A COMPREHENSIVE APPROACH TO ENTERPRISE NETWORK SECURITY MANAGEMENT

by

JOHN HOMER

B.S., Harding University, 2002
M.S., Kansas State University, 2006

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2009

Abstract

Enterprise network security management is a vitally important task, more so now than ever before. Networks grow ever larger and more complex, and corporations, universities, government agencies, *etc.* rely heavily on the availability of these networks. Security in enterprise networks is constantly threatened by thousands of known software vulnerabilities, with thousands more discovered annually in a wide variety of applications. An overwhelming amount of data is relevant to the ongoing protection of an enterprise network.

Previous works have addressed the identification of vulnerabilities in a given network and the aggregated collection of these vulnerabilities in an attack graph, clearly showing how an attacker might gain access to or control over network resources. These works, however, do little to address how to evaluate or properly utilize this information.

I have developed a comprehensive approach to enterprise network security management. Compared with previous methods, my approach realizes these issues as a uniform desire for provable mitigation of risk within an enterprise network. Attack graph simplification is used to improve user comprehension of the graph data and to enable more efficient use of the data in risk assessment. A sound and effective quantification of risk within the network produces values that can form a basis for valuation policies necessary for the application of a SAT solving technique. SAT solving resolves policy conflicts and produces an optimal reconfiguration, based on the provided values, which can be verified by a knowledgeable human user for accuracy and applicability within the context of the enterprise network. Empirical study shows the effectiveness and efficiency of these approaches, and also indicates promising directions for improvements to be explored in future works. Overall, this research comprises an important step toward a more automated security management initiative.

A COMPREHENSIVE APPROACH TO ENTERPRISE NETWORK SECURITY MANAGEMENT

by

JOHN HOMER

B.S., Harding University, 2002
M.S., Kansas State University, 2006

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2009

Approved by:

Major Professor
Xinming Ou

Copyright

Abstract

Enterprise network security management is a vitally important task, more so now than ever before. Networks grow ever larger and more complex, and corporations, universities, government agencies, *etc.* rely heavily on the availability of these networks. Security in enterprise networks is constantly threatened by thousands of known software vulnerabilities, with thousands more discovered annually in a wide variety of applications. An overwhelming amount of data is relevant to the ongoing protection of an enterprise network.

Previous works have addressed the identification of vulnerabilities in a given network and the aggregated collection of these vulnerabilities in an attack graph, clearly showing how an attacker might gain access to or control over network resources. These works, however, do little to address how to evaluate or properly utilize this information.

I have developed a comprehensive approach to enterprise network security management. Compared with previous methods, my approach realizes these issues as a uniform desire for provable mitigation of risk within an enterprise network. Attack graph simplification is used to improve user comprehension of the graph data and to enable more efficient use of the data in risk assessment. A sound and effective quantification of risk within the network produces values that can form a basis for valuation policies necessary for the application of a SAT solving technique. SAT solving resolves policy conflicts and produces an optimal reconfiguration, based on the provided values, which can be verified by a knowledgeable human user for accuracy and applicability within the context of the enterprise network. Empirical study shows the effectiveness and efficiency of these approaches, and also indicates promising directions for improvements to be explored in future works. Overall, this research comprises an important step toward a more automated security management initiative.

Table of Contents

Table of Contents vi

List of Figures viii

List of Algorithms ix

List of Tables x

Acknowledgements xi

Dedication xii

1 Introduction 1
- 1.1 Enterprise Network Security Management . . . 1
- 1.2 Related Work . . . 5
  - 1.2.1 Attack Graphs . . . 5
  - 1.2.2 Risk Assessment . . . 7
  - 1.2.3 Network Reconfiguration . . . 8
- 1.3 MulVAL . . . 10
- 1.4 Original Contributions . . . 13

2 Attack Graph Simplification 17
- 2.1 An Example . . . 19
- 2.2 Subnet-Based Trimming . . . 21
- 2.3 Dominator-Based Trimming . . . 26
- 2.4 Discussion . . . 32

3 Quantitative Risk Assessment 35
- 3.1 Component Metrics . . . 36
- 3.2 Sound Risk Assessment - A First Approach . . . 39
- 3.3 Sound and Practical Risk Assessment . . . 42
  - 3.3.1 Definitions . . . 43
  - 3.3.2 Example - Figure 3.1 . . . 48
  - 3.3.3 Example - Figure 3.2 . . . 51
- 3.4 Algorithms . . . 54
- 3.5 Scalability and Complexity . . . 65
- 3.6 Implementation and Testing . . . 67

4 SAT Solving Approaches to Context-Aware Enterprise Network Security Management **73**
4.1 An Example . . . 76
4.2 Transforming attack graphs to Boolean formulas . . . 79
4.3 Iterative UnSAT Core Elimination . . . 82
4.3.1 Decision Reduction through Appeal to a Partial Ordering . . . 85
4.4 MinCostSAT . . . 86
4.4.1 User queries . . . 88
4.4.2 Cost Policies . . . 89
4.4.3 Scalability . . . 90
4.5 Discussion . . . 92
4.6 Implementation and Testing . . . 94
4.6.1 Policy Construction . . . 94
4.6.2 Application of Iterative UnSAT Core Elimination . . . 96
4.6.3 Application of MinCostSAT approach . . . 97

5 Open Problems and Future Work **100**
5.1 Open Problems . . . 100
5.2 Future Work . . . 102

6 Conclusion **104**

Bibliography **112**

A Network Models (MulVAL) **113**

B Proofs **117**

List of Figures

1.1 Publications of Software Vulnerabilities, Annually . . . 2
1.2 MulVAL Logical Attack Graph Generation . . . 11
1.3 MulVAL Logical Attack Graph . . . 12
1.4 Envisioned Process for Enterprise Network Security Management . . . 16

2.1 Sample Graph - Useless Steps . . . 18
2.2 Energy Management Network . . . 19
2.3 Energy Management Network – Attack Graph . . . 22
2.4 Energy Management Network - Subnet Graph . . . 23
2.5 Energy Management Network - Attack Graph (Inter-subnet Trimming) . . . 25
2.6 Energy Management Network - Attack Graph (Subnet-based Trimming) . . . 26
2.7 Sample attack graph *(left);* simplified representation of same *(right)* . . . 30
2.8 Graphical dominator tree *(left);* logical dominator graph *(right)* . . . 32
2.9 Energy Management Network - Attack Graph (Domination-based Trimming) 33

3.1 Example: Acyclic Attack Graph . . . 44
3.2 Example: Cyclic Attack Graph . . . 52
3.3 Example: Cyclic Attack Graph - Unfolded . . . 52
3.4 Example for Risk Assessment . . . 68
3.5 Example – Attack Graph . . . 69
3.6 Example – Attack Graph - Trimmed . . . 69
3.7 Example – Attack Graph, after vulnerability patched . . . 71

4.1 Small Enterprise Network . . . 76
4.2 Small Enterprise Network – Attack Graph . . . 77
4.3 Attack Graph Representation . . . 78

A.1 Model for example shown in Section 2.1 . . . 114
A.2 Model for example shown in Section 3.6 . . . 115
A.3 Model for example shown in Section 4.1 . . . 116

List of Algorithms

2.1 Pseudocode for constructing a representative OR-graph 28
2.2 Pseudocode for identifying logical dominance relationships 31
3.1 Pseudocode for full graph unfolding 40
3.2 Pseudocode for risk assessment 55
3.3 Pseudocode for computing $evalProb(N)$ 57
3.4 Pseudocode for computing $evalCondProb(D, N)$ 59
3.5 Pseudocode for $evalCycle(C)$ - assessment over nodes in cycle 61
3.6 Pseudocode for $tracePaths(n, P, C)$ - tracing acyclic paths through cycle . . 63
3.7 Pseudocode for $evalCycleNode(D, N)$ 64

List of Tables

3.1 Values calculated while solving for $\phi(\overline{A}_4, \overline{A}_5)$ 50
3.2 Risk assessment calculations for Figure 3.1 51
3.3 Values calculated while solving for $\phi(\overline{P}_{2a}, \overline{P}_{2b})$ 54
3.4 Risk assessment calculations for Figure 3.2 54
3.5 Scalability test results - risk assessment 66
3.6 Risk assessment calculations for Figure 3.4 70
3.7 Risk assessment calculations for Figure 3.4, lower metric for VPN server . . . 70
3.8 Risk assessment calculations for Figure 3.4, after vulnerability patched . . . 71

4.1 Scalability test results - SAT solving 91
4.2 Cost policy for Figure 4.3 95

Acknowledgments

I would like to express my gratitude to my advisor, Simon Ou, for his continual encouragement and assistance over the past several years. It was his work that motivated me to venture into the field of enterprise network security analysis and management. I have learned much from his example and I am grateful to have had the opportunity to work closely with him. His patience and perseverance in aiding me along the way have been indispensable in bringing me to this point. I have the deepest respect for Simon, both as a teacher and a researcher, and I look forward to years of continued collaboration with him.

I would like to thank my committee members for their participation in this process and for their comments and suggestions. I am particularly grateful to David Schmidt, whose insight and knowledge have been invaluable.

Finally, I would like to acknowledge the debt owed to everyone who has made my time here memorable - the faculty, staff, and students comprising this department. I am grateful to each professor who has instructed and encouraged me over the past five years, and I hope that I have acquired some measure of their dedication to scholarship. For my friends here, I am grateful for the time we have shared in cooperative academic advancement, as well as the time spent cheerfully avoiding the same.

Dedication

This work is dedicated to my wife, Laura, a woman of steadfast character and breath-taking beauty. I could not have done this without her.

Chapter 1

Introduction

1.1 Enterprise Network Security Management

Enterprise network security management is an undertaking of paramount importance in the world today. Corporations, universities, schools and government depend on the availability and functionality of these networks for communication, data storage and retrieval, and many other day-to-day activities and responsibilities. Enterprise networks continue to grow in both size and complexity, as increasingly more tasks are automated and more information is stored and made available through the network. With this increase, concerns for network security continue to grow. Network security management has generally become an ever-changing, never completed chore.

Security risks exist in almost every enterprise network, often undetected or unknown to those assigned to maintain it. Vulnerabilities are regularly discovered in almost every software application. Figure 1.1 charts the discovery of software flaws, grouped by the year published. The number of vulnerabilities "published" (or discovered and made known to the public) increased dramatically several years ago and has remained consistently high. Extrapolating from the number of publications published as of April 2009, a comparably high number of vulnerabilities will be published in 2009. This trend seems likely to continue, with a large and increasing number of vulnerabilities discovered and published in each ensuing year.

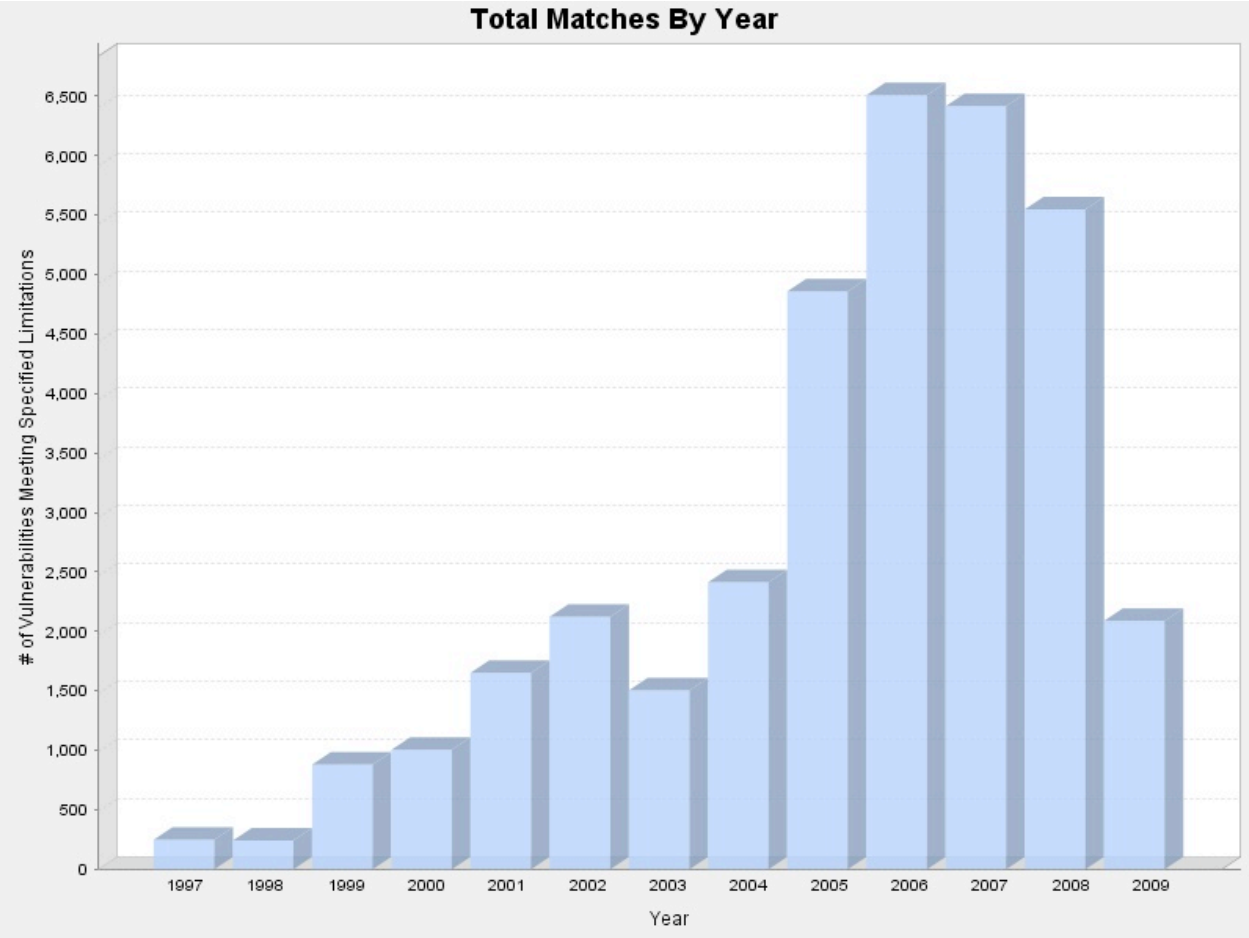


Figure 1.1: *Publications of Software Vulnerabilities, Annually*
[*Source:* `http://web.nvd.nist.gov`]

In a typical enterprise network, it is not merely the newly-published vulnerabilities that are a cause of concern. Vulnerabilities discovered several years ago may still persist, due to negligent patching protocols or even a lack of available patches. However, even keen attention to reviewing and maintaining enterprise network security cannot guarantee a freedom from security flaws or loopholes. The sheer number of notifications, vulnerability publications, and patches to be sifted and considered ensures that no human administrator can possibly be always and continuously aware of all the possible dangers and weaknesses in any given network configuration. Even a moderately-sized network can have many different attack paths which an attacker could exploit to gain unauthorized network access.

To make the issue more complicated, expectations and requirements for network usability are often in direct conflict with those of security. Enterprise network configuration management necessitates a careful balancing of these two interests. Without concern for usability, for example, security management becomes a trivial problem, resolvable by simply shutting down the network. This is clearly an untenable solution in almost any situation. Each enterprise network exists for a specific purpose, providing a means by which some group of intended users can perform certain tasks. Eliminating all threats to the security of the network will most likely interfere with this intended functionality. For this reason, even diligent security management rarely achieves zero vulnerabilities. Instead, security management must seek to mitigate risk within the network by making configuration changes in a context-aware manner.

Currently, the evaluation and management of security risks in an enterprise network is more an art than a science. System administrators operate by instinct and learned experience, often without any objective evaluation of their decisions or full knowledge of all ramifications of any changes. Even the most experienced administrator can accidentally introduce new vulnerabilities while trying to correct known issues. This dependence on human involvement practically guarantees security risks in almost any network.

Even assuming that a system administrator is aware of all current vulnerabilities in the enterprise network, there is currently no sound and quantitative measurement of risk within the network. Lacking such measurement, there is no reliable way to answer pertinent questions such as, "Where is our network most vulnerable?" and "If I change this configuration setting, will our network be more or less secure as a result?" Other individuals, lacking the experience of the administrator, have no verification that the administrator's intuition is correct. These questions require a quantitative model of security with clear measurement of risk and easy comparison of different network configurations.

Ideally, an administrator will be able to make proactive decisions toward better network security. A reliable quantification of risk together with structural knowledge about the

network provides the necessary foundation for these decisions. Even with full availability of this information, however, the problem remains non-trivial The complex interactions between settings and vulnerabilities form a difficult problem in seeking a solution that maintains essential network usability while reducing security risk.

It is clear, then, that enterprise network security management is a difficult and often unmanageable concern. I believe, however, that this issue can be trouble can be diminished. This dissertation addresses several key aspects of enterprise network security management, providing solutions for each and showing how they can be jointly applied.

1. Attack graph simplification. I will show the benefits attainable by reducing the amount of data included in the attack graph in two distinct techniques for attack graph trimming. By these techniques, (1) vital data can be made more clearly known to a human user and, (2) invalid or unusable data can be safely removed.

2. Sound quantification of risk. Without a reliable measurement of existing risk in the network, security management cannot move beyond decisions based on guesses and estimates. A provable calculation of security risk forms a necessary basis for stable decisions. I will present such an approach and demonstrate its effectiveness and applicability.

3. Automated, optimal nework reconfiguration. An optimal network configuration is the central desire in security management. I will show how two SAT solving techniques - MinCostSAT and UnSAT core - can be used to generate suggestions for network configuration changes to mitigage security concerns while sustaining the core functionality of the network.

These research aims represent different facets of a single goal - a comprehensive approach to enterprise network security management. Attack graph simplification provides for the human user an intuitive understanding of key security issues. Risk assessment produces sound measurement of the probability that any given attack can be successfully executed within

the network. These assessment values can be used as the basis for a security cost policy. Together with a parallel usability cost policy, SAT solving techniques can be used to construct a "best case" network configuration, considering and balancing the often conflicting desires for security and usability. Together, these research advances constitute a significant step toward greater automation and reliability in enterprise network management.

This development is important because it alleviates a human user's burden of searching for the best way to configure a system given a large number of complex security and usability constraints. Since configuration management is such an important and complex problem, it is unlikely that an automatic tool can take over a human system administrator's role overnight. Rather, I envision an incremental process whereby automated tools can assist a human user to manage an enterprise network, increasing the degree of automation as the tool is trained.

1.2 Related Work

1.2.1 Attack Graphs

Vulnerability analysis and attack graph generation have been well studied [2, 9, 10, 17, 18, 21, 23, 24, 39–41, 43, 44, 49–51, 53]. Tools have been developed to identify and report known vulnerabilities in running software versions, firewall policies, and other system settings. Attack graphs provide a visual representation of potential attack paths employing these vulnerabilities that could be followed to exploit system resources.

An early technique, developed by Phillips and Swiler [43, 51], created attack graphs in which nodes represent network states and arcs represent possible state transitions. Sheyner, *et al.* produced similar graphs by applying an approach based on symolic model checking [49, 50] These attack graphs, often called *state enumeration graphs* [37], included every possible system state attainable by an attacker moving through the network. Despite proposed techniques for reducing the overall size, these graphs suffered a fatal flaw in the exponential growth in size [2, 49].

To conquer this problem, a new approach was developed by Amman, *et al.* [2] based on the principle of *monotonicity* in attack steps, *i.e.,* a privilege once acquired by an attacker will not subsequently be forfeited in the acquisition of any additional privileges. Utilizing this principle, the same data contained in state enumeration graphs can be represented in *dependency attack graphs* with individual settings and privileges as vertices, rather than entire network states. In this type of approach, a graph vertex represents a system condition in some form of logical sentence, with graph edges representing causal relations between these system conditions. Various models of dependency attack graphs have since been developed, comparable in their approach and semantics but distinct in their presentation of the attack graph data. Beyond Amman's work, the different types of dependency attack graphs include the *exploit dependency graph* [37, 38], the *multiple prerequisite graph* [17], and the *logical attack graph* [40]. The MulVAL analysis engine and logical attack graph toolkit will be presented in more detail in Section 1.3.

More recent works have introduced improvements in the visualization of attack paths and the overall presentation of attack graph data. Lippman, *et al.* introduced the *predictive attack graph* to better display all hosts that can be comprised from a given starting location and to quickly (without rebuilding the graph) show the effects of patching or correcting existing vulnerabilities [24]. Noel, *et al.* suggested that complexity can be reduced through the use of protection domains to represent groups of machines with unrestricted interconnectivity [36, 37]. Lippmann, *et al.* proposed new visualization approaches to emphasize critical attack steps while clearly showing host-to-host reachability [58]. My work benefits from and builds upon results shown in most previous attack graph simplification works. My contribution is the use of attack graph trimming for specific purposes, such as improved data accessibility for human viewers or sound reductions for more efficient automated calculations.

1.2.2 Risk Assessment

The issue of security metrics has attracted much attention [20, 28, 29], and recent years have seen significant effort toward the development of quantitative security metrics [1, 4, 7, 26, 35, 42, 45]. Recent years have also seen effort on computing various metrics from attack graphs [12, 47, 54, 56, 57, 59]. My work builds upon results from some of these previous works and is unique in that it accurately accounts for both cyclic and shared dependencies in attack graphs.

Frigault, *et al.* [11, 12] utilizes the combination of attack graphs and Bayesian Network (BN) in measuring network security. Bayesian networks are well studied and many reliable, efficient implementations are commonly available for use; this allows an approach in which the BN is treated as a "black box" to assess network risk based on an input attack graph. Bayesian networks should satisfactorily account for shared dependencies between graph nodes, but BNs utilize directed acyclic graphs as input. This reliance on acyclic graphs is a key limitation, since cycles are quite common in attack graphs. My approach does not rely on Bayesian networks. Instead, I harness the key concept of d-separation in probabilistic reasoning (an underpinning of BN inference reasoning) and design customized algorithms for reasoning over attack graphs data sets. My algorithms correctly handle both shared dependencies and graph cycles, and also take into consideration several instrinsic properties of vulnerability assessment in attack graphs (*e.g.* that no real-time evidences need to be considered, the monotonicity property, *etc.*). In this way, I am able to eliminate some unnecessary overhead and complexity that would be present in a generic Bayesian network inference engine.

Wang, *et al.* [54] recognizes the presence of cycles in an attack graph and attempt to account for the presence of the cycle to ensure that only unique, acyclic paths to any given node are considered. However, their probability calculation seems to assume that probabilities along multiple paths leading to a node within a cycle are independent, an assumption that cannot generally be made within attack graphs. Thus, although their work presents

useful techniques for propagating probability over cycles, this work as well appears to suffer from an inherent unsoundness. My work correctly assesses probability within an attack graph cycle while still properly considering extra-cyclic shared dependencies, producing a sound result in all instances.

Anwar, *et al.* [3] introduce an approach to quantitatively assessing security risks for critical infrastructures. The approach captures both static and dynamic properties in the network, contained in network and workflow models. However, the work did not provide a mathematical model to explain what the calculated metrics mean. It is unclear from the work if their approach could be generically applied to real-world problem sets. My risk metric has a clear semantics based entirely upon the likelihood an attacker can succeed in achieving a privilege or carrying out an attack. Furthermore, I provide a sound relationship between the input component metrics and the cumulative metrics produced by the algorithm.

Sawilla and Ou [47] design an extended Google PageRank algorithm and apply it to attack graphs to calculate numeric ranks for the graph nodes in terms of their importance for an attacker to achieve his supposed goals. Mehta *et al.* also apply Google PageRank on a different type of attack graphs [31]. Numeric values computed from PageRank-like algorithms only indicate *relative* ranks and cannot be used to quantify absolute security risks [46]. My work provides a sound and practical method for quantifying the absolute security risks in an enterprise network. Discrete, absolute values are useful in further analysis of the network configuration and in improvements thereof.

1.2.3 Network Reconfiguration

Narain addressed general configuration management issues with the application of SAT solving techniques [33]. This work presented a framework for dealing with various requirements for connectivity, security, performance, and fault tolerance. The full first-order logic is used to model the specific constraints arising from these requirements. A solution (model) to the first-order formula is then sought using the Alloy[1] model-finder, which subsequently converts

[1] http://alloy.mit.edu/

the problem to Boolean formulas and utilizes SAT solving techniques. My work is similar, but addresses another important requirement (i.e., robustness against potential attacks). I express the system constraints in a Boolean formula converted from an attack graph. It appears possible that the two approaches could be integrated into a unified framework to address a more complete set of requirements regarding enterprise network configuration management.

Wang, *et al.* developed a graph search-based method for network hardening [55], with previous work by Noel, *et al.* [38]. An attacker's goal is expressed as a propositional formula on the initial conditions through recursive substitution during graph traversal. The formula is then converted into disjunctive normal form (DNF) and is used to identify all possible security hardening options, from which set an optimal solution is chosen. The converted DNF formula can be exponential in the size of the attack graph, as admitted in the paper. My approach is distinct from their work. In my approach, I formulate the reconfiguration issue as a MinCostSAT problem for a Boolean formula with a size *linear* in the size of the attack graph. I then utilize a modern, well developed SAT solver [25] to identify an optimal solution. My approach should show more efficient performance. Not only will the input formula be smaller in size, but also modern SAT solvers have been shown to be quite successful in efficiently handling large data sets arising from practical problem.s

Dewri, *et al.* formulates the security hardening problem as a *multi-objective optimization* problem [10]. They employ a genetic algorithm (GA) to search for a solution, using the cost for security hardening measures and the cost for potential damage (from successful attacks) as two objectives in the multi-objective optimization problem. My approach differs in my adaption of the security hardening issue to a MinCostSAT problem, allowing the application of MinCostSAT solving to find a *provably* minimum-cost (optimal) solution for the given network model. Genetics algorithms, however, cannot guarantee convergence to a solution that is globally optimal. Furthermore, MinCostSAT is a specific optimization problem that has been extensively studied and so is likely to prove more efficient than a general

optimization algorithm, such as GA, when the problem can be tailored to fit the expected structure, as is the case here. Adopting the SAT solving approach also allows the user to query against the results in addition to simply identifying the optimal reconfiguration solution. The benefit of multi-objective optimization is that a user can be presented multiple "optimal" trade-offs between the two objectives. It would be interesting to study whether the MinCostSAT techniques presented here can be extended to provide multiple trade-off solutions for the optimization problem.

1.3 MulVAL

MulVAL is a security analysis tool for automatically recognizing security vulnerabilities and multistage attacks that can potentially lead to exploitation of network resources [39]. The MulVAL attack graph toolkit uses this gathered data for the generation of a logical dependency attack graph [40]. My research and corresponding implementation utilize the MulVAL analysis engine and attack graph toolkit, although it easily could be applied to other attack graph models with similar semantics.

MulVAL's reasoning engine is declaratively specified in Datalog [6], a syntactic subset of the Prolog logic programming language. In MulVAL, a network model is constructed based on a security scan of the network, in which configuration settings such as host machines, active services, inter-host reachability permissions, *etc.* are represented as Datalog tuples. Known vulnerabilities are extracted from a database (*e.g.,* the National Vulnerability Database) and also represented as Datalog tuples. The network model and vulnerabilities data are employed as input for the analysis engine. By applying a collection of Datalog rules representing various attack techniques, MulVAL can identify and show how multi-host, multistage attacks can potentially enable successful attacks into the enterprise network toward a specified privilege (assumed to be the attacker's goal). The Datalog implementation guarantees inherent soundness of results as well as an efficient $O(N^2)$ running-time [40].

By tracing the attack steps potentially leading to obtainment of the goal privilege, Mul-

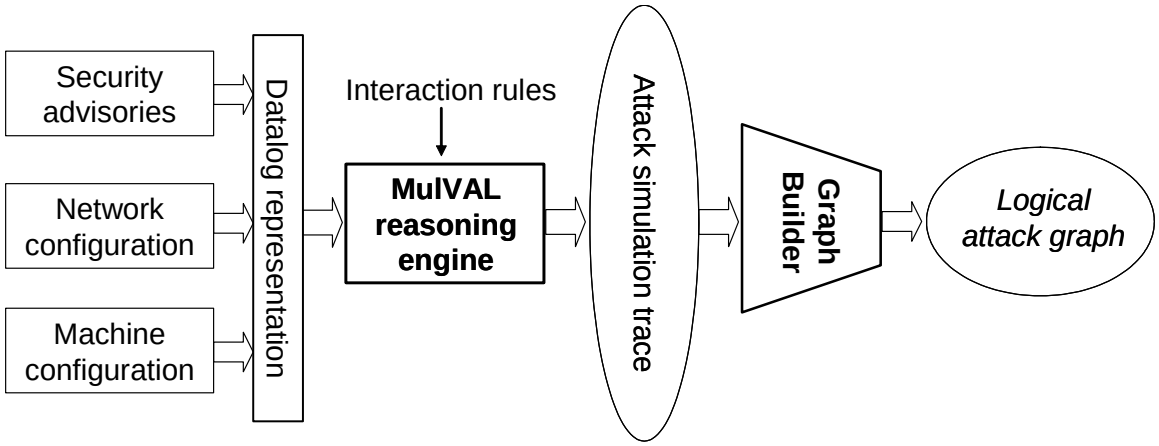


Figure 1.2: *MulVAL Logical Attack Graph Generation*

VAL creates a logical attack graph, essentially a proof graph of the logical derivation to the goal. The workflow for creating a MulVAL attack graph is shown in Figure 1.2.

The MulVAL logical attack graph can be defined $G = (G_D, G_C, G_F, G_E, G_L, G_G)$, where G_D, G_C, and G_F are three sets of disjoint nodes in the attack graph, so that the set of all graph nodes $G_N = G_D \cup G_C \cup G_F = G_N$; the set of directed arcs in the graph is $G_E \subset ((G_F \cup G_D) \times G_C) \cup (G_C \times G_D))$; G_L is a mapping from a node to its label; and, $G_G \in G_D$ is the attacker's goal [40].

