DISSERTATION

TOWARDS AN EFFICIENT VULNERABILITY ANALYSIS METHODOLOGY FOR BETTER SECURITY RISK MANAGEMENT

Submitted by

Nayot Poolsappasit

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2010

COLORADO STATE UNIVERSITY

July 13, 2010

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY NAYOT POOLSAPPASIT ENTITLED TOWARDS AN EFFICIENT VULNERABILITY ANALYSIS METHODOLOGY FOR BETTER SECURITY RISK MANAGEMENT BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Ross M. McConnell

Anura P. Jayasumana

Advisor: Indrajit Ray

Co-Advisor: Indrakshi Ray

Department Chair: L. Darrell Whitley

ABSTRACT OF DISSERTATION

TOWARDS AN EFFICIENT VULNERABILITY ANALYSIS METHODOLOGY FOR BETTER SECURITY RISK MANAGEMENT

Risk management is a process that allows IT managers to balance between cost of the protective measures and gains in mission capability. A system administrator has to make a decision and choose an appropriate security plan that maximizes the resource utilization. However, making the decision is not a trivial task. Most organizations have tight budgets for IT security; therefore, the chosen plan must be reviewed as thoroughly as other management decisions.

Unfortunately, even the best-practice security risk management frameworks do not provide adequate information for effective risk management. Vulnerability scanning and penetration testing that form the core of traditional risk management, identify only the set of system vulnerabilities. Given the complexity of today's network infrastructure, it is not enough to consider the presence or absence of vulnerabilities in isolation. Materializing a threat strongly requires the combination of multiple attacks using different vulnerabilities. Such a requirement is far beyond the capabilities of current day vulnerability scanners. Consequently, assessing the cost of an attack or cost of implementing appropriate security controls is possible only in a piecemeal manner.

In this work, we develop and formalize new network vulnerability analysis model. The model encodes in a concise manner, the contributions of different security conditions that lead to system compromise. We extend the model with a systematic risk assessment

methodlogy to support reasoning under uncertainty in an attempt to evaluate the vulnerability exploitation probability. We develop a cost model to quantify the potential loss and gain that can occur in a system if certain conditions are met (or protected). We also quantify the security control cost incurred to implement a set of security hardening measures. We propose solutions for the system administrator's decision problems covering the area of the risk analysis and risk mitigation analysis. Finally, we extend the vulnerability assessment model to the areas of intrusion detection and forensic investigation.

Nayot Poolsappasit
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 2010

ACKNOWLEDGEMENTS

A lot of people should be acknowledged for their contributions behind my research success and those who inspired and encouraged me in pursuing my Ph.D. First of all, I am heartily thankful to my academic advisor, Dr. Indrajit Ray, for his encouragement, guidance and support from the beginning to the final level. His perpetual energy and enthusiasm in research has motivated all his advisees, including me. As a result, my Ph.D. life became smooth and rewarding to me. I also especially want to thank my co-advisor, Dr. Indrakshi Ray, for her inspiration and encouragement. In particular, I would like to thank Dr. Indrakshi Ray for passing on an internship opportunity at SAP Research Lab, Palo Alto. I could not have been who I am today without her guidance. In fact, both Dr. Rays always help their students in anything beyond what they could expect from ordinary academic advisors. I am so fortunate to have them as mentors.

Dr. Ross McConnell and Dr. Anura Jayasumana deserve a special thank as my dissertation committee members. In particular, I would like to thank Dr. McConnell for his insights in algorithm and complexity analysis. His expertise helps me in making rigorous proofs behind many theories. I also want to thank, Rinku Dewri, my research partner for his expertise in Generic Algorithm. Together, we produced numerous research papers and contributions in risk management analysis. I always recognize Rinku as a better researcher, and strongly believe that he will become a famous professor in the future.

Finally, my deepest gratitude goes to my family for their unflagging love and support throughout my life. I am indebted to my mother, Sriwanna Poolsappasit, for her care and love. Since my father passed away, she has worked industriously to support the family of

three children and to provide the best possible environment for me to grow up. She has never complained in spite of the hardships in her life.

Finally, I can not ask for more from my wife, Ratree Poolsappasit. Although I can not say she is the best wife since I have no other, but I know that she will always be with me to share joy and sorrow. Thank for her patience and being with me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Every organization has objective, asset, and mission to protect. These days, all organizations make use of automated information technology (IT) systems to process their missions for greater benefits. The head of an organizational unit must ensure that the organization has the capabilities needed to accomplish its missions. From the security perspective, the organization needs the capabilities that allow it to maintain the desired level of security in the face of real world threats. Risk management plays a critical role in determining the security capabilities to protect an organizations information assets and carry on its missions from IT-related risks. An effective risk management is an essential part of a successful IT project.

Risk management is a process that allows IT managers to balance between cost of the protective measures and gains in mission capability. A system administrator has to make a decision and choose an appropriate security plan that maximizes the resource utilization. However, making the decision is not a trivial task. Most organizations have tight budgets for IT security; therefore, the chosen plan must be reviewed as thoroughly as other management decisions.

Risk management is broken into three components namely risk assessment, risk mitigation, and evaluation. Risk assessment is a process of determining the extent of negative impacts associated with the system. The output of this process helps decision maker to identify appropriate controls for reducing the risk in the risk mitigation process. Risk mitigation consists of several analysis methodologies which are used to prioritize risk and choosing the appropriate risk-reducing measures. The evaluation process includes a

process of risk acceptance which requires senior management to sign a statement accepting the residual risk and authorizing the security hardening operation. A well-structured risk management methodology, when used effectively, can help management identify appropriate controls for providing the mission-essential security capabilities.

In summary, risk management is the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level. Security Risk Management plays a vital role in protecting information, assets, business missions, and secrets from potential risks. Although each of these processes is equally important but we can not deny that the success key in IT security relies on the perfection of risk identification process. Unfortunately, even the best-practice risk management framework does not provide adequate information for effective risk management. More specifically, most traditional security risk management process ultimately boils down to only vulnerability identification for identifying the list of system vulnerabilities. Given the complexity of today's network infrastructure, it is not enough to consider the presence or absence of vulnerabilities in isolation. Materializing a threat strongly requires the combination of multiple attacks using different vulnerabilities. Such a requirement is far beyond what typical vulnerability scanner can give.

1.1 Problem Statement

The goal of vulnerability identification is to determine an extent of negative impacts associated with the system. The output of this process helps decision maker to identify appropriate controls for reducing the risk in the risk mitigation process. Typically, vulnerability assessment relies heavily on developing a list of system vulnerabilities using vulnerability scanner or penetration testing. My research shows that the traditional practice has at least two major drawbacks. First, traditional vulnerability scanners merely identify a list of individual vulnerabilities but are unable to depict the correlations between these vulnerabilities. Modern day attacks rely on exploiting multiple vulnerabilities in a correlated manner. Given the complexity of today's network infrastructure, it is not enough

to consider the presence or absence of vulnerabilities in isolation. Second, the value of the penetration testing relies on the thoroughness of the testing scenarios, testing level, and expertise of the testers. Penetration testing as practiced today is not a systematic approach and does not guarantee completeness. Consequently, since the risk mitigation analysis takes the input from the vulnerability assessment, then the cost-benefit analysis would be mostly flat in structure. Then the choice of security controls would then be determined by uncorrelated knowledges which do not describe the actual root cause of security risk.

What is needed is a systematic approach that models threats against the system. The model should allow system administrators to understand the actual outline of the network vulnerability and help them manage the risk in a proper manner. Researchers have proposed many vulnerability models using paradigms like attack graph [12, 38, 50, 62, 65] or attack tree [25, 43, 52, 60] to identify attack scenarios. But merely determining possible attack scenarios is not enough to help the system administrators make appropriate decisions. They are more interested in determining the best strategy to strengthen the system. Keeping this in mind, I propose the security model that not only captures ways in which the system can be attacked but also supports the calculations in risk analysis. The proposed model uses the concept of attack tree paradigm.

1.2 Research Goals and Contribution

Towards an Efficient vulnerability analysis methodology, there are three goals of the study.

1. To formalize the systematic model of the vulnerability assessment that outlines the universe of possible consequences following a successful attack. In addition, we wish to instantiate this abstraction to obtain a systematic method that automates the vulnerability assessment process.

2. To develop the pragmatic risk mitigation methodology to assist the security decision. In particular, we are interested to solve the system administrator's dilemma in choosing the best security hardening strategy that maximizes the resource utilization as an optimization problem and progressively transform it into the next to cater to more cost-benefit information as may be required by the decision maker.

3. Finally, we wish to evaluate the possibility of utilizing the proposed abstraction in the area of intrusion detection and forensic investigation.

Toward this end, this work makes five major contributions. First, it refines and formalizes the notion of network vulnerability assessment model so as to encode the contribution of different security conditions leading to system compromise. Second, it extends the assessment model to encode with the systematic risk assessment analysis to support the reasoning framework under uncertainty in an attempt to predict the vulnerability exploitation probability. Third, it proposes a cost model to quantify the potential loss and gain that can occur in a system if a certain condition are met (or protected). The model also quantifies the security control cost incurred to implement a set of security hardening measures. Forth, it proposes solutions for the system administrator's decision problems covering the area of the risk analysis and risk mitigation analysis. Fifth, it extends the vulnerability assessment model to the areas of intrusion detection and forensic investigation. Last but not the least we discuss our thoughts and observations regarding to the assessment model, in particular the critical sections and robust solutions, with a belief that such this observation will help the system administrator decide what methodology to adopt.

1.3 Dissertation Outline

The proposal is organized as the following. Chapter 2 presents the background of risk management process covering risk assessment, risk mitigation, and ISMS security

best-practice. Chapter 3 presents the literature review. It covers the related works in the areas of vulnerabilities model, assessment methodologies, intrusion detection, and forensic investigation. In Chapter 4, I give the test-bed network to illustrate the problem and address technical challenges in designing security hardening strategies. The network security risk assessment model is formulated in Chapter 5 and Chapter 6 describes an approach to represent and generate attack model given vulnerabilities and network topology information. In the Chapter 7, we formulate the problem of security risk analysis and develop a systematic approach to solve the problem using genetic algorithm and an attack tree model of the system. The risk model lends itself to optimal cost-benefit analysis in finding effective solutions for security hardening. Chapter 8 proposes dynamic risk assessment framework using Bayesian networks to enable a system administrator quantifies the chances of network compromise at various levels and shows how to use this information to develop a security mitigation and management plan. In Chapter 9 we develop an algorithm for intent-based intrusion detection system to detect threats from malicious insiders who can execute perfectly legitimate operations to compromise the system. The algorithm generates minimal forms of an attack tree customized for each user such that it can be used efficiently to monitor the users activities. If the users activities progress sufficiently up along the branches of the attack tree towards the goal of system compromise, the system can correctly predict the outcome and timely generates an alarm. Finally, chapter 11 summarizes the whole research and outlines the significants of network risk models and risk management process in supporting capabilities that allow IT business to maintain the desired level of security in the face of real world threats.

Chapter 2

BACKGROUND ON RISK MANAGEMENT

Risk management is a process that allows IT managers to balance between costs of protective measures and gains in mission capabilities. The objective of performing risk management is to enable the organization to accomplish its mission by increasing the security level of the IT systems, enabling management to make well-informed risk management decisions to justify the expenditures that are part of an IT budget and assisting management in authorizing or accrediting the IT systems on the basis of the supporting documentation resulting from the performance of risk management.

Risk management is broken into three major components namely risk assessment, risk mitigation, and evaluation. Each can be briefly described as follow.

2.1 Risk Assessment

Risk assessment is a process of determining the extent of negative impacts associated with the system. The output of this process helps decision maker identify appropriate controls for reducing the risk in the risk mitigation process. NIST SP800-30 [64] divides the risk assessment process into nine steps listed as follow:

Step 1 System Characterization: Identify the boundaries of the IT system along with the resources and the information that constitute the system. The goal of this step is to establish the scope of the risk assessment effort, delineate the operational authorization (or accreditation) boundaries, and provide information (e.g., hardware, software, system connectivity, and responsible division or support personnel) essential to define the risk.

Step 2 Threat Identification: Identify the potential threat-sources, motivations and actions. The goal of this step is to realize the threat model by listing potential threat-sources along with the motivations and actions that are applicable to the IT system being evaluated.

Step 3 Vulnerability Identification: Identify the flaws or misconfigurations that can be accidentally triggered or intentionally exploited. The goal of this step is to develop a list of system vulnerabilities that could be exploited by potential threat-sources.

Step 4 Control Analysis: The goal of this step is to analyze the security measures that already been implemented or planned to be implemented by the organization to minimize or eliminate the likelihood of a threat's exercising a system vulnerability.

Step 5 Likelihood Determination: Estimate the overall likelihood rating that indicates the probability that a potential vulnerability may be exploited under the threat environment (threat-sources, nature of vulnerability and level of security controls). The likelihood can be expressed as high, medium and low. NIST gives the descriptions of these quantitative measure in Table 2.1.

Likelihood Level	Definition
High	The threat-source is highly motivated and sufficiently capable, and controls to prevent the vulnerability from being exercised are ineffective.
Medium	The threat-source is motivated and capable, but controls are in place that may impede successful exercise of the vulnerability.
Low	The threat-source lacks motivation or capability, or controls are in place to prevent, or at least significantly impede, the vulnerability from being exercised.

Table 2.1: Likelihood Definitions (Source G. Stoneburner et. al.,"*Risk Management Guide for Information Technology Systems*", NIST SP 800-30, 2002).

Step 6 Impact Analysis: Estimate the level of risk by determining the adverse impact resulting from a successful threat exercise of a vulnerability. This information can be obtained from existing organization documents, such as the mission analysis report or asset criticality assessment report. A mission analysis prioritizes the impact

levels associated with the compromise of information assets. An asset criticality assessment identifies and prioritizes an organization's information assets (e.g., hardware, software, systems, services, and related technology assets) that support the organization's critical missions.

Step 7 Risk Determination: The final determination of mission risk is derived by multiplying the ratings assigned for likelihood and impact. The purpose of this step is to assess the level of risk to the IT system.

Step 8 Control Recommendations: The control recommendations are the results of the risk assessment process and are inputs to the risk mitigation process. During this step, the recommended procedural or technical security controls are identified to reduce the level of risk to an acceptable level. It should be noted that not all possible recommended controls can be implemented to reduce loss. A security administrator has to assess the security controls by conducting a cost-benefit analysis to verify that the costs of implementing the controls are worth for the investment. The cost-benefit analysis is covered in risk mitigation analysis.

Step 9 Results Documentation: Once the risk assessment has been completed, the results should be documented in an official report or briefing to help decision makers reach decisions on policy, procedural, budget, and system operational and management changes.

2.2 Risk Mitigation

Risk mitigation involves prioritizing, evaluating and implementing the appropriate risk-reducing controls recommended from the risk assessment process. Risk mitigation can be achieved through any of the following options:

- Research and Acknowledgment: To acknowledge the potential vulnerability or flaw and researching controls to correct the vulnerability

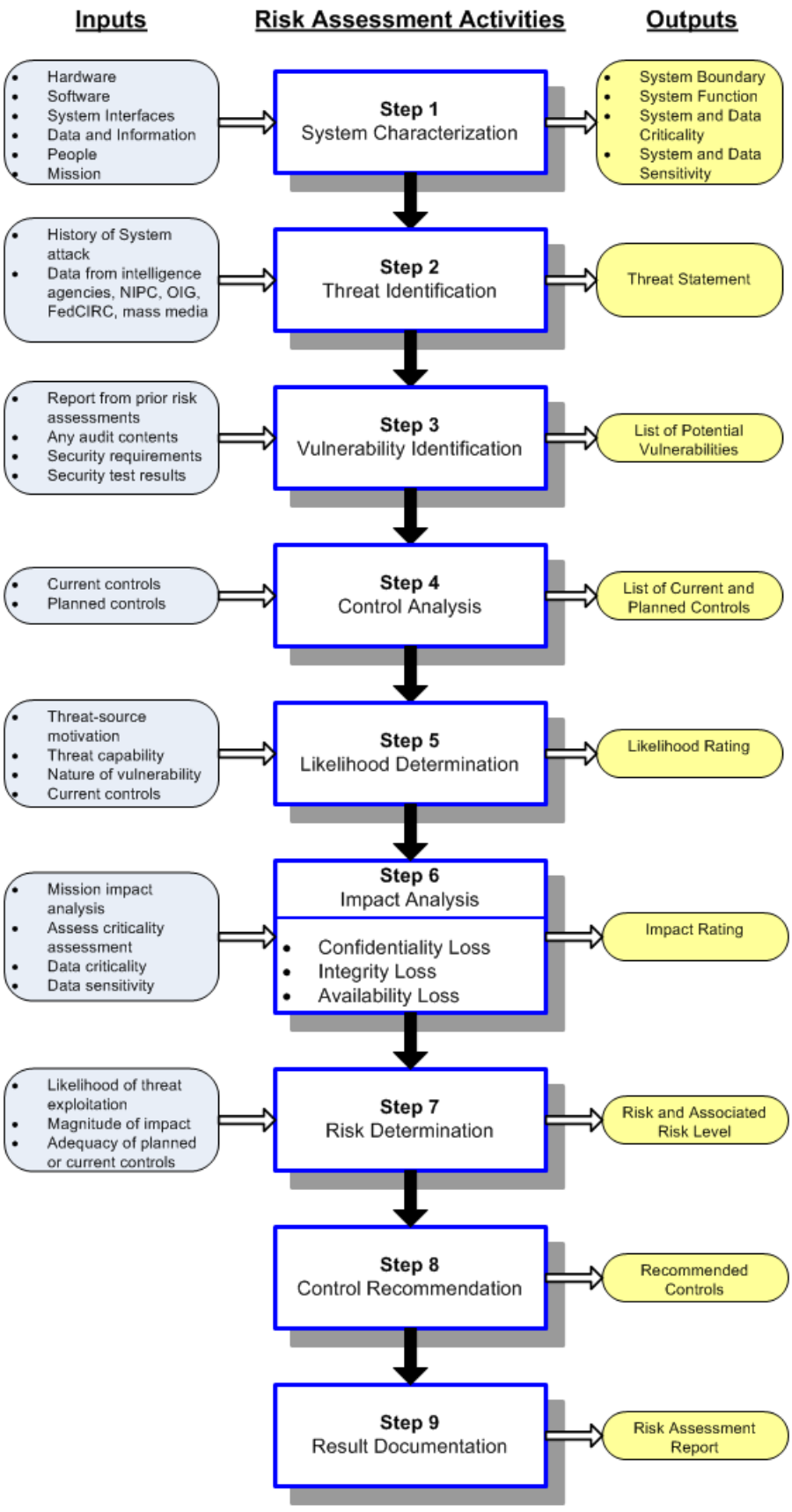


Figure 2.1: Risk Assessment Methodology Flow Chart (Adapted from G. Stoneburner et. al.,"*Risk Management Guide for Information Technology Systems*", NIST SP 800-30, 2002)

- Risk Avoidance: To avoid the risk by eliminating the risk cause and/or consequence (e.g., forgo certain functions of the system or shut down the system when risks are detected)

- Risk Limitation: To limit the risk by implementing controls that minimize the adverse impact of a threat's exercising a vulnerability (e.g., use of supporting, preventive, detective controls)

- Risk Tolerance: To implement controls to lower the risk to an acceptable level and accept the remaining risk that can potentially cause damages while continuing the operation of the IT system.

