Top-down Analyzers

While we have designed two algorithms, based on exact opposite ideologies, namely the *bottom-up* and the *top-down analyzer*. While the *bottom-up* approach goes from bigger to smaller policies, the *top-down* approach goes from small to bigger sized policies. Essentially, there is a wide gap in these two extreme approaches, which can be filled by a more elegant and hybrid approach, capitalizing on the positives of both these approaches. It would be interesting to see the performance of such an algorithm and how it fares compared to either or both of these policies.

BIBLIOGRAPHY

- AmenazaLtd. (2003). Creating secure systems through attack tree modeling. https://www.amenaza.com/downloads/docs/5StepAttackTree_WP.pdf.
- Ammann, P., Wijesekera, D., and Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. *Proceedings of the 9th ACM Conference on Computer and Communications Security*.
- Baldwin, R. (1994). Kuang: Rule based security checking. Technical report, MIT Lab for Computer Science.
- Bernstein, G. (2010). JUNG the Java Universal Network/Graph Framework—is a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. As an open-source library, JUNG provides a common framework for graph/network analysis and visualization. More info about JUNG at <http://jung.sourceforge.net/index.html>.
- Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, 21:135–191.
- Bouveret, S., Endriss, U., and Lang, J. (2009). Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, pages 67–72, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Brafman, R. I., Domshlak, C., and Shimony, S. E. (2006). On graphical modeling of preference and importance. *J. Artif. Int. Res.*, 25(1):389–424.

- Cashell, B., Jackson, W. D., Jickling, M., and Webel, B. (2004). The economic impact of cyber-attacks. Congressional Research Service, Library of Congress.
- Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In Brinksma, E. and Larsen, K., editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer Berlin Heidelberg.
- Clarke, E., Grumberg, O., and Peled, D. (2000). *Model Checking*. MIT Press.
- Cuppens, F., Mieke, A., and O’Berry, B. (2002). Alert correlation in cooperative intrusion detection framework. *Proceedings of the 2002 IEEE Symposium on Security and Privacy*.
- CVSS (2007). Common vulnerability scoring system (cvss). For more details refer to: <http://www.first.org/cvss> and <http://nvd.nist.gov/cvss.cfm?calculator&adv&version=2>.
- Dawkins, J., Campbell, C., and Hale, J. (2002). Modelling network attacks: Extending the attack tree paradigm. *Proceedings of the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*.
- Frigault, M., Wang, L., Singhal, A., and Jajodia, S. (2008). Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM Workshop on Quality of Protection, QoP ’08*, pages 23–30, New York, NY, USA. ACM.
- Goldsmith, J., Lang, J., Truszczynski, M., and Wilson, N. (2008). The computational complexity of dominance and consistency in cp-nets. *J. Artif. Intell. Res. (JAIR)*, 33:403–432.
- Graham, B. (2005). Hackers attack via chinese web sites. <http://www.washingtonpost.com/wp-dyn/content/article/2005/08/24/AR2005082402318.html>. Accessed: 2014-04-14.
- Hassan, M. M. et al. (2013). Current studies on intrusion detection system, genetic algorithm and fuzzy logic. *International Journal of Distributed & Parallel Systems*, 4(2).
- Homer, J. and Ou, X. (2009). Sat-solving approaches to context-aware enterprise network security management. *IEEE JSAC Special Issue on Network Infrastructure Configuration*, 27(3).

- Homer, J., Ou, X., and Schmidt, D. (2009). A sound and practical approach to quantifying security risk in enterprise networks. Technical report, Computing and Information Sciences Department, Kansas State University.
- Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S. R., and Singhal, A. (2013). Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security*, 21(4):561–597.
- Howard, M., Pincus, J., and Wing, J. (2005). Measuring relative attack surfaces. In Lee, D., Shieh, S., and Tygar, J., editors, *Computer Security in the 21st Century*, pages 109–137. Springer US.
- Howard, R. A. and Matheson, J. E. (1984). *Readings on the Principles and Applications of Decision Analysis*. Strategic Decision Group.
- Huang, H., Zhang, S., Ou, X., Prakash, A., and Sakallah, K. (2011). Distilling critical attack graph surface iteratively through minimum-cost sat solving. *27th Annual Computer Security Applications Conference (ACSAC)*.
- Jajodia, S., Noel, S., and O’Berry, B. (2005). Topological analysis of network attack vulnerability. *Managing Cyber Threats: Issues, Approaches and Challenges*.
- Kabiri, P. and Ghorbani, A. A. (2005). Research on intrusion detection and response: A survey. *International Journal of Network Security*, 1:84–102.
- Keramati, M. and Akbari, A. (2013). Cvss-based security metrics for quantitative analysis of attack graphs. In *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, pages 178–183.
- Kunz, W. and Pradhan, D. K. (1994). Recursive learning: A new implication technique for efficient solutions to cad problems - test, verification and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1143–1159.
- Lippmann, R. and Ingols, K. (2005). An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory.