G_C is the set of *attack-step nodes*, representing rules defining the means by which an attacker can gain privileges within the network. These nodes are conjunctive, or logical AND-nodes, meaning that for each node in G_C, in order for that node to be true, all of its immediate graphical predecessors (preconditions) must be true.

G_D is the set of *privilege nodes*, representing privileges within the network. These nodes are disjunctive, or logical OR-nodes, meaning that for each node in G_D, in order for that node to be true, at least one of its predecessors must be true.

G_F is the set of *configuration nodes*, representing facts about the current network configuration, such as inter-host reachability permissions, active software services, and other network settings. These nodes are source nodes in the graph, representing facts known to be true, and so do not have any predecessors.

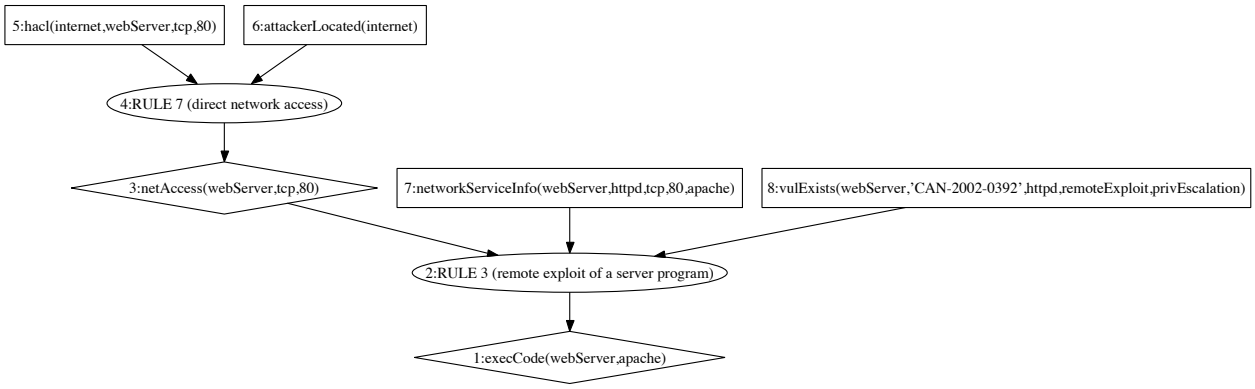


Figure 1.3: *MulVAL Logical Attack Graph*

G_E is the set of directed arcs (or edges) in the graph. Within the structure of a well-formed MulVAL attack graph, arcs indicate a causal relationship between nodes; an arc $a \rightarrow b$ indicates that a fully or partially enables b. For any node n, the set of immediate predecessors is $\{p \mid (p, n) \in G_E\}$. Attack-step nodes will have only privilege or configuration nodes as immediate predecessors; each of these predecessors represents a prerequisite condition necessary for the success of the represented attack step. Each attack-step node will have exactly one immediate successor, a privilege node. Privilege nodes will have only attack-step nodes as immediate predecessors; each of these predecessors represents an attack step whose execution gain an attacker the represented privilege. Each privilege node will have one or more immediate successors, except possibly the goal node G_G, which may not have any successors. Configuration nodes will never have predecessors, but each configuration node will have one or more immediate successors.

A sample attack graph is shown in Figure 1.3. In this graph, nodes 5-8 are configuration nodes, representing facts about the current system configuration; nodes 4 and 2 are attack-step nodes, showing how new privileges can be obtained by an attacker; nodes 3 and 1 are privilege nodes, representing network privileges obtainable by an attacker. For example, node 2 indicates that an attacker with network access to the web server (node 3), relying on this privilege and an existing vulnerability (node 8) on an active service on the web server (node 7), can obtain code-execution privileges on the web server (node 1).

An *attack path* is a minimal set of nodes and arcs that together show how a graph node can be reached. The set of all attack paths within the graph G is G_P and the set of paths leading to some node n is $G_P[n] \subseteq G_P$. An attack path is defined to be minimal, such that the same node cannot be reached by using only a proper subset of the nodes in the attack path. Please note that, because of the conjunctive attack-step nodes in the MulVAL attack graph, a more correct term would be "attack subgraph;" however, I will use the more widely recognized term "attack path." In Figure 1.3, because of the small size of the example, there is only one attack path showing how an attacker can gain privileges on the web server. This path contains all nodes in the attack graph.

1.4 Original Contributions

In this dissertation, I propose several approaches addressing various facets of enterprise network security management. Together, these approaches constitute a significant step toward automated management of enterprise network configuration with appropriate considerations of risk and security. I have utilized the MulVAL analysis engine and attack graph toolkit [40] in the development of this work, but my research could easily be applied to any attack graph model with comparable semantics. My contributions are the following:

1. I have proposed two approaches for reduction of attack graph data. The first approach identifies and removes any nodes and edges not contained in a straightforward path to the goal node(s). The second approach identifies and removes all provably unusable nodes and edges in the attack graph. These approaches are important because the balance between security and usability currently necessitates subjective judgment and a human needs to be involved in reaching the final decision. Without a clear understanding of the existing security problems, it is difficult for a human user to evaluate the proposed configuration changes and provide appropriate guidance.

2. Attack graphs illustrate the cumulative effect of attack steps, showing how individual steps can potentially enable an attacker to gain privileges deep within the network. The limitation of attack graphs, however, is the assumption that a vulnerability that exists can or will be exploited. In reality, there may be a wide range of probabilities that different attack steps could be profitably exploited by an attacker, dependent on the skill of the attacker and the difficulty of the exploit. Some published vulnerabilities, for example, have no known exploit or require specialized system knowledge unavailable outside of the enterprise network, and so these are unlikely to be used by an attacker. On the other hand, some vulnerabilities have widely known and easily available exploits, so that these vulnerabilities may be more likely to be utilized by an attacker. An attack graphs show what is possible without any indication of what is likely. I have defined and implemented an algorithm for a sound and effective quantification of risk in the enterprise network, employing individual component metrics (e.g., CVSS) and attack graph data. The values produced by this algorithm reflect the likelihood that privileges will be obtained by an attacker. Furthermore, I have provided formal proof of correctness for this algorithm.

3. Attack graphs are often of unmanageable size. Even a network of moderate size can have dozens of possible attack paths, overwhelming a human user with the amount of information presented. It is not easy for a human to determine from the information in the attack graph which configuration settings should be changed to best address the identified security problems. Without a clear understanding of the existing security problems, it is difficult for a human user to evaluate possible configuration changes and to verify that optimal changes are made. I have proposed an approach that transforms a network configuration problem into a SAT solving problem. Well-founded SAT solving techniques can then be applied to find an optimal reconfiguration solution. Where no satisfying solution exists, this approach utilizes the UnSAT core to present a human user with a minimal set of conflicting clauses, requiring that at least some of

these clauses be relaxed. Where at least one satisfying solution exists, this approach leverages the MinCostSAT [22] technique to find a mitigation solution that is optimal according to the various cost functions provided.

My overall contribution is the development of formal, logic-based approaches where attack graphs from security analysis are used to compute reconfiguration suggestions automatically, taking into account not only security requirements, but also requirements on usability and trade-offs between costs of security hardening, costs of possible damage due to successful attacks, and costs of loss in usability. Each of the pieces identified in the list above are fundamental to this development. The risk assessment valuations can be employed as the basis for cost values in the policies used. The SAT solving approach computes an optimal reconfiguration solution and presents this to the system administrator. The administrator consults trimmed attack graphs to gain an intuitive understanding of the problems present in the network and confirms that the reconfiguration suggested complies with all given policies. Where a suggested reconfiguration seems untenable, the cost policies can be modified to better reflect relative valuations of network resources. As these policies mature over time, they will more immediately provide desirable configurations.

These approaches contribute materially to the realization of a new vision for enterprise network security management, depicted in Figure 1.4.

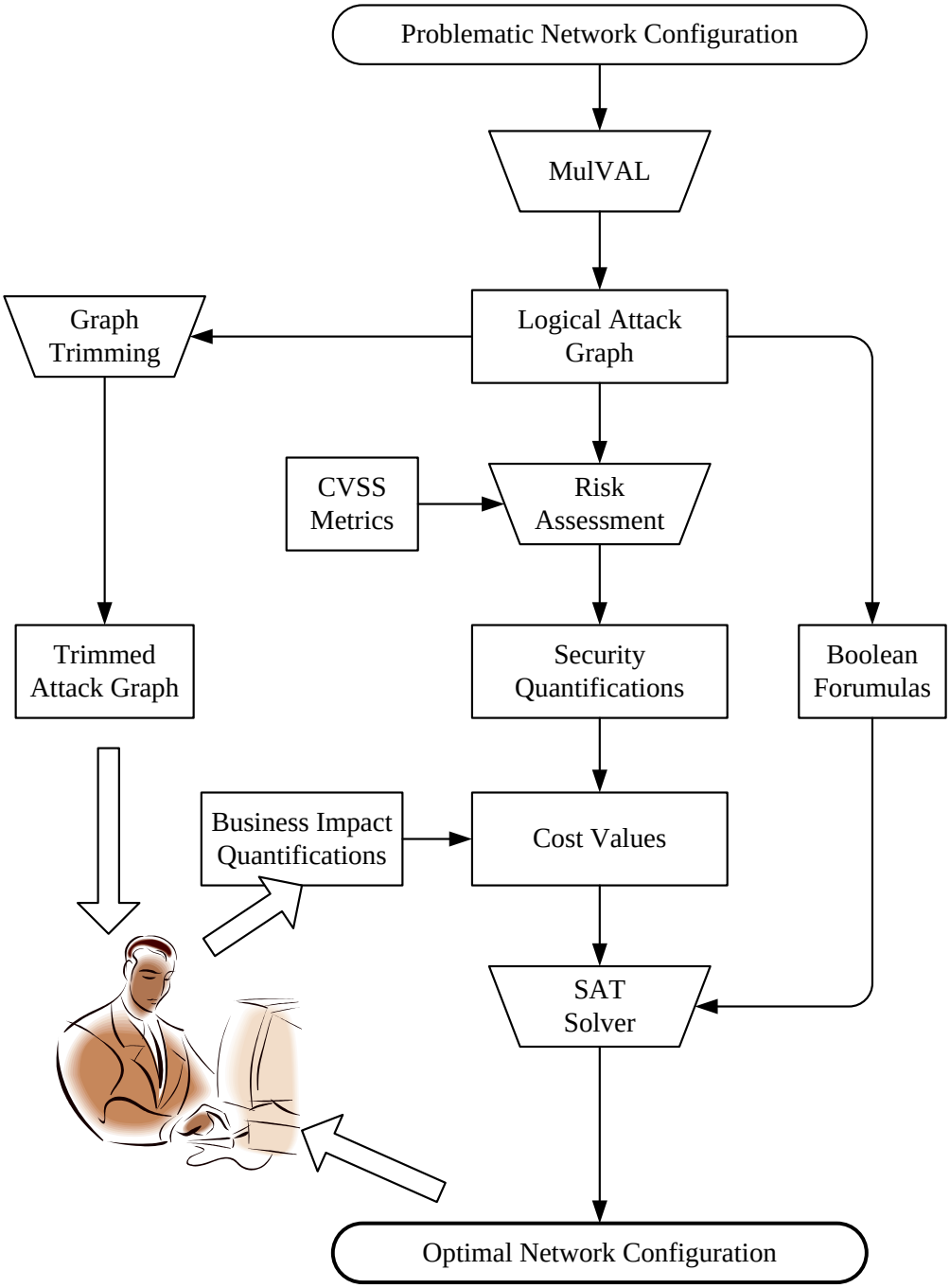


Figure 1.4: *Envisioned Process for Enterprise Network Security Management*

Chapter 2

Attack Graph Simplification

Attack graphs are generated from a representative model of an enterprise network system. The full attack graph includes all relevant data - configuration settings, attack steps potentially executed by an attacker, and privileges obtainable by an attacker. Many possible attack paths exist in the graph, each comprising a course of action an attacker might take to achieve the goal privilege. Because the attack graph contains all logically valid data, it is common to find strongly connected components (*i.e.,* cycles) within the graph. Cycles within the graph can be eliminated by removing one or more nodes within the cycle; however, some cycles should not be eliminated in this way, because important data within the attack graph would be lost. Considering this, we may assert that some nodes are less "useful" than other graph nodes, by various possible definitions of "usefulness" or "uselessness".

Consider the sample graph shown in Figure 2.1. In this simplified graph, the nodes will represent privileges and the edges will represent enabling attack steps allowing attainment of new privileges. An attacker beginning with privilege 1, then, can use this privilege to attain either or both of privileges 2 and 3; an attacker with privilege 3 can attain privileges 2 or 4, and so on. It is easily seen in this example that the transition from privilege $3 \rightarrow 2$ is not useful in any possible attack path leading to node 4. This attack step might then be identified as "useless" and removed from the graph without any real loss of information.

The immediately obvious solution for identifying these "useless" attack steps is to implement a breadth-first search algorithm to compute the distance of machine from the goal

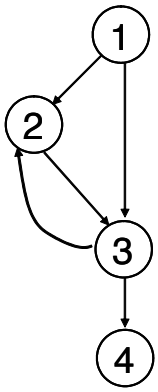


Figure 2.1: *Sample Graph - Useless Steps*

node, as measured by the minimum number of attack steps necessary to reach the goal. Node 2 in the above example would have a distance of 2 and node 3 would have a distance of 1. Attack steps that move from a node to another node with the same or greater distance (*e.g.* $3 \rightarrow 2$) could then be labelled "useless" and trimmed. While this approach would work in some cases, in many cases useful attack paths would be trimmed. For example, this algorithm would also trim the step $1 \rightarrow 2$, and thus lose the complete attack path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Actually, only attack paths with the shortest length will be preserved and all other paths will be trimmed. Such an approach might discard valuable information, especially in cases when an attacker might follow a longer attack path due to ease of exploit or better stealthiness along the longer path.

Another solution with initial appeal is to perform a simple depth-first search (DFS) and trim any back edges in the DFS tree. However, depending on the order of traversing a node's multiple children, either of the transitions $2 \rightarrow 3$ and $3 \rightarrow 2$ might be a back edge in the DFS tree. Only one of these transitions ($3 \rightarrow 2$) can be safely and correctly removed from the graph. Such an approach, then, will work no better than the breadth-first search.

In this chapter, I introduce two separate approaches to reducing the amount of data in an attack graph by recognizing and "trimming" (or removing) any nodes and edges that are "useless" by an associated definition. I will then compare the two approaches and discuss how and why each might be profitably employed.

2.1 An Example

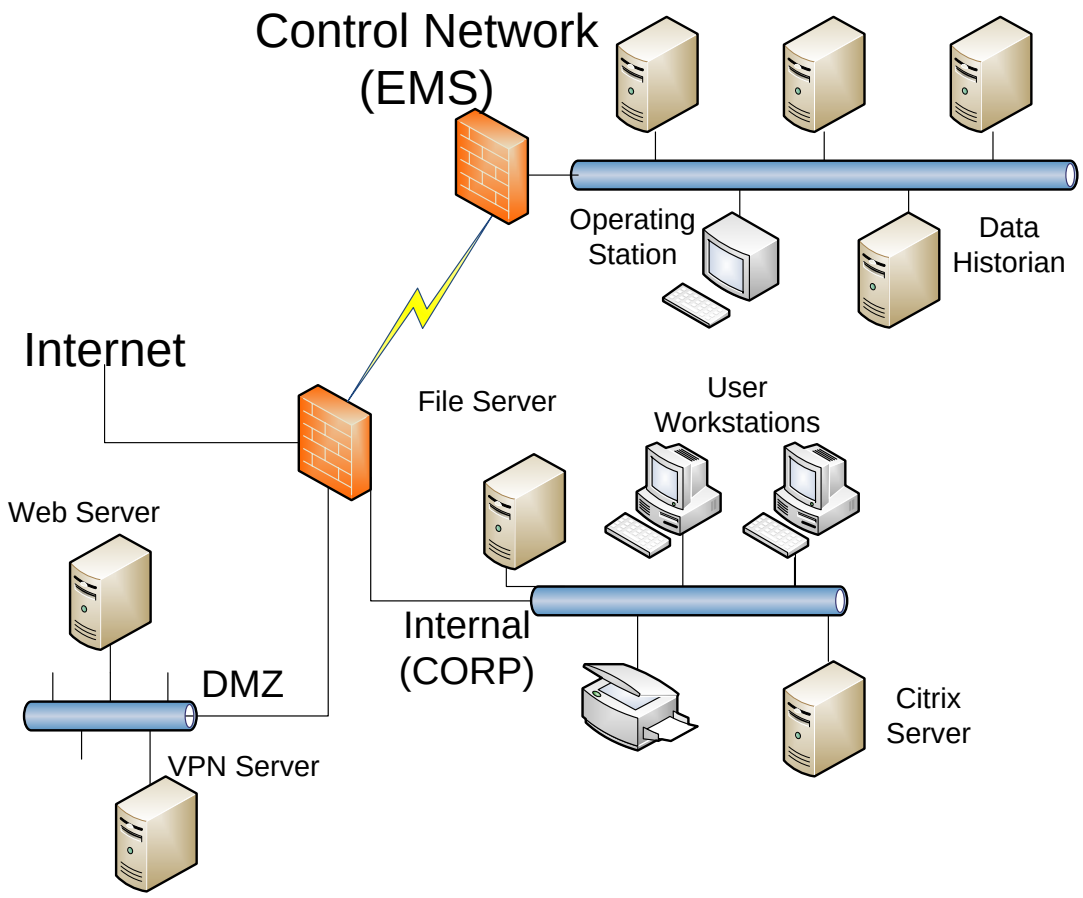


Figure 2.2: *Energy Management Network*

Figure 2.1 shows a very simple example of a cyclic graph. Generally, attack graphs are much larger and more complicated, necessitating a standardized and repeatable approach to attack graph simplification. Let us consider, then, the following example. Figure 2.2 shows a sample enterprise network model that is based on a real (and much larger) system. The specific problems encountered when applying attack graph techniques are the same for both the real and the adapted system.

Subnets: There are three subnets in the enterprise system: a DMZ (Demilitarized Zone), an internal subnet, and an EMS (Energy Management System) subnet, which is a control-

system network for power grids.

Hosts: The functionalities of most of the hosts are self-explanatory. One functionality of the EMS network is to gather real-time statistics, such as voltage, load, etc., from the physical power transmission and generation facilities. This information is the basis for generating various control signals to maintain appropriate operation of the power grid, and it is also useful for business transactions such as power brokerage between producers and consumers. The communication servers in the EMS subnet are responsible for communicating with the power grid physical infrastructures. The data historian is a database server that provides the power-grid statistics to be used for the various business/operation purposes.

Accessibility:

Src	Dst	Network access allowed
Internet	DMZ	all to web server all to VPN server
DMZ	internal	web server to file server VPN server to all
Internal	EMS	Citrix server to data historian

The table (above) shows network host accessibility, ignoring outbound access (such as from the Internal subnet to the Internet). Note that the Internet can only directly access the DMZ, and the Citrix server is the only host in the Internal subnet that can access the EMS network, and it can access only the data historian.

Threats: For this kind of system, security of the control system network is of paramount concern, since privileges on those machines could enable an attacker to assume control of physical infrastructures and drive them to a failed state (such as causing the turbine of a power generator to spin at a high speed until self-destruction). A number of mechanisms in this example are in place to reduce such threats as shown in the accessibility table.

Vulnerabilities: There are plenty of vulnerabilities in the system. The data historian runs a proprietary communication protocol with the communication servers that can be very easily compromised. All the machines in the system may have software vulnerabilities in

their various services and software applications. The users of the enterprise network may not be careful to protect their log-in credentials and could leak their user name and password to an attacker through various means, including social engineering. The VPN server in the DMZ can access every machine in the internal subnet. The web server in the DMZ can access the file server through the NFS file-sharing protocol. These vulnerabilities could affect the security of the enterprise network and the control system network.

The full MulVAL-generated attack graph for a model of this network is shown in Figure 2.3; the input model used to generate this attack graph is shown in Figure A.1. This attack graph contains 180 edges and 129 nodes, including 25 privilege nodes, 61 attack step nodes, and 43 configuration nodes. The size and complexity of this attack graph obviously makes it very difficult for a human user to comprehend all of the data displayed here.

2.2 Subnet-Based Trimming

This first approach to attack graph data reduction is intended to improve the usability of the attack graph for a human viewer (*e.g.,* a system administrator desiring to better understand existing security vulnerabilities in the network) [16]. Due to the size and complexity of an attack graph, often even for an enterprise network of moderate size, it can be difficult for a human user to quickly inspect and comprehend the attack paths present in the graph. This approach can provide a simplification of the original attack graph, highlighting critical attack steps and paths by removing nodes that provide data of less immediate importance. Reducing the graph in this manner can enable a human user to more quickly recognize the underlying, straightforward attack paths toward the goal privilege and the vulnerabilities enabling these paths.

Definition 1. *For subnet-based trimming, a "useless" attack step is defined to be any attack step that will not immediately provide an attacker with additional privilege that, together with already acquired privileges, is necessary for obtaining the goal privilege.*

Some attack steps, while valid from a logical point of view, may obscure the core security

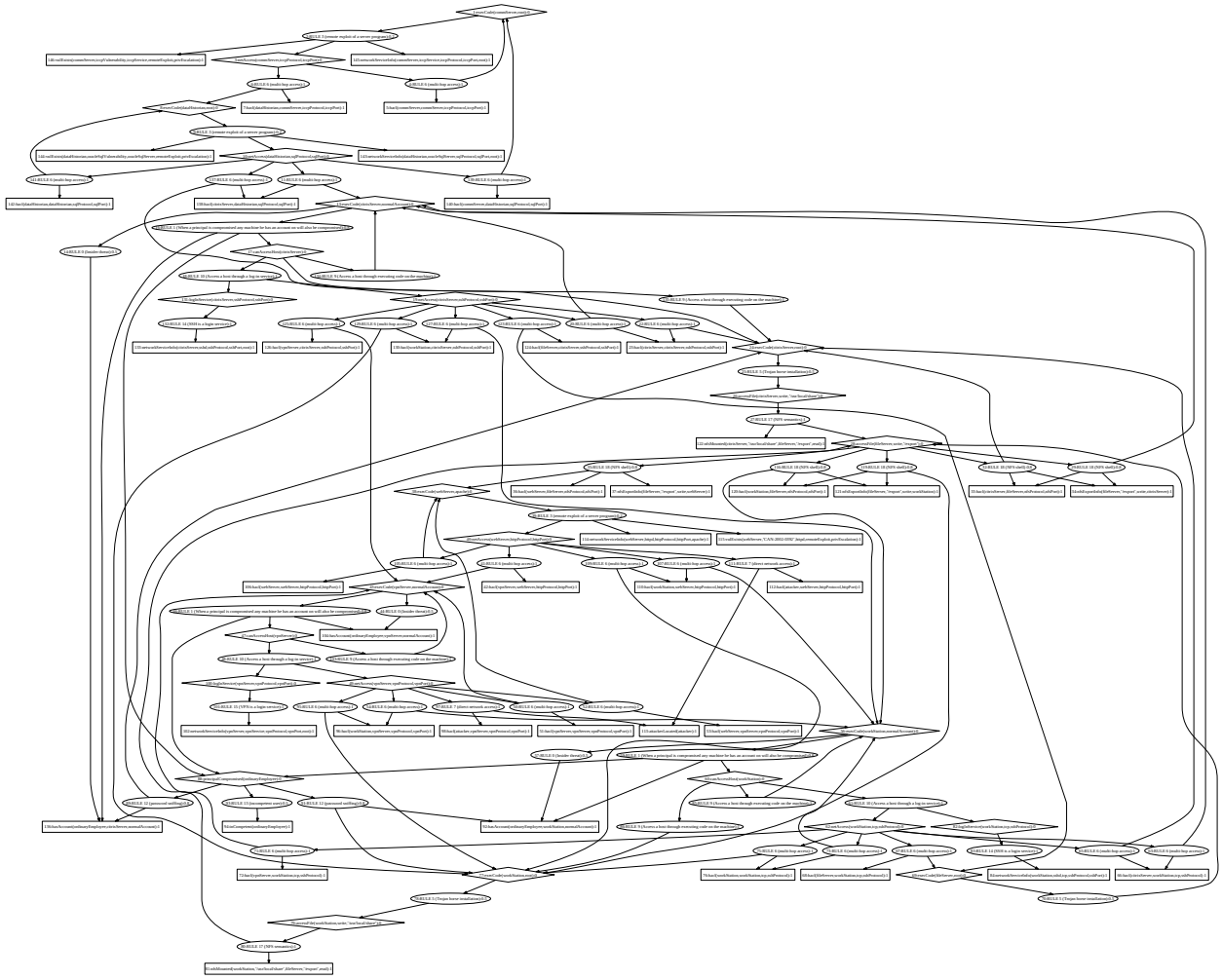


Figure 2.3: *Energy Management Network – Attack Graph*

issues in a given network configuration. While these steps may contain important information that would be necessary for a user desiring to identify and block every possible attack path, they can be distracting for a human user, drawing attention away from key problems. These "useless" attack steps share a common characteristic, in that they all show how an attacker could move through an enterprise network without penetrating "deeper" into the network (or closer to the goal privilege). In this approach, the classifications "useful" and "useless" are meant to reflect a prioritization of the data contained in the attack graph. It is important to emphasize that the so-called "useless" attack steps are valid and important to consider when determining an appropriate network reconfiguration for greater security.

However, when the user is first presented with the attack graph, understanding these paths is not crucial for understanding overall security threats.

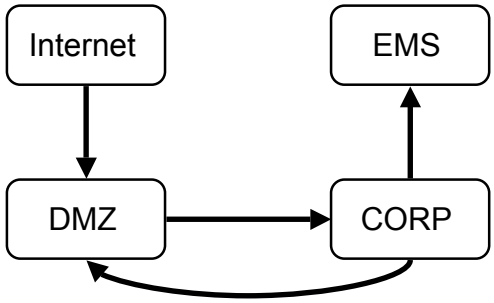


Figure 2.4: *Energy Management Network - Subnet Graph*

Subnet-based trimming is a two-level approach to identifying and removing "useless" attack steps, separately considering inter-subnet and intra-subnet attack graph edges. In this approach, I assume that within the data model of the enterprise network are identifiable subnets, sets of host machines with unlimited accessibility within the set.

The subnet-based trimming approach first requires the creation of a directed graph of the enterprise network in which the network subnets represented as nodes and the possible inter-subnet transitions as edges. In any case where some host machine in a subnet has reachability permissions to access some host machine in another subnet, an edge between the two subnets will be included in the graph. For example, Figure 2.4 shows the subnet graph derived from the Energy Management Network example depicted in Figure 2.2. Here, the citrix server (in the CORP subnet) has permission to access the data historian (in the EMS subnet), so an edge is added CORP $\rightarrow$ EMS, and so on.

From this subnet graph, we will then construct a dominator tree to recognize dominance and post-dominance relationships between subnets. For the dominator tree, the source node will be the assumed initial location of the attacker (e.g., the INTERNET subnet) and the sink node will be the subnet that contains the host machine on which the goal privilege resides (e.g., the EMS subnet).

Definition 2. *Let d, n be nodes in a directed graph. Node d* dominates *node n if every path from the source node to n goes through node d. This relationship is denoted "d* dom *n."*

Definition 3. *Let n, p be nodes in a directed graph. Node p* post-dominates *node n if every path from n to the sink node goes through p. This relationship is denoted "p* postdom *n."*

In the subnet graph of Figure 2.4, assuming that INTERNET is the source and EMS is the goal, some examples of dominance relationships are DMZ *dom* CORP and CORP *dom* EMS. We also have EMS *postdom* CORP and CORP *postdom* DMZ.

Considering any two subnets X and Y in the subnet graph, we will identify as "useless" all inter-subnet attack steps $X \rightarrow Y$ where Y *dom* X or X *postdom* Y. Intuitively, these are back edges in an attack path. If Y *dom* X, then an attacker who has gained privileges in subnet X must already have privileges in subnet Y or the attacker would not have been able to transition to X. In this case, moving "back" to Y will not enable an attacker to gain new privileges or opportunities. If X *postdom* Y, then moving from X to Y will not help the attacker either since he would have to return to X in order to reach his goal. Therefore, any transition between two hosts in different subnets that fits one or both of these cases is "useless" and will be trimmed from the attack graph. They are distracting for a human reader trying to comprehend other, more direct attack paths. In the attack graph shown in Figure 2.3, every transition from the workstation (in the CORP subnet) to hosts in the DMZ subnet will be trimmed, since DMZ dominates CORP. On the other hand, the transition from the citrix server to the data historian (CORP $\rightarrow$ EMS) will be retained. The attack graph in Figure 2.5 reflects inter-subnet trimming.