- Risk Transference: To transfer the risk by using other options to compensate for the loss, such as purchasing insurance.

The goal of risk mitigation is to consider in selecting any of these options. Certainly, it may not be practical to address all identified risks, so priority should be given to the vulnerabilities that have the potential to cause significant mission impact or harm. Also, in safeguarding an organization's missions and its IT systems, each organization has its own environment and objectives. Hence, the option(s) used to mitigate the risk and the methodologies may vary. The following are steps toward the risk mitigation analysis.

Step 1 Prioritize Actions: Based on the risk levels presented in the risk assessment report, the implementation actions are prioritized. In allocating resources, top priority should be given to risk items with are dangerously high. These items will require immediate corrective actions to protect an organization's interest and mission.

Step 2 Evaluate Recommended Control Options: The controls recommended in the risk assessment process may not be the most appropriate and feasible options for a specific organization and IT system. During this step, the feasibility (e.g., compatibility, user acceptance) and effectiveness (e.g., degree of protection and level of

risk mitigation) of the recommended control options are analyzed. The objective is to select the most appropriate control option for minimizing risk.

Step 3 Conduct Cost-Benefit Analysis: To aid management in decision making and to identify cost-effective controls, a cost-benefit analysis is conducted. Cost-benefit analysis is the process that allows system administrators to find a trade-off between the cost of implementing security hardening measures, and the gains in mission objectives results from implementing the security control. With cost-effectiveness in mind, system administrators can find out how to spend the resources to strengthen the system to balance the operation and economic cost of protective measures and achieve gains in mission capability.

Step 4 Select Control: On the basis of the results of the cost-benefit analysis, management determines the most cost-effective control(s) for reducing risk to the organization's mission. The controls selected should combine technical, operational, and management control elements to ensure adequate security for the IT system and the organization.

Step 5 Assign Responsibility: Appropriate persons who have the appropriate expertise and skill-sets to implement the selected control are identified, and responsibility is assigned.

Step 6 Develop a Safeguard Implementation Plan: The safeguard implementation plan prioritizes the implementation actions and projects the start and target completion dates. This plan will aid and expedite the risk mitigation process.

Step 7 Implement Selected Control(s): Executing selected controls in the safeguard implementation.

Depending on the situations, the implemented controls may lower the risk level but not eliminate the risk. The remaining risk after the implementation of security controls is

called “residual risk”. Hence, the residual risk is an outcome of the risk mitigation analysis. Figure 2.2 shows the flowchart of risk mitigation process. Note that, as mandated by OMB Circular A-130 [9], the intent of this process is to identify risks that are not fully addressed and to determine whether additional controls are needed to mitigate the risks identified in the IT system. For federal agencies, after the appropriate controls have been put in place for the identified risks, organization’s senior management needs to sign a statement accepting any residual risk and authorizing the operation of the new IT system or the continued processing of the existing IT system. If the residual risk has not been reduced to an acceptable level, the risk management cycle must be repeated to identify a way of lowering the residual risk to an acceptable level.

2.3 Risk Evaluation

This section emphasizes good practice and need for an ongoing risk evaluation and assessment and the factors that will lead to a successful risk management program. In most organizations, the network itself will continually be expanded and updated, its components changed, and its software applications replaced or updated with newer versions. In addition, personnel changes will occur and security policies are likely to change over time. These changes mean that new risks will surface and risks previously mitigated may again become a concern. Thus, the risk management process is ongoing and evolving. As mandated by OMB Circular A-130, the risk assessment process must regularly repeat at least every three years for federal agencies. Following the best practice, risk management should be conducted and integrated in the Software Development Life Cycle of IT systems, not because it is required by law or regulation, but because it supports the organization’s business objectives or mission. A successful risk management program will rely on (1) senior management’s commitment; (2) the full support and participation of the IT team; (3) the competence of the risk assessment team, which must have the expertise to apply the risk assessment methodology to a specific site and system; (4) the awareness and cooperation of members of the user community, who must follow procedures

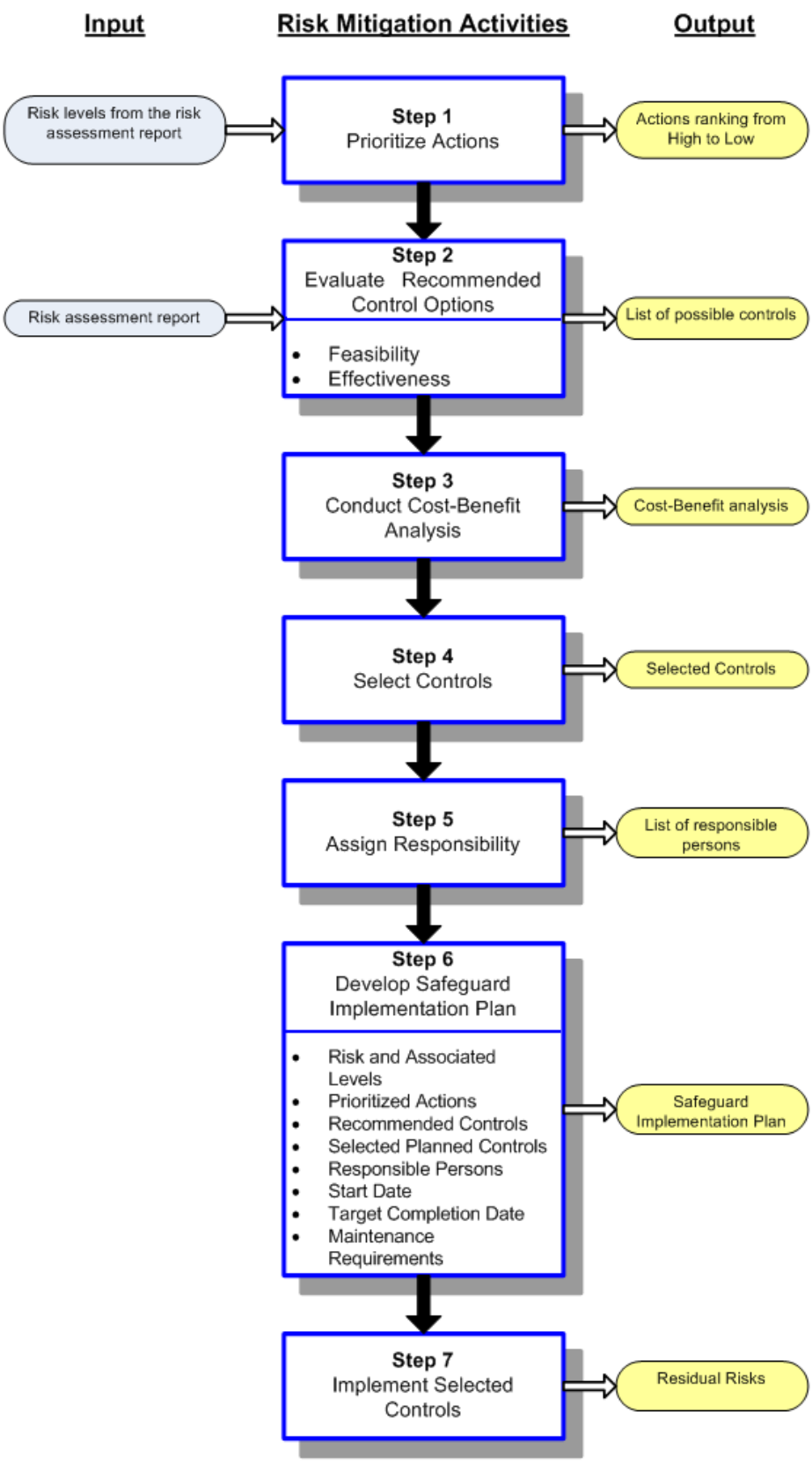


Figure 2.2: Risk Mitigation Methodology Flow Chart (Adapted from G. Stoneburner et. al.,"*Risk Management Guide for Information Technology Systems*", NIST SP 800-30, 2002)

and comply with the implemented controls to safeguard the mission of their organization; and (5) an ongoing evaluation and assessment of the IT-related mission risks. A well-structured risk management methodology, when used effectively, can help management identify appropriate controls for providing the mission-essential security capabilities.

Chapter 3

RELATED WORKS

Figure 3.1: Research Areas in Security Risk Management

Security risk management research is a complex area. It encompasses four aspects namely risk assessment, risk mitigation, security best-practice, and security technology. There are many research groups behind the development of security risk management research. This chapter presents works in the areas that are relevant to our research. We organize this chapter into four major areas including risk modeling, risk assessment analysis, intrusion detection, and forensic investigation.

3.1 Risk Modeling

Most traditional risk analysis frameworks rely heavily on the list of vulnerabilities obtained from the vulnerability scanner tools. A number of commercial and non-commercial scanning tools such as SAINT, Nessus, and Snort [6, 7, 29, 54, 70] evaluate the security risk of a network by developing a list of vulnerabilities per host basis. When evaluating the security of a network, it is not enough to simply consider the presence or absence of vulnerabilities in isolation. The major disadvantage of the vulnerability list is that it does not consider the network properties. Therefore, it cannot correlate local vulnerabilities to depict the global vulnerability introduced by the interconnections between hosts.

Graph-based models have been proposed to address this problem. Graph-based models have gained more acceptance as the method allows analysts to analyze the threats from both outside and inside of the network. In addition, Graph-base models can analyze risk to a specific network asset, or examine the universe of possible consequences following a successful attack. The system could be used to test the effectiveness of making configuration changes, implementing an intrusion detection system, etc. Among these representations, attack graph [37, 50, 53, 55, 63] and attack tree [31, 23, 60] are the two most general accepted models. Attack graphs depict ways in which an adversary exploits system vulnerabilities to achieve undesirable states. Paths in an attack graph represent different attack scenarios which attackers may use to achieve undesirable states (e.g. system compromise, DOS, information leakage, etc.). Node in attack graphs represents an individual attack.

On the other hand, attack trees have been proposed as a systematic method to specify system security based on varying attacks. An attack tree helps organize intrusion and/or misuse scenarios by utilizing known vulnerabilities in the system, analyzing system dependencies, and representing these dependencies in the form of an And-Or tree. An often-cited criticism of attack trees (vis-a-vis attack graphs) is that they are not able

to model cycles and time dependencies. However, we believe that this criticism is valid only in cases where attack trees are used to represent a sequence of operations leading to attacks, not when it is used to represent the dependency of system states. Another criticism on attack trees is that they tend to become more unwieldy than attack graphs. Our argument on this criticism is that both attack trees and attack graphs have their own advantage and disadvantage. In fact, it is merely a different view of the network security risk model. Therefore, it is not possible to make a sound judgment as to which model is better. An appropriate model is chosen by the favor and nature of usage.

3.2 Risk Assessment Analysis

The management of IT security risk is a major concern to organizations and governments worldwide. A lot of efforts are invested to improve the risk management process. Hence, numbers of risk assessment models have been proposed so far. In addition, we could see that the government sectors play an important role to develop and standardize the risk management frameworks. These models include CRAMM, EBIOS, and NIST [4, 44, 64]. With slight differences in the implementation, most traditional risk analysis frameworks evaluate the risk level following the classical equation:

$$Risk = Impact \times Likelihood \tag{3.1}$$

We have also found that all risk assessment frameworks do not specify which method to be used to mitigate risk. Instead, they encourage security administrators to choose appropriate methodologies to manage their own risks. A numbers of works have been proposed so far [3, 12, 53, 55, 37, 47, 69]. We can categorize these methodologies into minimization analysis, maximization analysis, optimization analysis, and simulation-based analysis.

The goal of minimization analysis is to identify the set of elements that, when they are implemented (or prevented), minimizes the expense. Some known analyses are least-cost analysis, least-effort analysis and minimal set of preventive measurements. Unlike

the minimization analysis, security administrators also use maximization analysis to find the attack scenarios that are most likely to happen. In this case, the probability is used to assess such a scenario. Note that choosing an appropriate data structure is critical. Many researchers have found that using the attack graph in the maximization analysis may suffer from NP-complete as, in many cases, maximizing the values is equivalent to choosing the longest path in the network. Phillips et al. [50] found that in some cases, we can transform the maximization to minimization problem by simply negating the assessment values. However, not all maximization problems can be converted to minimization analysis. Beside the minimization and maximization analysis, more often, the system administrator has to work within a given set of budget constraints. Given a set of security measures, the security administrators have to design a security plan that maximizes the use of these countermeasures and minimizes the cost of implementation. In its most general form, the problem is NP-hard. Phillips et al. [20] try to use bi-criteria shortest-path algorithms to compute the near-optimal cost-benefit analysis. However, scalability is an awful disadvantage to this approach.

The last group of risk analysis includes those that do not belong to any of the previous categories. Simulations are what-if analysis types that have their own specific use. For example, Dacier et al. [3] compute the Mean Time to Failure (METF) by simulating the network penetration from an attacker stand point. Jha et al. [37] introduce the reach ability analysis. Given a set of security controls, the reach ability analysis identifies whether the network is safe if a chosen set of security controls are implemented. In many occasions, security administrators may want to simulate the what-if analysis to assess the consequence of the changes such as network configuration, topology, or application, firewall technology, etc.

Of all different analysis presented, different models were designed to be used for each specific method and the performance of the analysis relies heavily on the chosen meta model. We have seen some attempts regarding the problem transformation but the

transformation can be implemented in some cases but not all. There is no sufficient research concerning the unified model for generic usages in risk analysis methodologies and the problem of security optimization tends to be the most challenging problem.

3.3 Intrusion Detection System

Intrusion detection systems can be broadly classified into two groups: knowledge-based systems and behavior-based systems [28]. Knowledge-based detectors [56, 61, 42] are the most popular techniques. A numbers of commercially available intrusion detection tools are knowledge-based detectors. These tools operate by processing system audit data for signatures of known attacks and/or specific outcomes of interest. The result of this processing is compared against signatures of specific attacks and vulnerabilities. A positive match signals an intrusion.

Knowledge-based detectors tend to be fairly accurate in the sense that they have low rates of false positives. However, they are limited by their inability to detect new attacks for which there are no known signatures. Any action that is not recognized as an attack is considered acceptable. Behavior-based detectors, on the other hand, take the paranoid approach everything that has not been witnessed before is considered dangerous. To operate, these systems compare the observed behavior of the system and/or the users against a model of normal (or expected) behavior. Any deviation from the normal behavior is considered an intrusion. Behavior-based systems are often considered complete, that is, all attacks (even previously not known attacks) can be caught. However, the accuracy of these systems is often low. Behavior-based systems need to undergo extensive training sessions to determine what constitutes normal behavior. During this phase these systems tend to generate false alarms at a very high rate. In addition, such systems require periodic on-line retraining. This results in either unavailability of the system or generation of false alarms till such time as the system is retrained.

The Intrusion Detection approach is not a perfect solution for security hardening by at least two concerns. First, it is not possible to make a sound and objective judgment

as to which type of intrusion detection system is better. In addition, an ideal intrusion detection system is the most difficult to build because it needs to address a set of rather tough and often contradictory requirements.

A second concern with intrusion detection systems is that they generate alerts only after they are able to see misused signatures or some deviations from norm. A malicious activity may result from a sequence of perfectly innocuous activities. Intrusion detection systems do not report on these activities mostly to prevent information overload for the system administrator. Thus the intrusion detection system generates an alarm only after the cause for alarm has occurred. In many situations however, this may already be too late.

3.4 Forensic Investigation

Digital forensic investigation is the process of identifying and preserving digital evidences to reveal the fact in computer crime incidences. Works previously done in this area are related to the development of standard investigation process [14, 49, 51], evidence extraction technology [48, 15, 69], and evidence preservation methods. This dissertation scopes itself into the area of evidence extraction technology. Following a large scale computer attack, an investigator often needs to make a reasoned determination of who is responsible for the attack incidence and how the system is compromised in order to assess the actual damage and determine how to reinstate back to the last normal state. Evidence extraction and analysis techniques offered by digital forensic tools can be broadly classified into two groups; pattern-matching-base and statistic-base methodologies.

Pattern matching approaches use tools such as *nmap*, *netcat*, *memdump*, or *CERTs liveview* [1, 5, 8, 11] to perform a keywords-search operation on potential evidence sources including standard operating data or application components for signatures of known attacks such as specific words, file classifications, images, etc. Even if the speed of standard operation is increased, the process remains too manual and relies heavily on

skill and experience of the investigators. In contrast, statistic-based analysis examines the potential evidence sources to identify the content that is distant away from others (e.g. a file that has been processed by unaccustomed process [15] or an image that is subjected to atypical image transformations [48, 69]. However, statistic analysis processes often fail to produce a satisfactory results and require substantial CPU resources to handle a huge amount of data. Hence, the data-reduction method and/or process distribution are often used to reduce case turnaround time.

In summary, much work needs to be done. We can summarize the current state and challenges facing evidence extraction and analysis as the following.

- There is currently no standard model for evidence analysis. The evidence sources often contain raw data which is hard for humans to interpret and there is no minimum requirement for information that needs to be analyzed.

- There is currently no established procedure for filtering and retrieving information from the potential evidence sources other than a sequential backward scan from the most recent entry. System administrators use ad-hoc regular expression searching commands to extract information from the log file. The process is too manual and time-consuming.

- When a security incident occurs, the system administrator does not usually have any information about what to look for in the system log. Most of the time the investigator has to rely on his/her experience or intuition for this purpose. Automated digital forensic tools are needed to release human interventions and give investigators more time to think.