- Madadhain, J., Fisher, D., Smyth, P., White, S., and Boey, Y. (2005). Analysis and visualization of network data using jung. *Journal of Statistical Software*, 10:1–35.
- Manadhata, P. and Wing, J. (2011). An attack surface metric. *Software Engineering, IEEE Transactions on*, 37(3):371–386.
- Manadhata, P. K., Wing, J. M., Flynn, M., and McQueen, M. (2006). Measuring the attack surfaces of two ftp daemons. In Karjoth, G. and Massacci, F., editors, *QoP*, pages 3–10. ACM.
- Mell, P., Scarfone, K., and Romanosky, S. (2007). *CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. FIRST: Forum of Incident Response and Security Teams.
- Narain, S., Levin, G., Malik, S., and Kaul, V. (2008). Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*, 16(3):235–258.
- Ning, P., Xu, D., Healey, C., and Amant, R. S. (2004). Building attack scenarios through integration of complimentary alert correration methods. *Proceedings of the 11th Annual Network and Distributed System Security Sumposium*.
- Noel, S., Jacobs, M., Kalapa, P., and Jajodia, S. (2005). Multiple coordinated views for network attack graphs. *VIZSEC '05 Proceedings of the IEEE Workshops on Visualization for Computer Security*, pages 12:1 – 12:8.
- Oster, Z., Santhanam, G., Basu, S., and Honavar, V. (2013). Model checking of qualitative sensitivity preferences to minimize credential disclosure. In Păsăreanu, C. and Salaiün, G., editors, *Formal Aspects of Component Software*, volume 7684 of *Lecture Notes in Computer Science*, pages 205–223. Springer Berlin Heidelberg.
- Ou, X., Boyer, W. F., and McQueen, M. A. (2006). A scalable approach to attack graph generation. *13th ACM Conference On Computer and Communications Security (CCS 2006)*.
- Phillips, C. and Swiler, L. (1998). A graph-based system for network-vulnerability analysis. *Proceedings of the New Security Paradigms Workshop*.

- Ramakrishnan, C. and Sekar, R. (2000). Model-based analysis of configuration vulnerabilities. *Proceedings of the 7th ACM Conference on Computer and Communication Security*.
- Ritchey, R. and Ammann, P. (2000). Using model checking to analyze network vulnerabilities. *Proceedings of the IEEE Symposium on Security and Privacy*.
- Santhanam, G. R., Basu, S., and Honavar, V. (2010). Dominance testing via model checking. *AAAI Conference on Artificial Intelligence*. Details about the i-PrefR tool, examples and download links can be found at, <http://fmg.cs.iastate.edu/project-pages/preference-reasoner/>.
- Santhanam, G. R., Oster, Z. J., and Basu, S. (2013). Identifying a preferred countermeasure strategy for attack graphs. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, CSIIRW '13*, pages 11:1–11:4, New York, NY, USA. ACM.
- Schiffman, M., Eschelbeck, G., Ahmad, D., Wright, A., and Romanosky, S. (2004). Cvss: A common vulnerability scoring system. *National Infrastructure Advisory Council (NIAC)*.
- Schneier, B. (1999). Attack trees: Modeling security threats. *Dr. Dobbs Journal*.
- Sheyner, O., Haynes, J., Jha, S., Lippmann, R., and Wing, J. (2002). Automated generation and analysis of attack graphs. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 254–265.
- Sheyner, O. M. (2004). Scenario graphs and attack graphs. Master’s thesis, Carnegie Mellon University.
- Smith, G. S. (2004). Recognizing and preparing loss estimates from cyber-attacks. *Information Systems Security*, 12(6):46–57.
- Stoffel, D., Kunz, W., and Gerber, S. (1995). And/or graphs. Technical report, Max Planck Fault Tolerant Computing Group, University of Potsdam.
- Swiler, L., Phillips, C., Ellis, D., and Chakerian, S. (2001). Computer-attack graph generation tool. volume 2, pages 307–321.

- Swiler, L., Phillips, C., and Gaylor, T. (1998). A graph-based network-vulnerability analysis system. Technical report, Sandia National Labs.
- Wang, L., Islam, T., Long, T., Singhal, A., and Jajodia, S. (2008). An attack graph-based probabilistic security metric. volume 5094 of *Lecture Notes in Computer Science*, pages 283–296. Springer Berlin Heidelberg.
- Wang, L., Noel, S., and Jajodia, S. (2006). Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824.
- Wang, L., Singhal, A., and Jajodia, S. (2007). Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM Workshop on Quality of Protection, QoP '07*, pages 49–54, New York, NY, USA. ACM.
- Wilson, N. (2004). Extending cp-nets with stronger conditional preference statements. In *in Proceedings of AAAI-04*, pages 735–741.
- yWorks (2013). yEd is a free of charge general-purpose diagramming program with a multi-document interface. yEd is a powerful desktop application that can be used to quickly and effectively generate high-quality diagrams. Create diagrams manually, or import your external data for analysis. More info about yEd at http://www.yworks.com/en/products_yed_about.html.
- Zerkle, D. and Levitt, K. (1996). Netkuang - a multi-host configuration vulnerability checker. *Proceedings of the 6th USENIX Unix Security Symposium*.