After applying the inter-subnet transition trimming, we then address intra-subnet transitions. An attack step between two machines $A \rightarrow B$ in the same subnet is retained in only two cases. First, if the subnet contains the goal machine, the transition is retained only if B is the goal. Any other transition within this subnet will be trimmed. In reality, an attacker might need to transition to other machines in the same subnet for the purpose of, *e.g.,* obtaining more computing power. However, these attack steps are not useful for a

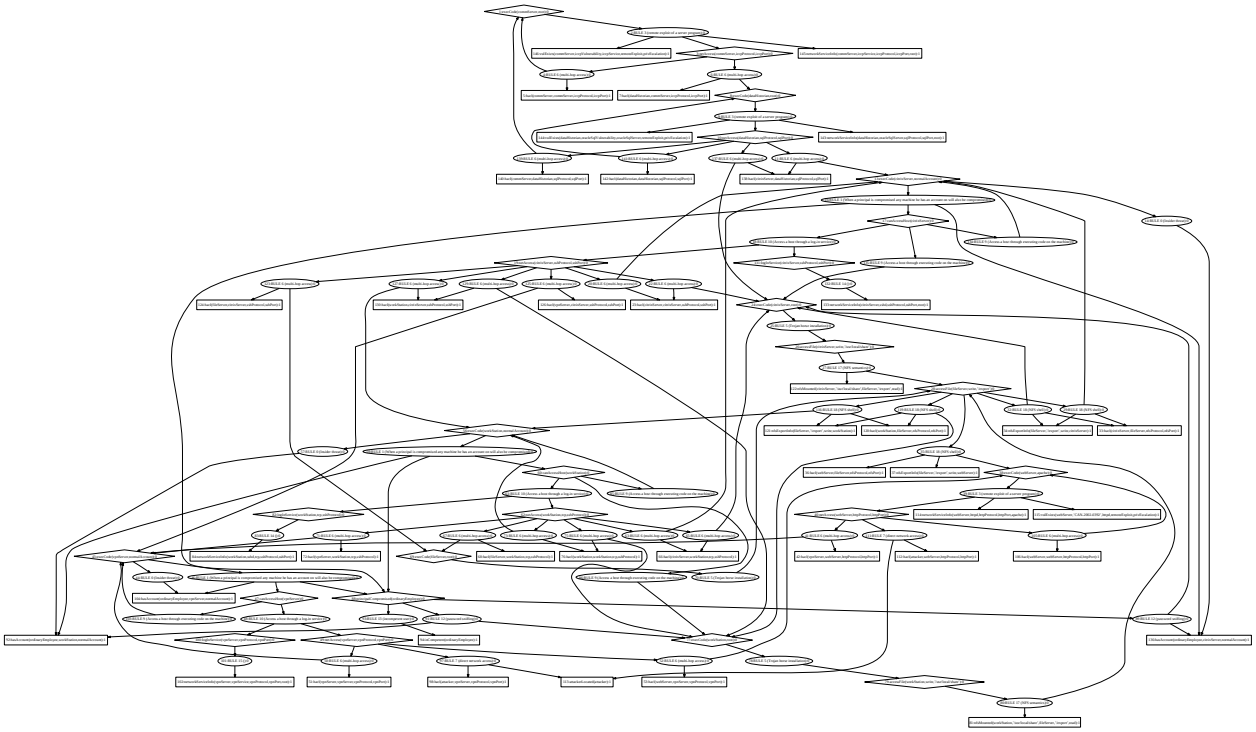


Figure 2.5: *Energy Management Network - Attack Graph (Inter-subnet Trimming)*

human to grasp the core security problem. Second, if the subnet does not contain the goal machine, the transition is useful only if B would provide an attacker with access to another subnet that would be deemed useful according to the subnet dominator tree, and even then only if that same access is not available from A. In the attack graph shown in Figure 2.3, the transition from the file server to the workstation is useless, since the workStation would not provide an attacker with new, useful access; however, the transition from the file server to the citrix server is useful, since, from the citrix server, an attacker could access the EMS subnet.

Figure 2.6 shows the resulting attack graph after both inter-subnet and intra-subnet trimming levels are applied to the example presented in Section 2.1. The trimmed attack graph now contains 110 arcs and 90 nodes, including 25 privilege nodes, 39 attack step nodes, and 26 configuration nodes. A total of 70 arcs and 39 nodes have been trimmed from the original attack graph, reducing the size by approximately 35 percent.

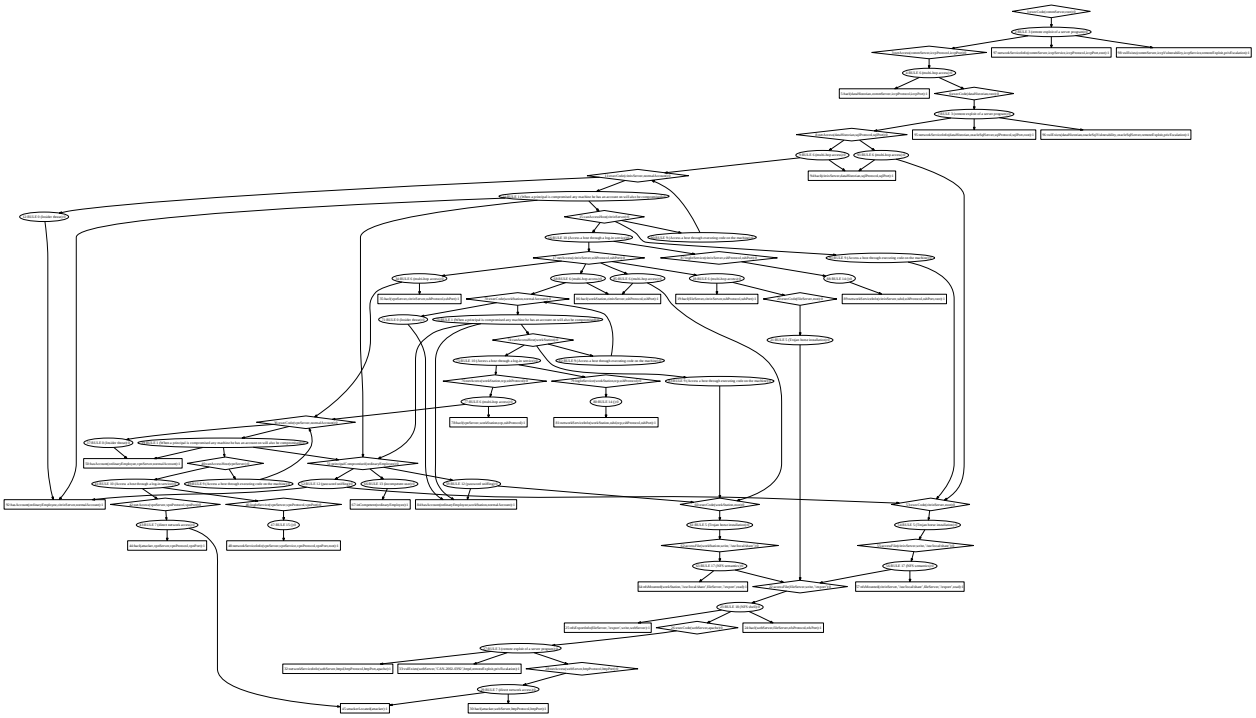


Figure 2.6: *Energy Management Network - Attack Graph (Subnet-based Trimming)*

Related works have introduced new visualization techniques for attack graphs [16, 37, 58]. Many of these techniques could be productively applied to the trimmed attack graph.

2.3 Dominator-Based Trimming

The second approach to attack graph data reduction, dominator-based trimming, employs graph-theoretic analysis techniques to identify provably "useless" nodes and edges within the attack graph. This approach is intended to reduce the amount of data within the attack graph while retaining all usable and productive attack steps. Because attack graphs include all possible attack steps, it is common to find cycles within the attack graph that lead only to privileges already gained, where some nodes within the cycle will need to appear in any attack path.

Definition 4. *For domination-based trimming, a "useless" node is defined to be any node within the attack graph that does not appear in an attack path leading to the goal node.*

There are some instances in which apparent cycles are actually useful, so it is not sufficient to simply attempt to break all cycles within the attack graph. However, by applying the concept of domination to the attack graph, we are able to identify and remove nodes and edges that are not part of any valid, acyclic attack path and thus are by this definition "useless" within the attack graph.

Definition 2, describing dominance in directed graphs, assumes as input a graph containing only OR-nodes. Since the MulVAL attack graph is an AND/OR-node graph, containing both OR-nodes and AND-nodes, it is necessary to define a dominance relationship between nodes within the attack graph and set forth an algorithm for identifying these relationships.

Definition 5. *Let d, n be nodes in a directed AND/OR-node graph. Node d* logically dominates *node n if every path from the source node to node n includes node d. This relationship is denoted $d \Leftarrow n$.*

Definition 6. *Let n, p be nodes in a directed AND/OR-node graph. Node p* logically post-dominates *node n if every path from n to the sink node includes node p. This relationship is denoted $n \Rightarrow p$.*

The definitions for logical dominance and post-dominance in the attack graph capture the same intuition as the more standard definition for dominance in an OR-node graph. If $d \Leftarrow n$ (node d logically dominates node n), then any attack path leading to node n must include node d. If $n \Rightarrow p$ (node p logically post-dominates node n), then any attack path leading from node n to the goal node must include node p.

The privilege nodes within the attack graph are OR-nodes, while the attack step nodes are AND-nodes. For this trimming approach, it is sufficient to consider domination strictly between OR-nodes, since the AND-nodes simply set forth the conditions allowing for transition between OR-nodes. It is helpful, then, to transform the attack graph into an OR-graph representation, to ease the computation of logical dominators in the graph, and then map these relationships back onto the full attack graph. Because of the well-founded structure of the attack graph, it is relatively straightforward to create this representation.

Algorithm 2.1 Pseudocode for constructing a representative OR-graph

```
1: Remove from the attack graph all configuration nodes and all arcs from these nodes
2: Add a virtual OR-node nil as a source node for the attack graph
3: for all attack steps e do
4:    P ← predecessor set ( privilege nodes with arcs leading to e )
5:    s ← privilege node gained by attack step e
6:    for all p ∈ P do
7:       Remove graph arc p → e
8:    end for
9:    Remove graph arc e → s
10:   if P = ∅ then { e has zero predecessors }
11:      Add new graph arc nil → s
12:   else if P = {p} then { e has exactly one predecessor }
13:      Add new graph arc p → s
14:   else { e has multiple predecessors }
15:      Create new graph node m, representing merging of all p ∈ P
16:      for all p ∈ P do
17:         Add new graph arc p → m
18:      end for
19:      Add new graph arc m → s
20:   end if
21: end for
```

Algorithm 2.1 sets forth a pseudocode representation for the construction of the OR-graph representation for a MulVAL attack graph. In this representation, the newly-created *nil* node will serve as the source node for the graph and the goal privilege node will continue to serve as the sink or exit node for the graph. All configuration and attack step nodes are removed from the graph. The causal relationships represented by arcs leading to and from attack step nodes are retained by creating new graph arcs directly between privilege nodes. This transformation is straightforward, except where an AND-node has multiple predecessors. In such cases, the algorithm will create a "merged" node, representing the possession of all of the predecessor OR-nodes; arcs will be added connecting each individual predecessor to this new merged node and connecting the merged node to the resulting privilege node. By this algorithm, then, the original AND/OR-node attack graph can be simulated by an OR-node graph.

The simulated OR-graph is not sufficient to capture the full depth of data included in the original graph, but will prove useful for identifying some properties, such as domination, that correlate to the data in the original graph. Graphical dominators and post-dominators in the simulated OR-graph can be identified according to Definitions 2 and 3. Logical dominators and post-dominators (specified in Definitions 5 and 6) in the full attack graph can be determined by mapping the simulated graph with its dominators back to the full attack graph. For individual graph nodes d, where d *dom* n in the simulated OR-graph, it will be true that $d \Leftarrow n$ in the attack graph. For merged graph nodes M, where m *dom* n in the simulated OR-graph, it will be true that for all $m \in M$, $M \Leftarrow n$ in the attack graph. Logical post-dominators in the attack graph can be similarly established based on graphical post-dominators in the simulated graph.

Figure 2.7 shows, on the left, an attack graph with goal G from which the configuration nodes have already been removed, leaving only the OR-nodes and AND-nodes, represented by the diamond and elliptical nodes, respectively. Figure 2.7 shows, on the right, the OR-graph representation of the same attack graph, after the application of Algorithm 2.1.

In Figure 2.7, the simplified graph representation has a new source node, *nil*, as well as appropriate connecting arcs to nodes A and B. All AND-nodes have been removed from the graph, and the associated relationships between their predecessors and successors have been retained by the addition of new graph arcs. In the original graph, for example, privilege E is reachable by attack step 8, which requires privilege C; in the modified graph, then, an arc directly connects $C \rightarrow E$. The new "merged" OR-node A, B represents an attacker's possession of both privileges A and B, necessary for acquiring privilege C.

Now that the attack graph is represented entirely by OR-nodes, the graphical dominator and post-dominator trees can be easily computed by applying a standard algorithm. Algorithm 2.2 will convert the graphical dominator tree T into a logical dominator graph G. Figures 2.8 displays the graphical dominator tree and logical domination graph. Logical post-dominance would be similarly established.

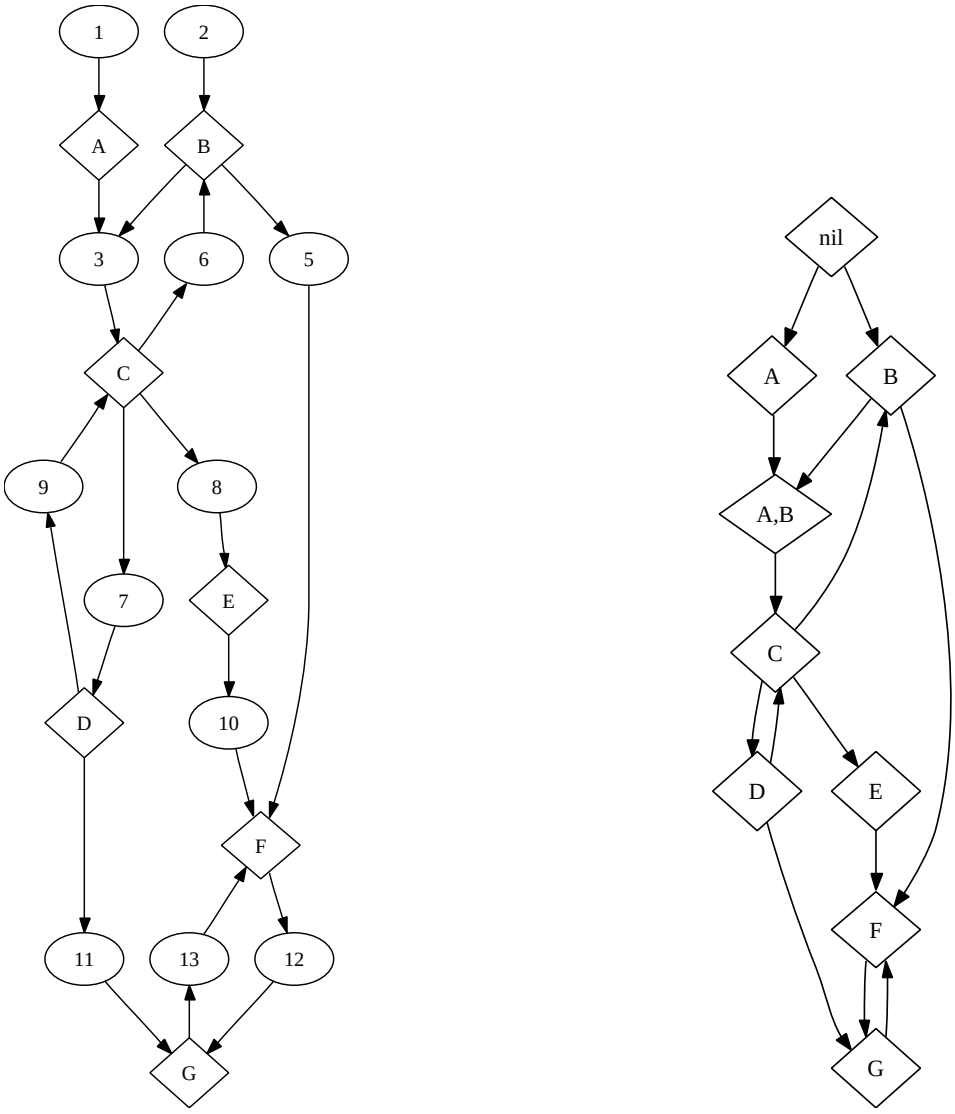


Figure 2.7: *Sample attack graph* (left); *simplified representation of same* (right)

Having identified logical dominance and post-dominance relationships within the graph, this data can be used to reduce the attack graph size by removing "useless" nodes and edges. In Figure 2.7, for example, attack step 9 indicates that an attacker with privilege D can obtain privilege C. Assuming the monotonicity property, however, the arcs $D \rightarrow 9$ and $9 \rightarrow C$ are useless, since an attacker with privilege D must already possess privilege C.

Employing knowledge of the transitive logical domination and post-domination relationships, "useless" nodes and arcs are easily identified within the attack graph. An attack step

Algorithm 2.2 Pseudocode for identifying logical dominance relationships

```
1: Construct graphical dominator tree T
2: Initialize empty logical dominator graph G
3: for all nodes d ∈ T do
4:    N ← {n | ∃ arc d → n in T}
5:    if d is an individual node then
6:       Create node d in G, if not already present
7:       for all n ∈ N do
8:          if n is an individual node then
9:             Create n in G, if not already present
10:            Add arc d → n in graph G
11:         end if
12:      end for
13:   else { d is a merged node }
14:      D ← individual nodes comprising merged node d
15:      for all d ∈ D do
16:         Create d in G, if not already present
17:         for all n ∈ N do
18:            if n is an individual node then
19:               Create n in G, if not already present
20:               Add arc d → n in graph G
21:            end if
22:         end for
23:      end for
24:   end if
25: end for
```

node E is "useless" and can be safely removed from the attack graph if it has a predecessor node P and a successor node S, where $S \Leftarrow P$ or $S \Rightarrow P$. In the case that S logically dominates P ($S \Leftarrow P$), privilege S must necessarily have been obtained already in any attack path leading to node P; thus, an attack step that enables an attacker with privilege P to obtain S must be "useless." Similarly, in the case that P logically post-dominates S ($S \Rightarrow P$), any acyclic attack path leading from node S to the goal node must again pass through node P; thus, an attack step that enables an attacker with privilege P to obtain S must be "useless." Once "useless" attack steps have been identified and removed from the attack graph, it is a straightforward matter to clean up the graph by removing arcs and nodes made useless by the previous removals.

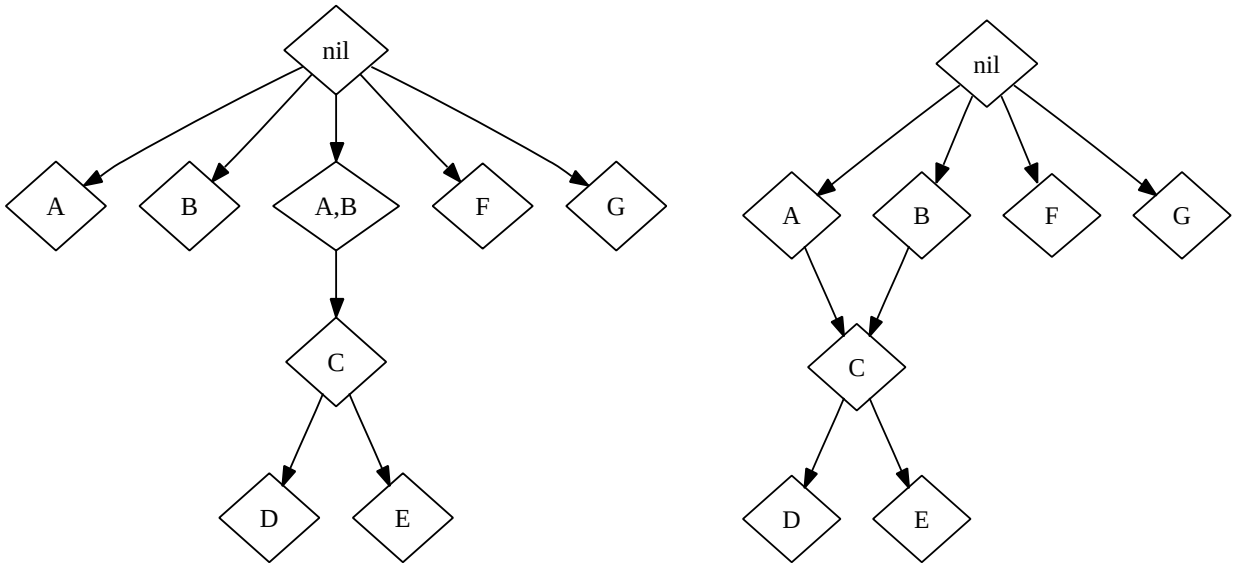


Figure 2.8: *Graphical dominator tree* (left); *logical dominator graph* (right)

Figure 2.9 shows the resulting attack graph after domination-based trimming has been applied to the example presented in Section 2.1. The trimmed attack graph now contains 145 arcs and 112 nodes, including 25 privilege nodes, 49 attack step nodes, and 38 configuration nodes. A total of 35 arcs and 17 nodes have been trimmed from the original attack graph, reducing the size by approximately 17 percent. Typically, this reduction will be less than the subnet-based trimming approach; the value of this trimming approach, however, is that it retains all provably valid acyclic attack paths.

2.4 Discussion

I have presented in this chapter two approaches to reducing the amount of data included in an attack graph. Each of the two approaches presents both advantages and disadvantages.

The first approach, subnet-based trimming, will often remove a significant number of nodes and edges from the attack graph. This serves to greatly reduce the size and complexity of the attack graph, so that the trimmed graph should be much easier for a human user to read and comprehend. However, this trimming approach can remove nodes and edges that are necessary for a complete understanding of network weaknesses and all possible attack

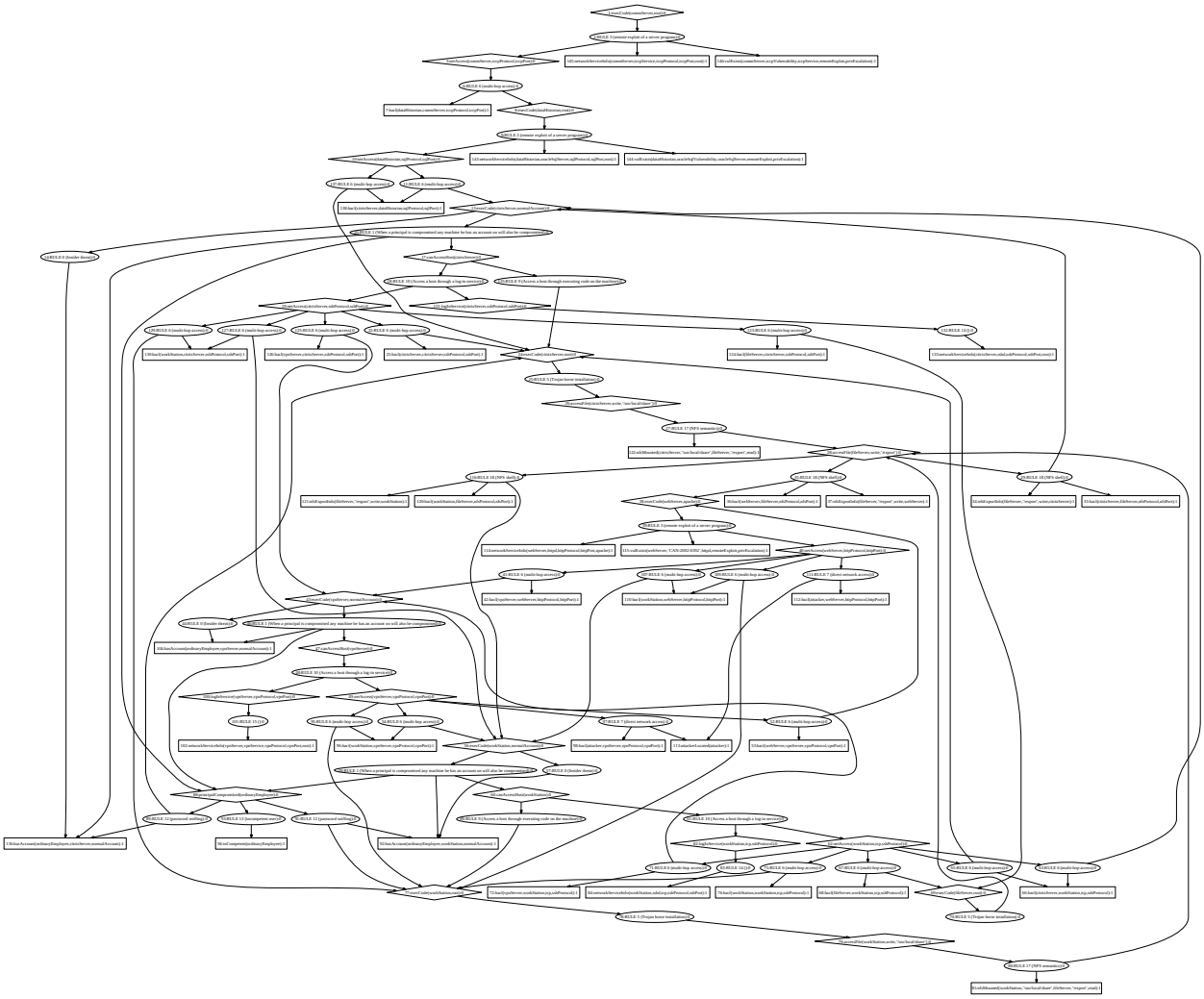


Figure 2.9: *Energy Management Network - Attack Graph (Domination-based Trimming)*

paths leading to the goal privilege. In the trimmed graph, only nodes lying on attack paths moving consistently forward toward the goal will be retained.

The second approach, domination-based trimming, typically removes a lesser number of nodes and edges from the attack graph. This will have only a small effect on the size and complexity of the graph, serving mainly to remove redundant or unnecessary data. This approach, however, is inherently sound. Trimming in this way will never remove any node or edge that lies on some potential, non-cyclic attack path. Thus, the attack graph data is provably reduced without loss of usable data.

I believe that each of these two trimming approaches serves an important role in enterprise network security management. Subnet-based trimming provides a simplified attack graph useful for presentation to a human user for quicker comprehension and better understanding. This trimming approach retains straightforward attack paths, highlighting key problems in network security. Domination-based trimming does not provide much advantage for a human user. It will, however, have some effect in reducing the complexity of risk assessment (Chapter 3) by removing nodes that will never provide any gain to an attacker. This provable reduction will not compromise risk assessment over the attack graph and so is useful in that regard.

Chapter 3

Quantitative Risk Assessment

The previous chapter addressed techniques for simplifying and reducing the size of the attack graph by "trimming" nodes and edges that are, by some definition, "useless." Once these techniques have been applied, however, there remains a fundamental question of how to best use the remaining data that has been retained to evaluate the current state of security in the network and address how best to improve upon that state. Currently, the evaluation and management of security risks in an enterprise network is more an art than a science. System administrators operate by instinct and learned experience, often without any objective evaluation of their decisions or a full knowledge of all ramifications of any changes. There is currently no sound and quantitative measurement of risk within an enterprise network system, no independent standard by which one configuration can be measurably compared to another. Lacking such measurement, there is no reliable way to answer pertinent questions such as, "Where is our network most vulnerable?" and "If I change this configuration setting, will our network be more or less secure as a result?" These questions require a quantitative model of security with clear and reliable measurement of risk.

In this chapter, I will introduce a sound and quantitative model for the measurement of risk in an enterprise network system. My approach is meant to quantify the inherent risk within an enterprise network system, considering only the underlying configuration of the network and *not* any type of real-time observations of system activity. This approach will

utilize the attack graph data set produced by MulVAL as well as input component metrics that provide discrete assessments of the probability a given attack step will succeed when the all necessary preconditions are met.

Developing a sound metric model based on attack graphs is not a trivial task. Enterprise networks are typically laid out in such a way as to include a great deal of interconnectedness between network hosts. Thus, multiple attack paths leading to a given network privilege are rarely independent. Furthermore, this interconnectedness will often lead to the appearance of cycles in an attack graph, but an accurate assessment of the probability that some privilege could be gained by an attacker should obviously not include a theoretical attack path in which an attacker can reach that privilege after having already gained it. Assuming monotonicity in the acquisition of network privileges [2], this attack path should be excluded from the calculations. I propose to treat both cyclic and acyclic attack graphs with proper management of shared dependencies. Specifically, assuming the input component metrics are correct, my model will always produce the correct probability assessment that an attacker can succeed in achieving a given privilege.

3.1 Component Metrics

Component metrics are essentially individual risk measurements, measuring the likelihood that a single attack step is successfully perpetrated within a network when all necessary preconditions are met. These component metrics consider specific qualities of vulnerabilities, such as the skill necessary to exploit the weakness, and this information can be used to assign a probability that the vulnerability will be exploited successfully. Every attack step can be assigned some discrete probability of success, ranging from zero to one. The likelihood of a successful attack will depend on the nature of the vulnerability being exploited as well as the skill and knowledge of the attacker.

Let us consider two quick scenarios as examples. In the first scenario, a vulnerability existing on an active service within the enterprise network is a widely published, easily

applied exploit. An attack step based on this vulnerability would have a high probability of success. In the second scenario, a vulnerability might exist in a technical sense but have no known attacks that can successfully exploit it, or perhaps successful attacks require detailed knowledge of inner system variables or other hidden data. An attack step based on this vulnerability would have a correspondingly lower probability of success.

Recently, there has been significant progress in standardizing and developing component metrics based on software vulnerabilities, such as the Common Vulnerability Scoring System (CVSS) [32, 48]. CVSS is a jointly developed, open industry standard for the quantification of security risks for software vulnerabilities. The CVSS database is maintained through the assistance and input provided by software vendors, security experts, and application users.

For each vulnerability, CVSS provides assessment for three distinct aspects: the Base, Temporal, and Environmental metric groups.

- The Base metric measures intrinsic and unchanging characteristics of a vulnerability, considering remote exploitability, complexity of a successful exploit, whether authentication is required, whether confidential data might be exposed, whether system availability is impacted, *etc.*

- The Temporal metric measures vulnerability characteristics that may change over time, considering the availability of a working exploit, the existence of a software patch or upgrade to address the issue, *etc.*

- The Environmental metric measures characteristics of the vulnerability that depend on a specific environment or implementation, considering the potential for collateral damage, what proportion of a system would be affected by an exploit, *etc.*

For network risk assessment, we will utilize the CVSS score derived from the Base metrics. The CVSS-defined score is represented as real number in the range [0,10]. There are several ways to transform this value into a percentage representing the likelihood of success for that exploit. For testing purposes, each vulnerability can be placed in one of three possible

classifications: low, medium, and high. For example, a low-risk vulnerability might have a score in the range [0,3.5], a medium-risk vulnerability might have a score in the range [3.5,7], and a high-risk vulnerability might have a score in the range [7,10].