Chapter 4

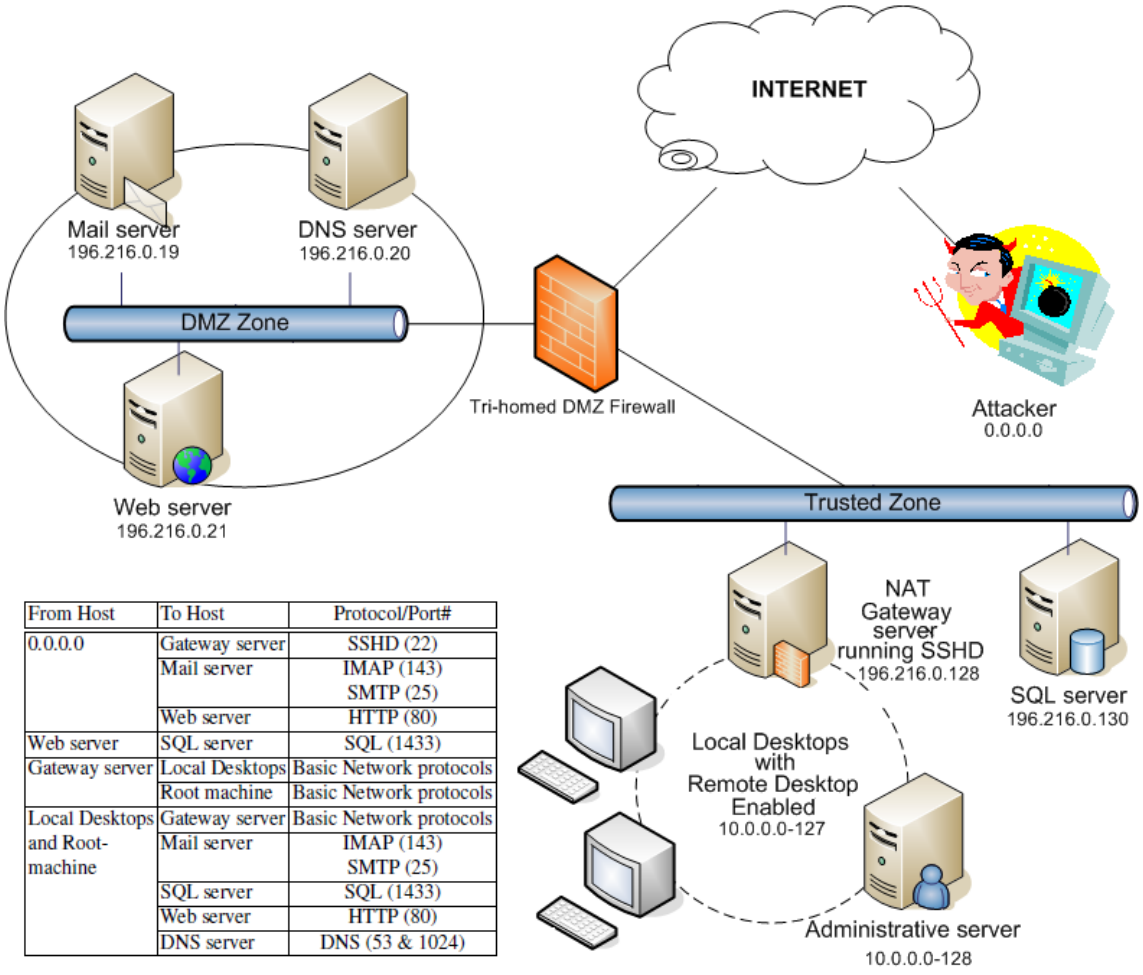
PROBLEM ILLUSTRATION

This research is mainly motivated by the dissatisfaction in mission objectives in risk management. We have found drawbacks in traditional security best-practice. Many security best-practices have identified methodologies to discover vulnerability so as to discover the threats. However, merely determining system vulnerabilities is not enough to help the system administrators reaching appropriate decisions. Security administrators are more interested in determining the best strategy to strengthen the system. Thus, they want a richer threat model to help them understand ways in which the system can be compromised and help them evaluate and manage the risk in a proper manner.

This chapter aims toward three goals. First, we introduce the test-bed network system to illustrate our problem formulation and solution. We, then, address technical challenges in designing security hardening strategies. Lastly, we identify the pitfalls found in the traditional risk management process.

4.1 Test-bed Network

We first analyze the test-bed system for potential security breaches and difficulties in designing the security hardening plan. Our test-bed network consists of eight hosts located within two subnets. The DMZ subnet consists of Web server, Mail server, and DNS server. This subnet is opened to the public. In the second subnet lies the SQL Server and several local desktops including the root machine. This subnet is the trusted zone. Hence, accesses from external sources are restricted. A DMZ tri-homed firewall is installed with policies to ensure that Web server, Mail server, and DNS server are



From Host	To Host	Protocol/Port#
0.0.0.0	Gateway server	SSHD (22)
	Mail server	IMAP (143) SMTP (25)
	Web server	HTTP (80)
Web server	SQL server	SQL (1433)
Gateway server	Local Desktops	Basic Network protocols
	Root machine	Basic Network protocols
Local Desktops and Root-machine	Gateway server	Basic Network protocols
	Mail server	IMAP (143) SMTP (25)
	SQL server	SQL (1433)
	Web server	HTTP (80)
	DNS server	DNS (53 & 1024)

Figure 4.1: Example Network Model

separated from the local network so that if one of these is compromised, the damage will only be limited to the DMZ zone. The access policies are represented as the connectivity shown in the inset table in Figure 4.1.

In terms of embedded security, the firewall has a strong set of policies (shown in the inset table in Figure 4.1) to prevent remote access to internal hosts. In particular, all machines in DMZ zone passively receive the service requests and only respond to the sender as needed. However, in order to accommodate the Web service's transactions, Web server is allowed to send SQL queries to the SQL server located in the trusted zone. Local machines are located behind the NAT firewall so that all communications to external parties are delivered through the Gateway server. In addition, all local desktops including the administrator machine have remote desktop enabled to facilitate remote operations for company's employees who may want to work from remote sites. The remote connections are monitored by the SSHD installed in the Gateway server.

4.2 Traditional Risk Management and Challenge in Risk Management

The typical risk management process consists of risk assessment, risk mitigation, and evaluation. Risk assessment is the process of identifying vulnerabilities presents in the system by the means of network penetration test, vulnerability scanner, security exposure, white box testing, or historical record. Table 4.1 shows the list of vulnerabilities resulted from this process. With further investigation, the security administrator can identify eight possible outcomes. The outcomes are ranked from information leakage to system compromised. Obviously, knowing 'what' outcome could happen is necessary for the security hardening but it is not sufficient for determining an effective security hardening plan. Section 4.3 addresses more detailed discussion covering this issue so as to give a motivation to developing a better threat model.

For the control analysis, we identified 13 security controls capable of reducing risk from initial vulnerabilities. Table 4.2 lists all these controls and their coverage. Since security controls are different in the cost and coverage they provide. It is a big challenge for

Host	Vulnerability	CVE#	Outcomes
Local desktops (10.0.0.1-127)	Remote login	CA 1996-83	remote-2-user
	LICQ Buffer Overflow (BOF)	CVE 2001-0439	remote-2-user
	MS Video ActiveX Stack BOF	CVE 2008-0015	remote-2-root
Admin machine (10.0.0.128)	RPC Marshalling Engine Vulnerability	CVE 2009-0568	local-2-root
Gateway server (196.216.0.128)	OpenSSL uses predicable random	CVE 2008-0166	information leakage
	Heap corruption in OpenSSH	CVE 2003-0693	local-2-root
	Improper cookies handler in OpenSSH	CVE 2007-4752	authentication bypass
SQL Server (196.216.0.130)	SQL Injection	CVE 2008-5416	remote-2-root
Mail Server (196.216.0.19)	Remote code execution in SMTP	CVE 2004-0840	remote-2-root
	Error message information leakage	CVE 2008-3060	account information theft
	Squid port scan vulnerability	CVE 2001-1030	information leakage
DNS Server (196.216.0.20)	DNS Cache Poisoning	CVE 2008-1447	integrity
Web Server (196.216.0.20)	IIS vulnerability in WebDAV service	CVE 2009-1535	remote-2-local authentication bypass

Table 4.1: Intial List of Vulnerabilities in Test-bed Network.

the security administrator to design the cost-effective security controls. The methodology the security administrator often uses is to estimate the risk from the individual vulnerability, prioritize risks by their values, and compare the cost and benefit of implementing a chosen set of security controls. Typically, the level of risk is computed by *Impact* and *Likelihood*.

However, there are some technical challenges in designing the security hardening plan. First, it is impossible to completely eliminate the risk as some vulnerabilities result from design flaws. Thus, it is not an easy task to redesign the system without affecting other business functionalities and there is no guarantee that the new design is not vulnerable to other attacks. Second, the system administrator has to work within a given set of budget constraints which may preclude her from implementing all suggested security controls. Designing a set of security policies for the organizations network safety has considerable implications on the organizations financial throughput.

Lastly, it is not an easy task to find the minimal security hardening plan since there is no one-to-one mapping between vulnerabilities and security controls. Some vulnerabilities can be eliminated by multiple security controls and some security controls, with higher costs, can have more coverage on multiple vulnerabilities. Thus, the system administrator needs to find a trade-off between the cost of implementing a subset of security

Vulnerability	Risk (CVSS)
CVE 2009-0568	10.0
CVE 2003-0693	10.0
CVE 2004-0840	10.0
CVE 2008-0015	9.3
CVE 2008-5416	9.0
CVE 2008-0166	7.8
CVE 2009-1535	7.6
CVE 2001-0439	7.5
CVE 2007-4752	7.5
CVE 2008-3060	7.5
CVE 2001-1030	7.5
CA 1996-0083	7.4
CVE 2008-1447	6.4

Control	Coverage
digital signature	CVE 2008-3060
disable WebDAV	CVE 2009-1535
query restriction	CVE 2008-5416
apply OpenSSH security patch	CVE 2003-0693, CVE 2007-4752
use POP3 instead	CVE 2008-3060
apply MS work around	CVE 2008-0015
disable portscan	CVE 2001-1030
apply MS09-004 work around	CVE 2008-5416
filtering external traffic	CA 1996-83, CVE 2004-0840, CVE 2009-1535
limit DNS access	CVE 2008-1447
encryption	CVE 2008-1447
add Network IDS	CVE 2001-1030, CVE 2008-3060, CVE 2009-0568
add Firewall	CVE 2001-0439

Table 4.2: (left)Vulnerability Sorted by CVSS Risk Score. (right)List of Security Controls and Their Coverages.

hardening measures and the damage that can potentially happen to the system if certain weak spots are left un-plugged in order to maximize the resource utilization. As such, this decision is not a trivial task.

4.3 Problem and Research Motivation

We have found many pitfalls in traditional security best-practice including NIST SP 800, ISO/IEC 27001, and BS7799. The major pitfall arises from the level of abstraction of the standard itself. Most standards define the framework at the abstraction level and leave out the implementation to field practices. As a result, one can easily misinterpret and mislead the abstraction. In this article, we categorize the drawbacks into three major aspects so as to highlight the problem and research motivation.

4.3.1 Pitfalls in Vulnerability Assessment

The major pitfall in traditional risk assessment is found in the process of risk assessment analysis. The threat models resulting from the vulnerability identification process are not rich enough for the security administrators to estimate the risk as well as conducting the risk mitigation analysis. Given the complexity of today's network infrastructure, it is not enough to consider the presence or absence of vulnerabilities in isolation. We have

observed that modern day attacks rely on exploiting multiple vulnerabilities in a correlated manner. Materializing a threat strongly requires the combination of multiple attacks using different vulnerabilities. As such, this requirement is far beyond what typical vulnerability scanners can offer.

To make the above critique more concrete, consider Table 4.1. The table identifies 13 vulnerabilities and 8 possible outcomes ranked from information leakage to system compromise. However, Knowing 'what' vulnerability presents in the network does not always tell tell "how" it is exploited. In fact, more than 20 attacks possible can be derived from the initial 13 vulnerabilities. Figure 4.2 illustrates these attacks. Hence, we can see that the true damage assessment is far more complicate than the one obtained from the traditional vulnerability list and the security mitigation analysis should yield different results from the one that is solely based on eliminating individual vulnerability.

4.3.2 Pitfalls in Risk Assessment Analysis

Many security best-practices estimate the risk by considering vulnerabilities on an individual basis. A network security administrator could be fooled in a situation where individual vulnerability risks are low but can be combined into compromising a critical resource. In other words, the *causal dependency* between vulnerabilities has been ignored in most existing security best-practices.

To illustrate this pitfall, the NIST's SP 800-30 security best-practice is chosen. The SP 800 measures level of risk using the Business Impact Analysis (BIA). In particular, the magnitude of impact is determined by the severity of the adverse outcome and the criticality of the target resources. Let assume that we want to analyze the impact of the vulnerability CVE 2003-0693. According to BIA, the magnitude of impact is computed from the severity of the outcome (in this case, root compromise) and the targeted resource (which is the Gateway server). Even if BIA impact estimation is flawless, the impact estimation of CVE 2003-0693 is underestimated. In fact, there are steps before executing

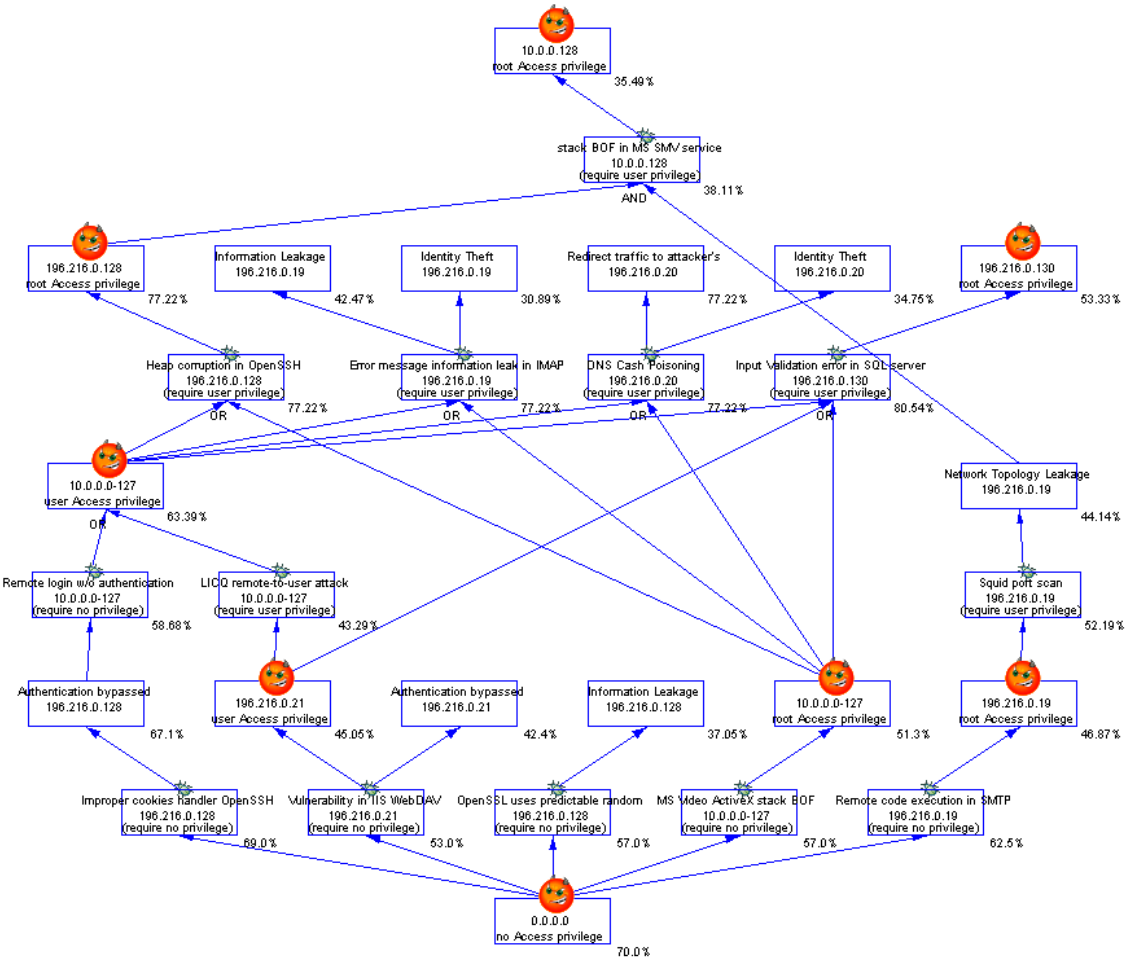


Figure 4.2: Unconditional Probability Distribution of the Test-bed Network

CVE 2003-0693. Fig 4.2 shows three attack scenarios that can compromise the Gateway server. One way is to compromise the local machine by either bypassing the authentication (CA 1996-83) or exploiting LICQ's vulnerability (CVE 2001-0439). Another way is to exploit the ActiveX vulnerability to compromise the local machine and then exploit the local connection to take the Gateway server using CVE 2003-0693. As a result there are several adverse impacts along the path that are overlooked by NIST's best-practice.

Similarly, there is a flaw in the likelihood estimation as well. The likelihood is meant to estimate the probability of the vulnerability exploitation given the attacker's capability, nature of vulnerability, and level of security controls. We discover that node dependency inside the system has a strong influence to the probability estimation. For example, the Remote login (CA 1996-83) execution by its nature is extremely sensitive. Hence, it is given the highest probability by many vulnerability exposure databases. However, in our test-bed network, CA 1996-83 can not be executed unless an attacker successfully bypasses the authentication process. According to the network configuration, the authentication bypass can only be achieved by exploiting CVE 2007-4752. If CVE 2007-4752 is hard to penetrate then it will greatly affect the attacker's capability which, in turn, reduces the attack Likelihood of CA 1996-83. We have not seen such an approach that takes node dependencies into consideration in the likelihood estimation.

In this study, we propose the threat model in Chapter 5 and Chapter 8. The proposed model allows us to understand ways in which the system can be compromised given a set of initial vulnerability, network configuration and initial security measures.

4.3.3 Administrator's Dilemma

Since the risk mitigation analysis takes the result from the risk assessment analysis and we already discussed the flaws in the traditional risk assessment analysis. It is clear that the security administrator will not get the best security hardening plan from a defected input. Beside, the risk mitigation itself is not a trivial task. Hence, it is important to

discuss one of the greatest challenges in risk mitigation analysis known as *Administrator's Dilemma* in this subsection.

The problem facing security managers while working on the risk mitigation analysis is not merely assessing the efficiency given a security plan but rather design the resource-effective security plan that reduces the system risk down to an acceptable level. It is important to note that the complete elimination is usually impractical or close to impossible. Hence, in order to defend against the attacks possible, a security administrator (as a decision maker) has to implement the security hardening plan from a variety of safeguard technologies, each has different cost and security coverage. For example, to defend against the ftp/.rhost exploit, one might choose to apply a security patch, firewall, or simply disable the FTP service. Each choice of action can have a different cost to spend with different outcomes. Besides, some measures have multiple coverage, but with higher costs. A security administrator has to make a decision and assesses the technologies in order to maximize the resource utilization.

At the same time, system administrator has to minimize the total cost of implementing these hardening measures given the costs for individual measures. Assuming that the vulnerability assessment analysis and control analysis are done correctly, finding the cost-effective security plan needs to select a subset of security controls to form a candidate plan. For each plan, assess the effectiveness of the plan and find the cheapest plan that reduces most of the risk. Unfortunately, given N security controls, there are as many as 2^N plans for the security administrator to evaluate. In addition, system administrators have to work within a fixed budget which may be less than the minimum cost of system hardening. Hence, the real problem is how to select a subset of security hardening measures so as to be within the budget and yet minimize the residual damage to the system caused by not plugging all required security risks.

In this study, we formalize the administrator dilemma problem in Chapter 7 as a Multi-Objective optimization problem and ropose the systematic approach to solve the

problem. We develop the heuristic algorithm to reduce the O(2^N) complexity yet capable to give the optimal result. In addition, the proposed methodology also provides a diversity of solutions for security administrator to choose to meet the budget constraints in real-world problems.

Chapter 5

ATTACK TREE MODEL

Attack tree represents all possible ways to achieve multi-stage attack in a network system. Typically, such a model is useful for an enterprise network where the network administrators wish to know the security risks due to the vulnerabilities present in a multi-host network. Of course, scanning for each host can reveal such vulnerabilities in the network, but eliminating all vulnerable software is not always possible. Also, it is not enough to remove the effects of combining multiple vulnerabilities, as some attacks can simply exploit the loop holes at the system architecture level to legitimately cause security violations. Thus, it is important to visualize the big picture of network attack scenarios in order to analyze vulnerabilities in enterprise networks.

In this chapter, we present a formal definition of an attack tree. All formal definitions introduced in this chapter serves as basic building blocks to explain vulnerability analysis, risk assessment, risk mitigation analysis, and forensic analysis in the sub sequence chapters.

5.1 Introduction

We can observe that vulnerabilities present in a network are often exploited in correlation. Given the complexity of today's network infrastructure, materializing a threat usually requires the combination of multiple attacks exploiting different vulnerabilities. Representing different scenarios under which an asset could be damaged thus becomes important for preventive analysis. Such representations not only provide a picture of the possible ways to compromise a system, but could also help determine a minimal set of

preventive actions. Given the normal operational state of a network, including the vulnerabilities present, an attack could possibly open up avenues to launch another attack, thereby taking the attacker a step closer to its goal. The presence of a vulnerability does not imply that it can always be exploited. A certain state of the network, viz. in terms of access privileges or machine connectivity, could be a prerequisite to be able to exploit a vulnerability. Once the vulnerability is exploited, the state of the network can change enabling the attacker to launch the next attack in the sequence. Such a pre-thought sequence of attacks gives rise to an *attack scenario*.

5.2 Description of an Attack Scenario

It is worth noting that the notion of a progressive attack induces a transitive relationship between the vulnerabilities present in the network and can be exploited while deciding on the security measures. Attack graph [12, 38, 47, 62] and attack tree [52, 60] representations have been proposed in network vulnerability management to demonstrate such cause-consequence relationships. The nodes in these data structures usually represent a certain network state of interest to an attacker, with edges connecting them to indicate the cause-consequence relationship. Although different attack scenarios are easily perceived in attack graphs, they can potentially suffer from a state space explosion problem. Ammann et al. [12] identified this problem and proposed an alternative formulation, with the assumption of monotonicity. The monotonicity property states that the consequence of an attack is always preserved once achieved. In other words, if a network attribute becomes true as a result of some attack, it remains so during subsequent attacks. Such an assumption can greatly reduce the number of nodes in the attack graph, although at the expense of further analysis required to determine the viable attack scenarios. An *exploit-dependency graph* [47] can be extracted from their representation to indicate the various conjunctive and disjunctive relationships between different nodes. For the purpose of this study, we adopt the attack tree representation since it presents a much clearer

picture of the different hierarchies present between attacker subgoals. The representation also helps us efficiently calculate the cost factors we are interested in. Different properties of the network effectuate different ways for an attacker to compromise a system. We first define an *attribute-template* that lets us generically categorize these network properties for further analysis.

Definition 1 ATTRIBUTE-TEMPLATE

An attribute-template is a generic property of the hardware or software configuration of a network which includes, but not limited to, the following:

- *system vulnerabilities (which are often reported in the vulnerability database such as BugTraq, CERT/CC, or netcat).*
- *network configuration such as open port, unsafe firewall configuration, etc.*
- *system configuration such as data accessibility, unsafe default configuration, or read-write permission in file structures.*
- *access privilege such as user account, guest account, or root account.*
- *connectivity*

Attribute-template lets us categorize most of the atomic properties of the network that might be of some use to an attacker. For example, *"running SSH1 v1.2.23 on FTP Server"* can be considered as an instance of the system vulnerabilities template. Similarly, *"user access on Terminal*" is an instance of the access privilege template. Such templates also let us specify the properties in propositional logic. We define an *attribute* with such a concept in mind.

Definition 2 ATTRIBUTE

An attribute is a propositional instance of an attribute-template. It can take either a true *or* false *value.*

The success or failure of an attacker reaching his goal depends mostly on what truth values the attributes in a network take. It also lays the foundations for a security manager to analyze the effects of falsifying some of the attributes by using some security policies. We formally define an attack tree model based on such attributes. Since we consider an attribute as an atomic property of a network taking either a *true* or *false* value, most of the definitions are written using propositional logic involving these attributes.

Definition 3 ATTACK

Let S be a set of attributes. We define Att to be a mapping $Att : S \times S \rightarrow \{true, false\}$ *and* $Att(s_{pre}, s_{post}) =$ *truth value of* s_{post}.

- $a = Att(s_{pre}, s_{post})$ *is an attack if* $s_{pre} \neq s_{post} \wedge a \equiv s_{pre} \leftrightarrow s_{post}$. s_{pre} *and* s_{post} *are then respectively called a precondition and postcondition of the attack, denoted by pre(a) and post(a) respectively.*

- $Att(s_{pre}, s_{post})$ *is a* ϕ*–attack if* $\exists$*non-empty* $S' \subset S | Att(s_{pre}, s_{post}) \equiv \bigwedge_i s_i \wedge s_{pre} \leftrightarrow s_{post}$ *where* $s_i (\neq s_{pre}) \in S'$.

An attack relates the truth values of two different attributes so as to embed a cause-consequence relationship between the two. For example (see Figure 5.1), for the attributes $s_{pre} =$ "*vulnerable to MATU FTP attack on machine A*" and $s_{post} =$ "*root compromise on machine A*", $Att(s_{pre}, s_{post})$ is an attack – the FTP buffer overflow attack. We would like to clarify here that the bi-conditional logical connective "$\leftrightarrow$" between s_{pre} and s_{post} does not imply that s_{post} can be set to *true* only by using $Att(s_{pre}, s_{post})$; rather it means that given the FTP BOF attack, the only way to make s_{post} *true* is by having s_{pre} *true*. In fact, $Att($"*vulnerable to ssh BOF attack on machine A*"$,s_{post})$ is also a potential attack. The ϕ–attack is included to account for attributes whose truth values do not have any direct relationship. However, an indirect relationship can be established collectively. For example, the attributes $s_{pre_1} =$ "*running FTP prior to V.1.23 on machine A*" and $s_{pre_2} =$ "*connectivity(machine B, machine A)*" cannot individually influence the truth value of s_{pre}, but can

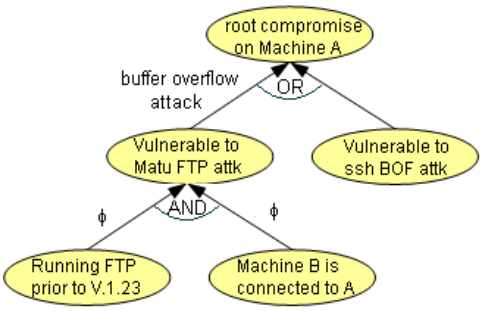


Figure 5.1: Example Attack Tree

collectively make *"vulnerable to MATU FTP attack on machine A" true*, given they are individually *true*. In such a case, $Att(s_{pre_1}, s_{pre})$ and $Att(s_{pre_2}, s_{pre})$ are ϕ–attacks.

5.3 The Formal Definition of an Attack Tree

Attack trees have been previously proposed in [25, 43, 52, 60] as a systematic method to specify system security based on varying attacks. They help organize intrusion and/or misuse scenarios by

1. utilizing known vulnerabilities and/or weak spots in the system, and
2. analyzing system dependencies and weak links and representing these dependencies in the form of an And-Or tree.

For every system that needs to be defended, there is a different attack tree. The nodes of the tree are used to represent the stages towards an attack. The root node of the tree represents the attacker's ultimate goal, namely, to cause damage to the system. The interior nodes, including leaf-nodes, represent possible system states (that is subgoals) during the execution of an attack. Following is the formal definition of an attack tree.

Definition 4 ATTACK TREE

Let A be the set of attacks, including the ϕ–attack. An attack tree is a tuple $AT = (s_{root}, S, \tau, \varepsilon)$, where

1. s_{root} *is an attribute which the attacker wants to become* true.

2. $S = N_{internal} \cup N_{external} \cup \{s_{root}\}$ *is a multi-set of attributes.* $N_{external}$ *denotes the multi-set of attributes* s_i *for which* $\nexists a \in A|\ s_i \in post(a)$. $N_{internal}$ *denotes the multi-set of attributes* s_j *for which* $\exists a_1, a_2 \in A|\ [s_j \in pre(a_1) \wedge s_j \in post(a_2)]$.

3. $\tau \subseteq S \times S$. *An ordered pair* $(s_{pre}, s_{post}) \in \tau$ *if* $\exists a \in A|[s_{pre} \in pre(a) \wedge s_{post} \in post(a)]$. *Further, if* $s_i \in S$ *and has multiplicity n, then* $\exists s_1, s_2, \ldots, s_n \in S|\ (s_i, s_1), (s_i, s_2), \ldots, (s_i, s_n) \in \tau$, *and*

4. ε *is a set of decomposition tuples of the form* $\langle s_j, d_j \rangle$ *defined for all* $s_j \in N_{internal} \cup \{s_{root}\}$ *and* $d_j \in \{AND, OR\}$. d_j *is AND when* $\bigwedge_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ *is* true, *and OR when* $\bigvee_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ *is* true.

Fig. 5.2 shows a simple attack tree for a hypothetical system. In this figure, the goal G0 is the attacker's ultimate objective, namely to breach the system security. Each internal node is a sub-goal state that takes the attacker towards the ultimate goal. The leaf nodes (represented by double rectangles) are the initial vulnerabilities present in the system and constitute stepping stones for the attacks. Branches represent a change of state caused by one or more actions taken by the attacker. Change in state is represented by either AND-branches or OR-branches. Nodes may be decomposed as

1. a sequence of events (attacks), all of which must be achieved for this sub-goal to succeed; this is represented by the events being combined by AND branches at the node; or

2. a set of events (attacks), any one of which occurring will result in the sub-goal succeeding; this is represented by the events being combined by OR branches at the node.

In Definition 4, the set of ordered pairs, τ, reflect the edges in the tree. The existence of an edge between two nodes implies that there is a direct or indirect relationship between their truth values, signified by the decomposition at each node. The *AND* decomposition

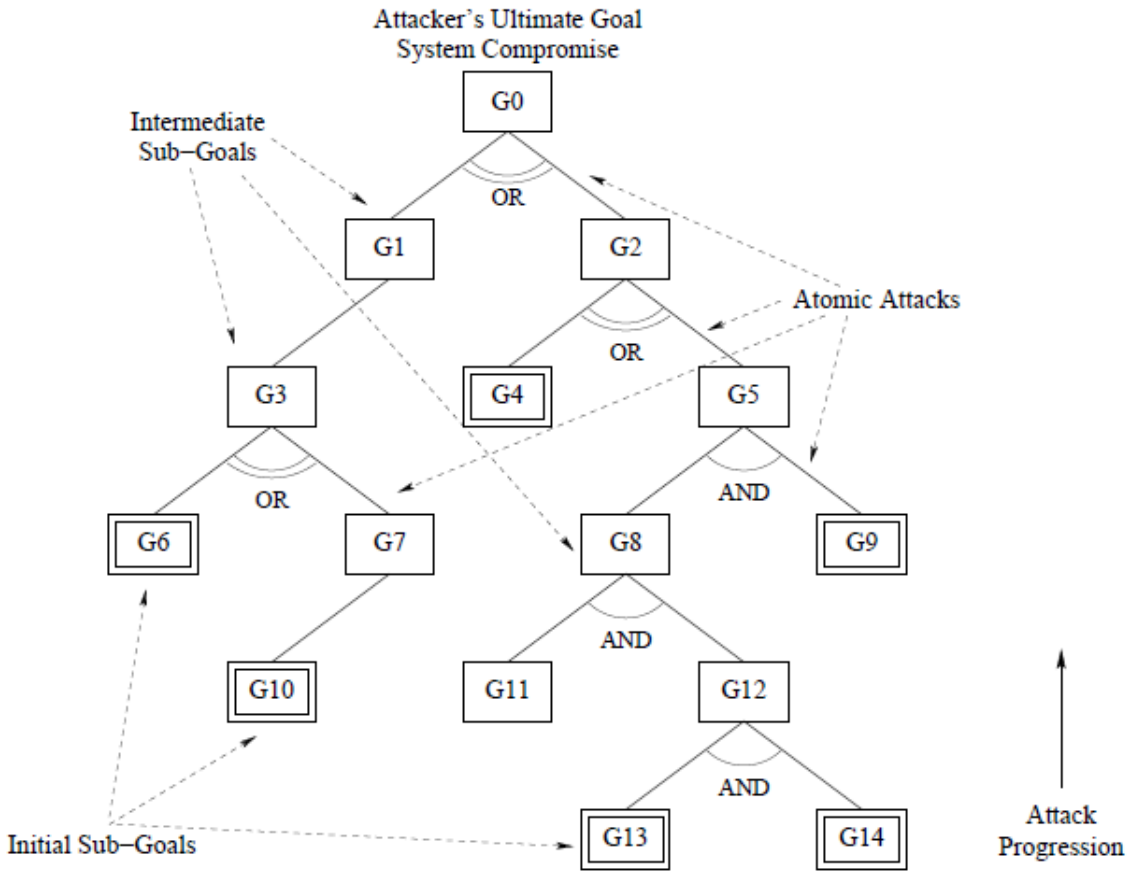


Figure 5.2: Simple Attack Tree Corresponding to a Hypothetical System

at a node requires all child nodes to have a truth value of *true* for it to be *true.* The *OR* decomposition at a node requires only one child node to have a truth value of *true* for it to be *true.* In risk analysis, the attack tree can be augmented to provide a better quantitative representation. In particular, nodes in the attack tree can be assigned with different metrics of interest signifying the amount of value which the analyst wishes to assess. For example, any Boolean value, such as possible or impossible, easy or difficult, detectable or undetectable, etc. can be assigned to the nodes and yields Boolean result to the analysis. Assigning Boolean expression is not the only way an attack tree can be used. It is possible to assign nodes in the tree with a quantitative value such as cost or chance of success. Such kinds of trees help the analyst to determine the cheapest attack

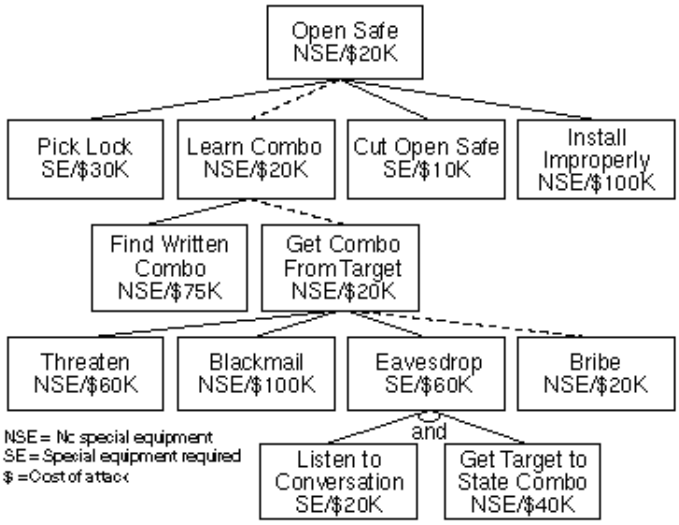


Figure 5.3: Illustration of the Combination of Quantitative and Qualitative Applications on an Attack Tree

scenarios or can answer other what-if analyses. In addition, nodes can be assigned with many different values corresponding to many different variables, with both Boolean and quantitative values. These combinations can answer even ad hoc queries. Figure 5.3 shows an example of an attack tree with various assigned values. These values are used in combination to answer the ad hoc questions such as what are the cheapest ways to attack with no special equipment required (NSE), or what is the most likely undetectable attack etc. We will illustrate these qualitative and quantitative applications of the attack tree model in the subsequent chapters.

Chapter 6

CONSTRUCTION OF ATTACK TREE MODEL

Understanding the big picture of network attack scenarios is an important process for analyzing vulnerabilities in enterprise networks. Such a model is useful for enterprise networks where the network administrators wish to know the security risks due to the vulnerabilities present in the multi-host network.

Constructing attack models by hand, however, is tedious, error-prone, and impractical for attack scenarios larger than a hundred nodes. Previous work on attack tree and attack graph models has not provided an account of the scalability of the graph generating process, and there is often a lack of logical formalism in the representation of attack models. This results in the attack model being difficult to use and be understanded by the users.

We implemented a comprehensive attack tree analysis tool that discovers all possible paths in which the system can be compromised. This is a custom tool written in Java programming language with full-featured user interface and attack tree visualization capabilities. In this chapter, we demonstrate how to prepare the trace information in generating an attack scenario, and how to use the trace to generate a logical attack tree in polynomial time. We generate an attack tree for the test-bed network given in the previous chapter and prove that our attack graph generation algorithm is efficient.