One common criticism of security metrics addresses the imprecision in risk measurements for network vulnerabilities. Although work continues to be done in refining these metrics, they are, at best, rough estimates of what is likely. Their value lies in the comparative interrelations of different vulnerabilities, knowing that a "high" probability vulnerability is more likely to be exploited than a "medium" or "low" probability vulnerability. While it is true that the component metrics, and as a result the computed cumulative metrics, are inevitably imprecise, we contend that it is necessary to make initial steps toward a comprehensive risk assessment, rather than remaining constrained to precisely measurable properties. Utilized in a sound approach, the input measurements and the metric model can be refined with practice and experimentation, improving on the usefulness and applicability of the approach as a whole. Further discussion on the ongoing refinement of component metrics is included in Section 5.1.

The key limitation of CVSS is that the computed scores represent only the likelihood of success for individual attack steps, without consideration of the probability of actions performed to attain the preconditions necessary for an attack step. It is easy to see in an attack graph how an attacker might perform multi-step attacks to penetrate deeper into a network than may seem immediately possible. The probability of success for a multi-step attack is actually an aggregate calculation over the probabilities for each individual step in the path. Because each score reflects only the probability of a single step, CVSS (or any comparable component metric system) is insufficient in itself to fully quantify risk in an enterprise network. I will utilize the CVSS scores for individual attack steps, but will establish my own sound basis for an extended calculation over these values.

3.2 Sound Risk Assessment - A First Approach

I believe that a sound approach is needed to fully address the need for quantitative risk assessment over an enterprise network. My first attempts toward this end focused on the use of Bayesian networks (BN), a powerful and well-studied graphical model utilizing directed graphs with associated probabilities.

Definition 7. *A Bayesian network* [19] *consists of the following:*

- *A set of variables and a set of directed edges between variables.*
- *Each variable has a finite set of mutually exclusive states.*
- *The variables together with the directed edges form an acyclic directed graph*
- *To each variable A with predecessors $B_1, \ldots, B_n$, a conditional probability table $P[A|B_1, \ldots, B_n]$ is attached.*

Bayesian networks provide a great deal of probabilistic reasoning power, allowing inference and learning over provided evidences. Bayesian networks incorporate causality in the underlying graph structure, with individual metrics applied to graph nodes. Many practical and well-tested implementations of BNs are publicly available. Using BNs could provide a "black box" approach, in which an attack graph would be used as input and the probability calculations would be handled completely by the BN.

Although Bayesian networks are certainly capable for this task, there are some concerns about the suitability of such an approach. Much of the innate power of Bayesian networks is not needed for our purposes, since the attack graph provides a static view of the network configuration and will not add any new evidences over time. Bayesian networks require a directed, acyclic graph to be used as the underlying graph structure; this is the most substantial difficulty with utilizing BNs for quantifying risk over an attack graph, which often contains one or more cycles between graph nodes.

It is possible, of course, to take a possibly cyclic directed attack graph and represent that data as a directed acyclic graph. Essentially, unfolding the attack graph involves a backward-progression through the graph, creating new, unique instances of each attack step and privilege node visited and tracing which nodes have already been visited to ensure that no cycles are included. Configuration nodes, of course, do not create cycles in the graph and so will not need unique instances.

Algorithm 3.1 shows the general procedure for unfolding an attack graph.

Algorithm 3.1 Pseudocode for full graph unfolding

```
1: Initialize set of attack graph nodes N with all configuration nodes from original attack
   graph
2: Initialize empty set of attack graph arcs E
3: Create node g, representing attacker's goal
4: Add g to set N
5: Seed Queue with (g, emptySet)
6: while Queue is not empty do
7:    (node, path) ← pop Queue {node is a privilege node; path is the set of nodes in path
      to g}
8:    for all attack steps e leading to node do
9:       s ← set of preconditions for e
10:      if path ∩ s = ∅ then
11:         Create a new instance e′ of attack step e
12:         Add e′ to node set N
13:         Add new arc connecting e′ → node to arc set E
14:         for all n ∈ s do
15:            if n is a privilege node then
16:               Create a new instance n′ of privilege node n
17:               Add n′ to node set N
18:               Add new arc connecting n′ → e′ to arc set E
19:               Push (n′, path ∪ {node}) to Queue
20:            else { n is a configuration node }
21:               Add new arc connecting n → e′ to arc set E
22:            end if
23:         end for
24:      end if
25:   end for
26: end while
```

The algorithm begins by seeding a Queue with a tuple containing the goal node g and an empty node set. The algorithm will iterate until the Queue is found to be empty. In each iteration, a tuple $(node, path)$ will be popped from the front of the Queue; *node* is a unique instance of a privilege node within the attack graph and *path* is the set of privilege nodes visited along a unique path from this node instance to the goal node. The algorithm will consider all attack-steps e that would result in an acquisition of privilege n. For an attack-step e with predecessor s, where no privilege in *path* appears in s, a new instance of e is created and added to the unfolded graph with an arc connecting it to *node*. New instances are also created of its predecessor nodes and appropriate arcs are added to the unfolded graph. All privilege nodes in set s are added to the Queue with an updated path, to be considered in turn. The algorithm will continue to iterate until all paths have reached back to the graph source, where no prior privileges must be held.

This algorithm will create a directed acyclic graph equivalent to the original (possibly cyclic) attack graph in that all unique, acyclic attack paths in the original attack graph will be retained in the new, unfolded attack graph. However, depending on the number and size of cycles in the original graph, the unfolded graph may have many more nodes and edges than the original. For the sample network shown in Section 2.1, the attack graph contained 86 non-configuration nodes and 180 arcs; unfolding this attack graph produces a new, acyclic graph with 1,720 nodes and 2,598 arcs. The size of the unfolded graph is dependent upon the interconnectedness of the network, so that a more strongly connected network will produce a larger unfolded attack graph. If an attack graph is already acyclic, then the unfolded graph will contain the same number of nodes and arcs. In worst case, however, the size of the unfolded graph could be exponential in the size of the original graph. For example, a network consisting of eight fully connected machines produces an attack graph of 94 non-configuration nodes and 237 arcs; the unfolded attack graph contains 66,541 nodes and 119,381 arcs. This unfolding clearly constitutes an unmanageable increase in the size and complexity of the attack graph.

One possible optimization of this algorithm is to first identify the strongly connected components (SCC) in the graph and then unfold only the SCCs. By unfolding only the cyclic subgraphs and then connecting them appropriately with the rest of the (acyclic) graph, we can keep the unfolded graph smaller than by directly applying the unfolding algorithm to the whole graph. Unfortunately, this optimization, called a "partial unfolding," does not resolve the inherent scalability issue. The increase in the size of the attack graph comes largely from within the cycles, so that a very strongly-connected attack graph will still show a large increase in the number of nodes, even with only a partial unfolding. Many attack graphs include a small number of large cycles, in which case this optimization would not provide much reduction in the size of the unfolded graph.

Although Bayesian networks initially seem well suited for implementing risk assessment over an attack graph, there are strong objections to this type of approach. In order to utilize the power of BNs, the attack graph must be unfolded to an acyclic representation and the unfolded attack graph will often be prohibitively large. It seems necessary, then, to design an approach that utilizes the inherent capability of probabilistic reasoning without being constrained to acyclic graphs. Fortunately, the well-founded structure of attack graphs makes this a viable option, as I will show in the rest of this chapter.

3.3 Sound and Practical Risk Assessment

My first attempted approach toward quantitative risk assessment over an attack graph, presented in Section 3.2, was based on Bayesian networks, but required an unfolding of a cyclic attack graph; this approach is sound but proves impractical, because of the dramatic increase in the size of an unfolded attack graph. I will present in this section a different approach, employing many of the same underlying concepts of probabilistic reasoning, that will be both sound and practical. This approach relies upon the well-formed structure of the MulVAL attack graph (Section 1.3) and automatically computes the absolute risk value for each obtainable privilege within a network.

3.3.1 Definitions

To present my approach, I will first define several further graph properties and values, as well as several helpful functions. This approach will use a MulVAL attack graph with the configuration nodes removed, because these nodes will not affect the current probability of a privilege being obtained by an attacker. Once the configuration nodes are removed, a new privilege node will be added to serve as the root node in the graph; the new root of the graph, $G_R \in G_D$, represents a lack of network privileges, an attacker's assumed starting point. The set of all nodes with multiple successor nodes, hereafter referred to as "branch nodes," is $G_B \subset G_D$.

Definition 8. *Conditional probability is the probability of some variable B, given the truth of another variable A; this is expressed* $Pr[B|A]$.

G_M is the set of relevant component metric values (*e.g.*, based on a CVSS score); each attack-step node $c \in G_C$ will have an assigned metric value, $G_M[c]$, representing the conditional probability that the single attack step will succeed when all of the prerequisite conditions are met. Thus, for some attack-step node e with predecessor set p, $G_M[c] = Pr[e|p]$. Additionally, the attack graph will have an assumed prior risk value, G_V, representing the probability of an attack being attempted against the network; for testing, we assume $G_V = 1$.

I will utilize the notion of *d-separation* within a causal network (such as a Bayesian network). D-separation is a graphical property that can be used to establish conditional independence between node sets, so that knowing a value for one node set will not affect the probable value for the other node set. For risk assessment within attack graphs, however, the general definition of graphical d-separation can be further reduced. In this approach, there will be no joint assessment over node sets with serial or converging connections, so that the only case applicable for d-separation in attack graphs deals with divergent paths. Therefore, the concept of d-separation is refined here for my specific application, relying on the well-formed structure of an attack graph.

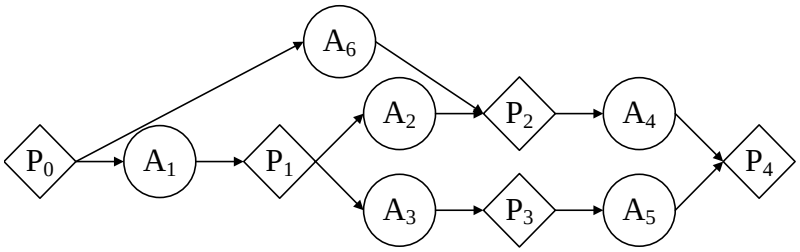


Figure 3.1: *Example: Acyclic Attack Graph*

Definition 9. *Within a logical attack graph, for the purposes of risk assessment, two distinct nodesets A and B are* d-separated *by an intermediate nodeset $V \subseteq G_B$ (distinct from A and B) if along every diverging path between A and B, there is some $v \in V$ such that v is the point of divergence.*

Because of the structure of the attack graph, only *shared dependencies* (points of divergence in shared paths) need to be considered in the construction of a d-separating set. For example, in Figure 3.1, there is a diverging path $A_2 \leftarrow P_1 \rightarrow A_3$ between nodes A_2 and A_3; the point of divergence, P_1, d-separates these two nodes. Even though A_2 and A_3 are not independent (they are both influenced by P_1), when the value of P_1 is fixed they become conditionally independent.

In an attack graph with the configuration nodes removed, the divergence nodes must be the branch nodes, which are all privilege nodes. Then the d-separating set D for two nodesets must be a subset of G_B and the elements in D must "block" all diverging paths between the two node sets. To discover a set of nodes that d-separates two node sets, we only need to consider branch nodes along the paths to the nodes. Take A_4 and A_5 in Figure 3.1 as an example; the set of branch nodes along the paths to A_4 and A_5 is $D = \{P_0, P_1\}$. We can see from the graph that D d-separates A_4 and A_5, so once the nodes in D are fixed, A_4 and A_5 become conditionally independent.

Theorem 1. $\forall D, N \subseteq G_N, Pr[N] = \sum_D Pr[N|D] \cdot Pr[D]$

Proof:

$$\begin{aligned}\sum_{D} Pr[N|D] \cdot Pr[D] &= \sum_{D} Pr[N, D] \\ &= Pr[N]\end{aligned}$$

By Bayes theorem, $Pr[N|D] \cdot Pr[D]$ produces the joint probability $Pr[N, D]$. Summing over all possible values of D will *marginalize* set D from the joint distribution $Pr[N, D]$ and produce $Pr[N]$.

Theorem 2. *Let D, N be node sets such that D d-separates all $n \in N$. Then all nodes $n \in N$ are* conditionally independent *given D: $Pr[N|D] = \prod_{n \in N} Pr[n|D]$.*

This theorem follows directly from the property of d-separation and conditional independence.

In order to calculate the risk value or probability for each $n \in G_N$ for some attack graph G, it is convenient to define and employ the following notations:

Function 1. *For every node $n \in G_N$, $\phi(n)$ represents the absolute probability that node n is true (i.e., the probability that the privilege/attack step represented by node n will be successfully gained/launched). Similarly, $\phi(\overline{n})$ represents the probability that node n is false. $\phi(n) + \phi(\overline{n}) = 1$. For a node set $N \subseteq G_N$, $\phi(N)$ represents the probability that each $n \in N$ will be true or false as specified in set N.*

For example, in Figure 3.1, $\phi(P_4)$ is the probability that node P_4 is true, while $\phi(\overline{P}_4)$ is the probability that P_4 is false. Since P_4 must be either true or false, these two values must have a sum of one.

$\phi(\overline{A}_4, \overline{A}_5)$ is the joint probability that nodes A_4 and A_5, are both false; since there are shared dependencies between these nodes, calculating the precise value of $\phi(\overline{A}_4, \overline{A}_5)$ would require a d-separating set.

Function 2. *For every node $n \in G_N$ and node set $A \subseteq G_N$, $\psi(A, n)$ represents the conditional probability that node n is true if the values of all $a \in A$ are fixed as specified in set A. For a node set $N \subseteq G_N$, $\psi(A, N)$ represents the conditional probability that each $n \in N$*

will be true or false as specified in set N if the values of all $a \in A$ are fixed as specified in set A.

For example, in Figure 3.1, $\psi(\{P_0, P_1\}, P_2)$ is the probability that P_2 is true given that P_0 and P_1 are true. This eliminates any influence that A_1 has on node P_2, by assuming that P_1 is true.

Function 3. *For every node $n \in G_N$, $\chi(n) = \{b | b \in G_B$ and $\exists P \in G_P[n], b \in P\}$*

$\chi(n)$ is the set of all branching nodes within the graph that appear in at least one path to n and so may affect the probability of n. For example, in Figure 3.1, $\chi(A_6) = \{P_0\}$ and $\chi(A_2) = \{P_0, P_1\}$.

Function 4. *For every node $n \in G_N$, $\delta(n) = \{d | d \in G_B$ and $d \Leftarrow n\}$.*

$\delta(n)$ is the set of all branch nodes that logically dominate n (Definition 5), so that $\forall d \in \delta(n),\ \psi(\overline{d}, n) = 0$. In other words, n is false if any $d \in \delta(n)$ is false. Clearly, the set of nodes that must affect n is a subset of the all the nodes that may affect n, so $\delta(n) \subseteq \chi(n)$.

Employing these notations, I will now consider how to calculate these values for nodes within an attack graph.

Proposition 1. *For any privilege node $n \in G_D$ with immediate predecessor set W,*

$$\begin{aligned}
\phi(n) &= 1 - \phi(\overline{W}) \\
\psi(A, n) &= 1 - \psi(A, \overline{W}) \\
\chi(n) &= \bigcup_{w \in W} \chi(w) \\
\delta(n) &= \bigcap_{w \in W} \delta(w)
\end{aligned}$$

A privilege node n will be true when at least one of its predecessors are true; conversely, it will be false only when all of its predecessors are false. Similarly, given set A, n is true

except when all predecessors are false. The χ set for n is the set of branch nodes that affect at least one path to some $w \in W$ and so at least one path to n; the δ set for n is the set of branch nodes that affect *all* paths to n (and so logically dominate n). Since the predecessors of a privilege node are all attack nodes, they cannot be branch nodes themselves.

Proposition 2. *For any privilege node $n \in G_D$ with immediate predecessor set $\{w\}$,*

$$\phi(n) = \phi(w)$$
$$\psi(A, n) = \psi(A, w)$$
$$\chi(n) = \chi(w)$$
$$\delta(n) = \delta(w)$$

Proposition 2 is obviously a special case of Proposition 1, where the computation of values is simplified upon the assumption of exactly one predecessor, w. By Proposition 1, $\phi(n) = 1 - \phi(\overline{w})$, but since $\phi(w) + \phi(\overline{w}) = 1$, we know that $\phi(n) = 1 - (1 - \phi(w) = \phi(w)$. $\chi(n) = \bigcup_{w \in W} \chi(w) = \chi(w)$. Since all paths to n lead through w, $\delta(n) = \delta(w)$.

Proposition 3. *For any attack-step node n with immediate predecessor set W,*

$$\phi(n) = G_M[n] \cdot \phi(W)$$
$$\psi(A, n) = G_M[n] \cdot \psi(A, W)$$
$$\chi(n) = (\bigcup_{w \in W} \chi(w)) \cup (G_B \cap W)$$
$$\delta(n) = (\bigcup_{w \in P} \delta(w)) \cup (G_B \cap W)$$

When all predecessors are true, an attack step node n is true with conditional probability $G_M[n]$. Similarly, given set A, n is true with conditional probability $G_M[n]$ when all predecessors are conditionally true. Since n requires that all predecessors be true, the χ set for n is the set of branch nodes that affect at least one of its predecessors together with any of its predecessors that are branch nodes (and so affect n); the δ set for n is the set of branch

nodes that logically dominate *any* of its predecessors as well as any of its predecessors that are branch nodes (and so logically dominate n).

Proposition 4. *For any attack-step node $n \in G_C$ with immediate predecessor set $\{w\}$,*

$$\phi(n) = G_M[n] \cdot \phi(w)$$
$$\psi(A, n) = G_M[n] \cdot \psi(A, w)$$
$$\chi(n) = \chi(w) \cup (G_B \cap w)$$
$$\delta(n) = \delta(w) \cup (G_B \cap w)$$

Proposition 4 is obviously a special case of Proposition 3, where the computation of values is simplified upon the assumption of exactly one predecessor, w.

3.3.2 Example - Figure 3.1

We will now work through the sample graph shown in Figure 3.1, demonstrating my approach and showing its effectiveness at recognizing and correctly handling shared dependencies within the graph. A table showing all calculated values will be included at the end of this example. For the purposes of this example, each attack-step node will be assigned a symbolic component metric value, so that $\forall A_i \in G_C, G_M[A_i] = e_i$.

We begin with the root node, P_0. The probability that node P_0 is true is G_V, so $\phi(P_0) = 1$. As the root node, P_0 has no preceding nodes, so $\chi(P_0) = \{\ \}$ and $\delta(P_0) = \{\ \}$.

Now that we have calculated $\phi(P_0)$, we can calculate for either A_1 or A_6. Let us next calculate for A_6. Proposition 4 specifies the calculation for an attack-step node with exactly one predecessor. Thus, $\phi(A_6) = G_M[A_6] \cdot \phi(P_0) = e_6 \cdot 1 = e_6$; $\chi(A_6) = \delta(A_6) = \{P_0\}$.

We cannot yet evaluate node P_2, because not all of its predecessors have been evaluated. We will return, then, to evaluate A_1. Similarly to A_6, $\phi(A_1) = G_M[A_1] \cdot \phi(P_0) = e_1$; $\chi(A_1) = \delta(A_1) = \{P_0\}$.

We can now evaluate node P_1. Proposition 2 specifies the calculation for a privilege node with exactly one predecessor. Thus, $\phi(P_1) = \phi(A_1) = e_1$; $\chi(P_1) = \delta(P_1) = \{P_0\}$.

From this point, we could evaluate either A_2 or A_3. Let us next calculate for A_2. $\phi(A_2) = G_M[A_2] \cdot \phi(P_1) = e_2 \cdot e_1 = e_1 e_2$; $\chi(A_2) = \delta(A_2) = \{P_0, P_1\}$.

Now both predecessors to P_2 have been solved and we can calculate for this node. Proposition 1 specifies the calculation for a privilege node with multiple predecessors. So, $\phi(P_2) = 1 - \phi(\{\overline{A}_2, \overline{A}_6\})$. In previous cases with single predecessors, we already knew the probability of the predecessor, but in this case we do not yet know the joint probability of $\phi(\{\overline{A}_2, \overline{A}_6\})$ and so must solve for it.

To calculate $\phi(\{\overline{A}_2, \overline{A}_6\})$, since they have a shared dependency, we must identify a d-separating set for these two nodes. One such set can be found by taking the intersection of the χ sets for these nodes, so that $D = \chi(A_2) \cap \chi(A_6) = \{P_0\}$. In this way, D contains all branch nodes that can affect the values of both A_2 and A_6. The set D, then, contains all branch nodes that diverge to paths leading to A_2 and A_6, which should be sufficient to d-separate the nodes (Definition 9). Using set D, we can now solve for $\phi(\{\overline{A}_2, \overline{A}_6\})$.

$$
\begin{aligned}
\phi(\overline{A}_2, \overline{A}_6) &= \sum_{P_0} \psi(\{P_0\}, \overline{A}_2, \overline{A}_6)\phi(\{P_0\}) && (\textit{Theorem 1}) \\
&= \sum_{P_0} \psi(\{P_0\}, \overline{A}_2)\psi(\{P_0\}, \overline{A}_6)\phi(\{P_0\}) && (\textit{Theorem 2}) \\
&= \psi(\{P_0\}, \overline{A}_2)\psi(\{P_0\}, \overline{A}_6)\phi(\{P_0\}) + \\
&\quad \psi(\{\overline{P}_0\}, \overline{A}_2)\psi(\{\overline{P}_0\}, \overline{A}_6)\phi(\{\overline{P}_0\}) \\
&= (1 - e_1 e_2)(1 - e_6)(1) + (1)(1)(0) && (\textit{Propositions } 1-4) \\
&= 1 - e_1 e_2 - e_6 + e_1 e_2 e_6
\end{aligned}
$$

Then, $\phi(P_2) = 1 - \phi(\{\overline{A}_2, \overline{A}_6\}) = 1 - (1 - e_1 e_2 - e_6 + e_1 e_2 e_6) = e_1 e_2 + e_6 - e_1 e_2 e_6$. Also, $\chi(P_2) = \chi(A_2) \cup \chi(A_6) = \{P_0, P_1\}$, and $\delta(P_2) = \delta(A_2) \cap \delta(A_6) = \{P_0\}$.

Nodes A_3, A_4, A_5, P_3 are calculated very similarly to nodes we've already seen here, so we will skip over the details of these. The resulting values are in Table 3.2.

Finally, we evaluate for node P_4, a privilege node with multiple predecessors, so again we will apply Proposition 1: $\phi(P_4) = 1 - \phi(\{\overline{A}_4, \overline{A}_5\})$. The d-separating set for $\{A_4, A_5\}$ is

$D = \chi(A_4) \cap \chi(A_5) = \{P_0, P_1\}$. The term values for this calculation are shown in Table 3.1.

$$\begin{aligned}
\phi(\overline{A}_4, \overline{A}_5) &= \sum_{P_0,P_1} \psi(\{P_0,P_1\}, \overline{A}_4)\psi(\{P_0,P_1\}, \overline{A}_5)\phi(\{P_0,P_1\}) \\
&= \psi(\{P_0,P_1\}, \overline{A}_4)\psi(\{P_0,P_1\}, \overline{A}_5)\phi(\{P_0,P_1\}) + \\
&\quad \psi(\{P_0,\overline{P}_1\}, \overline{A}_4)\psi(\{P_0,\overline{P}_1\}, \overline{A}_5)\phi(\{P_0,\overline{P}_1\}) + \\
&\quad \psi(\{\overline{P}_0,P_1\}, \overline{A}_4)\psi(\{\overline{P}_0,P_1\}, \overline{A}_5)\phi(\{\overline{P}_0,P_1\}) + \\
&\quad \psi(\{\overline{P}_0,\overline{P}_1\}, \overline{A}_4)\psi(\{\overline{P}_0,\overline{P}_1\}, \overline{A}_5)\phi(\{\overline{P}_0,\overline{P}_1\}) \\
&= (1 - e_4(e_2 + e_6 - e_2e_6))(1 - e_3e_5)(e_1) + (1 - e_4e_6)(1)(1 - e_1) + (1)(1)(0) + (1)(1)(0) \\
&= 1 - e_1e_2e_4 + e_1e_2e_4e_6 - e_1e_3e_5 - e_4e_6 + e_1e_3e_4e_5(e_2 + e_6 - e_2e_6)
\end{aligned}$$

Then, $\phi(P_4) = 1 - \phi(\overline{A}_4, \overline{A}_5) = e_4e_6 + e_1(e_2e_4 + e_3e_5) - e_1e_2e_4e_6 - e_1e_3e_4e_5(e_2 + e_6 - e_2e_6)$. Also, $\chi(P_4) = \chi(A_4) \cup \chi(A_5) = \{P_0, P_1\}$, and $\delta(P_2) = \delta(A_2) \cap \delta(A_6) = \{P_0\}$.

Table 3.1: *Values calculated while solving for* $\phi(\overline{A}_4, \overline{A}_5)$

D	$\psi(D, \overline{A}_4)$	$\psi(D, \overline{A}_5)$	$\phi(D)$
$\{P_0, P_1\}$	$(1 - e_4(e_2 + e_6 - e_2e_6))$	$(1 - e_3e_5)$	e_1
$\{P_0, \overline{P}_1\}$	$(1 - e_4e_6)$	1	$1 - e_1$
$\{\overline{P}_0, P_1\}$	1	1	0
$\{\overline{P}_0, \overline{P}_1\}$	1	1	0

We have now solved for the probability of each node in the graph, and the computed ϕ values for individual nodes are shown in Table 3.2, together with the χ and δ sets for each node. In my implementation, joint and conditional probability values are calculated only as needed, to reduce the amount of computation performed; I also apply dynamic programming techniques to store and re-use calculated values, as needed.

Table 3.2: *Risk assessment calculations for Figure 3.1*

N	$\phi(N)$	$\delta(N)$	$\chi(N)$
P_0	1	$\{\}$	$\{\}$
P_1	e_1	$\{P_0\}$	$\{P_0\}$
P_2	$(e_1e_2 + e_6 - e_1e_2e_6)$	$\{P_0\}$	$\{P_0, P_1\}$
P_3	e_1e_3	$\{P_0, P_1\}$	$\{P_0, P_1\}$
P_4	$(e_4e_6 + e_1(e_2e_4 + e_3e_5) - e_1e_2e_4e_6 - e_1e_3e_4e_5(e_2 + e_6 - e_2e_6))$	$\{P_0\}$	$\{P_0, P_1\}$
A_1	e_1	$\{P_0\}$	$\{P_0\}$
A_2	e_1e_2	$\{P_0\}$	$\{P_0, P_1\}$
A_3	e_1e_3	$\{P_0, P_1\}$	$\{P_0, P_1\}$
A_4	$e_4(e_1e_2 + e_6 - e_1e_2e_6)$	$\{P_0\}$	$\{P_0, P_1\}$
A_5	$e_1e_3e_5$	$\{P_0, P_1\}$	$\{P_0, P_1\}$
A_6	e_1	$\{P_0\}$	$\{P_0\}$

3.3.3 Example - Figure 3.2

The last section demonstrated an approach to quantitative risk assessment over an attack graph. Within a cycle, however, recursing backward through predecessor sets will create an infinite loop. It is clear that a modified approach is needed for cyclic graphs.

Figure 3.2 contains an attack graph that I will use as an aid to explain and demonstrate quantitative risk assessment over an attack graph containing cycles. This attack graph contains one cycle, node set $\{P_2, P_3, A_4, A_5\}$.

Figure 3.3 shows the same attack graph after the partial unfolding algorithm (Section 3.2) has been applied. This unfolded attack graph is now acyclic, and shows all of the paths that traverse the cycle. Node P_2, for example, can be reached as P_{2a} or P_{2b}; the dotted-line arcs indicate that reaching either of these instances means that P_2 has been reached.

We will now work through the sample graph shown in Figure 3.2, demonstrating our approach and showing its effectiveness at recognizing and correctly handling cycles within the graph. A table showing all calculated values will be included at the end of this example. The same values can be generated by applying the acyclic approach to Figure 3.3.

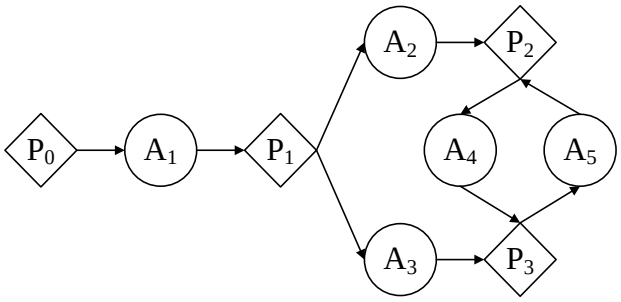


Figure 3.2: *Example: Cyclic Attack Graph*

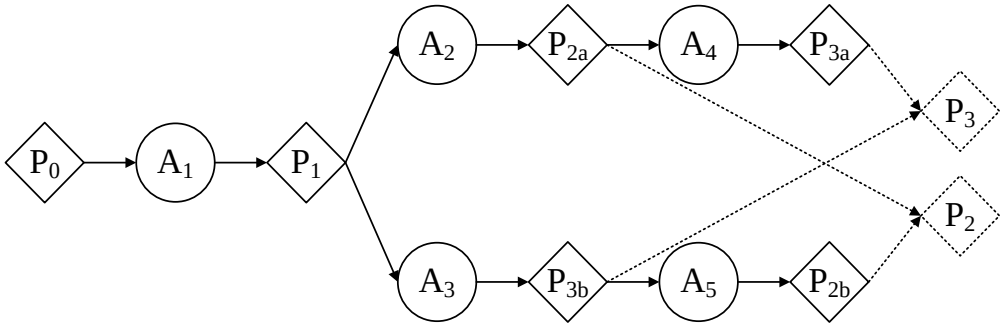


Figure 3.3: *Example: Cyclic Attack Graph - Unfolded*

Intuitively, we will identify all acyclic paths traversing the cycle and use this knowledge to logically identify unique possible instances of cyclic nodes. We can then d-separate over the set of reaching paths to calculate the probability that the cyclic node will be reached.