6.1 Introduction

Many researchers in this area [12, 25, 38, 43, 50, 60, 62, 65] have found that there are potentially strong connections between series of attacks in that the consequence of

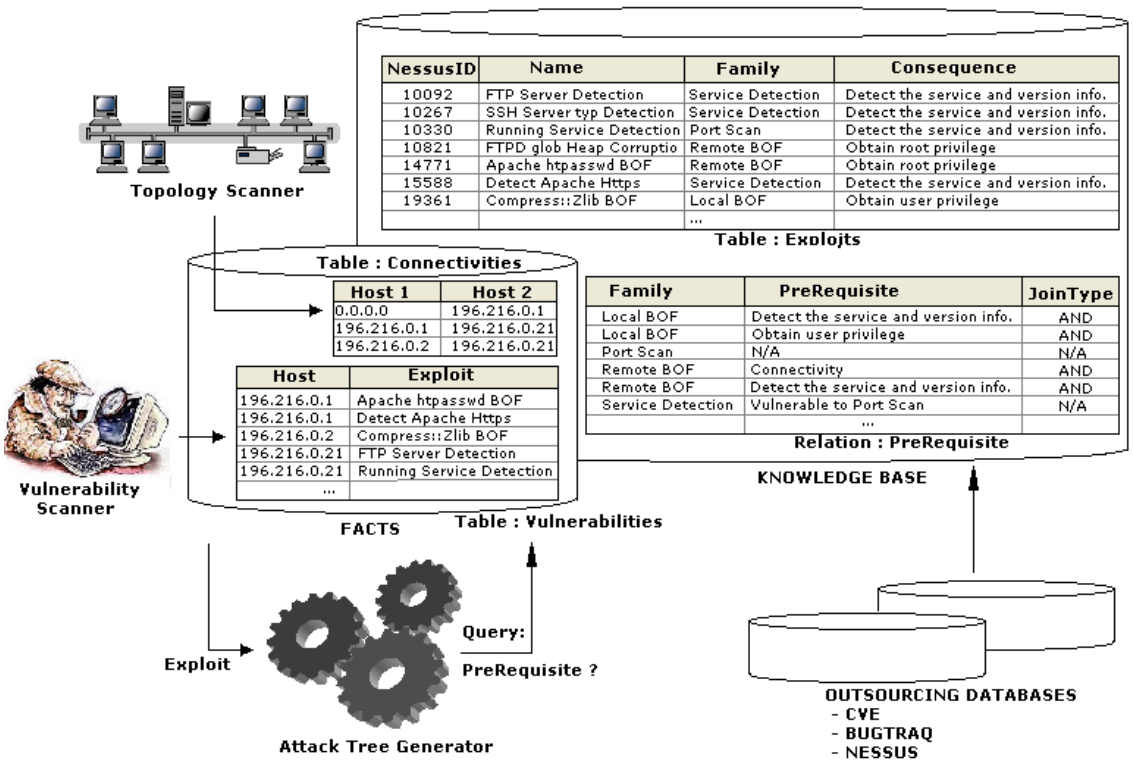


Figure 6.1: Database Architecture of the Attack Tree Risk Analysis Tool

one attack enables the execution of another attack. This relationship is known as prerequisite and consequence relation [12, 38, 46, 50, 53]. Hence, an infinite number of sophisticated attacks can be generated from chaining individual vulnerability exploitations through these relations as one (atomic) attack serves as a prerequisite for another.

Figure 6.1 presents the data diagram showing the flow of information during the generation of an attack tree. The chapter also describes the algorithm to generate an attack tree from the list of vulnerabilities and network topology. The vulnerability list represents facts about flaws in the system design and specifications. This list is prepared by vulnerabilities scanner tools such as *saint* [70], *snort* [54], *nessus* [29], etc. Instances of Vulnerability will be extracted from the report and stored in the SQL database table *vulnerabilities*.

For the Network topology, we can obtain the information from a variety of network topology scanner tools such as TraceRoute [32], Nmap [8], nttlscan [45], etc. Thus the connectivity information between machines in the network of interest can be identified.

Once the facts of the system vulnerabilities and network topology have been established, the tool binds these facts with the relationship information from the *knowledge base repository*. The knowledge base repository stores information relating to the dependencies between vulnerabilities and their exploitations. This information is readily available from many different sources such as the CERT Coordination Center Knowledge base (http://www.cert.org/kb/), the (US) National Vulnerability Database (formerly known as the ICAT database –http://nvd.nist.gov), or the SANS Institute. Based on this information, we can build the inter connection from the characteristics of vulnerability exploitation (e.g., local buffer overflow attack, remote-to-user attack, network sniffing attack, etc.). In particular, the dependencies between vulnerability exploitation and attack pattern (attack category) can be drawn directly from the outsourcing databases. The remaining challenge is how to establish the prerequisite relationship. Section 6.2.1 describes this methodology in more detail.

6.2 Implementation

In this section, we describe the system architecture of the Attack tree risk analysis tool. The system architecture consists of the software architecture and the database architecture. Due to the complexity of the design, Figure 6.3 can only illustrate the software architecture of the tool. The database architecture is shown in detail by Figure 6.2. Note that we shall assume that the readers of this article are already familiar with the terms *vulnerability, exploitation, and attack*. These terms, however, are described in Chapter 12.

This section is organized as follows. We begin with the descriptive design of the database meta model. This consists of the data schema of the Fact module and Knowledge

module. We, then, describe the application architecture which interacts with the database and describe how the attack tree model is generated from a list of initial vulnerabilities and network topology information.

Before moving to the detailed description of meta data design, let's begin with the tuple's convention, as it is the base structure of all elements in this SQL database. A tuple consists of data fields which describe the type of information stored in the data record. Each record has a unique identifier field. The identifier field is written in an underlined italic font. In the case of table association, combination of foreign keys is used to uniquely identify the tuple. The foreign key is identified by italic font. Figure 6.2 shows the data model in the design of the attack visualization tool. The data model is implemented as tables in a regular SQL database. In this model, an *attribute* is a basic building block of vulnerability: a prerequisite, and a consequence of attack. In this model, *attribute* serves as the primitive data type to describe states in attack tree. Specific vulnerability exploitation requires certain conditions for an attacker to execute an attack; this relationship is known as a *prerequisite*. Similarly, an execution of an exploit produce specific outcomes; this relation is called the *consequence*.

Next we describe tables and table associations in the design of the SQL database. The database repository is composed of *the Fact Module* and *Knowledge-base Module*. We begin with the design of *the Fact Module* which comprises of three tables: vulnerabilities, hosts, and connectivity followed by the design of *Knowledge-base Module*.

6.2.1 The Design of Fact Module

We design the Fact Module to represent information that can be drawn directly from the design specification of the evaluated network system. The data module consists of hosts, vulnerabilities, and connectivity tables.

Definition 5 HOST

Host is described by a tuple of { *IP, Name* } *representing the physical machine in the network. A set of host H is a table that consists of 'IP' field and 'Name' field.*

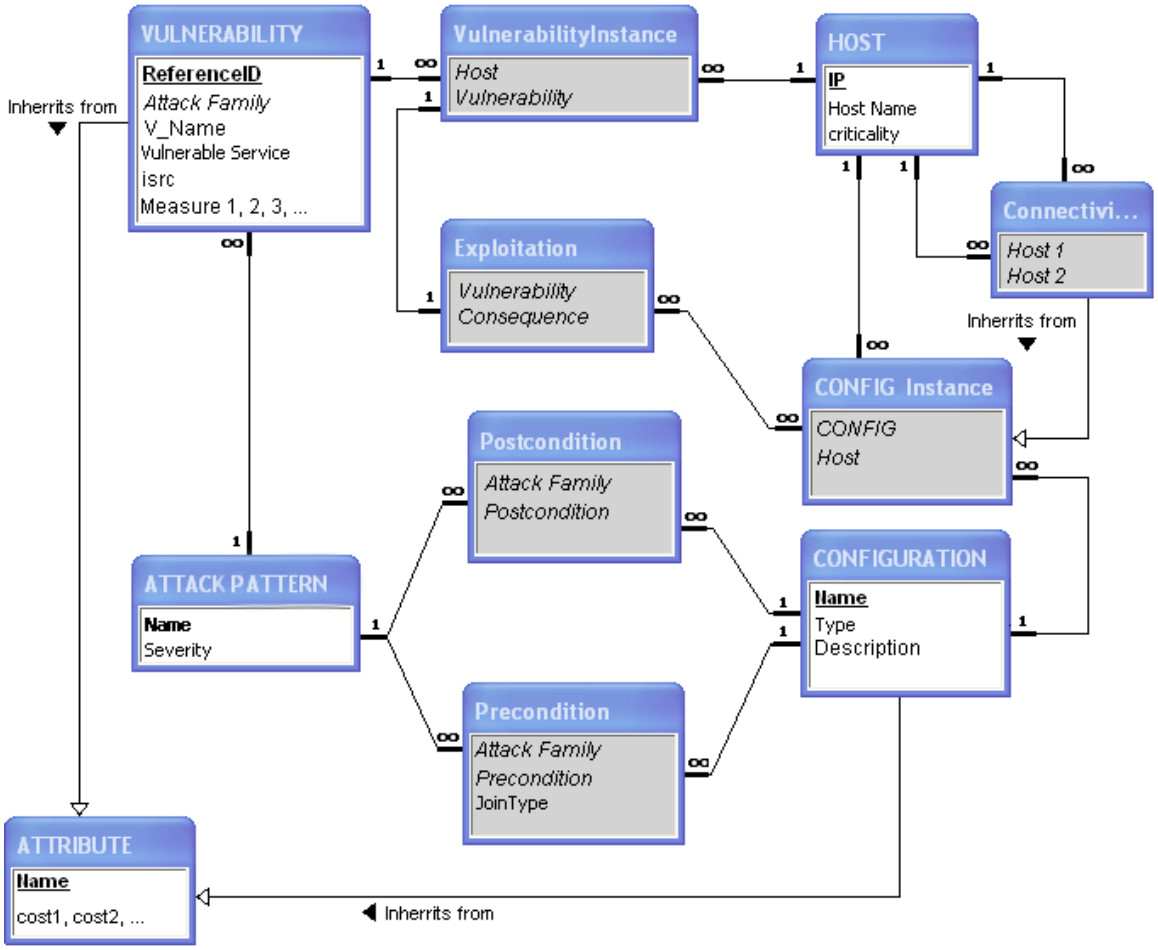


Figure 6.2: The Design of the Database Metada

Definition 6 VULNERABILITY

Vulnerability is an attribute denoted by { *ReferenceID, V_Name, service_name, isrc, measure 1,2,3...* } *representing information about the weakness of the host that may allow an attacker to execute an exploit through a particular input-source isrc. In our current version, we adopt the CVSS scoring system in our risk measurement. Hence, measures 1, 2, 3... correspond to CVSS security metrics of interest.*

Definition 7 CONFIGURATION

System Configuration is an attribute denoted by { *Name, Description* } *describing a system configuration setting required in order to successfully exploit the vulnerability.*

Definition 8 VULNERABILITY INSTANCE

Vulnerability Instance is a table association defined as $\{$ *Vulnerability, Host* $\}$ *representing the facts about local vulnerability present in a specific host machine.*

Definition 9 CONFIGURATION INSTANCE (CONFIG INSTANCE)

Configuration Instance is a table association denoted by $\{$ $Configuration, Host$ $\}$ *representing the facts about local configuration in a particular host. These configurations are either "is required by" or "is a consequence of" a certain vulnerability exploitation.*

Definition 10 CONNECTIVITY

Let H be a set of hosts. Network Connectivity is the table association denoted by $\{$ *host_1, host_2* $\}$. *The connectivity table encodes the connection between host_i and host_j* ($i \neq j$) *in the network topology.*

Note that we impose the strict order in the connectivity tuple, which means that there exists a process in *host_1* capable of sending information to *host_2* but this does not need to have the same capability in the opposite direction. In the attack tree generation algorithm, we are interested in identifying all entities that can reach a given target. Let $\mathcal{H}$ be the host target of interest, the list of hosts that can inject the stream of information to $\mathcal{H}$ (denoted by $CON_{\mathcal{H}}$) can be identified by the following SQL statement.

SQL Statement 6.2.1

$$\begin{array}{rcll} CON_{\mathcal{H}} & = & SELECT & host\_1 \\ & & FROM & ConnectivityTable \\ & & WHERE & host\_2 = \mathcal{H} \end{array}$$

6.2.2 The Design of the Knowledge-base Module

The Knowledge-base Module supports content management of the information about vulnerability exploitations. To the best of our knowledge, more than 40,000 vulnerabilities have been reported, and such reports are growing at an alarming rate. Thus, it is not

efficient to gather information for every exploits. We observe that exploits have common characteristics. Thus, they can be classified by their *attack patterns* (see Chapter 12).

In this project, we employ Common Attack Pattern Enumeration and Classification (CAPEC) to define attack patterns. CAPEC [2] is sponsored by the Department of Homeland Security as part of the Software Assurance strategic initiative of the National Cyber Security Division. An attack pattern is the mechanism to capture common characteristics used in exploiting the system. Exploits in the same attack pattern have a set of preconditions and post conditions in common. Since the number of patterns is much less than the number of exploits, it is more scalable to link dependency among exploits to construct an attack tree by the pattern of attack rather than by the instances of exploit. The prerequisite relationships require human intervention but this is the only part of the tool that requires preparation, and it is more scalable [1] and reusable.

Definition 11 ATTACK PATTERN

Attack pattern is denoted by { *P_Name, Severity* } *representing classification of taxonomy that describes common methods for exploiting the system.*

Definition 12 PRECONDITION AND POSTCONDITION

Given a set of attack patterns P and a set of configurations C. Precondition is a table association between P and C. Precondition is denoted by a tuple { *AttackPattern* :: *P_Name, CONFIGURATION* :: *Name as Precondition , JoinType* }.

Similarly, Postcondition is a table association between P and C denoted by { *Attackpattern* :: *P_Name*, *CONFIGURATION* :: *Name as Postcondition* }. The knowledge about the preconditions and postconditions can be drawn from

[1] especially if we are only concerned with those attack patterns that are only found in the evaluated network

public vulnerability bulletin such as CAPEC (http://capec.mitre.org/), Bugtraq (http://www.securityfocus.com), the (US) National Vulnerability Database (http://nvd.nist.gov), etc.

Definition 13 EXPLOITATION

Given a set of vulnerability instance (VI) and configuration instance (CI). The vulnerability exploitation is a table association between VI and CI. Vulnerability Exploitation is an instantiation that describes an occurrence of state transition between vulnerability and its post-conditions in a particular host machine. An ordered tuple $\{VI, CI\}$ *is an exploit if and only if there exists a vulnerability instance* v *and configuration instance* c *such that* host(v) = host(c).

Relations between exploits, pre-conditions and post-conditions are key components in generating an attack tree. Given an instance of post-condition, we can backtrack from the post-condition table to the vulnerabilities table (through attack pattern) in the Fact Module to determine whether there exists such a vulnerability in the system. The following SQL statement is used by an attack generation Algorithm 1 to identify the vulnerability instances, denoted by $VULN_C$, given a Configuration Instance C as a post-condition.

SQL Statement 6.2.2

$$\begin{array}{rcll} VULN_C & = & SELECT & Vulnerability, Host \\ & & FROM & VulnerabilityInstance\ JOIN\ Exploitation \\ & & & JOIN\ CONFIG\ Instance \\ & & WHERE & [CONFIG\ Instance].CONFIG = C.CONFIG\ \ AND \\ & & & [CONFIG\ Instance].Host = C.Host \end{array}$$

Similarly, given the vulnerability instance $\mathcal{V}$, the following SQL statement is used to identify a set of preconditions at the instance level where an attacker can begin launching an attack from.

SQL Statement 6.2.3

$$
\begin{array}{rcll}
PREC_{\mathcal{V}} & = & SELECT & CONFIG, Host \\
 & & FROM & CONFIG\ Instance\ \ JOIN\ \ CONFIGURATION \\
 & & & JOIN\ \ Pre\ condition\ \ JOIN\ \ ATTACK\ PATTERN \\
 & & & JOIN\ \ VULNERABILITY\ \ JOIN\ \ Vulnerability\ Instance \\
 & & WHERE & [CONFIG\ Instance].Host = \mathcal{V}.Host\ \ AND \\
 & & & [CONFIG\ Instance].Vulnerability = \mathcal{V}.Vulnerability
\end{array}
$$

Note that a single vulnerability exploitation can cause multiple outcomes and that there exist multiple attributes required in executing an exploit. In particular, an exploit may require multiple conditions to be satisfied or only one of its preconditions to be satisfied in order to launch an attack. The attack tree encodes such a relation by a logical operator AND-OR. The field *JoinType* in the Precondition table association captures this aspect. We use the logical operator AND to encode the relationship where multiple conditions are all required in order to execute an exploit and use the logical operator OR to address the case in which an exploit can be executed if at least one of its preconditions is satisfied.

Note that in the case where both AND and OR operators are needed to launch an attack, we use the "dummy node" to partially combine preconditions with AND-decomposition or OR-decomposition as it is appropriate.

6.2.3 Attack Tree Generation Algorithm

We implemented the attack generation tools in Java. The design UML is shown in Figure 6.3. The tool is comprised of a platform-dependent parser, Database Interface, Attack Tree Generator, and ViewPanel. A Parser class is used to read the information about vulnerabilities from public vulnerability bulletin and parse this information to the database. In this project we implemented NessusParser to read data from the nessus vulnerability report. A Database Interface class is used to perform queries to the Database. The Attack Tree Generator class generates a Dependency Graph and the ViewPanel draws an attack tree from the Dependency Graph. Note that the EventHandler and Risk Analyzer classes are not relevant to attack tree generation. Their abstractions, however, will be revisited when we are discussing risk analysis methodologies using the attack tree as the risk model. The attack tree generation process begins with the ultimate goal. In this step,

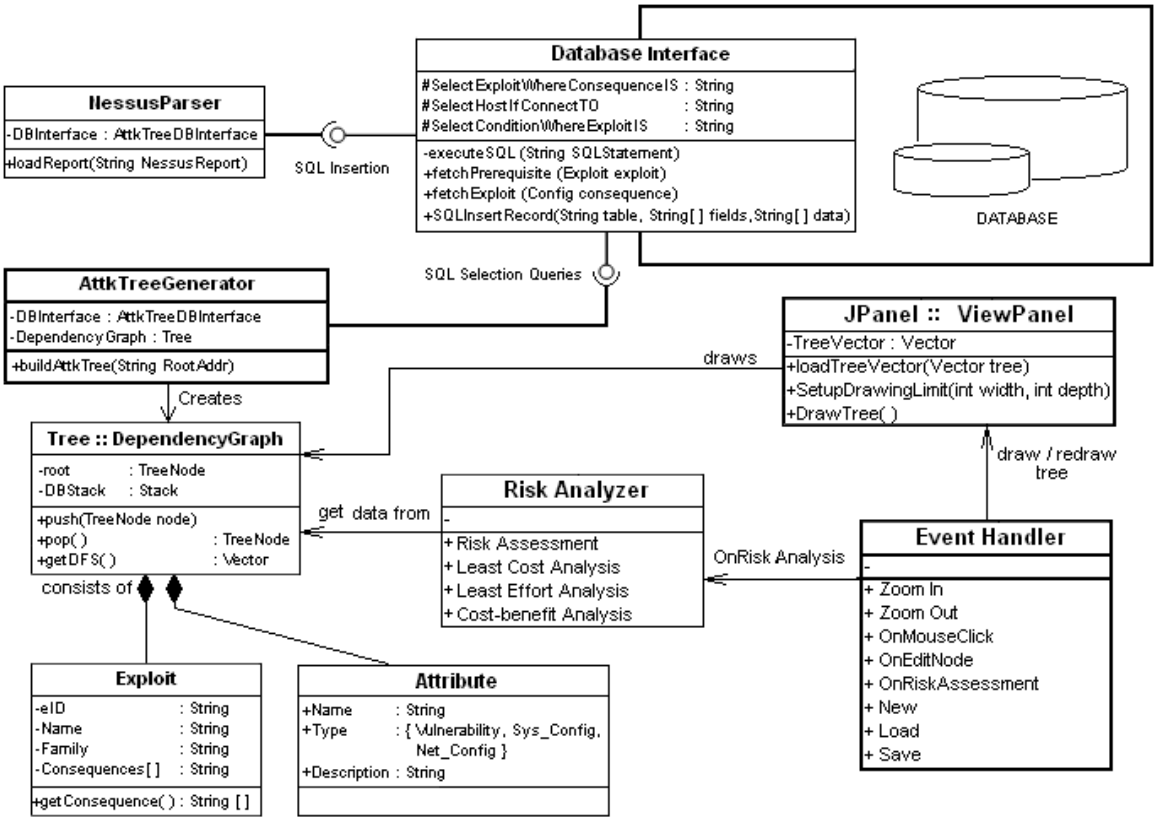


Figure 6.3: The Designed UML Diagram of the Attack Tree Generating Tool

the security analyst selects an ultimate assertion (often "root access privilege") on the machine which she wants to protect most as an ultimate goal.

An attack tree generation pseudo code is shown in Algorithm 1. The algorithm works as follows. First, it executes SQL query 6.2.2 given C = ultimate goal to identify all vulnerability instances that can cause a security breach allowing attackers to reach the ultimate goal. Then for each vulnerability $n \in VULN_C$, the algorithm searches for the preconditions by executing the SQL query 6.2.3. All preconditions discovered in this step are attached to DBStack. In the subsequent iterations, preconditions from the previous iteration then become root nodes of the subtrees. The algorithm repeats the previous steps by executing SQL queries 6.2.2 and 6.2.3 to find vulnerabilities and their preconditions on each subtree. The preconditions found in this iteration become root nodes of the smaller

subtrees, and then the algorithm repeats itself until it can not find any preconditions or vulnerabilities using all available resources. Nodes found in this step become the leaf nodes. The above paragraph describes the typical routine of the algorithm. There is, however, one exception in this routine. If the vulnerability exploitation requires connectivity (see line 21), the algorithm searches the connectivity table (query 6.2.1) to locate machines that can remotely launch an attack to the victim machine. For each machine, the procedure looks for any vulnerability exploitation that can cause the system compromise, puts these exploits in the DBStack, and switches back to the typical search.

6.3 Complexity Analysis

In this subsection, we shall prove that the attack generation algorithm scales with the number of attributes. To begin the complexity analysis, one explicit assumption Algorithm 1 adopts from [12] is *monotonicity*. Monotonicity implies that once a particular state has been reached, it can not be undone. In other words, attack progress never backtracks. With this assumption, we assume the existence of a procedure CHAIN that takes a tree node and returns a set of predecessor nodes by backtracking up to the root node. We use the procedure CHAIN to enforce *monotonicity* constraints so that the path in the attack tree contains no duplication thus it is monotonic. Although Monotonicity may not hold for some cases such cases can be simplified to monotonicity. For example the attacker may execute a denial-of-service attack to temporarily disable the local DNS service and steal sensitive information as described in CVE-2006-0351. The attacker can not deny the fact that an attack progress has exploited the denial-of-service state of attack in an exploit model.

We begin the complexity analysis by assuming the set $A = \{a_1, a_2, \ldots, a_n\}$ be the set of attributes. Set A consists of three compartments: Vulnerability (V), Configuration (F), and Connectivity (C) such that for $V \neq F \neq C$, $A = V \cup F \cup C$. We observe that SQL queries 6.2.1, 6.2.2, and 6.2.3 interact with different compartments of A so that we can safely bound the complexity of SQL executions to $O(N)$ where $N = |A|$.

Algorithm 1 Attack Tree Generation Algorithm

{**Description:**Attack tree generation algorithm generates attack tree from a given ultimate goal (denoted by s_{root}), vulnerability instances, configuration instances, and network configuration attributes.
The algorithm assumes the existence of a procedure called CHAIN that takes a node n corresponding to the state in attack tree and returns a set of nodes resulting from backtracking edges in τ from node n back to s_{root}. }
{**Input:** s_{root}, DATABASE VulnerabilitiesDB}
{**Output:** : The Attack Tree ($AT = (s_{root}, S, \tau, \in)$)}
BEGIN
Let DBStack be a stack that queues nodes in the attack tree that is under exploration.

$VULN_{\mathcal{C}} \leftarrow$ execute SQL statement 6.2.2 given $\mathcal{C} \leftarrow s_{root}$
for all n $\in VULN_{\mathcal{C}}$ | n $\notin$ CHAIN($\mathcal{C}$) **do**
  add (n,$\mathcal{C}$) to τ and add n to S
  DBStack.put(n)
end for
while DBStack is not Empty **do**
  $\mathcal{C} \leftarrow$ DBStack.pop()
  if $\mathcal{C}$ is vulnerability **then**
    $PREC_{\mathcal{V}} \leftarrow$ execute SQL statement 6.2.3 given $\mathcal{C}$
    for all n $\in PREC_{\mathcal{V}}$ | n $\notin$ CHAIN($\mathcal{C}$) **do**
      add (n,$\mathcal{C}$) to τ and add n to S
      DBStack.put(n)
    end for
  else
    if $\mathcal{C}$ = "Connectivity" **then**
      $\mathcal{H} \leftarrow \mathcal{C}$.hostIP
      $CON_{\mathcal{H}} \leftarrow$ execute SQL statement 6.2.1 given $\mathcal{H}$
      for all $h \in CON_{\mathcal{H}}$ **do**
        $VULN_{\mathcal{C}} \leftarrow$ execute SQL statement 6.2.2 given $\mathcal{C} \leftarrow$ { root compromise, h.hostIP } OR { user compromise, h.hostIP }
        for all n $\in VULN_{\mathcal{C}}$ | n $\notin$ CHAIN($\mathcal{C}$) **do**
          add (n,$\mathcal{C}$) to τ and add n to S
          DBStack.put(n)
        end for
      end for
    else
      $VULN_{\mathcal{C}} \leftarrow$ execute SQL statement 6.2.2 given $\mathcal{C}$
      for all n $\in VULN_{\mathcal{C}}$ | n $\notin$ CHAIN($\mathcal{C}$) **do**
        add (n,$\mathcal{C}$) to τ and add n to S
        DBStack.put(n)
      end for
    end if
  end if
end while
END

Note that the monotonicity constraint plays a critical role in guaranteeing that each path contains no duplicate nodes. In other words, it guarantees that the search space is reduced, thus each path has a search complexity of $O(N^2)$. Since there are at most n branches, Algorithm 1 is bounded to $O(N^3)$. Another way to view the algorithm is from the pseudo code. We organize Algorithm 1 into three states: initialization, maintenance, and termination.

- **Initialization:** Prior to the first iteration (line 5 - 11), we execute SQL 6.2.2. The results are stored in DBStack. The overall complexity is $O(N)$.

- **Maintenance:** In each loop iteration (line 12 to 39), a tree node C is removed from the DBStack. Node C can be either vulnerability (line 14), precondition (line 32), or connectivity (line 21). In the cases where C is vulnerability or state condition, the procedure only executes one SQL query and enforces monotonicity constraints to prevent cycle (line 16 and line 33). Both executions are bounded to $O(N)$. In the case where node C is a connectivity, we execute SQL 6.2.1 and then SQL 6.2.2 on each result returned from 6.2.1. In this case, the procedure is bounded to $O(N^2)$.

- **Termination:** The algorithm is terminated when the DBStack is empty. We use the DBStack to store attributes discovered by SQL queries. With monotonicity constraints, the while loop (line 12) which bounds the number of iterations in the algorithm is bounded to $O(N)$. Since, the maximum bound in the Maintenance state is $O(N^2)$, the the overall complexity of the algorithm is $O(N^3)$.

6.4 Tool Illustration

We conducted the experiment on the test-bed network as described in Chapter 4. The test-bed network has 14 vulnerabilities as shown in Table 4.1. In this test, Nessus vulnerability scanner is chosen to produce a proof-of-concept vulnerability report to be parsed to the tool. A sample Nessus report is provided in Figure 6.4. The first step in this

Figure 6.4: The Nessus Report

experiment is to parse the Vulnerability report into the Database and construct the Fact Database. We implemented the NessusParser in Java to read the Nessus Report and parse all vulnerabilities and exploits reported in this document to the Database.

The next step is to build a Precondition and Postcondition table. In this step, National Vulnerability Database (CVE) is chosen as the Nessus report refers to CVE. Since CVE stores data in XML format, we implemented XMLparser using a Java standard XML library. Note that although the information about host machines can be gathered during this process, the connectivity among those hosts is not reported by the Nessus report. Therefore, we had to enter the connectivity information manually. We hope to integrate this tool with the topology scanner and successfully parse the connectivity in an automatic manner in the future. Figure 6.5 shows the screen capture of four major database components: *precondition*, *postcondition*, *VulnerabilityInstance*, and *Connectivity* tables at the end of this stage. Once the database is prepared, we execute an attack tree generation algorithm. In this test, we picked host 10.0.0.128 as the root machine. Algorithm 1 identifies 4 attack

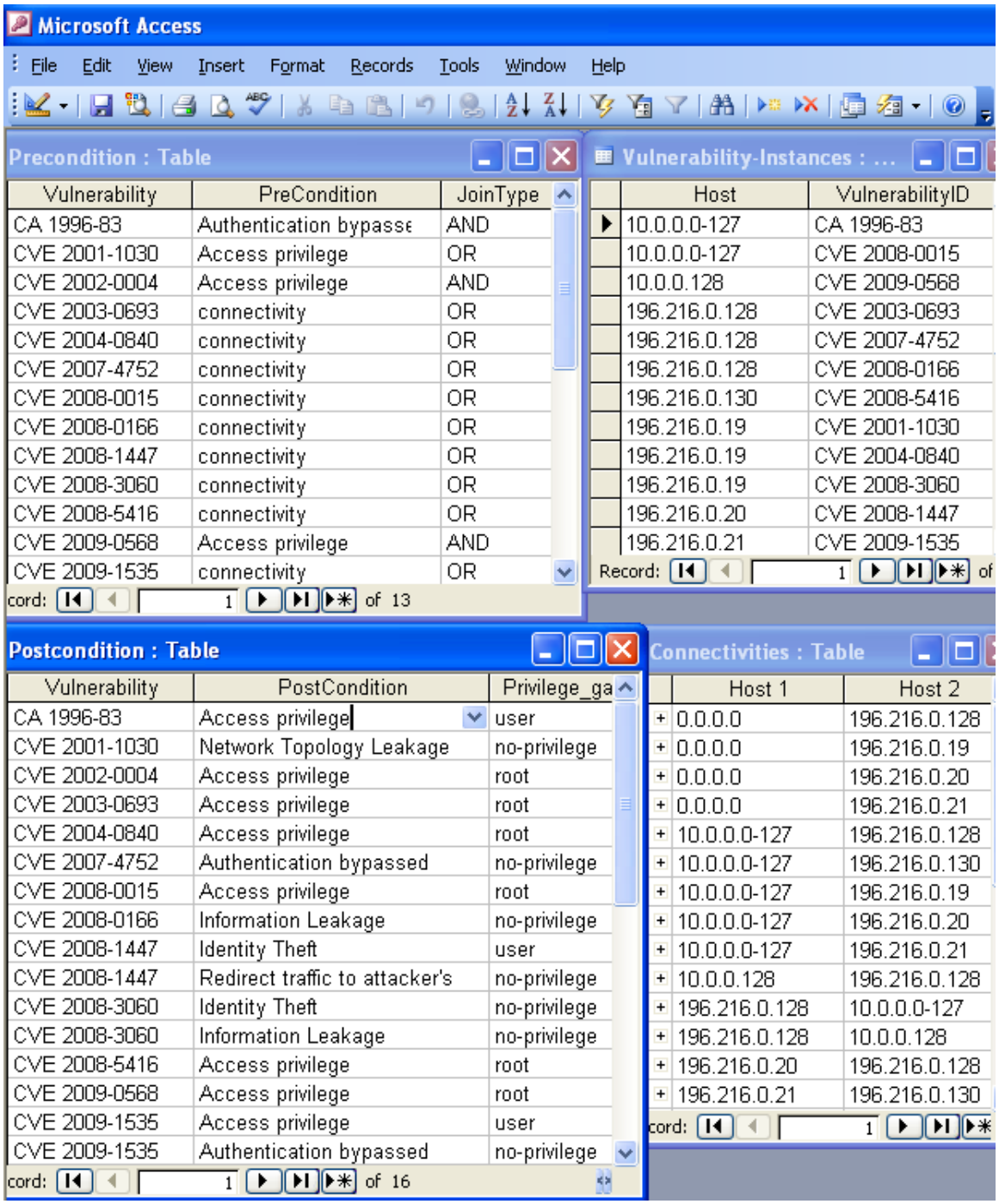


Figure 6.5: The screen capture of the system vulnerabilities database being parsed from the Nessus Report

scenarios in which the root machine can be compromised. Figure 6.6 simplified the risk model of the test-bed network. Note that the actual attack tree model is shown in the upper-left corner.

The original attack tree consists of 33 nodes covering four distinctive attack scenarios targeting the root machine. From the model, the attacker takes advantage of public access of the web server (1296.216.0.21) and exploits the IIS vulnerability (CVE 2009-1535) to gain user privilege on the server (as illustrated in sub scenario A). After compromising the Web server, the attacker can either execute stack BOF (ActiveX vulnerability), LICQ remote-2-user attack, or simply exploit the trust between local machines to bypass the authentication mechanism. Any of these methods allows attackers to compromise the local machine. Then the attacker executes a Heap corruption BOF attack to compromise the Gateway server (196.216.0.128) from the local machine and then uses the Gateway server to compromise the root machine.

An attack tree model resulting from this stage shows ways in which the target system can be compromised. The model itself can be augmented to provide a better quantitative representation and give significant benefit to risk assessment analysis. In the subsequent chapters, we will illustrate how an attack model can be used in various risk assessment analyses.

6.5 Chapter Summary

In this chapter, we propose an attack tree analysis tool that discovers ways in which the system can be compromised and we model network penetration from the attacker's prospective. We implemented this tool in Java language, with full-feature user interface and attack tree visualization capability.

Our attack tree analysis tool models the network topology configuration. This configuration is then subjected to simulated attacks from the exploit database. Exploits are modeled in terms of preconditions and postconditions. When all preconditions for an

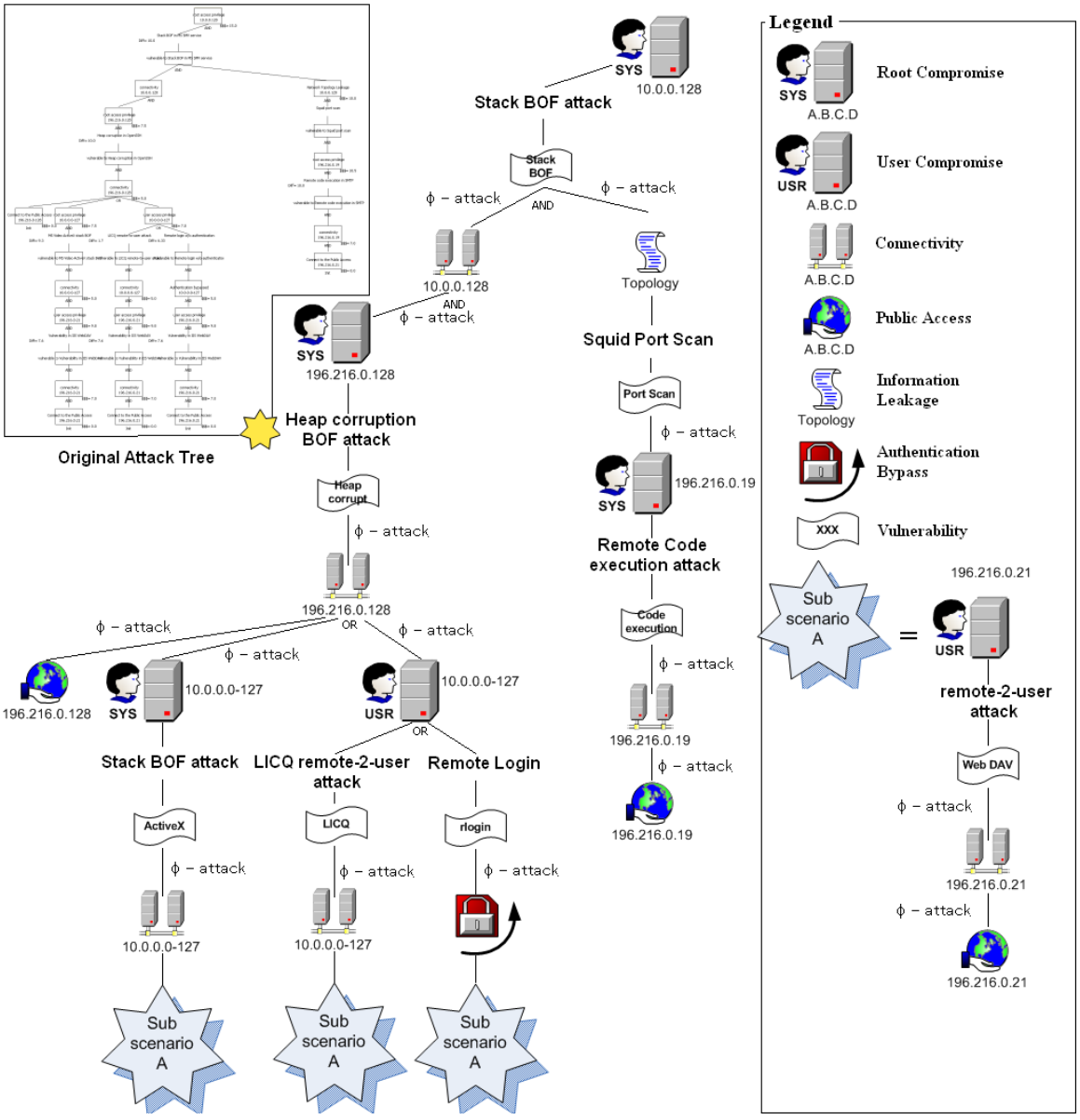


Figure 6.6: The Simplified Attack Tree of the Test-bed Network

exploit are met, the vulnerability exploitation is successful, and its postconditions are induced. These postconditions, in turn, provide a stepping stone for other exploits. The resulting set of exploits, joined by their precondition/postcondition dependencies, form the graphical attack model. The graphical model reveals dependencies among exploits and system configurations thus allowing the security administrator to accurately identify weakness in the network design, predict all possible outcomes, and decide appropriate defensive measures.