Nodes P_0, A_1, P_1, A_2, A_3 will be calculated very much as demonstrated in Section 3.3.2 and so I will not go through the detailed calculations for those nodes. Once these have been calculated, however, the remaining graph nodes comprise a cycle and therefore must be handled differently. In my implementation, I have chosen to use Tarjan's algorithm for the identification of strongly connected graph components [52].

We can simplify the handling of the cycle by calculating values only for cyclic nodes with multiple immediate predecessors. Considering Propositions 2 and 4, it is easily seen that the probability for a node with exactly one predecessor is dependent only on the value of that predecessor. If the probability is known for that predecessor, it does not matter if

the node is or is not contained in a cycle. In this example, the cyclic nodes with multiple predecessors are P_2 and P_3.

First, we will trace all acyclic paths through the cycle, to determine all valid ways that these nodes can be reached. This trace essentially performs a logical unfolding of the graph, marking unique passes through each node. The acyclic paths through this cycle are:

$$P_{2a} = \{A_2\} \qquad P_{3a} = \{A_2, P_{2a}, A_4\}$$
$$P_{3b} = \{A_3\} \qquad P_{2b} = \{A_3, P_{3b}, A_5\}$$

There are two unique instances of node P_2 in this logical unfolding of the graph, P_{2a} and P_{2b}. The probability that node P_2 is true will equal the probability that at least one of these instances is true, or one minus the probability that both instances are false. Then $\phi(P_2) = 1 - \phi(\overline{P}_{2A}, \overline{P}_{2B})$.

Definition 10. *For some set of cyclic attack graph nodes $C \subset G_N$, the* entry nodes *are the set of nodes Q such that $Q \cap C = \{\}$ and $\forall q \in Q, \exists c \in C, (q, c) \in G_E$. That is, the entry nodes are not in the cycle, but each entry node does have an arc leading into the cycle.*

To calculate $\phi(\overline{P}_{2A}, \overline{P}_{2B})$, we must calculate the joint probability of the set of entry nodes $\{A_2, A_5\}$ and we will also need to identify a d-separating set D within the cycle, to ensure that the instances of P_2 are conditionally independent. In this case, $D = Path(P_{2A}) \cap Path(P_{2B}) = \emptyset$; because the cycle is so small, these partial paths are already conditionally independent, given the entry points into the cycle. The values calculated for each term in the equation below are shown in Table 3.3.

$$\begin{aligned}
\phi(\overline{P}_{2A}, \overline{P}_{2B}) &= \textstyle\sum_{A_2,A_3} \phi(\{A_2, A_3\})\psi(\{A_2, A_3\}, \overline{P}_{2A})\psi(\{A_2, A_3\}, \overline{P}_{2B}) \\
&= \phi(\{A_2, A_3\})\psi(\{A_2, A_3\}, \overline{P}_{2A})\psi(\{A_2, A_3\}, \overline{P}_{2B}) + \\
&\quad \phi(\{A_2, \overline{A}_3\})\psi(\{A_2, \overline{A}_3\}, \overline{P}_{2A})\psi(\{A_2, \overline{A}_3\}, \overline{P}_{2B}) + \\
&\quad \phi(\{\overline{A}_2, A_3\})\psi(\{\overline{A}_2, A_3\}, \overline{P}_{2A})\psi(\{\overline{A}_2, A_3\}, \overline{P}_{2B}) + \\
&\quad \phi(\{\overline{A}_2, \overline{A}_3\})\psi(\{\overline{A}_2, \overline{A}_3\}, \overline{P}_{2A})\psi(\{\overline{A}_2, \overline{A}_3\}, \overline{P}_{2B}) \\
&= (0) + (0) + (e_1e_3(1 - e_2))(1)(1 - e_4) + (e_1(1 - e_2)(1 - e_3) + 1 - e_1)(1)(1) \\
&= 1 - e_1e_2 - e_1e_3e_4 + e_1e_2e_3e_4
\end{aligned}$$

Table 3.3: *Values calculated while solving for* $\phi(\overline{P}_{2a}, \overline{P}_{2b})$

D	$\psi(D, \overline{P}_{2a})$	$\psi(D, \overline{P}_{2b})$	$\phi(D)$
$\{A_2, A_3\}$	0	$1 - e_5$	$e_1e_2e_3$
$\{A_2, \overline{A}_3\}$	0	1	$e_1e_2(1 - e_3)$
$\{\overline{A}_2, A_3\}$	1	$1 - e_5$	$e_1e_3(1 - e_2)$
$\{\overline{A}_2, \overline{A}_3\}$	1	1	$e_1(1 - e_2)(1 - e_3)$

So, $\phi(P_2) = 1 - (1 - e_1e_2 - e_1e_3e_4 + e_1e_2e_3e_4) = e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4$. By a similar calculation, $\phi(P_3) = e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5$. Once these have been solved, it is easy to see that $\phi(A_4) = G_M[A_4] \cdot \phi(P_3) = e_4(e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5)$ and $\phi(A_5) = G_M[A_5] \cdot \phi(P_2) = e_5(e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4)$.

The full results are shown in Table 3.4. The cyclic nodes do not have χ or δ sets, since these values are not used to compute cyclic nodes.

Table 3.4: *Risk assessment calculations for Figure 3.2*

N	$\phi(N)$	$\delta(N)$	$\chi(N)$
P_0	1	$\{\}$	$\{\}$
P_1	e_1	$\{\}$	$\{\}$
P_2	$e_1(e_2 + e_3e_5 - e_2e_3e_5)$	–	–
P_3	$e_1(e_3 + e_2e_4 - e_2e_3e_4)$	–	–
A_1	e_1	$\{\}$	$\{\}$
A_2	e_1e_2	$\{P_1\}$	$\{P_1\}$
A_3	e_1e_3	$\{P_1\}$	$\{P_1\}$
A_4	$e_1e_4(e_2 + e_3e_5 - e_2e_3e_5)$	–	–
A_5	$e_1e_5(e_3 + e_2e_4 - e_2e_3e_4)$	–	–

3.4 Algorithms

I will now present several algorithms for correct risk assessment over an attack graph data set. These algorithms are based in probabilistic reasoning, much like a Bayesian network, but are designed to handle cycles, as well. Well-founded data-flow analysis techniques are used to

examine cyclic data and evaluate over it. The algorithms also employ dynamic programming practices, storing all values for future reference to eliminate repeated calculations.

Algorithm 3.2 Pseudocode for risk assessment

1: Identify cyclic subsets
2: $n \leftarrow G_R$ {Begin with graph root node}
3: $\phi(n) \leftarrow G_V$
4: $\chi(n) \leftarrow \emptyset$
5: $\delta(n) \leftarrow \emptyset$
6: $U \leftarrow G_N - n$ { Initialize set of unevaluated nodes, excluding root node}
7: **while** $U \neq \{\ \}$ **do** { some nodes remained unevaluated }
8: **if** $\exists\ n \mid n \in U, \forall p \mid [p,n] \in G_E, p \notin U$ **then** { node n is ready to be evaluated }
9: $P \leftarrow \{p \mid [p,n] \in G_E\}$ { Immediate predecessors of n }
10: **if** $n \in G_D$ **then**
11: $\phi(n) = 1 - evalProb(\overline{P})$ { Call Algorithm 3.3 }
12: $\chi(n) = \{\bigcup_{p \in P} \chi(p)\}$
13: $\delta(n) = \{\bigcap_{p \in P} \delta(p)\}$
14: **if** $n \in G_B$ **then**
15: $\psi(n,n) \leftarrow 1$
16: **end if**
17: **else** { $n \in G_C$ }
18: $\phi(n) \leftarrow G_M[n] \cdot evalProb(P)$ { Call Algorithm 3.3 }
19: $\chi(n) = (G_B \cap P) \cup \bigcup_{p \in P} \chi(p)$
20: $\delta(n) = (G_B \cap P) \cup \bigcup_{p \in P} \delta(p)$
21: **end if**
22: $U \leftarrow U - n$ { Mark node n as evaluated }
23: **else** { a cycle is ready for evaluation }
24: $C \leftarrow$ cyclic set ready for evaluation (all non-cyclic predecessors evaluated)
25: $M \leftarrow evalCycle(C)$ { Call Algorithm 3.5 }
26: $U \leftarrow U \setminus M$ { Mark node set M as evaluated }
27: **end if**
28: **end while**

Algorithm 3.2 presents the core algorithm for risk assessment over an attack graph. This algorithm consists primarily of a controlling loop that will iteratively consider each individual non-cyclic node or set of cyclic nodes in the graph. This loop will terminate only

when a risk assessment evaluation has been performed for every node.

The algorithm assumes a MulVAL attack graph node set, with configuration nodes removed and a new privilege node (representing the attacker's starting point) added to serve as a root node for the graph. Cycles in the graph are identified using Tarjan's algorithm for strongly connected components [52]. Before entering the control loop, the data set is initialized for the graph root node. The probability that this node is true is exactly the same probability that the network will be attacked - G_V. Because this is the root node, the χ and δ sets are empty; no other nodes will affect the root node.

Within the loop, we attempt to find an individual, non-cyclic node n that is itself unevaluated but for which all predecessors have already been evaluated. If we find an OR-node n, we know n will be always true except when *all* of its predecessors P are false (Proposition 1). We can thus solve for the probability of n by recursively solving for the joint probability of the negation of set P. The χ set of n, those branching nodes that *may* affect n, can be found by a union over the χ sets for all predecessors. The δ set of n, those branching nodes that *must* affect n, can be found by intersecting the δ sets for predecessors. Because n is an OR-node, it may be affected by any branching node affecting at least one predecessor, but it must be affected by any branching node affecting all of its predecessors.

If we find an AND-node n, we know that n will be true with probability $G_M[n]$ when all of its predecessors P are true (Proposition 3). We can thus solve for the probability of n by recursively solving for the joint probability of set P. The χ set of n can be found by a union over the χ sets for all of its predecessors, adding, of course, any predecessors that are themselves branching nodes. The δ set of n can also be found by a union over the δ sets for all of its predecessors. This set construction differs from the δ set for an OR-node, because each AND-node is necessarily affected by all branching nodes affecting at least one predecessor.

If no individual node n can be found ready for evaluation, it must be the case that at least one cycle in the graph is ready for evaluation. These graph cycles are easily identified

using Tarjan's algorithm for the discovery of strongly connected components [52]. A cycle C is "ready" for evaluation when all entry nodes to the cycle (Definition 10) have been evaluated. In this case, Algorithm 3.5 will be called to handle the cyclic node set C; the return value will be set $M \subset C$, the subset of nodes in set C that have multiple immediate predecessors. As discussed earlier, only multi-predecessor nodes within a cycle must be specially treated. Once these are solved, all remaining (single-predecessor) cyclic nodes can be solved as in Proposition 2 or 4 without consideration of the cycle.

After a cycle has been evaluated, it will be abstracted to a single node for representation in the graph. In this way, successor nodes following the cycle can be treated acyclically. Where a node has multiple predecessors within a cycle, the cyclic nodes will be d-separated and the joint probability calculated.

After each loop iteration, a single node n or a node set M will be removed from the set of remaining, unevaluated nodes. The loop will re-iterate at this point, continuing until calculations have been made for all nodes in the graph.

Algorithm 3.3 Pseudocode for computing $evalProb(N)$

Require: Parameter N, such that
$N = \{n_0, n_1, ..., n_j\} \subseteq G_N$, such that $\forall n_i \in N, \phi(n)$ has already been evaluated

if $\phi(N)$ previously calculated **then**
  return $\phi(N)$ from table of stored values
end if
{ Find d-separating set D for node set N }
$D \leftarrow \bigcup_{m,n \in N} \chi(m) \cap \chi(n)$
if $D = \emptyset$ **then**
  { No d-separating set, so nodes are independent }
  return $\prod_{n \in N} \phi(n)$
else
  { Calculate conditional probabilities given D }
  return $\sum_{D} evalCondProb(D, N) \cdot evalProb(D)$
end if

Algorithm 3.3 - $\phi(N)$ - calculates the joint probability of an acyclic node set N. This

algorithm will be called from the main algorithm [Algorithm 3.2] and $\phi'(N)$ [Algorithm 3.5]. When called, this algorithm first checks if the requested calculation has already been performed and the answer stored for future use. If so, the stored answer is retrieved and returned, without repeating the same calculation.

Assuming that the requested value is not already calculated, we must find a d-separating set D such that all $n \in N$ are conditionally independent given D. It is sufficient here to construct a set D of all branching nodes that may affect two or more elements in N.

If D is an empty set, then all $n \in N$ are fully independent. In this case, the probability of N is equal to the product of the probabilities of all $n \in N$.

Otherwise, when D is not empty, the probability of set N is equal to the summation over all possible values of D of the conditional probability of N given D multiplied by the probability of D. In this way, the d-separating set D is marginalized out of the calculation, providing a joint probability value for set N.

Algorithm 3.4 Pseudocode for computing $evalCondProb(D, N)$

Require: Parameters D, N, such that
$N = \{n_0, n_1, ..., n_j\}, N \subseteq G_N, |N| \geq 1$, such that $\forall n_i \in N, \phi(n)$ has been evaluated
$D = \{d_0, d_1, ..., d_k\}, D \subseteq G_N$, such that D d-separates all $n \in N$

1: **if** $\psi(D, N)$ previously calculated **then**
2: **return** $\psi(D, N)$ from table of stored values
3: **end if**
4:
5: **if** $|N| > 1$ **then**
6: **return** $\prod_{n \in N} evalCondProb(D, \{n\})$
7:
8: **else if** $N = \{\overline{n}\}$ **then** { N contains exactly negative element }
9: **return** $1 - evalCondProb(D, \{n\})$
10:
11: **else** { $N = \{n\}$, so that N contains exactly positive element }
12: $J \leftarrow \{j \mid j \in D\}$ { All positive elements in D }
13: $K \leftarrow \{k \mid \overline{k} \in D\}$ { All negative elements in D }
14:
15: **if** $n \in J$ **then**
16: **return** 1
17: **end if**
18:
19: { If n or a dominator of n is negated in D }
20: **if** $n \in K$ or $K \cap \delta(n) \neq \emptyset$ **then**
21: **return** 0
22: **end if**
23:
24: { If set D does not affect the value of n }
25: **if** $D \cap \chi(n) = \emptyset$ **then**
26: **return** $\phi(n)$
27: **end if**
28:
29: $P \leftarrow \{p \mid \exists n \in N, [p, n] \in G_E\}$ { Immediate predecessors of N }
30: **if** $n \in G_D$ **then**
31: **return** $1 - evalCondProb(D, \overline{P})$
32: **else** { $n \in G_C$ }
33: **return** $G_M[n] \cdot evalCondProb(D, P)$
34: **end if**
35: **end if**

Algorithm 3.4 - $evalCondProb(D, N)$ - calculates the conditional probability of node set

N given d-separating set D. This algorithm will be called from $evalProb(N)$ [Algorithm 3.3] to aid in the computation of joint probabilities for node sets. Like Algorithm 3.3, this algorithm first checks if the requested value has already been calculated and stored; if this is the case, the saved value is retrieved and returned.

If the node set N contains multiple nodes, then the conditional probability of N given D is equal to the product of the conditional probability of n given D, over all $n \in N$. This is true because of the d-separating nature of D, so that all $n \in N$ are conditionally independent given D.

If N contains exactly one node n, and that node is negated (or disabled), then the conditional probability of $\overline{n}$ is equal to one minus the conditional probability of n. The algorithm will be recursively called to calculate this value.

We must first consider some base cases that may reduce the calculations performed. If N contains exactly one node n, and that node is enabled, we must calculate the conditional probability of n given set D. If $n \in D$, then n is forced to be true and so the probability of n is one; this value will be returned accordingly. If $\overline{n} \in D$ or the negation of some other dominator of n is in D, then n must necessarily be false and have probability zero; this value will be returned in such a case. If there are no shared elements between D and $\chi(n)$, then D has no effect on the probability of n; in this case,the conditional probability of n given D is equal to the total probability of n, and so this value (already calculated) is returned.

Finally, if none of these base cases are satisfied, we must move backward in the graph, considering the predecessors of n. If n is an OR-node, then n is true except when all of its predecessors are false given D. If n is an AND-node, then n is true with probability $G_M[n]$ when all of its predecessors are true given D.

This function will continue to make recursive calls until a base case is satisfied, and a final answer can be returned. Formal proofs are provided in Appendix B, showing that this function, though recursive, will always terminate successfully.

Algorithm 3.5 Pseudocode for $evalCycle(C)$ - assessment over nodes in cycle

Require: Parameters C, such that
$C = \{c_0, c_1, ..., c_k\} \subset G_N$ comprises a strongly connected component in G

{ Trace all acyclic reaching paths }
for all $p \mid p \notin C,\ \exists a.(a \in C, [p, a] \in G_E)$ **do**
    $tracePaths(a, \{p\}, C)$ { Algorithm 3.6 }
end for

{Solve all nodes in cycle with multiple predecessors}
$Q \leftarrow$ set of entry nodes (non-cyclic nodes, leading into cycle)
$M \leftarrow \{m \mid m \in C,\ \exists p, q.(p \neq q, [p, m] \in G_E, [q, m] \in G_E)\}$
for all $m \in M$ **do**
    $P \leftarrow$ set of acyclic paths to m through cycle (identified by $tracePaths$ call)
    $V \leftarrow$ set of possible instances of m (such that each $v \in V$ is attainable by exactly one path $p \in P$)
    { Find set of nodes appearing in multiple acyclic paths in set P }
    $T \leftarrow \bigcup_{p,q \in P} (p \cap q)$
    { Identify d-separating set within cycle for all paths to m }
    $D \leftarrow (G_C \cap T) \setminus Q$
    { Find the probability of m, transitively d-separating by Q and D }
    if $m \in G_D$ **then** { m is an OR-node }
        $$\phi(m) \leftarrow 1 - \sum_{Q} \left(evalProb(Q) \cdot \sum_{D} \left[\left(\prod_{d \in D} (G_M[d]) \right) \cdot evalCycleNode(Q \cup D, \overline{V}) \right] \right)$$
    else { m is an AND-node }
        $$\phi(m) \leftarrow G_M[m] \cdot \left[1 - \sum_{Q} \left(evalProb(Q) \cdot \sum_{D} \left[\left(\prod_{d \in D} (G_M[d]) \right) \cdot evalCycleNode(Q \cup D, \overline{V}) \right] \right) \right]$$
    end if
end for
return M

Algorithm 3.5 controls risk assessment calculation for certain nodes within a cyclic node set C. This algorithm is called from the main algorithm [Algorithm 3.2] to consider cyclic components within the attack graph. Given a cyclic node set C, this algorithm will first identify all acyclic reaching paths that lead from outside of the cycle to a node within the cycle [Algorithm 3.6].

Once all acyclic paths through the cycle C have been identified, this algorithm will calculate a risk assessment value for each $m \in C$ such that m has multiple predecessors

(multiple arcs leading to m). It is sufficient to specially handle these nodes within the cycle and allow single-predecessor nodes to be handled by Algorithm 3.2. In applying Propositions 2 and 4 to single-predecessor nodes, no differentiation needs to be made for acyclic or cyclic nodes [54]. This optimization will reduce the overall running time of the algorithm.

For each multi-source node $m \in C$, there is a set V of "partial values," or different instantiations of m reached by different paths. The node m will be true except when all of these "partial values" are unreachable. To find a d-separating set D within the cycle for all $v \in V$, it is sufficient to identify all attack-step nodes appearing on multiple paths in P, excluding the entry nodes Q (which are not in the cycle). Because all possible acyclic paths are considered, the set of attack-step nodes lying along each path is unique to that path. We will not use the full probability of these nodes, but only the individual component metrics associated with them, calculating the likelihood of success for each path within the cycle. Together with a set of cycle entry nodes Q, we can now solve for the probability of node m.

If m is a privilege node, then m is true except when all $v \in V$ are false. To find this value, we must marginalize over sets Q and D, eliminating their conditional influence on m. If m is an attack-step node, it is similarly computed, considering also the individual component metric value of m - $G_M[m]$. Because we are using only the component metric values for D, the order and derivation of each attack path is unimportant; only the product of the individual probabilities is needed.

Once probabilities have been calculated for all multi-predecessor nodes $M \subset C$, the set M will be returned to the main algorithm to be marked as evaluated. Again, all single-predecessor cycle nodes will be treated as acyclic, to reduce overall run-time of the algorithm. In this way, the cycle is properly evaluated, so that no node influences its own probability.

Algorithm 3.6 Pseudocode for $tracePaths(n, P, C)$ - tracing acyclic paths through cycle

Require: Parameters n, P, C, such that
$n \in C$,
$P = \{p_0, p_1, ..., p_m\} \subset Q \cup C$ is a set of nodes comprising an acyclic path to node n,
$C = \{c_0, c_1, ..., c_k\}$ is a strongly connected component $C \subset G_N$

1: { Get set of successors to n not appearing in path set P }
2: $S \leftarrow \{s \mid s \in C, s \notin P, [n, s] \in G_E\}$
3:
4: Store set P as partial path to node n
5: **for all** $s \in S$ **do**
6: $\quad tracePaths(s, P \cup n, C)$
7: **end for**

Algorithm 3.6 - $tracePaths(n, P, C)$ - performs a logical "unfolding" of the strongly connected graph component C, comparable to the graphical unfolding presented in Section 3.2 in that all acyclic reaching paths are identified for each node $c \in C$.

The paths are primed by cycle entry nodes. As a node is visited, a unique, non-cyclic path set (sufficient for obtaining one instance of that node) will be saved for future reference, representing one non-cyclic path to that node. This algorithm will never produce duplicate paths to any one node. Whenever the algorithm discovers a connecting node that is already in the path set P, this avenue of exploration is ceased, to prevent cyclic paths from occurring. In this way, all of the exploratory paths will eventually end, so the algorithm will terminate successfully. These partial paths and values are stored for use in calculating probability for multi-predecessor cycle nodes in Algorithm 3.5.

Algorithm 3.7 Pseudocode for $evalCycleNode(D, N)$

Require: Parameters D, N, such that
D d-separates all $n \in N$, and
$N = \{n_0, n_1, ..., n_j\}$ are partial values for some node $n \in C$

1: **if** $|N| > 1$ **then**
2: **return** $\prod_{n \in N} evalCycleNode(D, \{n\})$
3: **else if** $N = \{\overline{n}\}$ **then**
4: **return** $1 - evalCycleNode(D, \{n\})$
5: **else** { $N = \{n\}$, so that N contains exactly one element, enabled }
6: $P \leftarrow$ partial path (set of attack-step nodes) leading to node instance n
7: $J \leftarrow \{j \mid j \in D\}$ { All enabled nodes in D }
8: $K \leftarrow \{k \mid \overline{k} \in D\}$ { All disabled nodes in D }
9:
10: { If node in path is negated, n cannot be reached by path P }
11: **if** $K \cap P \neq \emptyset$ **then**
12: **return** 0
13: **end if**
14:
15: { Discard any path nodes forced true }
16: $P \leftarrow P \setminus J$
17:
18: **return** $\prod_{p \in P} G_M[p]$
19: **end if**

Algorithm 3.7 - $evalCycleNode(D, N)$ - calculates conditional probability for a set N of node instances given a d-separating set D. D is the union of the set of cycle entry nodes and the attack-step nodes appearing on multiple partial paths within the cycle. This algorithm is called from $evalCycle(N)$ [Algorithm 3.5]. It is similar in many respects to $evalCondProb(D, N)$ [Algorithm 3.4], but utilizes the acyclic reaching paths identified within the cycle to perform its probability calculations.

If node set N contains multiple nodes, then the product of the conditional probability for all $n \in N$ is calculated and returned. If node set N contains exactly one node n, and that node is disabled, then $\overline{n}$ holds except when n is conditionally true. This value, the inverse, will be calculated and returned.

If node set N contains exactly one node n, and that node is enabled, we must calculate a conditional probability for n given D. Set P contains all attack-step nodes in entry nodes or cycle nodes that lie along the acyclic reaching path to node instance n. If some node in path P is negated in D, then n cannot be reached by path P and so has probability zero. Otherwise, we must remove from P all nodes that are forced true in D, since these can have no effect on the probability of n by path P.

Once all such fixed values have been accounted for, we know that the node is reachable along path P. Because the attack-step nodes in P are treated independently, knowing the probability that all will jointly succeed gives us the likelihood that an attacker will succeed along path P. The algorithm therefore calculates the product of the component metric values for all remaining nodes in path P; this value is the probability that n is true by path P, given set D.

Together, Algorithms 3.2 - 3.7 can correctly evaluate for the probability of every node within an attack graph, properly considering and allowing for shared dependencies and the presence of cycles. Formal proofs of the soundness of this approach are provided in Appendix B. I will next consider the efficiency and scalability of these algorithms.

3.5 Scalability and Complexity

In order to test the scalability of our approach, we constructed several testing models based on networks of varying sizes and complexity, created MulVAL input files representing each network, and evaluated them with a preliminary implementation of our algorithms. The results of these tests are displayed in Table 3.5.

The limiting factors in the current algorithm and implementation are the size of the d-separating set (the number of nodes which must be marginalized in calculating conditional probability values) and the number of paths that must be considered in the calculation of each multi-predecessor node within a cycle. As any of these increase, the number of recursive calls made by the algorithm increases and so the evaluation time also grows.

Table 3.5: *Scalability test results - risk assessment*

Scenario	Testing model	Number of network hosts	Number of nodes in largest cycle	Number of cycle entry nodes	Number of multi-predecessor nodes in cycle	Number of unique acyclic paths per node	Number of d-separating nodes within cycle	Time required to evaluate single node (average)	Total algorithm run-time
Realistic models	A	9	28	4	4	16	3	< 1 sec	0:04
	B	10	40	4	8	20	6	0:02	0:15
	C	9	46	5	11	56	9	0:46	8:31
Cliques	D	6	40	5	5	65	4	0:01	0:06
	E	7	54	6	6	326	5	0:25	2:33
	F	8	70	7	7	1957	6	7:45	54:40

Models A, B, and C contain realistic network models. Model C is based on a real control system network for power grids, which cannot be revealed here. Model B is adapted from model C, adding an additional host and altering the accessibility configurations between hosts. Model A is still more sparsely connected, so that no large cycles predominate in the graph.

Models D, E, and F contain cliques, in which every network host is accessible from every other host, so a single strongly-connected component (cycle) contains almost every node in the attack graph (except the configuration nodes and the privilege representing an attacker's initial access to the network). In these scenarios, each host has exactly one remote-exploitable vulnerability that can provide an attacker with code-execution privileges. These models represent the worst-case situation in network connectivity, because a more strongly-connected attack graph will include a greater number of acyclic paths reaching to each graph node, making the assessment more complex. In a clique, each additional node will dramatically increase the number of acyclic paths per node; the number of entry nodes and the size of the d-separating set also increase.

Each of these models contains a clique, in which every network host is accessible from

every other host, so a single strongly-connected component contains almost every node in the attack graph (except the configuration nodes and the privilege representing an attacker's initial access to the network). Each host has exactly one remote-exploitable vulnerability that can provide an attacker with code-execution privileges. As the network size increases, so do the number of nodes in the cycle and the number of multi-predecessor nodes that are evaluated specially, as well as the number of acyclic paths reaching these nodes.

These performance tests show that the approach in its current implementation is practical for some realistic models, but may not scale well over enterprise network models containing very large, strongly-connected cycles in the attack graph. Some possible further optimizations and other future work are addressed in Chapter 5.

3.6 Implementation and Testing

Figure 3.4 shows a sample enterprise network, which I will use to demonstrate the application of my risk assessment algorithms.

Hosts and Accessibility: There are five hosts (or groups of hosts) with this enterprise network. The DMZ subnet and the VPN server are directly accessible from the Internet. The DMZ subnet, which contains only the web server, is permitted to access the database server through the firewall. The VPN server, which also serves as a firewall to protect internal subnets, is able to access the two subnets of user workstations, Group 1 and Group 2. Although there are multiple machines in each of these subnets, it is sufficient to model each group as a single host. Groups 1 and 2 also have permissions to access the database server, each through a unique application and port. The database server is permitted to access the user workstation subnets.

Vulnerabilities: All of the machines in this network have at least one vulnerability that potentially allows for remote exploitation, given an attacker code-execution privileges on the machine. The database server actually hosts three vulnerabilities, one on each of the

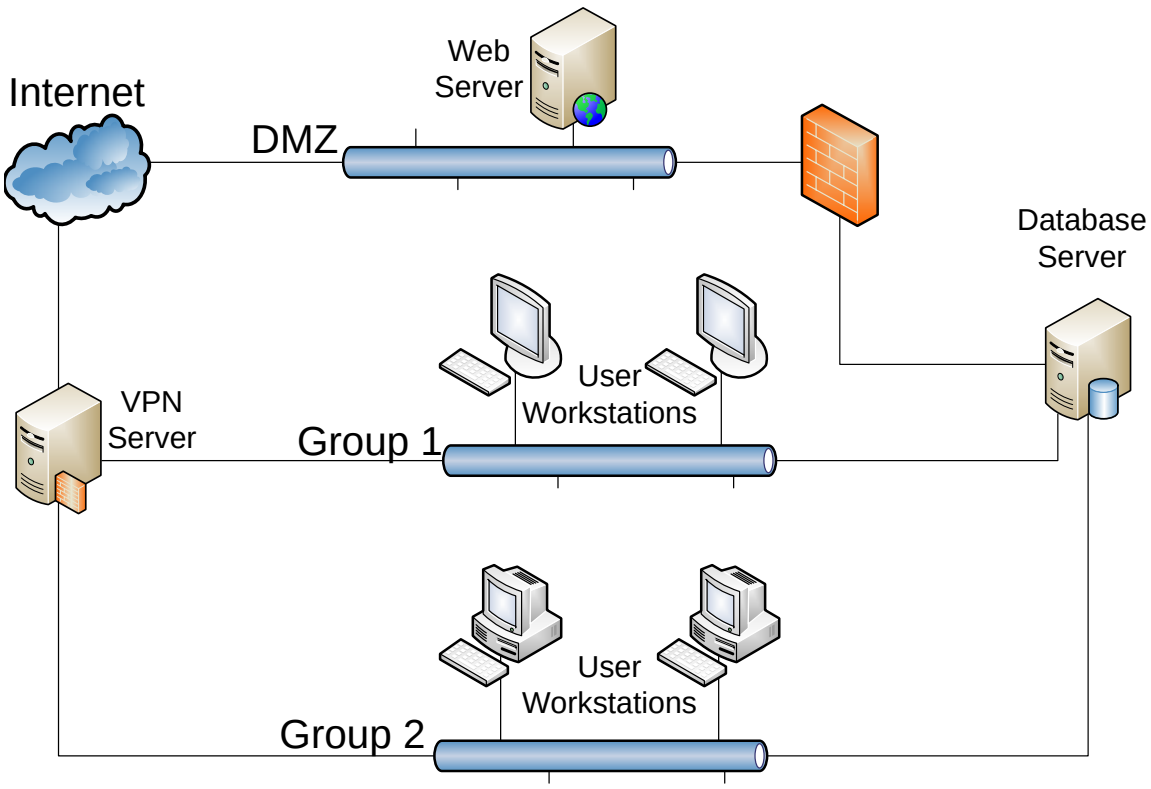


Figure 3.4: *Example for Risk Assessment*

active applications accessed by the web server, Group 1, and Group 2. Any one of these faults are sufficient for an attacker to gain control of the database server.