Chapter 7

OPTIMAL SECURITY HARDENING ON ATTACK TREE MODELS OF NETWORKS

Researchers have previously looked into the problem of determining whether a given set of security hardening measures can effectively make a networked system secure. Many of them also addressed the problem of minimizing the total cost of implementing these hardening measures given costs for individual measures. However, system administrators are often faced with a more challenging problem since they have to work within a fixed budget which may be less than the minimum cost of system hardening. Their problem is how to select a subset of security hardening measures so as to be within the budget and yet minimize the residual damage to the system caused by not plugging all required security holes. In this work, we formulate the problem as a multi-objective optimization problem and develop a systematic approach to solve the problem using non-dominated sorting genetic algorithm and an attack tree model of the system.

7.1 Introduction

Network-based computer systems form an integral part of any information technology infrastructure today. The different levels of connectivity between these systems directly facilitates the circulation of information within an organization, thereby reducing invaluable wait time and increasing the overall throughput. As an organizations operational capacity becomes more and more dependent on networked computing systems, the need to maintain accessibility to the resources associated with such systems has become

an important necessity. Any weakness or vulnerability that could result in the breakdown of the network has direct consequence on the amount of yield manageable by the organization. This in turn requires the organization to not only consider the advantages of utilizing a networked system, but also consider the costs associated with managing the system. With cost-effectiveness occurring as a major factor in deciding the extent to which an organization would secure its network, it is not sufficient to detect the presence or absence of a vulnerability and implement a security measure to rectify it. Further analysis is required to understand the contribution of the vulnerabilities towards any possible damage to the organizations assets. Often, vulnerabilities are not exploited in isolation, but rather used in groups to compromise a system. Similarly, security policies can have a coverage for multiple vulnerabilities. Thus, cost-effective security management requires researchers to evaluate the different scenarios that could lead to the damage of a secured asset, and then come up with an optimal set of security policies to defend such assets.

Researchers have proposed building security models for networked systems using paradigms like attack graphs [12, 38, 50, 62, 65]and attack trees [25, 43, 52, 60] and then finding attack paths in these models to determine scenarios that could lead to damage. However, merely determining possible attack paths does not help the system administrators much. They are more interested in determining the best possible way of defending their network in terms of an enumerated set of hardening options [47]. Moreover, the system administrator has to work within a given set of budget constraints which may preclude her from implementing all possible hardening measures or even measures that cover all the weak spots. Deciding a set of security policies for the organizations network safety has considerable implications on the organizations financial throughput. Thus, the system administrator needs to find a tradeoff between the cost of implementing a subset of security hardening measures (from a set of measures that can potentially close all attack paths), and the damage that can potentially happen to the system if certain weak spots are left un-plugged. In addition, the system administrator may also want to determine optimal

robust solutions. These are sets of security hardening measures that have the property that even if some of measures within a set fail, the system is still not compromised.

We believe that the problem should be addressed in a more systematic manner, utilizing the different tools of optimization at hand. A decision maker would possibly make a better choice by successively evaluating different levels of optimization possible, rather than accepting a solution from an off-the-shelf optimizer. Towards this end, the current work makes four major contributions. First, we refine and formalize the notion of attack trees so as to encode the contribution of different security conditions leading to system compromise. Next, we develop a model to quantify the potential damage that can occur in a system from the attack modeled by the system attack tree. We also quantify the security control cost incurred to implement a set of security hardening measures. Third, we model the system administrators decision problem as three successively refined optimization problems. We progressively transform one problem into the next to cater to more cost-benefit information as may be required by the decision maker. Last but not the least we discuss our thoughts and observations regarding the solutions, in particular the robust solutions identified by our optimization process, with a belief that such discussion will help the system administrator decide what methodology to adopt.

The rest of the chapter is organized as follows. We discuss some of the previous works related to determining optimum security hardening measures in section 7.2. Section 7.3 gives some background information about multi-objective optimization and the genetic algorithms that we use in this work. In section 7.5, we revisit a test-bed network to illustrate the problem. Section 7.6 presents the cost model use in this problem. The three optimization problems are presented in section 7.7 with results and discussion following in section 7.8. Finally we conclude in section 7.9.

7.2 Related Works

Network vulnerability management has been previously addressed in a variety of ways. Noel et al. used exploit dependency graphs in [47] to compute minimum cost-hardening measures. Given a set of initial conditions in the graph, they computed boolean assignments to these conditions, enforced by some hardening measure, so as to minimize the total cost of those measures. As pointed out in their work, these initial conditions (or leaf nodes in an attack tree) are the only type of network security conditions under our strict control. Hardening measures applied to internal nodes can potentially be by-passed by an attacker by adopting a different attack path. Jha et.al [37] did not consider any cost for the hardening measures. Rather, their approach involved finding the minimal set of atomic attacks critical for reaching the goal and then finding the minimal set of security measures that cover the minimal set of atomic attacks. Although such analysis is meant for providing solutions that guarantee complete network safety, the hardening measures provided may still not be feasible within the financial limitations of an organization. Under such circumstances, a decision maker must perform a cost-benefit analysis to understand the trade-off between hardening costs and network safety. Furthermore, a minimum cost hardening measure set only means that the root goal is safe, and some residual damage may still remain in the network. Owing to these real world concerns, network vulnerability management should not always be considered as a single-objective optimization problem.

A multi-objective formulation of the problem was presented by Gupta et al. in [34]. They considered a generic set of security policies capable of covering one or more generic vulnerabilities. A security policy could also create possible vulnerabilities, thereby resulting in some residual vulnerabilities even after the application of security policies. The multi-objective problem then was to minimize the cost of implementing the security policies, as well as the weighted residual vulnerabilities. However, the authors finally scalarized the two objectives into a single objective using relative weights for the objectives.

7.3 BackGround on Multi-Objective Analysis

In real world scenarios, often a problem is formulated to cater to several criteria or design objectives, and a decision choice to optimize these objectives is sought for. An optimum design problem must then be solved with multiple objectives and constraints taken into consideration. This type of decision making problems falls under the board category of multi-criteria, multi-objective, or vector optimization problem.

Multi-objective optimization differs from single-objective ones in the cardinality of the optimal set of solutions. Single objective optimization techniques are aimed towards finding the global optima; whereas in the case of multi-objective optimization, there is no such concept of a single optimum solution. This is due to the fact that a solution that optimizes one of the objectives may not have the desired effect on the others. As a result, it is not always possible to determine an optimum that corresponds in the same way to all the objectives under consideration. Decision making under such situations thus require some domain expertise to choose from multiple trade-off solutions depending on the feasibility of implementation. Formally, we can state the multi-objective optimization problem (MOOP) as follows:

Definition 14 GENERAL MOOP

Find the vector $\overrightarrow{x^*} = [x_1^*, x_2^*, \ldots, x_n^*]^T$ *which optimizes the M-dimensional vector function*

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})]^T$$

satisfying p inequality and q equality constraints

$$\begin{aligned} g_i(\vec{x}) &\geq 0 \qquad i = 1, \ldots, p \\ h_i(\vec{x}) &= 0 \qquad i = 1, \ldots, q \end{aligned}$$

where $\overrightarrow{x} = [x_1, x_2, \ldots, x_n]^T$ *is the vector of decision variables and M is the number of objectives in the problem.*

Due to the conflicting nature of the objective functions, a simple objective value comparison cannot be performed to compare two feasible solutions to this problem. Most multi-objective algorithms thus use the concept of dominance to compare feasible solutions.

Definition 15 DOMINANCE AND PARETO-OPTIMAL SET

In a minimization problem, a feasible solution vector $\vec{x}$ is said to dominate another feasible solution vector $\vec{y}$ if 1. $\forall i \varepsilon \{1,2,\ldots,M\} \quad f_i(\vec{x}) \leq f_i(\vec{y})$ *and* 2. $\exists j \varepsilon \{1,2,\ldots,M\} \quad f_j(\vec{x}) < f_j(\vec{y})$ *$\vec{y}$ is then said be dominated by $\vec{x}$. If the two conditions do not hold, $\vec{x}$ and $\vec{y}$ are said to be non-dominated w.r.t. each other. Further, the set of all non-dominated solutions obtained over the entire feasible region constitutes the Pareto-optimal set.*

In other words, a Pareto-optimal solution is as good as the other feasible solutions in all the objective functions, and better in at least one of them. Solutions in the Pareto optimal set have no apparent relationship with each other, except for their membership in the set. The surface generated by these solutions in the objective space is called the Pareto-front or Pareto-surface. Fig. 7.1 shows the Pareto front for a hypothetical two-objective problem, with the dominance relationships between three feasible solutions. The classical way to solve a multi-objective optimization problem is to follow the preference-based approach. A relative weight vector for the objectives can help reduce the problem to a single-objective instance, or impose orderings over the preference given to different objectives. However, such methods fail to provide a global picture of the choices available to the decision maker. In fact, the decision of preference has to be made before starting the optimization process. Relatively newer methods have been proposed to make the decision process more interactive.

Evolutionary algorithms for multi-objective optimization (EMO) have been extensively studied and applied to a wide spectrum of real-world problems. One of the major advantages of using evolutionary algorithms for optimization is their ability to scan

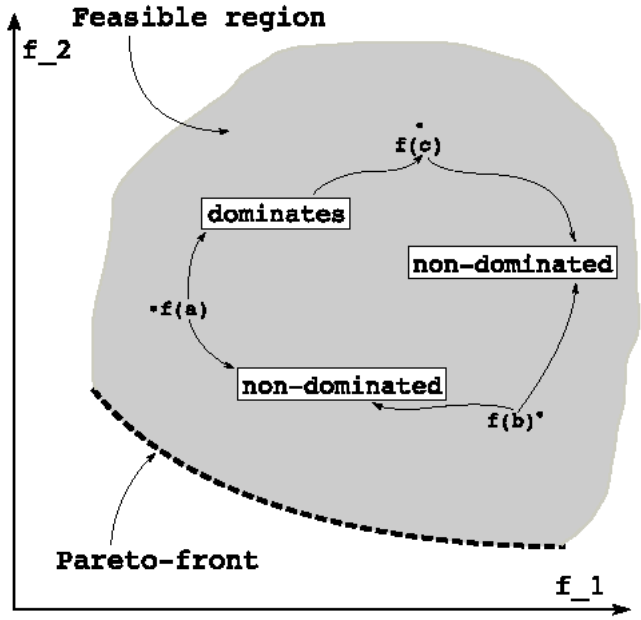


Figure 7.1: Pareto-front for a Hypothetical Two Objective Problem.

through the global search space simultaneously, instead of restricting to localized regions of gradient shifts. An EMO works with a population of trial solutions, trying to converge on to the Pareto-optimal set by filtering out the infeasible or dominated ones. Having multiple solutions from a single run of an EMO is not only an efficient approach, but also helps a decision maker obtain an intuitive understanding of the different trade-off options available at hand. The effectiveness of an EMO is thus characterized by its ability to converge to the true Pareto-front and maintain a good distribution of solutions on the front.

A number of algorithms have been proposed in this context [19, 39]. We employ the Non-dominated Sorting Genetic Algorithm (NSGA-II) [27] for the multi-objective optimization in this study. NSGA-II has gained a wide popularity in the multi-objective optimization community, partly because of its efficiency in terms of the convergence and diversity of solutions obtained, and partly due to its extensive application to solve real-world problems.

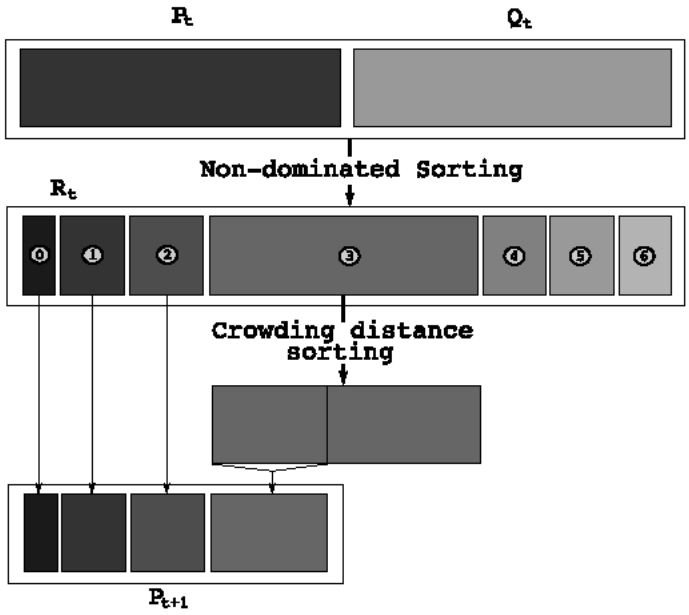


Figure 7.2: One Generation of NSGA-II.

7.4 Non-dominated Sorting GA (NSGA-II)

Similar to a simple genetic algorithm [33], NSGA-II starts with a population P0 of N random solutions. A generation index $t = 0, 1, \ldots, Gen_{MAX}$ keeps track of the number of iterations of the algorithm. Each generation of NSGA-II then proceeds as follows. An offspring population Q_t is first created from the parent population P_t by applying the usual genetic operations of selection, crossover and mutation [33]. The parent and offspring populations are combined to form a population $R_t = P_t \cup Q_t$ of size $2N$. A non-dominated sorting is applied to Rt to rank each solution based on the number of solutions that dominate it. A rank k solution indicates that there are k other solutions that dominate it. In the presence of constraints, the infeasible solutions are given unique ranks higher than the highest feasible solution rank. The ranking starts in ascending order from the infeasible solution with least constraint violation. The population P_{t+1} is generated by selecting N solutions from R_t. The preference of a solution is decided based on its rank: lower the rank, higher the preference. By combining the parent and offspring population,

and selecting from them using a non-dominance ranking, NSGA-II implements an elite-preservation strategy where the best solutions obtained are always passed on to the next generation. However, since not all solutions from Rt can be accommodated in P_{t+1}, a choice is likely to be made when the number of solutions of the currently considered rank is more than the remaining positions in P_{t+1}. Instead of making an arbitrary choice, NSGA-II uses an explicit diversity-preservation mechanism. The mechanism, based on a crowding distance metric [27], gives more preference to a solution with a lesser density of solutions surrounding it, thereby enforcing diversity in the population. The NSGA-II crowding distance metric for a solution is the sum of the average side-lengths of the cuboid generated by its neighboring solutions. Figure 7.2 depicts a single generation of the algorithm. For a problem with M objectives, the overall complexity of NSGA-II is $O(MN^2)$.

7.5 A Simple Network Model

In this chapter, we consider the test-bed network as previously discussed in the Chapter 4. The test-bed network consists of eight hosts located with in two subnets. The DMZ subnet consists of Web server, Mail server, and DNS server. This subnet is opened to the public. The second subnet lies the SQL Server and several local desktops including the root machine. This subnet is the trusted zone. Hence, the access from the external are restricted. A DMZ tri-homed firewall is installed with policies to ensure that Web server, Mail server, and DNS server located in the DMZ network are separated from local network so that if one of these is compromised, the damage will only be limited to the DMZ zone. Figure 7.3 simplifies the the attack tree model of the test-bed network. We use an in-house tool to generate this attack tree. The attack tree consists of 33 nodes covering four distinctive attack scenarios targeting the administrative machine. From the model, attacker takes advantage on public access of the web server (129.216.0.21) and exploits the IIS vulnerability (CVE 2009-1535) to gain user privilege on the server (as illustrate in

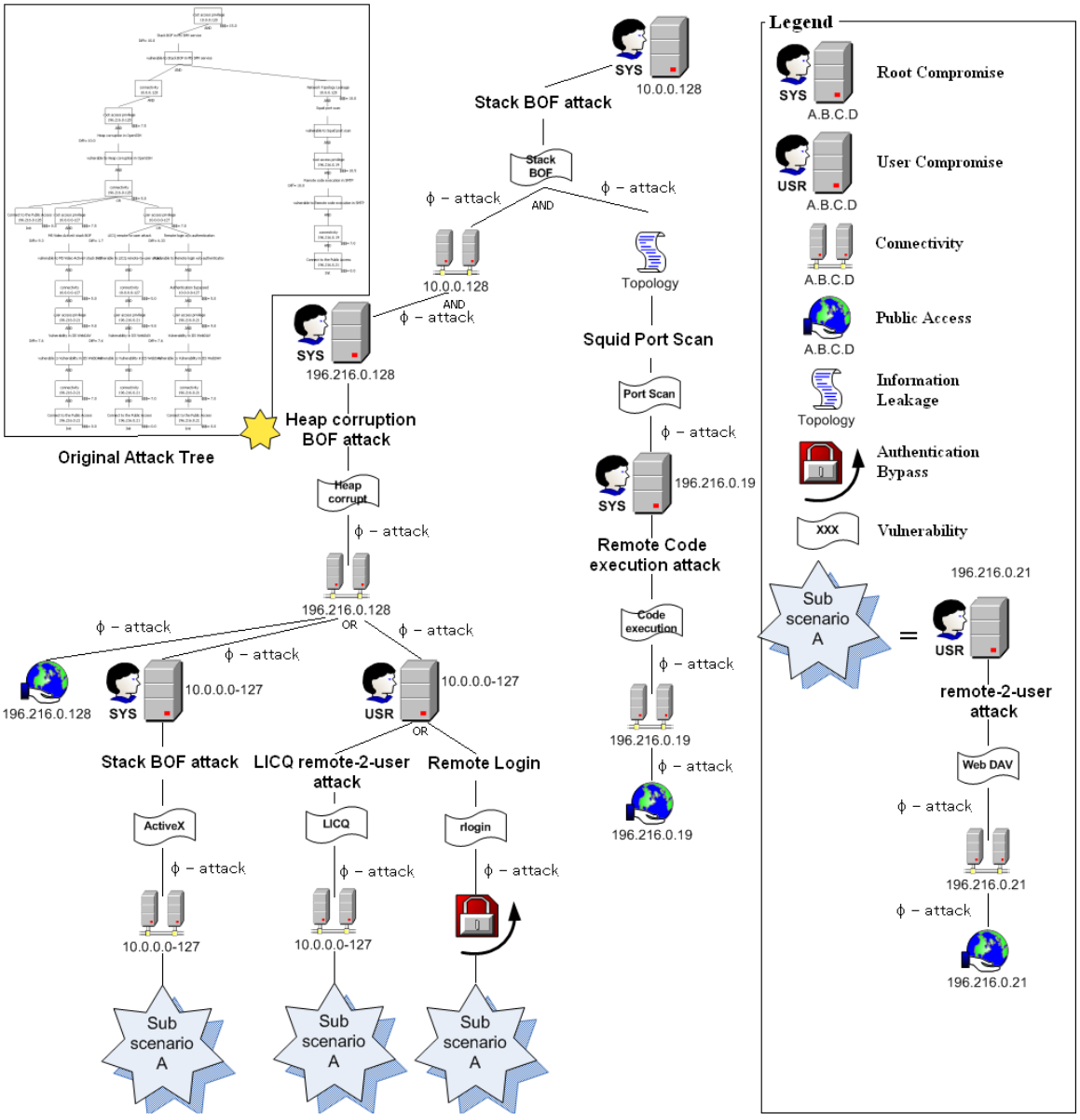


Figure 7.3: The Simplified Attack Tree of the Test-bed Network

sub scenario A). After compromising Web server, attacker can either execute stack BOF (ActiveX vulnerability), LICQ remote-2-user attack, or simply exploits trust between local machines into bypassing the authentication mechanism. Any of these method allows attackers to compromise the local machine. Then attacker executes Heap corruption BOF attack to compromise Gateway server (196.216.0.128) from local machine and then use the Gateway server to compromise the root machine.