We will assume that an attacker's goal is to gain code-execution privileges on the database server. The full MulVAL-generated attack graph for this enterprise network model is shown in Figure 3.5; the input model used to generate this attack graph is shown in Figure A.2. The attack graph includes twelve privilege nodes, sixteen attack-step nodes, and twenty-four configuration nodes.

A close examination of the attack graph shown in Figure 3.5 reveals that two of the attack-steps show how an attacker can utilize privileges on the database server to gain privileges on the user workstation groups. Since the database server is the presumed goal, these attack-steps are not helpful in showing how an attacker can reach the goal. Applying

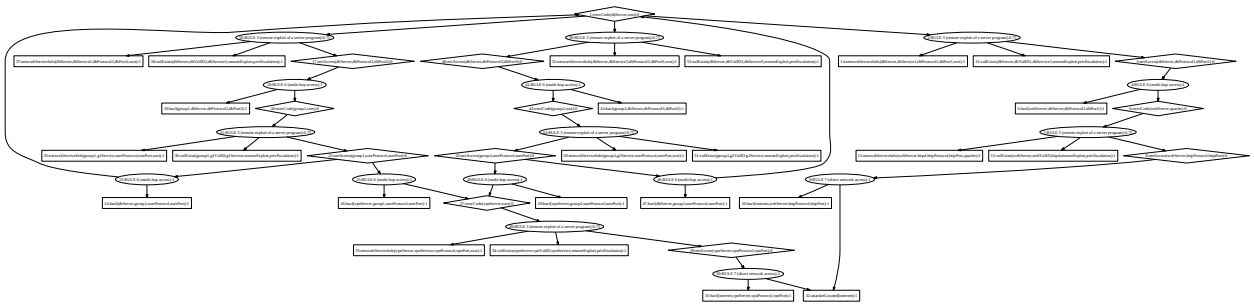


Figure 3.5: *Example – Attack Graph*

domination-based trimming (Section 2.3) removes these "useless" attack steps from the graph; the trimmed attack graph is shown in Figure 3.6. The trimmed attack graph includes twelve privilege nodes, fourteen attack-step nodes, and twenty-two configuration nodes.

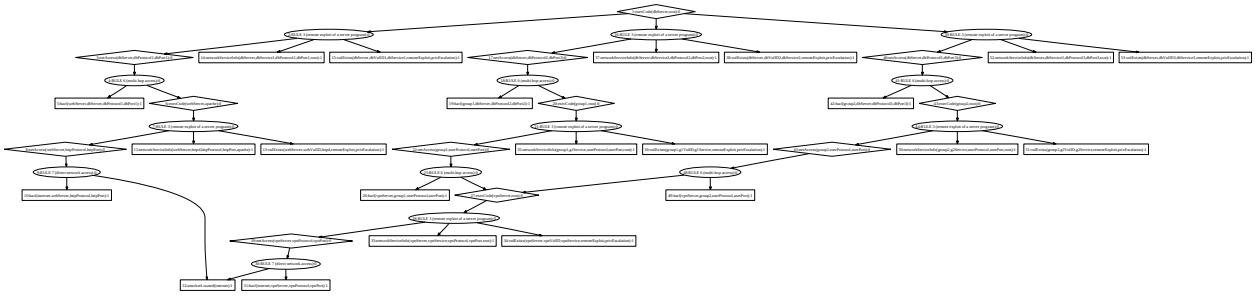


Figure 3.6: *Example – Attack Graph - Trimmed*

For simplicity, we will assume that every vulnerability in the network has a known exploit with 0.75 probability of success. In other words, for each vulnerability in the network, when all necessary preconditions are met, an attacker will succeed in an exploit utilizing that vulnerability with 75% likelihood.

Considering the network topology, a network administrator would likely assume that the web server and VPN server at the most vulnerable, while the groups of user workstations and the database server are more secure, since these are not directly accessible from outside the network. This assumption seems intuitively correct, but a careful and complete evaluation of the probability of exploitation for each machine shows a different state of affairs.

The full risk assessment results for Figure 3.4 are shown in Table 3.6. Here we see

Table 3.6: *Risk assessment calculations for Figure 3.4*

Network host	Probability of exploitation (all metrics equal)
Web server	0.75
VPN server	0.75
User Group 1	0.5625
User Group 2	0.5626
Database server	0.8278

that the database server is the *most* vulnerable machine in the network, due to the several distinct attack paths leading to an attacker gaining privilege on this machine. Because the database server is compromised by a successful exploitation of one or more of these possible attack paths, the chance that an attacker will successfully gain code-execution privileges is higher than might be intuitively expected.

The risk assessment values are correct only for the given component metrics and the current network configuration. Alterations in either of these inputs can have a great effect on the probability of exploitation within the network.

Consider the same scenario, except that the probability of success for the vulnerability on the VPN server is 0.25, much lower than the other probabilities, each 0.75. The change in this component metric value will affect not only the risk assessment for the VPN server, but also for all hosts that have attack paths through the VPN server.

Table 3.7: *Risk assessment calculations for Figure 3.4, lower metric for VPN server*

Network host	Probability of exploitation (all metrics equal)	Probability of exploitation (VPN metric lower)
Web server	0.75	0.75
VPN server	0.75	0.25
User Group 1	0.5625	0.1875
User Group 2	0.5625	0.1875
Database server	0.8278	0.6509

Table 3.7 contains both the original and the updated risk assessment values. As can be seen in this table, the lower risk value for the VPN server affects several attack paths leading to different nodes. The probability that the web server will be exploited remains unchanged, since no attack path leads from the VPN server to the web server. The other probability values have changed, however; the probability values are reduced for the VPN server, both user groups, and even the database server.

Network configuration changes can also have drastic effects on the risk assessment. For example, suppose that the software vulnerability in the database service accessed from the web server is patched, so that there is now no attack path leading from the web server to the database server. The updated (and trimmed) attack path is shown in Figure 3.7.

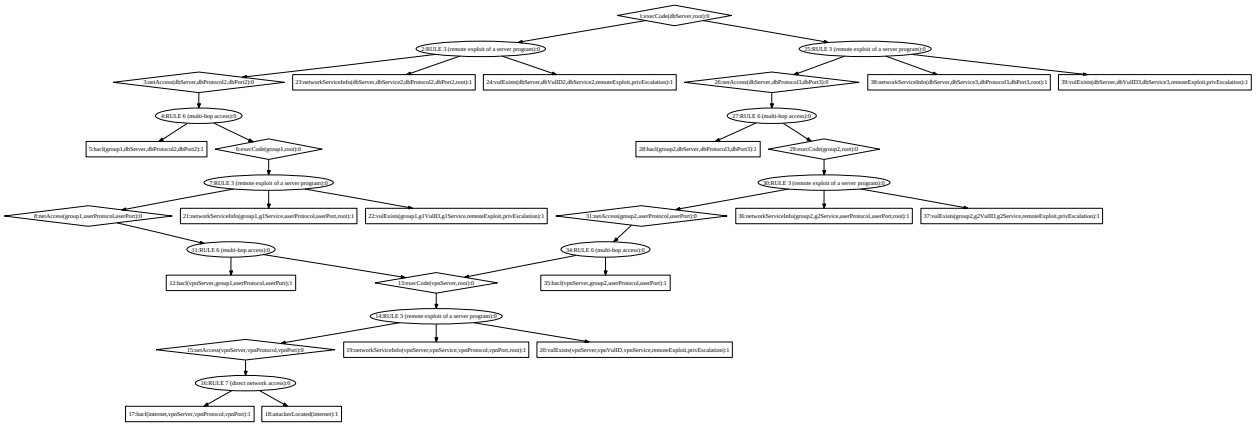


Figure 3.7: *Example – Attack Graph, after vulnerability patched*

Table 3.8: *Risk assessment calculations for Figure 3.4, after vulnerability patched*

Network host	Probability of exploitation (all metrics equal)	Probability of exploitation (VPN metric lower)	Probability of exploitation (patch vulnerability)
Web server	0.75	0.75	0.75
VPN server	0.75	0.25	0.75
User Group 1	0.5625	0.1875	0.5625
User Group 2	0.5625	0.1875	0.5625
Database server	0.8278	0.6509	0.6064

Employing again the universal 0.75 component metric value for all vulnerabilities, the risk assessment results for this scenario will be as shown in Table 3.8, together with previous risk assessment values calculated for this example. As seen in this table, the removal of one of the three attack steps leading to privileges on the database server has noticeably reduced the probability that an attacker might gain these privileges. The other assessment values are unchanged, since this attack step leads only to the database server.

Manual manipulation of the network configuration is arduous and not guaranteed to produce optimal results. In the next chapter, however, I will introduce an approach by which optimal reconfiguration suggestions can be automatically generated, to reduce security in a manner cognizant of the context of the network.

Chapter 4

SAT Solving Approaches to Context-Aware Enterprise Network Security Management

In the endeavor for enterprise network security management, the sheer amount of information available to a human user can be quite overwhelming. Facing this level of complexity, it is often nearly impossible for a human user to fully and accurately identify and address core security problems. Previous chapters have addressed techniques for attack graph simplification and quantitative risk assessment. These techniques can help to alleviate the burden on a human user, but do not provide a practical approach to reconfiguring the network to mitigate the discovered security risks. Every security deficiency is at root an issue of enterprise network configuration management, but it can be extremely difficult to discover which configuration settings are best changed to resolve the issue.

To make matters even more complicated, security issues cannot be identified and treated without considering all of the ramifications of changes made in the enterprise network. Requirements for system usability are often directly at odds with demands for security. If one needed to consider only security requirements, then configuration management would become a trivial problem. In that case, simply shutting down the whole network would easily resolve any security issues. Such heavy-handed solutions are rarely viable, however; basic usability of the network must be maintained.

Configuration changes aimed at correcting security flaws must be made in a context-aware manner, carefully balancing expectations of both usability and security within the enterprise network system. Each enterprise network is assembled and maintained to meet specific needs. These basic functions must be upheld, even while working to reduce security threats. This balance can be extremely fragile and easily lost.

In this chapter, I will introduce an innovative approach to context-aware enterprise network security management. My approach employs well-studied SAT solving techniques for automated network configuration management, balancing usability and security policies to find an optimal configuration solution [15]. Although the underlying problem is NP-hard, modern SAT solvers have been very successful in practice. Some implementations are able to calculate over Boolean formulas with millions of variables and clauses with a run-time measurable in seconds.

This research has been formulated using the MulVAL analysis engine and attack graph toolkit [40, 41], but could easily be applied to any attack graph data set with comparable semantics. The well-formed attack graph structure provides a solid foundation for weighing security concerns against network functionality in search of an optimal reconfiguration.

I have developed a systematic approach to aid a human user in confronting the difficulty of security risk mitigation with minimal network disruption. The MulVAL toolkit accepts the initial, problematic network configuration model and constructs a representative logical proof graph (or logical attack graph) which identifies all potential attack paths by which an attacker might exploit system resources. Because of its consistent structure, the attack graph is easily converted into a system of Boolean formulas in conjunctive normal form that relates configuration settings and attacker actions to the potential effects of successful attack steps. Security and usability requirements and valuations, provided by the human user, are also encoded in conjunctive normal form and appended to the Boolean formulas.

A combined formula is constructed, representing the enterprise network model as well as security and usability policies. This new formula can then be processed by a SAT solver to

discover a satisfying solution, conforming to all necessary relations dictated by the model and upholding the given policies. In this chapter, I will show how several different SAT solving techniques can be used to ensure that an optimally desirable solution is returned and how policy conflicts might be handled.

The SAT solver provides an immediate degree of automation in the search for an optimal network reconfiguration. I will show how a human user can further train the SAT solver over time as to the relative value of various system resources and usages. Working interactively, the human user is able to quickly identify and resolve network security issues without unknowingly lessening the system usability. As the tool is trained and refined, the degree of automation should increase, producing sound and desirable reconfiguration suggestions with minimal human involvement.

I will show the application of two SAT solving techniques:

1. *UnSAT core elimination* examines conflicting subsets of configuration settings and policy requirements that cause the entire formula R to be unsatisfiable. By considering and making decisions within the UnSAT core, the complexity of a reconfiguration dilemma is narrowed to a straightforward choice between specific options. Decision-making can be further reduced by appeal to earlier choices. To achieve this optimization, past policy decisions by the human user are placed in a partial-ordering lattice and are used to further reduce the scope of the decisions presented to the user. This technique will resolve conflicts when no satisfying solution exists.

2. *MinCostSAT* utilizes user-provided discrete cost values, associated with changing a given configuration setting or allowing an attacker a specified amount of network access, to discover a mitigation solution that minimizes the associated cost in terms of both security risk and usability impairment. If the stated cost policies accurately reflect relative values of system resources, the suggested reconfiguration is guaranteed to be optimal. This technique will find an optimal solution when at least one satisfying solution exists.

Together, these techniques can be employed to address security flaws present in a network. My approach considers situations in which no satisfying solution exists, due to policy conflicts, and situations in which an optimal solution must be selected from a collection of satisfying solutions. Furthermore, this approach allows a user to provide instant feedback to the SAT solver, so that constraints on usability, cost of deployment, and potential damage due to successful attacks can all be optimized in a unified framework.

4.1 An Example

Figure 4.1 shows a small and fairly simple enterprise network, which I will use to demonstrate my approach to enterprise network reconfiguration. This example is employed chiefly for its size, which makes it easier to comprehend in full than a larger example.

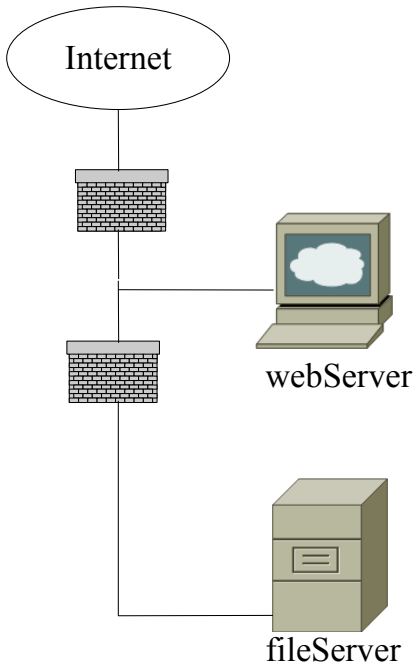


Figure 4.1: *Small Enterprise Network*

Hosts and Accessibility: There are only two hosts within this enterprise network. The web server is directly accessible from the Internet and has access permissions to both the

Internet and the file server. The file server is directly accessible from the web server and has access permissions to both the Internet and the web server.

Vulnerabilities: There is a vulnerability in a service running on the web server that potentially allows for remote exploitation, giving an attacker code-execution privileges on the machine. The file server has two security weaknesses. First, it too has a remote vulnerability in an active service; second, there is a flaw in the NFS service configuration that could potentially allow an attacker to gain unauthorized access to the file server. Either of these faults are sufficient for an attacker to gain code-execution privileges on the file server.

The full MulVAL-generated attack graph for this enterprise network model is shown in Figure 4.2; the input model used to generate this attack graph is shown in Figure A.3. Even for this small example, the attack graph includes five privilege nodes, nine attack-step nodes, and twelve configuration nodes. This size is not beyond the scope of human comprehension, but it is easily seen that a larger network might well produce an attack graph too large to be readily understood by a human user.

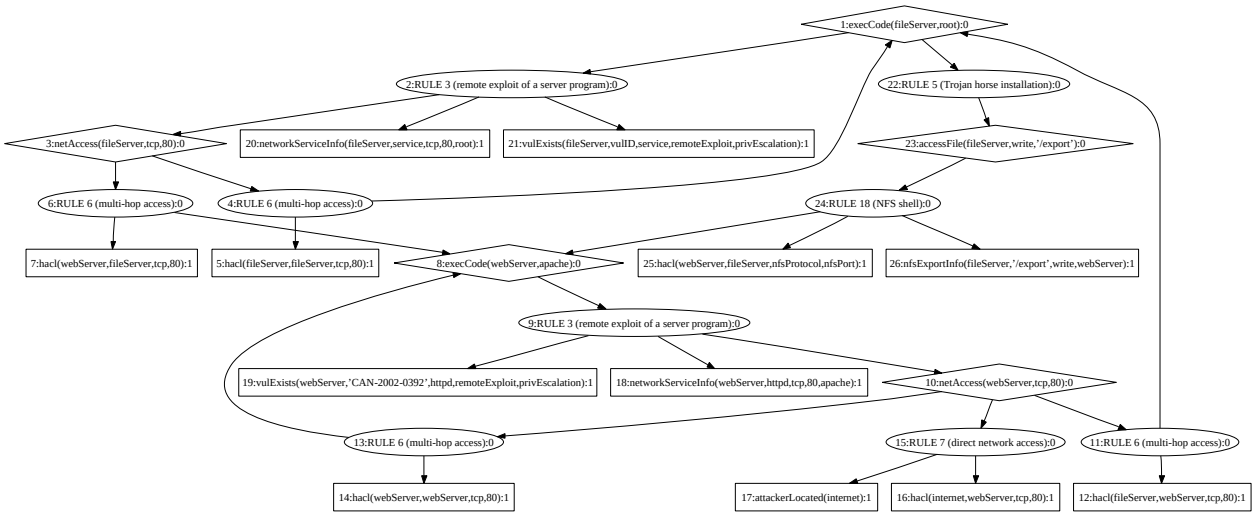
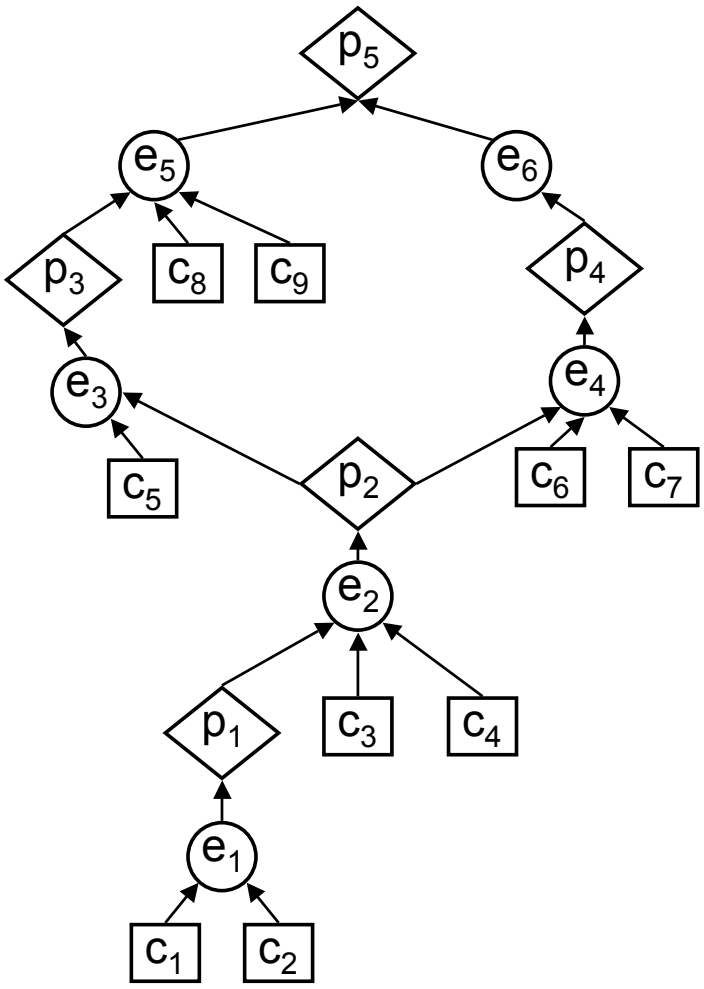


Figure 4.2: *Small Enterprise Network – Attack Graph*

For easier readability, I have constructed a simplified representation of the attack graph, shown in Figure 4.3. This new attack graph image has been simplified by application of the

domination-based trimming technique (Section 2.3), removing all nodes and edges that do not appear in any valid attack path. Also, I have added an identifying label for each node and provided a key for these labels, to better read the graph. I will refer to this attack graph representation as I describe the SAT solving techniques for network reconfiguration.



p_5 : Execute code on file server
p_4 : Access file on file server
p_3 : Network access to file server
p_2 : Execute code on web server
p_1 : Network access to web server

e_6 : Trojan horse installation
e_5 : Remote exploit
e_4 : NFS shell attack
e_3 : Access via network
e_2 : Remote exploit
e_1 : Access via network

c_9 : Vulnerability exists
c_8 : Active service
c_7 : NFS table settings
c_6 : Permit access from web server to file server (NFS)
c_5 : Permit access from web server to file server (TCP)
c_4 : Vulnerability exists
c_3 : Active service (HTTPD)
c_2 : Permit access from Internet to web server
c_1 : Attacker located in Internet

Figure 4.3: *Attack Graph Representation*

4.2 Transforming attack graphs to Boolean formulas

In order to use these SAT solving techniques, we first extract the causality relationships represented in a MulVAL attack graph and express them as a Boolean formula. In this way, we capture logical interrelations between configuration settings and potential network privileges. Figure 4.3 depicts a simplified MulVAL attack graph, showing how an attacker might gain code-execution privileges on the web server (p_2) or the file server (p_5). I will use this graph as an example to explain the transformation to a Boolean formula.

Nodes p_1, p_2, p_3, p_4, p_5 represent network privileges that an attacker could potentially obtain by exploiting vulnerabilities and weaknesses in the current network configuration. Nodes $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9$ represent system configuration settings and other facts about the network, including inter-host accessibility permissions, software services actively running on hosts, and the presence of vulnerabilities in software on network hosts. Nodes $e_1, e_2, e_3, e_4, e_5, e_6$ represent attack steps, each reflecting a causal relationship between the predecessors of the attack-step node and its successor. For example, attack-step node e_1 indicates that if an attacker has access to the Internet (c_1) and access is allowed to the web server from the Internet (c_2), then an attacker will have network access to the web server (p_1). Logically, this relationship is an implication, so that if all of the necessary predecessors are simultaneously true, then the resulting privilege could be obtained by an attacker. This can be expressed in the following formula:

$$c_1 \wedge c_2 \Rightarrow p_1$$

Or, equivalently, we can represent this relationship as a disjunction of negations. Considered in this way, it must be that either one or more of the preconditions is false or that the result is true. This is expressed as follows:

$$\neg c_1 \vee \neg c_2 \vee p_1$$

The other attack step nodes are similarly converted to construct the following formulas:

$$\begin{aligned} e_1 &= \neg c_1 \vee \neg c_2 \vee p_1 \\ e_2 &= \neg p_1 \vee \neg c_3 \vee \neg c_4 \vee p_2 \\ e_3 &= \neg p_2 \vee \neg c_5 \vee p_3 \\ e_4 &= \neg p_2 \vee \neg c_6 \vee \neg c_7 \vee p_4 \\ e_5 &= \neg p_3 \vee \neg c_8 \vee \neg c_9 \vee p_5 \\ e_6 &= \neg p_4 \vee p_5 \end{aligned}$$

With this set of formulas, we can now represent the logical relationships within the entire attack graph data set as a single formula in conjunctive normal form (CNF), as:

$$\begin{aligned} R &= e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6 \\ &= (\neg c_1 \vee \neg c_2 \vee p_1) \wedge (\neg p_1 \vee \neg c_3 \vee \neg c_4 \vee p_2) \wedge (\neg p_2 \vee \neg c_5 \vee p_3) \wedge \\ &\quad (\neg p_2 \vee \neg c_6 \vee \neg c_7 \vee p_4) \wedge (\neg p_3 \vee \neg c_8 \vee \neg c_9 \vee p_5) \wedge (\neg p_4 \vee p_5) \end{aligned}$$

Then R is a Boolean formula in CNF whose size is linear in the size of the attack graph.[1]

Clearly, a new configuration that eliminates the node representing an attacker's presence (c_1) is unrealistic. The assumed threat of an attacker cannot simply be "disabled" like a system configuration setting, so variable c_1 must be true. For this example, we will assume that that the vulnerability on the web server does not currently have an available patch, so variable c_4 must be true. Let us redefine $R = e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6 \wedge c_1 \wedge c_4$. Formula R now encompasses not only the causality relationships between configuration data and potential attacker privileges, but also all immutable data within the model. Any realistic network configuration must satisfy the constraints in R.

Beyond simply conforming to logical constraints, it is desirable to find a new network configuration that considers security concerns. A security policy must be created within the context of each individual network, considering what is expected for that network. For example, an administrator desiring absolute security, so that an attacker cannot gain any

[1] A MulVAL attack graph's size is quadratic in the size of the number of hosts in the network [40], so the CNF representation would also be quadratic in the size of the network.

privileges within the network, could define the security policy as:

$$S = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5$$

Some privilege variables, however, do not represent real exploitations within the network. For example, privilege p_1 represents network access to the web server. This privilege is necessary not only for attackers seeking to enter the network, but for anyone wishing to view data on the web server. A more realistic security policy might seek only to prevent an attacker from gaining code-execution privileges within the network. This security policy could be defined as:

$$S = \neg p_2 \wedge \neg p_5$$

Let $C = R \wedge S$ be a formula representing a valid and fully secure network configuration, in which an attacker cannot gain any network privileges. Because privilege variables are the results of logical implications based on the attack graph, a privilege variable cannot be negated without removing some or all of the preconditions currently enabling it. In other words, to prevent an attacker from gaining privileges within the network, some of the configuration settings that currently enable these privileges to be obtained must be changed.

In the current network configuration, all configuration settings $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9$ are assigned the truth value T. Thus, privileges p_1, p_2, p_3, p_4, p_5 currently must also be assigned T, in order to satisfy R. If we desire that any privilege be false (assigned F) while satisfying the constraints of R, at least some of $c_2, c_4, c_5, c_6, c_7, c_8, c_9$ must be assigned F. Negating a configuration variable represents a change in the network configuration so that this setting is no longer active. For example, since c_4 represents the existence of a software vulnerability on an active service running on the web server, the negation of that node would mean patching or otherwise removing the vulnerability; c_5 represents a reachability relationship between the web server and the file server, so disabling (or negating) that node would mean changing the access permissions between these hosts. A satisfying solution to C would be the negation of a sufficient set of configuration settings such that an attacker is prevented from gaining either p_2 or p_5.

Usability must also be considered, however, in order that the network can be used to fulfill its intended purpose. A usability policy, comparable to the security policy, can be created to specify which functionalities should not be removed. In the Boolean formula, the usability policy will contain variable assignments in which configuration variables are forced to be true. For example, an administrator desiring to maintain the usefulness of the web server would require preservation of general accessibility to the web server as well as the necessary software services. In this case, the usability policy could be defined as:

$$U = c_2 \wedge c_3$$

A desirable network configuration must satisfy $C = R \wedge S \wedge U$, so that both security and usability of the network are considered. In seeking an optimal configuration for an enterprise network system, our goal is not necessarily to eliminate all security risks. Instead, we would seek to mitigate the potential cost in damage from a successful attack while also limiting the losses arising from decreased network usability. If the cost (in terms of business functionality) of completely securing the network against attackers is much higher than the potential losses from attacks, it may be a better solution simply to acknowledge and tolerate some possibility that an attacker might obtain minor privileges on the network.

A reliable and automated approach is needed to address concerns in real-size enterprise networks. The interplay between security and usability is often quite complicated, well beyond the capability of human discernment. I will show two applications of SAT solving techniques - iterative UnSAT Core elimination and MinCostSAT solving - that can be used to resolve network misconfigurations in a context-aware manner, balancing the dual costs of potential damage and loss of usefulness.

4.3 Iterative UnSAT Core Elimination

A satisfying solution is not guaranteed to exist. Conflicts between the need for network usability and the desire for network security may eliminate any possible solution. In this

event, the iterative UnSAT Core elimination technique can be employed for the identification and resolution of such conflicts.

Definition 11. *An UnSAT core is a subset of the original CNF clauses that is, in itself, unsatisfiable [13].*

When a SAT solver finds a set of clauses to be unsatisfiable, a byproduct of this decision is the UnSAT core, a subset of clauses that is unsatisfiable. Logically, given an unsatisfiable Boolean formula C, the UnSAT core $\mu = \mu_1, \mu_2, \ldots, \mu_m \subseteq C$ where μ is itself unsatisfiable. In other words, C will remain unsatisfiable so long as μ remains unchanged. For the purposes of testing, I have made use of the zChaff SAT solver [25], generating the UnSAT core using the zChaff zcore function.

Iterative UnSAT core elimination will be applied only when no satisfying solution can be found. C typically becomes unsatisfiable when one aspect of the overall system policy (usability) demands that a certain variable be true while another aspect (security) necessitates that the same variable be false. This approach recognizes the conflict and seeks to restore an acceptable balance between security and usability; when one or more satisfying solutions exists, the MinCostSAT technique (Section 4.4) can be applied to discover a "best" solution.

The UnSAT core $\mu = R_u \wedge S_u \wedge U_u$, where $R_u \subseteq R$, $S_u \subseteq S$, and $U_u \subseteq U$. Each UnSAT core, then, will have some subset of derivation clauses based on the MulVAL logic rules as well as the security and usability policies specified by the user. Together, these subsets comprise an unsatisfiable instance.

For example, let $C = R \ \wedge \ S \ \wedge \ U$, where:

$$\begin{aligned} R = &\ e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6 \wedge c_1 \wedge c_4 \\ S = &\ \neg p_2 \wedge \neg p_5 \\ U = &\ c_2 \wedge c_3 \end{aligned}$$

In this example, an UnSAT core would be identified as $\mu = (e_1 \wedge e_2 \wedge c_1 \wedge c_4) \wedge (\neg p_2) \wedge (c_2 \wedge c_3)$. The security policy dictates that the attacker will not be able to gain code-execution privileges on the web server ($\neg p_2$); the usability policy dictates that the web server be

accessible from the Internet and that the httpd service be maintained on the web server. Based on the logical implications in R, together with the upholding of c_1 and c_4, it is easily seen that these configuration settings (c_1, c_2, c_3, c_4) are precisely the settings that can enable an attacker to obtain privileges on the web server; this is the conflict identified in μ.

Obviously, the user cannot change the logical foundations of the MulVAl derivation rules (e_1, e_2) or capriciously eliminate the threat of an attacker (c_1) or a vulnerability for which a patch does not exist (c_4), so the UnSAT core μ cannot be resolved by altering any of R_u. To make the necessary reconfiguration decision, this approach will request from the user an immediate decision of the relative importance among the elements of $S_u = \neg p_2$ and $U_u = c_2 \wedge c_3$.

The user would be prompted to judge the relative importance of the conflicting policies; the less important policy element would be "relaxed" (removed from C) so that its value is no longer forced true or false. Relaxing (or allowing to falsify) a policy does not necessarily mean that all of the elements will be disabled; instead, only the configuration variables that conflict with the remaining elements will be disabled. The other policy variables will remain unchanged (forced true or false).

It is possible that multiple UnSAT cores exist in a single Boolean formula. In my approach, each UnSAT core is iteratively presented, prompting the user to decide for each core which policy constraint should be relaxed. In this way, a satisfiable configuration solution can eventually be reached.

So long as security and usability policies do not conflict, the user is not asked to decide the relative values of any two policies. These decisions are only faced when an actual conflict has arisen, so the human user makes only necessary choices about system resource valuations.

In utilizing this approach to enterprise network security management, the human user is not expected to fully comprehend the ramifications and effects, both positive and negative, of all possible changes in network configuration, but only to make decisions on the

immediate relative values of specific instances of usability and security. In this way, we reduce an extremely complex problem to one of more manageable proportions, automating the enforcement of both security and usability policies while introducing a method by which conflicts can be quickly and verifiably resolved.

4.3.1 Decision Reduction through Appeal to a Partial Ordering

To further reduce the breadth of decisions faced by the human user, I will show that a partial ordering can be established to arbitrate the relative priorities between specific variables appearing in conflicting policies.

Definition 12. *A partial order is a binary relationship $\leq$ over a set P which is reflexive, antisymmetric, and transitive.*

We will employ a partial ordering (denoted by "$\leq$") over the set of variables contained in C. Thus, for any variables $\{a, b, c\}$, it is true that $a \leq a$ (reflexivity), if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry), and if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity). This relationship reflects the relative significance between two variables.

Each time the human user is presented with the causes of an unsatisfiable conflict and selects one or more of those constraints to be relaxed, this decision is recorded as a partial ordering between the relaxed variable and each of the other variables in the original conflict. In ordering the formula variables, this technique assumes that the constraints that the user allows to be relaxed have a lower priority than any clauses that were not relaxed; this ordering is duly recorded. For some UnSAT core, let $\mu = x \wedge y \wedge z$, and assume that the user, when prompted, elects to relax constraint x. Then $x \leq y$ and $x \leq z$. It is important to note that an ordering is not established at this time between y and z; in this decision, no evidences have been provided for the relative prioritization between these variables.

In future decisions, then, in which two constraints appear for which an ordering is already known, the constraint with higher priority will not be offered to the user as a possibility for relaxation. Assume that at some future point, an UnSAT core is identified, where

$\mu = v \wedge x \wedge y$. The ordering $x \leq y$ is already known, based on a previous decision, so before the user is presented with a decision, constraint x is first relaxed. If the formula remains unsatisfiable, then the user is presented only with the decision $v \vee y$. In this way, conflicts are reduced to comparisons between configuration settings or policy requirements for which relative priorities are not known. Over time, the size and number of decisions required by the human user will decrease as more conflicts can be resolved based on the knowledge of past decisions. There may also be some opportunity for the application of a learning algorithm to this problem, so that conflicts can be resolved without human intervention on the basis of similarity to past ordering decisions, even when the conflicting variables have not been definitely ordered.

4.4 MinCostSAT

Conflicts within the policies can be settled through iterative UnSAT core elimination. Once all conflicts have been resolved, one or more satisfying solutions will exist for the Boolean formula C. MinCostSAT is another SAT solving technique, one that can be used to discover an optimal (minimal cost) solution. By applying this technique, the suggested network reconfiguration will not only conform to the stated policies S and U, but will do so in the most desirable manner possible, based on discrete "costs" or valuations provided for each variable in formula C.

MinCostSAT is a Boolean satisfiability problem which seeks the satisfying assignment with a minimal associated cost [22]. Mathematically, given a Boolean formula C with n variables $x_1, x_2, \ldots, x_n$ with associated costs $c_i \geq 0$ (so that each x_i has exactly one corresponding cost c_i), MinCostSAT finds a variable assignment set $X \in \{0,1\}^n$ such that X is a satisfying solution for C and the total cost C is minimized, for

$$C = \sum_{i=1}^{n} c_i x_i$$

where $x_i \in \{0,1\}$ (corresponding to truth values { F, T }).

MinCostSAT has been thoroughly studied by the SAT solving community [8, 14, 22, 27]. For testing purposes, I have used the MinCostChaff solver [14], a MinCostSAT solver implementation based on the zChaff SAT solver [25].

Within the MinCostSAT problem, the optimal solution satisfies the Boolean formula while minimizing the associated cost for variables that are enabled (assigned T). This cost assignment matches the semantics for privilege variables, for which a T assignment implies that an attacker might potentially gain that privilege and thereby cause some damage within the system.

We desire exactly the opposite effect in dealing with configuration variables. The "cost" to the system occurs when a configuration setting must be changed, altering the overall usability of the network system and possibly affecting legitimate users. For configuration variables, then, the associated cost is incurred when the variable is disabled (assigned F).

To correctly model this difference in cost exposure between privilege and configuration variables, we will transform the formula to use the negation of a Boolean variable to represent each configuration node in the attack graph. By this means, when the variable is assigned T, the corresponding change in the system configuration would be the disabling of the corresponding configuration setting, which should incur the associated cost.

Returning again to the example in Figure 4.3, the new formula derived from the attack graph will be:

$$\tilde{R} = \widetilde{e_1} \wedge \widetilde{e_2} \wedge \widetilde{e_3} \wedge \widetilde{e_4} \wedge \widetilde{e_5} \wedge \widetilde{e_6} \wedge \widetilde{c_1}$$

where

$$\begin{aligned}
\widetilde{e_1} &= \widetilde{c_1} \vee \widetilde{c_2} \vee p_1 \\
\widetilde{e_2} &= \neg p_1 \vee \widetilde{c_3} \vee \widetilde{c_4} \vee p_2 \\
\widetilde{e_3} &= \neg p_2 \vee \widetilde{c_5} \vee p_3 \\
\widetilde{e_4} &= \neg p_2 \vee \widetilde{c_6} \vee \widetilde{c_7} \vee p_4 \\
\widetilde{e_5} &= \neg p_3 \vee \widetilde{c_8} \vee \widetilde{c_9} \vee p_5 \\
\widetilde{e_6} &= \neg p_4 \vee p_5
\end{aligned}$$

Here $\widetilde{c_1}, \widetilde{c_2}, \widetilde{c_3}, \widetilde{c_4}, \widetilde{c_5}, \widetilde{c_6}, \widetilde{c_7}, \widetilde{c_8}, \widetilde{c_9}$ are the new Boolean variables for the configuration nodes, whose T assignment will mean that the node is disabled.

We can now use this new formula to construct $\tilde{C} = \tilde{R} \wedge S \wedge U$. Given $\tilde{C}$ as input, the MinCostSAT solution would be, in fact, the optimal satisfying solution for the enterprise network system's reconfiguration, assuming that the costs are correctly assigned for all of the variables.

In $\tilde{C}$, the policies S and U are strictly enforced; in this way, no satisfying solution will ever be discovered in which any policy element is contradicted. Although I am allowing these variables within the formula to be forced absolutely true or false, it is important to note that this is not truly necessary within the MinCostSAT problem.

A comparable effect (*e.g.*, the falsification of p_2, p_5) can be caused by assigning an unrealistically high cost to the variable, so that there is no minimal cost solution in which the policy is countered (*e.g.*, p_2, p_5 are assigned T). I believe that it is the case, however, that human users will desire that certain configurations are *always* maintained or that certain privileges are *never* obtainable by attackers and so I have chosen to allow these variables to be absolutely assigned T or F within the policies. This allowance ensures that no reconfiguration suggestion will ever reverse these settings.

It is important to note that the full power of the MinCostSAT solver is employed when there is as little forcing of variables as possible. Rather than reliance upon large and elaborate policies, the interrelationship of the associated variable costs should be allowed to dictate the optimal reconfiguration.

4.4.1 User queries

With the expressiveness of the Boolean formulas and the inherent power of a SAT solver, a system administrator is able to perform useful queries to explore security within the enterprise network.

For example, an administrator might ask, "What is the most cost efficient way to reconfigure my network system if I want to guarantee that the database server will not be compromised by an attacker?" To find this configuration, the administrator would force the

Boolean variable x that corresponds to the privilege `execCode(databaseServer, root)` to be false, by appending $\neg x$ to the system formula, so $C = R \wedge \neg x$. The new formula C comprises the input for the MinCostSAT solver.

As another example, an administrator might ask, "Can I make the file server secure while still allowing the web server to be accessed from the Internet?" To find this configuration, the administrator might force false the variable corresponding to the file server privilege and force true the configuration settings corresponding to web server reachability from the Internet. These and similar questions are easily answered by employing the power of the MinCostSAT solver.

In this way, security and usability constraints in an administrator's query can be straightforwardly specified in Datalog, similar to the rest of the MulVAL rules, and automatically transformed into additional clauses in the Boolean formula to be solved by the MinCostSAT solver. This kind of constraint can be expressed as a part of the configuration policy.

4.4.2 Cost Policies

The accuracy and dependability of the reconfiguration suggestions produced by the MinCostSAT solver are entirely reliant upon the precision of the given cost assignments. If the cost parameters are realistic, then the reconfiguration suggestions will be fully in line with what a human examiner would manually determine after long and hard work. The development of these cost policies, then, is of paramount importance to the useful application of MinCostSAT solving for network reconfiguration.

There is much latitude in the development of cost policies for use with the MinCostSAT solver. Indeed, an administrator can construct cost policies based on any self-consistent criteria that can be used for measuring both potential damages due to security risk and effects to business functionality. For example, the cost policies might try to relate to real dollars. The cost for a privilege variable would correspond to the monetary losses that the network owner might expect to incur if an attacker obtained that privilege. The cost for

a configuration variable would correspond to the cost in man-hours for the time needed to patch a software vulnerability or to train users to perform their duties in an alternate manner, after the network configuration has been changed.

Cost policies can be manually constructed and maintained, or they can be automatically built and updated. I believe that risk assessment values (produced by the algorithms presented in Chapter 3) can serve as the basis for costs associated with network privileges. There are a number of ways in which this transformation can be made. For example, to generate costs for an attacker gaining code-execution privileges on a machine, an administrator might rank each network machine on a scale of one to three, where one indicates a machine with no sensitive data, two indicates a machine with some sensitive data, and three indicates a machine with highly sensitive or proprietary data. This rank value, $rank$, could be multiplied with the risk assessment, $risk$, and a baseline value $base$ (*e.g.*, 100). Using this system, a configuration that allows an attacker to gain code-execution privileges on a machine with $rank = 2$ and $risk = 0.65$ would incur a cost of $rank \cdot risk \cdot base = 2 \cdot 0.65 \cdot 100 = 130$. A machine with $rank = 3$ and $risk = 0.5$ would incur a cost of $3 \cdot 0.5 \cdot 100 = 150$. The ranking scale could easily be expanded beyond three values or the baseline value could be changed. This serves only as a suggestion of how the risk assessment values might be utilized in MinCostSAT cost policies. It is important to note that the actual values assigned to each variable are largely irrelevant, so long as the values are correct in proportion to one another.

4.4.3 Scalability

In order to test the scalability of my MinCostSAT-based approach for network reconfiguration, I constructed simulated enterprise networks with two different sizes:

I: 100 host machines, evenly divided in 10 subnets

II: 250 host machines, evenly divided in 25 subnets

I also conducted tests using two different cost functions:

A: All variables were assigned an equal cost. The principal effect of this cost policy would be simply to minimize the sum of the number of configuration changes made plus the number of privileges possibly obtainable by an attacker. This policy is quite simplistic, minimizing the number of changes without considering the type.

B: Clauses representing execute privileges on a machine were assigned costs based on the machines position in the network relative to the goal. The primary effect of this cost policy would be to have increasingly high costs for penetrations deeper into the network. The costs for blocking network access to hosts or disabling network services were significant. All other changes had equal, low cost.

The tests were conducted on a Linux machine with Opteron Dual-Core 2214 2.2 GHz CPU, with 16GB memory, and running gentoo Linux with kernel version 2.6.18-hardened-r6.

Table 4.1: *Scalability test results - SAT solving*

Size	Configuration	Number of variables	Number of clauses	Run-time (sec)
I	A	11,853	12,053	0.11
I	B	11,853	12,053	0.21
II	A	70,803	72,553	3.03
II	B	70,803	72,553	6.49

The simulated networks on which I performed the above tests were certainly not representative of realistic enterprise network settings. However, I believe that the performance in the tests indicates that SAT solver implementations should be powerful enough to handle the configuration management problem based on attack graphs. Related work has also shown the efficient application of SAT solving techniques for configuration management. The ConfigAssure system [33] utilizes the constraint solvers provided in Alloy to identify network component configurations that satisfy the given model.

4.5 Discussion

MinCostSAT requires cost functions that assign numeric values to every configuration setting and security or usability policy; these cost valuations may be difficult to assign fairly to all resources. Once decided, however, a minimum-cost configuration can be determined at any time, simply by comparing the relative costs of potential security breaches to usability requirements.

One difficulty with MinCostSAT, in application to network configuration decisions, is determining the basis for the cost functions. Although any metric can be utilized, I suggest an approach wherein a dollar amount is assigned, indicating, for example, the potential liability if an attacker gains access to a specific server or the cost of applying a fix to a known vulnerability.

Referring to the UnSAT core for conflict resolution, however, requires no up-front cost assignments, relying instead on immediate decisions made only when a conflict is discovered. I believe also that introducing a form of machine learning may assist the reasoning engine in ordering priorities of clauses that have not been previously directly compared, to further reduce the decisions a user faces. The nature of the UnSAT core, however, necessitates that a human user carefully examine all conflicts and make decisions as needed for all occurring conflicts.

Both approaches to configuration resolution, MinCostSAT and the UnSAT core, carry advantages and disadvantages. The optimal approach may be a combination of the two SAT techniques, in which a user would utilize the UnSAT core to prompt user decisions about relative values between constraints and then feed this formula into MinCostSAT, using a constant cost function to find a satisfiable solution that considers user decisions about valuations and disables the minimum configuration settings to meet this goal, as determined by the cost assignments.

An automated reasoning system, such as described here, can compare security and usability policies, identify conflicts, prompt for human prioritization between the two, and

alter the necessary underlying configuration settings needed to resolve this conflict without introducing new conflicts into the system.

Related work by Wang, *et al.* [55] also seeks a minimal-cost reconfiguration solution for securing an enterprise network against intrusion. Their work utilizes a graph traversal algorithm to construct a Boolean formula representing configuration changes that are sufficient to prevent the goal privilege from being obtained. This formula is converted to disjunctive normal form (DNF), so that each clause represents a reconfiguration option, and a minimum-cost option is identified from these choices. The DNF formula is potentially exponential in the number of configuration nodes in the original attack graph, so this approach has an exponential complexity, in the worst case. Furthermore, their approach seeks only a solution sufficient to prevent an attacker's gaining of the goal privilege. Their approach does not allow that hardening against an attacker's obtainment of the goal privilege may carry lower cost than the cost incurred by changing the configuration to prevent this access. My work provides a more defensible approach to network configuration management, seeking an overall minimum cost solution considering all configuration settings and privileges.

Dewri, *et al.* [10] recognize that security management should seek a minimal cost solution for hardening against all privileges potentially obtainable by an attacker. They formulate the problem as a multi-objective optimization and apply a genetic algorithm to solve it. Genetic algorithms converge toward an optimal solution, but are not guaranteed to produce the optimal solution; MinCostSAT, however, will produce a provably optimal solution.

Narain, *et al.* [34] proposed the use of the UNSAT core for the reconciliation of security and functionality policies in the ConfigAssure system for a more general logic (*i.e.,* an arithmetic quantifier-free form). ConfigAssure uses the Prolog language, allowing for a greater possible breadth of policies to be specified. However, this work requires manual specification of the configuration parameters that can be changed to reconcile policy conflicts, while my work utilizes the MulVAL attack graph semantics to automatically identify these restraints. An interesting opportunity for future research would be a combination of these separate

approaches (MulVal and ConfigAssure) into a single system

4.6 Implementation and Testing

In this section, I will show the full application of these techniques to the sample graph shown in Figure 4.3. I will explain first how the various system policies were constructed for this test; next, I will demonstrate the usefulness of the iterative UnSAT core elimination approach; and, finally, I will present the MinCostSAT approach for producing network reconfiguration suggestions.

4.6.1 Policy Construction

Three types of system policies have been described in this chapter:

- *Security policy* - Specifies network privileges that should never be acquired by an attacker. It is likely that this policy does not demand full security, but includes only privileges that truly must *not* be held by an attacker.
- *Usability policy* - Specifies network configuration settings that should not be altered. Similar to the security policy, it is important that this policy does not insist upon all current network permissions, but includes only the configurations that are fundamentally necessary to the continued usefulness of the network.
- *Cost policy* - Assigns discrete cost values to each variables in the formula C. For privilege variables, the value is determined by the cost incurred if an attacker gains that privilege. For configuration variables, the value is determined by the cost incurred by disabling that configuration setting, thereby altering or reducing the usability of the network.

Let security policy $S = \neg p_2 \wedge \neg p_5$. This policy states that an attacker should not acquire code-execution privileges on either the web server or the file server. The other privilege

variables - p_1, p_3, p_4 - represent basic network accessibility privileges and are unimportant so long as the attacker cannot gain control of either host machine.

Let usability policy $U = U_1 \wedge U_2$, where $U_1 = c_5 \wedge c_8$ and $U_2 = c_2 \wedge c_3$. This policy states that the accessibility and active services should be maintained in the current configuration for both the web server and the file server.

The cost policy that will be used in this example is defined in Table 4.2. Although some of these variables are currently forced true or false in the security and usability policies, costs should be assigned almost all variables anyway, in case policy conflicts force relaxations of existing constraints. The exceptions, of course, are variables that will never be relaxed, such as c_4, a vulnerability for which no patch currently exists.

Table 4.2: *Cost policy for Figure 4.3*

Privilege	**Variables**	**Cost**
Execute code on file server	p_5	1000
Execute code on web server	p_2	50
File access on file server	p_4	50
Host network access	p_1, p_3	5
Configuration Setting	**Variables**	**Cost**
Inter-host reachability	c_2, c_5, c_6	100
Active service on web server	c_3	100
Active service on file server	c_8	50
Vulnerability on file server	c_9	20
NFS table on file server	c_7	10

These three policies - security, usability, and cost - will form the foundation for all further assessment of the network reconfiguration and for determining the optimal reconfiguration. It is important that the cost policy, especially, accurately portrays the relative costs incurred by a successful attack or by preventive changes in network configuration. Reconfiguration suggestions are produced on the basis of the cost policy, so setting costs too high or too low will produce inaccurate suggestions.

4.6.2 Application of Iterative UnSAT Core Elimination

In applying the UnSAT core approach to this example, I began with an empty partial-ordering lattice. An empty lattice is the initial point for a network when this technique is first applied, so there is no historical data. At this point, my security policy states that an attacker should not gain executive privileges on either the web server or file server, while my usability policy states that accessibility and active services should not be changed on either the web server or file server. The total representative formula is $C = R \wedge S \wedge U$, where:

$$\begin{aligned} R = &\quad e_1 \wedge e_2 \wedge e_3 \wedge e_4 \wedge e_5 \wedge e_6 \wedge c_1 \wedge c_4 \\ S = &\quad \neg p_2 \wedge \neg p_5 \\ U = &\quad (c_2 \wedge c_3) \wedge (c_5 \wedge c_8) \end{aligned}$$

To find an optimal network reconfiguration, we must first establish that there is at least one satisfying solution for the formula C. We will submit formula C to the SAT solver to see if an UnSAT core is generated or if a solution can be found.

Not surprisingly, we will immediately encounter an UnSAT core. We are prompted to choose from among the following constraints, some of which must be relaxed:

1. $[c_2]$ Network reachability allowed from Internet to web server
2. $[c_3]$ Active service running on web server
3. $[\neg p_2]$ Attacker should not be able to acquire executive privileges on web server

It is easily seen that e_1, e_2, c_1, c_4 in R specify that if c_2, c_3 are forced to be true, then p_2 must necessarily be obtainable by an attacker. One of these policy requirements must be relaxed, so that this unsatisfiability is eliminated. The key question, then, is whether it is more acceptable to lose the usefulness of the web server (by relaxing c_2 or c_3) or to acknowledge some risk that the web server may be compromised by an attacker (by relaxing $\neg p_2$).

Since the web server is the public portal to information about the organization, it is unlikely that it would be acceptable to lose this resource. We may also assume that traffic

to the web server is likely to be closely watched by an administrator, so any attack on the web server will likely be quickly observed and countered. Bearing this in mind, let us suppose that we decide to accept some risk to the web server in favor of retaining the benefits gained by hosting a website. We choose, then, to relax $\neg p_2$, so now $S = \neg p_5$. In a new configuration complying with the updated security policy, we tolerate some possibility that an attacker will gain privileges on the web server, but will remain confident that the file server is secured against outside attack.

To reduce the possibility of future decisions, we will store the results of this decision in a partial ordering lattice. We now know that $c_2 > \neg p_2$ and $c_3 > \neg p_2$. Based on this decision, we have not learned anything about the relative valuations between c_2, c_3, c_4, but only their individual relationships to $\neg p_2$. If, in the future, an UnSAT core is identified in which either c_2 or c_3 appear together with $\neg p_2$, then the configuration setting will not be shown as an option, since this historical data indicates that $\neg p_2$ will be relaxed instead. Other elements of the UnSAT core for which partial ordering is not known will still be presented to the user for judgement.

Formula C will be updated with the new S value and again submitted to the SAT solver to check for a satisfying solution. Running the SAT solver with the updated policy constraints, we can now find a satisfying solution, a system configuration that adheres to the remaining policy constraints. The usefulness of the iterative UnSAT core elimination approach lies in the resolution of internal policy conflicts. Once we have determined that a satisfying solution exists, we must apply the MinCostSAT approach to determine which of the possible solutions is optimal, according to the cost policy.

4.6.3 Application of MinCostSAT approach

We will continue with our examination of the example shown in Figure 4.3 by applying the MinCostSAT solving technique described in Section 4.4. Having already utilized the iterative UnSAT core elimination approach, we know that there is now at least one satis-

fying solution for the combined formula C. MinCostSAT solving can discover the optimal satisfying solution.

In order to better test the MinCostSAT solving approach, let $C = R \wedge S$, where $S = \neg p_5$. Here, we have reduced the C formula to the logical relationships between configuration and privilege variables (R) together with a minimal security policy enforcement, that an attacker never gain control over the file server (S). Although the MinCostSAT can produce an optimal configuration even when constrained by variables forced true or false, the comparative power of this technique is more fully employed when the decisions rely not on forced values but on costs. In this example, I have removed the usability policy from C, relying on MinCostSAT to utilize the cost specifications for configuration variables to identify an optimal solution.

As partial input for the MinCostSAT solver, we will use the cost policy specified in Table 4.2. This policy specifies the costs incurred by a change in any of the configuration settings or by allowing an attacker to gain any of the privileges.

The MinCostSAT solver will find an optimal configuration based on these cost valuations. If in applying this technique, the reconfiguration suggestions do not seem to be in line with expectations, the user should carefully review the assigned cost values. It may be that the current cost values are inaccurate and a correction will produce an expected result; if the costs do not accurately reflect the true value of the configuration settings or the true cost of the privileges, then the suggested reconfiguration may not be accurate within the real-world context of the enterprise network. On the other hand, it may be that the values are accurate but the subtle interplay of the attack graph model returns a correct but counter-intuitive solution. In larger networks, especially, an intuitive assessment might easily omit important considerations and thus, in the absence of an automated calculation, lead to the usage of an insecure configuration.

Based on the currently active security and usability policies, the MinCostSAT solver will suggest the following configuration changes: $\neg c_7 \wedge \neg c_9$. The minimal cost solution allows

some risk that an attacker might obtain code-execution privileges on the web server (p_1, p_2) and even some network access to the file server (p_3), but changes the configuration - by patching or otherwise removing the vulnerability on the file server (c_9) and adjusting the NFS table settings (c_9) - to ensure that the attacker never gains control of the file server. Looking again at the cost policy shown in Table 4.2, it can be seen that this reconfiguration suggestion entails a cost of 90 ($5 + 5 + 50 + 10 + 20$). This solution is minimum cost and, if the evaluations are accurate, optimal within the context of the network.

Other solutions exist, of course, but every other possible solution results in a higher cost. One such solution might change the reachability permissions between the web server and file server (c_5) and change the NFS table settings (c_7). In this solution, the attacker is still allowed some privileges on the web server, but is prevented from gaining even network access to the file server (privilege p_3). This solution thus restricts an attacker's potential privilege set further, but we must consider the total effect. Changing the reachability permissions represented by c_5 could have large business impact on the network, lessening its usability; this incurs a high cost, as reflected in the cost policy. The total cost of this solution is 165 ($5 + 50 + 100 + 10$), higher than the optimal solution already shown.

It is important to note that assigning different costs can easily produce different suggestions. The cost for changing a specific configuration parameter and the cost caused by a potential attacker privilege will vary from one organization to another. There is no one-size-fits-all cost function suitable for all enterprise systems, and the user of the tool will have to define costs based on local requirements and policies. This potential variation in "correct" or "optimal" solutions reflects the need for a context-aware approach to network reconfiguration. Within a specific context, however, an optimal reconfiguration can be discovered and proven by the techniques presented in this chapter.

Chapter 5

Open Problems and Future Work

5.1 Open Problems

Scalability remains a concern in the enterprise network risk assessment algorithm. The running time of the algorithm depends less on the size of the data set than on its interconnectedness. A large but weakly connected data set will probably be calculated faster than a smaller but more strongly connected data set. Improvements in handling larger and more strongly connected network models will expand the applicability of the approach.

I have already made efforts to optimize the treatment of cycles within the attack graph, by special handling within a cycle of only the nodes with multiple predecessors. This reduces the number of nodes for which paths must be identified and considered, thus reducing the overall run-time of the algorithm. Domination-based trimming can also have some effect in reducing the size of cycles.

I believe that opportunity exists for further refinement of the algorithm toward lower run-time and more efficient handling of cycles and larger data sets. There may be a smaller d-separating set distinguishing the separate paths to a cyclic node. Employing a smaller d-separating set would likely lessen the run-time for calculations over cycles by reducing the size of the assumed set used in calculating conditional probabilities. In the implementation, there is also an opportunity to exploit parallelism in the algorithm to utilize multiple CPU cores or even cluster computing techniques.

Another concern in the risk assessment algorithm is the reliability of the input component metrics. There is some skepticism on the feasibility of security metrics given the current software and system architectures [5]. While we may still be far from achieving an objective quantitative metric for a system's overall security, a practical method for quantifying risks in an enterprise network based on known information about potential vulnerabilities is highly valuable in practice. Although they constitute only one aspect of a system's overall security, such risk metrics can produce highly needed automated guidance on how to spend the limited IT management resources and how to balance security and usability in a meaningful manner. I believe that the inherent soundness of my approach to calculating risk in the network will serve to highlight gross errors in the input data. Any such issues, once noted, could then be evaluated and corrected in the input; over time, we may assume, the construction of component metrics will become more precise and reliable. Any such advancement will clearly serve to improve the component metric input data and, in turn, provide obvious benefits in the accuracy of the results of a sound risk assessment algorithm. My work provides the important enabling technology for automated decision making through sound models and algorithms for quantifying security risks using attack graphs and CVSS metrics.

For the SAT-solving approaches to enterprise network reconfigurations presented in Chapter 4, there is opportunity to create the policies at a higher level. Rather than addressing basic Boolean variables, a policy could instead define high-level services and requirements. For example, a usability policy might define "public access to company information," incorporating the accessibility of the web server, the necessary services to retrieve and present data, *etc.*. In the event that the security and usability policies conflict, the user would choose between high-level requirements, rather than needing to consider low-level configuration settings.

Also, I have detailed only the production of some of the values necessary for the input cost policy. The computation of values to reflect usability costs is not addressed by my research. Rather, I leave this as an open problem for researchers better acquainted with

quantification of business impact, functionality, *etc.* I welcome any contributions toward this end.

5.2 Future Work

Future work along this line of research will focus on improving the connectivity between the various pieces, smoothing the flow through the whole process of enterprise network security management. My work will obviously benefit from continued enhancements in attack graph production and presentation for any type of dependency attack graph with semantics comparable to MulVAL.

Improvements in the specificity of network modeling can deliver a greater degree of detail for consideration in the automatic computation of secure network configurations, ensuring a solution of greater accuracy and relevance.

MulVAL does not currently model an intrusion detection system (IDS) that might exist in an enterprise network. An IDS can limit an attacker's ability to penetrate a network without detection, so including an IDS in the Boolean formulas representing the attack graph could change the "optimal" configuration. For example, an administrator might be more willing to leave an open vulnerability in the network if he believes that the IDS will alert him of an attacker's attempts to exploit that vulnerability. Including these and other, similar network defensive features in the network model underlying the attack graph generation can advance the scope and detail of the results. Such improvements in the scope and detail of the attack graph data set will be reflected in reconfiguration suggestions that are more appropriate in the full context of the enterprise network.

McQueen, *et al.* [30] have looked into estimating the likelihood that a vulnerability is currently present but undiscovered, as yet, in a specific software application. This estimate might serve as the basis for the hypothesized presence of unknown vulnerabilities in an enterprise network system, so that an administrator can anticipate the presence and effect of such vulnerabilities and ensure that the system is secure even if an attacker gains knowledge

of a zero-day vulnerability. The likelihood of undiscovered vulnerabilities in the network can be included in the risk assessment calculations for a more comprehension evalaution of the probability that the network can be exploited or a specific privilege gained by an attacker. The inclusion of these estimated probabilities would break the soundness of my risk assessment algorithms, but could potentially expand the effectiveness of my work by considering possibilities beyond what is known to be true. This idea is intriguing and is worth considering for future work in risk assessment and network reconfiguration.

Beyond expanding the network model considerations and improving the general efficiency of the calculations, the chief need in considering future work is knowledgeable input from experienced network administrators. A unified tool implementing my research should be distributed and allowed to be tested by system administrators, the intended users, to generate provide feedback and suggestions. They may be able to identify other aspects of usability or security management that should be included in the framework, provide comments about the accuracy of the risk assessment componenet metrics or probability calculations (based on their experienced intuition), and review the network reconfiguration suggestions for relevance and applicability within the modeled network. Testing and real-world use should, over time, further fine-tune the algorithms and techniques to produce more optimal network configurations the first time when the tool is run for a network.

Chapter 6

Conclusion

In this dissertation, I have presented several complementary techniques for improving enterprise network security management. I have based my work on previous works, most notably attack graphs.

In Chapter 2, I introduced two approaches by which the data set of an attack graph can be reduced or "trimmed." These two approaches, subnet-based trimming and domination-based trimming, are each useful for different purposes. Subnet-based trimming will greatly reduce the size of most attack graphs and thus make the remaining information more readily accessible to a human viewer. It may also, however, remove some valid data from the attack graph. Domination-based trimming will provably reduce the attack graph data, removing only cyclic nodes and edges that would not provide a distinct attack path to the goal. This approach will trim less data from the graph, so this trimmed graph is better used in other evaluations, discussed in later chapters.

In Chapter 3, I presented a sound and quantitative approach for risk assessment in an enterprise network. Using an existing component metric system, such as CVSS, this approach will calculate the probability that a given privilege in the enterprise network would be gained by an attacker. Furthermore, I have shown that this approach properly considers both shared dependencies between attack graph nodes and sets of cyclic nodes within the attack graph. Proofs of correctness for this algorithm can be found in Appendix B.

In Chapter 4, I introduced two approaches to network reconfiguration based on different

SAT-solving techniques. The first approach identifies conflicts in a configuration policy and presents an unsatisfiable subset of the policy, known as an UnSAT core, to the user, who must decide which element of the policy to relax. If multiple UnSAT cores exist, they would be iteratively presented and eliminated. The second approach utilizes a given cost policy, specifying discrete values for each network configuration setting and privilege, and a MinCostSAT solver to find a minimal cost configuration. These two approaches are complementary. When conflicts exist, UnSAT core elimination can be used to remove all conflicts. Once a satisfying solution does exist, the MinCostSAT technique will ensure that a minimal cost solution is returned.

Together, these different approaches constitute a single, comprehensive approach to enterprise network security management. This is my vision of enterprise network security management shown in Figure 1.4. Imagine, for example, a system administrator wanting to understand and correct current vulnerabilities in his network. He would employ an existing analysis tool, such as MulVAL, to build a network model and generate an attack graph. From this data, an improved visualization could be built, probably with subnet-based trimming to reduce the data, and presented to the administrator. The original attack graph data would be trimmed using domination-based trimming and would then be used as input for the risk assessment algorithm. Using the results of this algorithm, costs could be determined for network privileges and added to an overall cost policy that also addressed configuration settings, network usability, *etc.* This cost policy would form the input for an automated network reconfiguration using the two complementary SAT-solving techniques. A new, suggested configuration would be presented to the administrator. Based on his knowledge of the enterprise network and also using the visualization, he would decide to accept or reject the new configuration. If rejecting it, he might alter the cost policy to better reflect valuations among system resources. The automatic reconfiguration tool can be run repeatedly until an acceptable solution is found, properly balancing system security and usability.

Bibliography

[1] Al-Shaer, E., L. Khan, and M. S. Ahmed (2008). A comprehensive objective network security metric framework for proactive security configuration. In *ACM Cyber Security and Information Intelligence Research Workshop*.

[2] Ammann, P., D. Wijesekera, and S. Kaushik (2002, November). Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC.

[3] Anwar, Z., R. Shankesi, and R. H. Campbell (2008, July). Automatic security assessment of critical cyber-infrastructures. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[4] Balzarotti, D., M. Monga, and S. Sicari (2005). Assessing the risk of using vulnerable components. In *Proceedings of the 2nd ACM workshop on Quality of protection*.

[5] Bellovin, S. (2006). On the brittleness of software and the infeasibility of security metrics. *IEEE Security & Privacy*.

[6] Ceri, S., G. Gottlob, and L. Tanca (1989). What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions Knowledge and Data Engineering 1*(1), 146–166.

[7] Chew, E., M. Swanson, K. Stine, N. Bartol, A. Brown, and W. Robinson (2008, July). *Performance Measurement Guide for Information Security*. National Institute of Standards and Technology. NIST Special Publication 800-55 Revision 1.

[8] Coudert, O. (1996). On solving covering problems. In *33rd Design Automation Conference (DAC'96)*, pp. 197–202.

[9] Dawkins, J. and J. Hale (2004, April). A systematic approach to multi-stage network attack analysis. In *Proceedings of Second IEEE International Information Assurance Workshop*, pp. 48 – 56.

[10] Dewri, R., N. Poolsappasit, I. Ray, and D. Whitley (2007). Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*.

[11] Frigault, M. and L. Wang (2008). Measuring network security using bayesian network-based attack graphs. In *Proceedings of the 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*.

[12] Frigault, M., L. Wang, A. Singhal, and S. Jajodia (2008). Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*.

[13] Fu, Z. and S. Malik (2006a). On solving the partial MAX-SAT problem. In A. Biere and C. P. Gomes (Eds.), *Proceedings of Theory and Applications of Satisfiability Testing - SAT 2006*, pp. 252–265.

[14] Fu, Z. and S. Malik (2006b). Solving the minimum-cost satisfiability problem using sat based branch and bound search. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'06)*, San Jose, CA, USA.

[15] Homer, J. and X. Ou (To appear). SAT-solving approaches to context-aware enterprise network security management. *IEEE JSAC Special Issue on Network Infrastructure Configuration*.

[16] Homer, J., A. Varikuti, X. Ou, and M. A. McQueen (2008). Improving attack graph visualization through data reduction and attack grouping. In *The 5th International Workshop on Visualization for Cyber Security (VizSEC)*.

[17] Ingols, K., R. Lippmann, and K. Piwowarski (2006, December). Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida.

[18] Jajodia, S., S. Noel, and B. O'Berry (2003). Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic (Eds.), *Managing Cyber Threats: Issues, Approaches and Challanges*, Chapter 5. Kluwer Academic Publisher.

[19] Jensen, F. V. and T. D. Nielsen (2007). *Bayesian Networks and Decision Graphs* (2 ed.). Springer Verlag.

[20] Jr., D. G., K. S. Hoo, and A. Jaquith (2003). Information security: Why the future belongs to the quants. *IEEE SECURITY & PRIVACY*.

[21] Li, W., R. B. Vaughn, and Y. S. Dandass (2006). An approach to model network exploitations using exploitation graphs. *SIMULATION 82*(8), 523–541.

[22] Li, X. Y. (2004). *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. Ph. D. thesis, North Carolina State University, Raleigh, North Carolina.

[23] Lippmann, R. and K. W. Ingols (2005, March). An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory.

[24] Lippmann, R. P., K. W. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham (2005, October). Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory.

[25] Mahajan, Y. S., Z. Fu, and S. Malik (2004). Zchaff2004: An efficient SAT solver. In *Lecture Notes in Computer Science SAT 2004 Special Volume*, pp. 360–375. LNCS 3542.

[26] Manadhata, P., J. Wing, M. Flynn, and M. McQueen (2006). Measuring the attack surfaces of two FTP daemons. In *Proceedings of the 2nd ACM workshop on Quality of protection*.

[27] Manquinho, V. M. and J. ao P. Marques-Silva (2002). Search pruning techniques in SAT-based branch-and-bound algorithmsfor the binate covering problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21*, 505–516.

[28] McHugh, J. (2006). Quality of protection: measuring the unmeasurable? In *Proceedings of the 2nd ACM workshop on Quality of protection (QoP)*, Alexandria, Virginia, USA.

[29] McHugh, J. and J. Tippett (Eds.) (2001, May). *Workshop on Information-Security-System Rating and Ranking (WISSRR)*. Applied Computer Security Associates.

[30] McQueen, M., T. McQueen, W. Boyer, and M. Chaffin (2009). Empirical estimates and observations of 0day vulnerabilities. In *42nd Hawaii International Conference on System Sciences*.

[31] Mehta, V., C. Bartzis, H. Zhu, E. Clarke, and J. Wing (2006, September). Ranking attack graphs. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*.

[32] Mell, P., K. Scarfone, and S. Romanosky (2007, June). *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Forum of Incident Response and Security Teams (FIRST).

[33] Narain, S. (2005). Network configuration management via model finding. In *Proceedings of the 19th conference on Large Installation System Administration Conference (LISA)*.

[34] Narain, S., G. Levin, S. Malik, and V. Kaul (2008). Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*.

[35] National Institute of Standards and Technology (1985). *Technology assessment: Methods for measuring the level of computer security*. National Institute of Standards and Technology. NIST Special Publication 500-133.

[36] Noel, S., M. Jacobs, P. Kalapa, and S. Jajodia (2005). Multiple coordinated views

for network attack graphs. In *IEEE Workshop on Visualization for Computer Security (VizSEC 2005)*.

[37] Noel, S. and S. Jajodia (2004). Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, New York, NY, USA, pp. 109–118. ACM Press.

[38] Noel, S., S. Jajodia, B. O'Berry, and M. Jacobs (2003, December). Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC)*.

[39] Ou, X. (2005). *A logic-programming approach to network security analysis*. Ph. D. thesis, Princeton University.

[40] Ou, X., W. F. Boyer, and M. A. McQueen (2006). A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pp. 336–345.

[41] Ou, X., S. Govindavajhala, and A. W. Appel (2005). MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*.

[42] Pamula, J., S. Jajodia, P. Ammann, and V. Swarup (2006). A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*.

[43] Phillips, C. and L. P. Swiler (1998). A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79. ACM Press.

[44] Saha, D. (2008). Extending logical attack graphs for efficient vulnerability analysis.

In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*.

[45] Salim, M., E. Al-Shaer, and L. Khan (2008). A novel quantitative approach for measuring network security. In *INFOCOM 2008 Mini Conference*.

[46] Sawilla, R. and X. Ou (2007). Googling attack graphs. Technical report, Defence R & D Canada – Ottawa.

[47] Sawilla, R. and X. Ou (2008, October). Identifying critical attack assets in dependency attack graphs. In *13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain.

[48] Schiffman, M., G. Eschelbeck, D. Ahmad, A. Wright, and S. Romanosky (2004). *CVSS: A Common Vulnerability Scoring System*. National Infrastructure Advisory Council (NIAC).

[49] Sheyner, O. (2004, April). *Scenario Graphs and Attack Graphs*. Ph. D. thesis, Carnegie Mellon.

[50] Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. M. Wing (2002). Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 254–265.

[51] Swiler, L. P., C. Phillips, D. Ellis, and S. Chakerian (2001, June). Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, Volume 2.

[52] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing 1*(2), 146–160.

[53] Tidwell, T., R. Larson, K. Fitch, and J. Hale (2001, June). Modeling Internet attacks.

In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, West Point, NY.

[54] Wang, L., T. Islam, T. Long, A. Singhal, and S. Jajodia (2008). An attack graph-based probabilistic security metric. In *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*.

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

[56] Wang, L., A. Singhal, and S. Jajodia (2007a). Measuring network security using attack graphs. In *Third Workshop on Quality of Protection (QoP)*.

[57] Wang, L., A. Singhal, and S. Jajodia (2007b). Measuring the overall security of network configurations using attack graphs. In *Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'07)*.

[58] Williams, L., R. Lippmann, and K. Ingols (2007). An interactive attack graph cascade and reachability display. In *IEEE Workshop on Visualization for Computer Security (VizSEC 2007)*.

[59] Williams, L., R. Lippmann, and K. Ingols (2008). Garnet: A graphical attack graph and reachability network evaluation tool. In *The 5th International Workshop on Visualization for Cyber Security (VizSEC)*.

Appendix A

Network Models (MulVAL)

MulVAL uses text input files to model enterprise networks. This appendix contains the input used to generate MulVAL attack graphs for examples shown throughout this dissertation.

```
attackerLocated(attacker).

hasAccount(ordinaryEmployee, vpnServer, normalAccount).
hasAccount(ordinaryEmployee, workStation, normalAccount).
hasAccount(ordinaryEmployee, citrixServer, normalAccount).
inCompetent(ordinaryEmployee).

/* Traffic between machines on the same subnet is not filtered */
hacl(H1, H2, _, _) :-
inSubnet(H1, Subnet),
inSubnet(H2, Subnet).

inSubnet(attacker, internet).
inSubnet(webServer, dmz).
inSubnet(vpnServer, dmz).
inSubnet(fileServer, corp).
inSubnet(workStation, corp).
inSubnet(printer, corp).
inSubnet(citrixServer, corp).
inSubnet(operatingStation, ems).
inSubnet(dataHistorian, ems).
inSubnet(commServer, ems).

gateway( outerFirewall ).
gateway( innerFirewall ).
connection( internet, outerFirewall ).
connection( dmz, outerFirewall ).
connection( corp, outerFirewall ).
connection( outerFirewall, innerFirewall ).
connection( ems, innerFirewall ).
```

```
hacl(attacker, webServer, httpProtocol, httpPort).
hacl(attacker, vpnServer, vpnProtocol, vpnPort).
hacl(webServer, fileServer, nfsProtocol, nfsPort).

hacl(vpnServer, H, _, _) :-
inSubnet(H, corp).

hacl(workStation, H, _, _) :-
inSubnet(H, dmz).
hacl(workStation, attacker, _, _).
hacl(citrixServer, dataHistorian, sqlProtocol, sqlPort).

/* configuration information of vpnServer */
networkServiceInfo(vpnServer, vpnService, vpnProtocol, vpnPort, root).

/* configuration information of webServer */
vulExists(webServer, 'CAN-2002-0392', httpd).
cvss('CAN-2002-0392',h).
vulProperty('CAN-2002-0392', remoteExploit, privEscalation).
networkServiceInfo(webServer, httpd, httpProtocol, httpPort, apache).

/* configuration information of fileServer */
networkServiceInfo(fileServer, mountd, rpc, 100005, root).
nfsExportInfo(fileServer, '/export', _anyAccess, workStation).
nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
nfsExportInfo(fileServer, '/export', _anyAccess, citrixServer).

/* configuration information of workStation */
nfsMounted(workStation, '/usr/local/share', fileServer, '/export', read).
networkServiceInfo(workStation, sshd, tcp, sshProtocol, sshPort).

/* configuration information of citrixServer */
nfsMounted(citrixServer, '/usr/local/share', fileServer, '/export', read).
networkServiceInfo(citrixServer, sshd, sshProtocol, sshPort, root).

/* configuration information of dataHistorian */
vulExists(dataHistorian, oracleSqlVulnerability, oracleSqlServer).
cvss(oracleSqlVulnerability,m).
vulProperty(oracleSqlVulnerability, remoteExploit, privEscalation).
networkServiceInfo(dataHistorian, oracleSqlServer, sqlProtocol, sqlPort, root).

/* configuration information of commServer */
vulExists(commServer, iccpVulnerability, iccpService).
cvss(iccpVulnerability,m).
vulProperty(iccpVulnerability, remoteExploit, privEscalation).
networkServiceInfo(commServer, iccpService, iccpProtocol, iccpPort, root).
```

Figure A.1: *Model for example shown in Section 2.1*

```
attackerLocated(internet).

hacl( internet, webServer, httpProtocol, httpPort ).
hacl( internet, vpnServer, vpnProtocol, vpnPort ).
hacl( webServer, dbServer, dbProtocol1, dbPort1 ).
hacl( vpnServer, group1, userProtocol, userPort ).
hacl( vpnServer, group2, userProtocol, userPort ).
hacl( group1, dbServer, dbProtocol2, dbPort2 ).
hacl( group2, dbServer, dbProtocol3, dbPort3 ).
hacl( dbServer, group1, userProtocol, userPort ).
hacl( dbServer, group2, userProtocol, userPort ).

networkServiceInfo( webServer, httpd, httpProtocol, httpPort, apache ).
vulExists( webServer, webVulID, httpd ).
vulProperty( webVulID, remoteExploit, privEscalation ).
cvss(webVulID,h).

networkServiceInfo( vpnServer, vpnService, vpnProtocol, vpnPort, root ).
vulExists( vpnServer, vpnVulID, vpnService ).
vulProperty( vpnVulID, remoteExploit, privEscalation ).
cvss(vpnVulID,h).

networkServiceInfo( group1, g1Service, userProtocol, userPort, root ).
vulExists( group1, g1VulID, g1Service ).
vulProperty( g1VulID, remoteExploit, privEscalation ).
cvss(g1VulID,h).

networkServiceInfo( group2, g2Service, userProtocol, userPort, root ).
vulExists( group2, g2VulID, g2Service ).
vulProperty( g2VulID, remoteExploit, privEscalation ).
cvss(g2VulID,h).

networkServiceInfo( dbServer, dbService1, dbProtocol1, dbPort1, root ).
vulExists( dbServer, dbVulID1, dbService1 ).
vulProperty( dbVulID1, remoteExploit, privEscalation ).
cvss(dbVulID1,h).

networkServiceInfo( dbServer, dbService2, dbProtocol2, dbPort2, root ).
vulExists( dbServer, dbVulID2, dbService2 ).
vulProperty( dbVulID2, remoteExploit, privEscalation ).
cvss(dbVulID2,h).

networkServiceInfo( dbServer, dbService3, dbProtocol3, dbPort3, root ).
vulExists( dbServer, dbVulID3, dbService3 ).
vulProperty( dbVulID3, remoteExploit, privEscalation ).
cvss(dbVulID3,h).
```

Figure A.2: *Model for example shown in Section 3.6*

```
attackerLocated(internet).

hacl(internet, webServer, tcp, 80).
hacl(webServer, fileServer, nfsProtocol, nfsPort).

hacl(webServer, _,  _, _).
hacl(fileServer, _, _, _).
hacl(H,H,_,_).

/* configuration information of webServer */
vulExists(webServer, 'CAN-2002-0392', httpd).
vulProperty('CAN-2002-0392', remoteExploit, privEscalation).
networkServiceInfo(webServer , httpd, tcp , 80 , apache).
cvss('CAN-2002-0392',h).

/* configuration information of fileServer */
networkServiceInfo(fileServer, mountd, rpc, 100005, root).
nfsExportInfo(fileServer, '/export', read, webServer).
nfsExportInfo(fileServer, '/export', write, webServer).

vulExists(fileServer, vulID, service).
vulProperty(vulID, remoteExploit, privEscalation).
networkServiceInfo(fileServer, service, tcp, 80, root).
cvss(vulID,m).
```

Figure A.3: *Model for example shown in Section 4.1*

Appendix B

Proofs

I provide here a formal proof of correctness for my risk assessment approach, as presented in Algorithms 3.2 - 3.7.

Definition 13. *For two distinct nodes a, b in the graph, let us define a relation $a < b$ if and only if node a appears on some path to node b; because the graph is acyclic, if $a < b$, then b cannot appear on any path to a. Intuitively, this ordering states that node a is "nearer" to the graph source node than node b along at least one path.*

Definition 14. *For two nodesets A, B in the graph, let us define a relation $A \leq B$ if and only if $\forall(a \in A, b \in B), a = b$ or $a < b$, by Definition 13; then all nodes in A either appear in B or precede all nodes in B.*

The relation $A \leq B$ satisfies the three properties of a partial order:

1. Reflexivity: For all sets S, $S \leq S$.
2. Antisymmetry: For two sets A, B, if $A \leq B$ and $B \leq A$, then $A = B$.
3. Transitivity: For three sets A, B, C, if $A \leq B$ and $B \leq C$, then $A \leq C$.

So, the relation $\leq$ constitutes a partial ordering over sets of nodes.

Definition 15. *For two nodesets A, B in the graph, $A < B$ if and only if $A \leq B$ and $A \neq B$. Retaining the transitivity property of the $\leq$ relation and applied over to a finite set of nodes, it is easily seen that the relation $<$ has no infinite sequence of sets.*

Lemma 1. *By definition 15, it is clear that for any set N and set P of all immediate or transitive predecessors to $n \in N$, we can know that $P < N$. For a set D that d-separates N, we know that $D \subset P$, so clearly $D < N$.*

Lemma 2. *Algorithm 3.4 computes $\psi(D, N)$, the conditional probability of nodeset N given nodeset D. The algorithm requires that D d-separates N. I will show here that, given well-formed input, this algorithm will always terminate with correct results.*

Base case 1 (Line 15): *Let $N = \{n\}$ and $n \in D$. It is clearly true that $\psi(\{n \cup B\}, \{n\}) = 1$ for any nodeset B. Thus, $\psi(D, N) = 1$.*

Base case 2 (Line 20): *Let $N = \{n\}$ and $\bar{n} \in D$. It is clearly true that $\psi(\{\bar{n} \cup B\}, \{n\}) = 0$ for any nodeset B. Thus, $\psi(D, N) = 0$.*

Base case 3 (Line 20): *Let $N = \{n\}$ and for some $m \in \chi(n)$, $\bar{m} \in D$. Since m dominates n, it is clearly true that $\psi(\{\bar{m} \cup B\}, \{n\}) = 0$ for any nodeset B. Thus, $\psi(D, N) = 0$.*

Base case 4 (Line 26): *Let $N = \{n\}$, such that $D \cap \chi(n) = \emptyset$. Then no $d \in D$ appears on any path to n, so n is independent of D. Thus, by (some def or lemma), $\psi(D, N) = \phi(N)$.*

Induction Hypothesis: *Assume that the algorithm terminates correctly for every $evalCondProb(D, P)$, where $P < N$.*

Induction Step: *Assume the induction hypothesis. I will show that the algorithm terminates correctly for $evalCondProb(D, N)$, addressing four cases:*

(1) $|N| > 1$. *By Theorem 2, we know that* $\psi(D, N) = \prod_{n \in N} \psi(D, \{n\})$. *By Definition 15, we know that* $\forall (n \in N), \{n\} < N$. *Then by the induction hypothesis, the algorithm will correctly terminate.*

(2) $N = \{\bar{n}\}$. *Because node n must be strictly either true or false, it is easily seen that* $\psi(D, \{n\}) + \psi(D, \{\bar{n}\}) = 1$. *Thus,* $\psi(D, \{\bar{n}\}) = 1 - \psi(D, \{n\})$. *In this algorithm, a recursive call will never be made in which a positive node is negated, so clearly this recursive call in which a negated node is made positive will occur at most once per node, and by the induction hypothesis, the algorithm will correctly terminate.*

(3) $N = \{n\}, n \in G_D$. *Assume that nodeset P is the set of immediate predecessors to N. By Lemma 1,* $P < N$. *Then by the induction hypothesis, this recursive call will return a correct result for* $evalCondProb(D, P)$, *and by Proposition 1, this result can be used to correctly compute* $\psi(D, N)$, *so the algorithm will correctly terminate.*

(4) $N = \{n\}, n \in G_C$. *Assume that nodeset P is the set of immediate predecessors to N. By Lemma 1,* $P < N$. *Then by the induction hypothesis, this recursive call will return a correct result for* $evalCondProb(D, P)$, *and by Proposition 3, this result can be used to correctly compute* $\psi(D, N)$, *so the algorithm will correctly terminate.*

Thus, this algorithm will always return a correct assessment for the conditional probability of set N, given set D.

Lemma 3. *Algorithm 3.3 computes* $\phi(N)$, *the probability of nodeset N. I will show here that, given well-formed input, this algorithm will always terminate with correct results.*

Base case (Line 8): *Let N be a nodeset such that all* $n \in N$ *are independent (with no d-separating set D). In the structure of the approach, all the individual probabilities*

of all nodes $n_1, n_2, ..., n_j$ will already be computed before these nodes can comprise a nodeset N. Thus, $\phi(n)$ is known for all $n \in N$. If all $n \in N$ are independent, then $\phi(N)$ is equal to the product of the individual probabilities. Thus, the algorithm terminates with a correct result.

Induction Hypothesis: *Assume that the algorithm terminates correctly for some nodeset D.*

Induction Step: *Assume the induction hypothesis for nodeset D, such that D d-separates N. By Lemma 1, $D < N$. Then by Theorem 1, Lemma 2, and the induction hypothesis, the algorithm will correctly terminate.*

Lemma 4. *Algorithm 3.7 computes for D, N the conditional probability of nodeset N given nodeset D, where nodeset N is wholly contained within a strongly connected component in the graph. The algorithm requires that D d-separates N. I will show here that, given well-formed input, this algorithm will always terminate with correct results.*

Base case (Line 5): *Let $N = \{n\}$ and let D be a nodeset such that D d-separates N. Because this algorithm is dealing with nodes within a strongly connected component in the graph, set D contains a subset of extra-cyclic nodes sufficient to d-separate the cycle from the rest of the graph and another, distinct subset of cyclic nodes lying along a path P to $\{n\}$. Then the probability of n is simply the probability that all other nodes in P are true. These nodes are independent, so the product is computed and returned. Thus, the algorithm returns a correct result.*

Induction Hypothesis: *Assume that the algorithm terminates correctly for some nodesets $D, \{n\}$.*

Induction Step: *Assume the induction hypothesis. I will show that the algorithm terminates correctly for some sets D, N such that D d-separates N. I will address two cases:*

(1) $|N| > 1$. *By Theorem 2, we know that* $\psi(D, N) = \prod_{n \in N} \psi(D, \{n\})$ *and by Definition 15, we know that* $\forall (n \in N), D < \{n\}$. *Then by the induction hypothesis, the algorithm will correctly terminate.*

(2) $N = \{\bar{n}\}$. *Because node n must be strictly either true or false, it is easily seen that* $\psi(D, \{n\}) + \psi(D, \{\bar{n}\}) = 1$. *Thus,* $\psi(D, \{\bar{n}\}) = 1 - \psi(D, \{n\})$. *In this algorithm, a recursive call will never be made in which a positive node is negated, so clearly this recursive call in which a negated node is made positive will occur at most once per node, and by the induction hypothesis, the algorithm will correctly terminate.*

Lemma 5. *Algorithm 3.6 will identify and record all acyclic paths leading through the set of cyclic nodes. In each call, the algorithm will record a unique, single instance of an acyclic path leading to some node n within a graph cycle. Because there are a finite number of paths, the function must terminate. Furthermore, because all possible, acyclic paths are explored, the function will correctly idenitfy unique acyclic paths to each node in the cycle.*

Lemma 6. *Algorithm 3.5 will correctly assess a subset of nodes in a cycle.*

By Lemma 5, all acyclic paths within the cycle will be correctly identified. This algorithm will loop over the finite set M *of cycle nodes with multiple immediate predecessors, guaranteeing termination of the algorithm. For each* $m \in M$, *the algorithm considers all acyclic paths leading to* m *to compute* $\phi(m)$. *By Definition 9 and Lemma 4, the calculated result will be correct. Thus, the algorithm correctly assesses each cyclic node with multiple immediate predecessors.*

Lemma 7. *Algorithm 3.2 will correctly calculate the probability of each node within the graph.*

This algorithm contains a single loop which will iterate until set U *is empty. In each iteration of the loop, one or more nodes will be removed from* U. *Thus,* U *is strictly decreasing and must eventually be empty, so this algorithm will terminate.*

I will consider the correctness of the assessment in four cases:

(1) *Assume $n \in G_D$, where n has an immediate predecessor p such that $\phi(p) = 1$. Clearly, $\phi(n) = 1$, since n will be true when at least one predecessor is true and p is always true.*

(2) *Assume $n \in G_D$. Then by Proposition 1 and Lemma 3, the algorithm correctly assesses the probability of n.*

(3) *Assume $n \in G_C$. Then by Proposition 3 and Lemma 3, the algorithm correctly assesses the probability of n.*

(4) *Assume $C \subset G_N$, such that C comprises a strongly connected component in the graph. Then by Lemma 6, the algorithm correctly assesses all nodes in C with multiple immediate predecessors.*

Thus, this algorithm will always produce a correct result.

Proof of correctness for the risk assessment technique presented in Chapter 3. The risk assessment technique comprising Algorithms 3.2 - 3.7 will always terminate with correct evaluations for every node in the graph. By Lemma 7, all nodes in the graph will be correctly evaluated. Therefore, the overall technique will always produce correct assessment values.

□