According to the definition of an attack tree (see Definition 4). The multi-set S forms the nodes of the tree. The multi-set $N_{external}$ represents the leaf nodes of the tree. These nodes reflect the initial vulnerabilities present in a network and prone to exploits. Since, an attribute can be a precondition for more than one attack, it might have to be duplicated, hence forming a multi-set. The attribute "Authentication bypass" in the example is one such attribute. The set of ordered pairs, τ, reflect the edges in the tree. The existence of an edge between two nodes imply that there is a direct or indirect relationship between their truth values, signified by the decomposition at each node. The *AND* decomposition at a node requires all child nodes to have a truth value of true for it to be true. The *OR* decomposition at a node requires only one child node to have a truth value of true for it to be true. Using these decompositions, the truth value of an attribute $s_j \in N_{internal} \cup \{s_{root}\}$ can be evaluated after assigning a set of truth values to the attributes $s_i \in N_{external}$.

7.6 Cost Model

In order to defend against the attacks possible, a security manager (decision maker) can choose to implement a variety of safeguard technologies, each of which comes with different cost and coverages. For example, to defend against the ftp/.rhost exploit, one might choose to apply a security patch, disable the FTP service, or simply tighten the write protection on the .rhost directory. Each choice of action can have a different cost. Besides, some measures have multiple coverages, but with higher costs. A security manager has to make a decision and choose to implement a subset of these policies in order to maximize the resource utilization.

However, this decision is not a trivial task. Security planning begins with risk assessment which determines threats, loss expectancy, potential safeguards and installation costs. Many researchers have studied risk assessment schemes, including the National Institute of Standards and Technology (NIST). For simplicity, the security manager can choose to evaluate the risks by considering a relative magnitude of loss and hardening costs [16, 40, 64]. However, relative-cost approaches do not provide sufficient information to prioritize security measures especially when the organization faces resource constraints. We implement Butlers multi-attribute risk assessment framework [butler02a, butler02b] to develop quantitative risk assessments for our security optimization. First we define the notion of a security control in the context of the attack tree definition.

Definition 16 SECURITY CONTROL

Given an attack tree $(s_{root}, S, \tau, \varepsilon)$*, the mapping* $SC : N_{external} \rightarrow \{true, false\}$ *is a security control if* $\exists\, s_i \in N_{external} = false$.

In other words, a security control is a preventive measure to falsify one or more attributes so as to stop an attacker from reaching its goal. Further, in the presence of multiple security controls SC_k, the truth value of an attribute $s_i \in N_{external}$ is taken as $\bigwedge_k SC_k(s_i)$. Given a security control SC, the set of all $s_i \in N_{external} \mid SC(s_i = false)$ is called the *coverage* of SC. Hence, for a given set of security controls we can define the coverage matrix specifying the coverage of each control. For a given set of m security controls, we use the boolean vector $\vec{T} = (T_1, T_2, \ldots, T_m)$ to indicate if a security control is chosen by a security manager. Note that the choice of this vector indirectly specifies which attributes in the attack tree would be *false* to begin with.

7.6.1 Evaluating Potential Damage

The potential damage, P_j , represents a unitless damage value that an organization might have to incur in the event that an attribute s_j becomes true. Based on Butlers

framework, we specify below the four steps to calculating the potential damage for an attribute s_j.

1. Identify potential consequences of having a true value for the attribute, induced by some attack. In our case, we have identified five outcomes lost revenue ($$$), non-productive downtime (hrs), damage recovery ($$$), public embarrassment (severity scale) and law penalty (severity scale) denoted by x_{1j} , x_{2j} , x_{3j} , x_{4j} and x_{5j}.

2. Estimate the expected number of attack occurrence, $Freq_j$, resulting in the consequences. A security manager can estimate the expected number of attack from the organization-based historical data or public historical data. [1]

3. Assess a single value function, $V_{ij}(x_{ij})$, for each possible consequence. The purpose of this function is to normalize different unit measures so that the values can be summed together under a single standard scale.

$$V_{ij}(x_{ij}) = \frac{x_{ij}}{Max_j x_{ij}} \times 100 \quad , 1 \leq i \leq 5 \tag{7.1}$$

4. Assign a preference weight factor, W_i, to each possible consequence. The weight factor represents an organizations concerns for different outcomes. A security manager can rank each outcome on a scale of 1 to 100. The outcome with the most concern would receive 100 points. The manager ranks the other attributes relative to the first. Finally, the ranks are normalized and set as W_i.

The potential damage for the attribute can then be calculated from the following equation.

$$P_j = Freq_j \times \sum_{i=1}^{5} W_i V_{ij}(x_{ij}) \tag{7.2}$$

When using an attack tree, a better quantitative representation of the cost is obtained by considering the residual damage once a set of security policies are implemented. Hence,

---

[1] Also known as an incident report published annually in many sites such as CERT/CC or SANS.ORG.

we augment each attribute in the attack tree with a value signifying the amount of potential damage residing in the subtree rooted at the attribute and the attribute itself.

Definition 17 AUGMENTED-ATTACK TREE

Let $AT = (s_{root}, S, \tau, \varepsilon)$ be an attack tree. An augmented attack tree $AT_{aug} = AT \mid \langle I, V \rangle$ is obtained by associating a tuple $\langle I, V \rangle$ to each $s_i \in S$, where

1. *I_i is an indicator variable for the attribute s_i, where*

$$I_i = \begin{cases} 0 & , \quad if \ \ s_i \ \ is \ \ false \\ 1 & , \quad if \ \ s_i \ \ is \ \ true \end{cases}$$

2. *V_i is a value associated with the attribute s_i.*

In this work, all attributes $s_i \in N_{external}$ are given a zero value. The value associated with $s_j \in N_{internal} \cup \{s_{root}\}$ is then computed recursively as follows.

$$V_j = \begin{cases} \sum\limits_{k|(s_k, s_j) \in \tau} V_k & + I_j P_j \quad if \ \ d_j \ \ is \ \ AND \\ \\ \mathit{Max}\limits_{k|(s_k, s_j) \in \tau} V_k & + I_j P_j \quad if \ \ d_j \ \ is \ \ OR \end{cases} \tag{7.3}$$

Ideally, P_j is same for all identical attributes in the multi-set. We took a "panic approach" in calculating the value at each node, meaning that given multiple subtrees are rooted at an attribute with an OR decomposition, we choose the maximum value. The residual damage of the augmented tree is then defined as follows.

Definition 18 RESIDUAL DAMAGE

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon) \mid \langle I, V \rangle$ and a vector $\vec{T} = (T_i), T_i \in \{0,1\}; 1 \leq i \leq m$, the residual damage is defined as the value associated with s_{root}, i.e.,

$$RD(\vec{T}) = V_{root}$$

7.6.2 Evaluating Security Cost

Similar to the potential damage, the security manager first lists possible security costs for the implementation of a security control, assigns the weight factor on them, and computes the normalized value. The only difference is that there is no expected number of occurrence needed in the evaluation of security cost. In our case, we have identified five different costs to implementing a security control installation cost ($$$), operation cost ($$$), system downtime (hrs), incompatibility cost (scale), and training cost ($$$). The overall cost C_j , for the security control SC_j , is then computed in a similar manner as for potential damage, with an expected frequency of 1. The total security cost for a set of security controls implemented is then defined as follows.

Definition 19 TOTAL SECURITY CONTROL COST

Given a set of m security controls, each having a cost C_i ; $1 \leq i \leq m$, and a vector $\vec{T} = (T_i)$, $T_i \in \{0,1\}$, the total security control cost is defined as

$$SCC(\vec{T}) = \sum_{i=1}^{m} (T_i C_i)$$

7.7 Problem Formulation

The two objectives we consider in this study are the total security control cost and the residual damage in the attack tree of our example network model. For the attack tree shown in Figure 7.3, we identified 19 different security controls possible by patching or disabling of different services, as well as by changing file access permissions. These controls are listed in Table 7.7.We also tried to maintain some relative order of importance between the different services in a real world scenario when computing the potential damage and security control costs.

Problem 1 THE SINGLE-OBJECTIVE OPTIMIZATION PROBLEM

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon) \mid \langle I, V \rangle$ and m security controls, find a vector $\vec{T_i^*} = (T_i^*), T_i^* \in \{0,1\}; 1 \leq i \leq m$, which minimizes the function

Security Control	Action	Security Control	Action
SC_1/SC_2	Disable/Patch WebDAV @ 196.216.0.21	SC_{11}	Add Gateway Firewall @ 10.0.0.0/8
SC_3/SC_4	Disable/Patch LICQ @ 10.0.0.0-127	SC_{12}/SC_{13}	Disable/Patch SMTP @ 196.216.0.19
SC_5	Patch Marshalling Engine @ 10.0.0.128	SC_{14}/SC_{15}	Disable/Patch OpenSSH @ 196.216.0.128
SC_6/SC_7	Disable/Patch port scan service @ 196.216.0.19	SC_{16}	Disable remote shell service @ 10.0.0.0-127
SC_8	Disconnect Internet @ 196.216.0.128	SC_{17}	Disconnect Internet @ 196.216.0.19
SC_9	Change to POP3 protocol @ 196.216.0.19	SC_{18}	Patch Active X @ 10.0.0.0-127
SC_{10}	Enforce digital signature @ 196.216.0.19	SC_{19}	Disconnect Internet @ 196.216.0.21

$$\alpha RD(\vec{T}^*) + \beta SCC(\vec{T}^*)$$

where, α *and* β *are preference weights for the residual damage and the total cost of security control respectively,* $0 \leq \alpha, \beta \leq 1$ *and* $\alpha + \beta = 1$.

The single-objective problem is the most likely approach to be taken by a decision maker. Given only two objectives, a preference based approach might seem to provide a solution in accordance with general intuition. However, as we find in the case of our example network model, the quality of the solution obtained can be quite sensitive to the assignment of the weights. To demonstrate this affect, we run multiple instances of the problem using different combination of values for α and β. α is varied in the range of [0, 1] in steps of 0.05. β is always set to $1 - \alpha$.

Problem 2 THE MULTI-OBJECTIVE OPTIMIZATION PROBLEM

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon) \mid \langle I, V \rangle$ *and m security controls, find a vector* $\vec{T}^* = (T_i^*), T_i^* \in \{0,1\}; 1 \leq i \leq m$, *which minimizes the total security control cost and the residual damage.*

The next level of sophistication is added by formulating the minimization as a multi-objective optimization problem. The multi-objective approach alleviates the requirement to specify any weight parameters and hence a better global picture of the solutions can be obtained.

Problem 3 THE MULTI-OBJECTIVE ROBUST OPTIMIZATION PROBLEM

Let $\vec{T} = (T_i)$ *be a boolean vector. A perturbed assignment of radius r,* $\vec{T}_r$, *is then obtained*

by inverting the value of at most r elements of the vector $\vec{(}T)$. *The robust optimization problem can then be defined as follows.*

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon) \mid \langle I, V \rangle$ *and m security controls, find a vector* $\vec{T}^* = (T_i^*), T_i^* \in \{0,1\}; 1 \leq i \leq m$, *which minimizes the total security control cost and the residual damage, satisfying the constraint*

$$\underset{\vec{T}_r^*}{Max}\ RD(\vec{T}_r^*) - RD(\vec{T}^*) \leq D$$

where, D is the maximum perturbation allowed in the residual damage.

The third problem is formulated to further strengthen the decision process by determining robust solutions to the problem. Robust solutions are less sensitive to failures in security controls and hence subside any repeated requirements to reevaluate solutions in the event of a security control failure. We use a simple genetic algorithm (SGA) to solve Problem 1. NSGA-II is used to solve Problem 2 and 3.

The algorithm parameters are set as follows: population size = 200, number of generations = 200, crossover probability = 0.9, and mutation probability = 0.1. We ran each instance of the algorithms five times to check for any sensitivity of the solutions obtained from different initial populations. Since the solutions always converged to the same optima, we dismiss the presence of such sensitivity.

7.8 Results and Discussion

We first study the sensitivity of NSGA-II and SGA to their parameters. Increasing the population size from 200 to 400 gives us a faster convergence rate, although the solutions reported still remains the same. The effect of changing the crossover probability in the range of 0.7 to 0.9 does not lead to any significant change of the solutions obtained. Similar results were observed when changing the mutation probability from 0.1 to 0.05. The solutions also do not change when the number of generations is changed from 150 to 300. Since we did not observe any significant change in the solutions by varying the

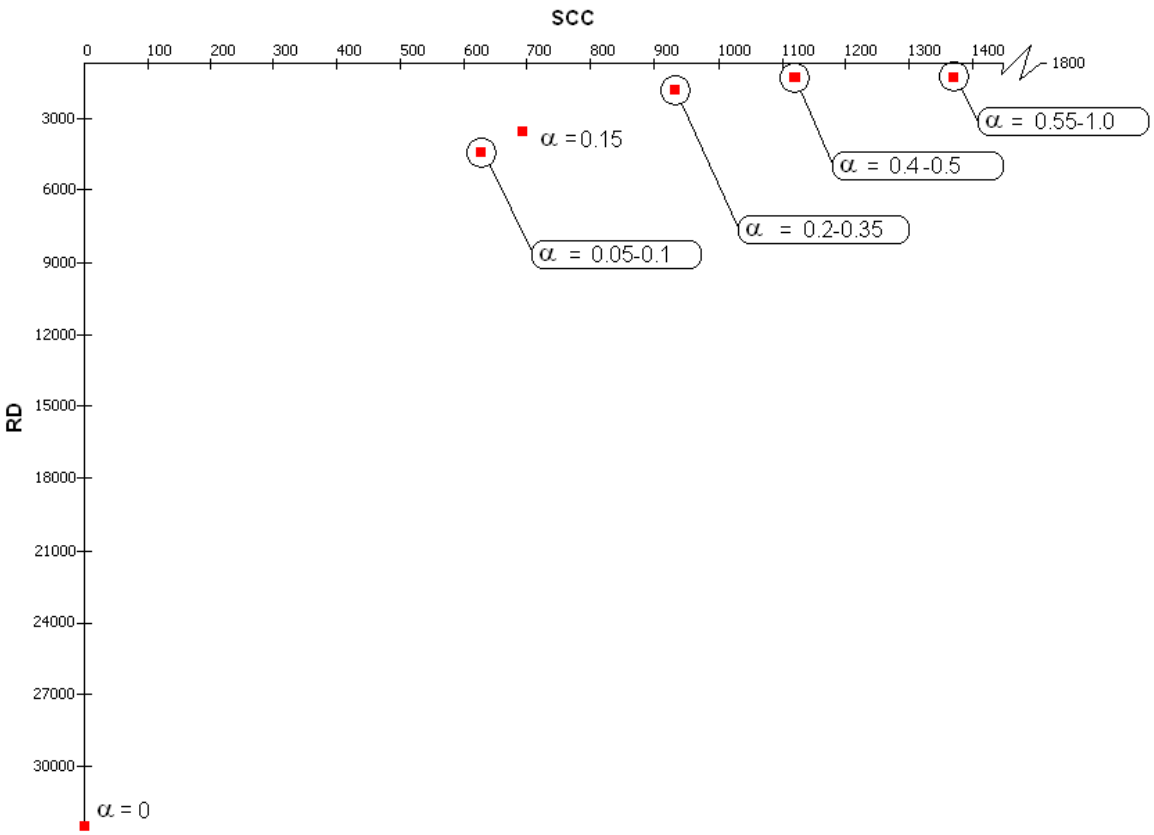


Figure 7.4: SGA Solutions to Problem 1 with α Varied from 0 to 1 in Steps of 0.05.

algorithm parameters, the following results are presented as obtained by setting the parameters as chosen in the previous section. It is usually suggested that the preference based approach should normalize the functions before combining them into a single function. However, we did not see any change in the solutions in the normalized version of Problem 1. Figure 7.4 shows the solutions obtained from various runs of SGA in Problem 1 with varying α. A decision maker, in general, might want to assign equal weights to both the objective functions, i.e. set $\alpha = 0.5$. It is clear from the figure that such as assignment do not necessarily provide the desired balance between the residual damage and the total security control cost. Furthermore, such balance is also not obtainable by assigning weight values in the neighborhood of 0.5.

The solutions obtained are quite sensitive to the weights, and in this case, much higher preference must be given to the total security control cost to find the other possible solutions. Since the weights do not always influence the objectives in the desired manner, understanding their effect is not a trivial task for a decision maker. It is also not possible to always do an exhaustive analysis of the affect of the weights on the objectives. Given such situations, the decision maker should consider obtaining a global picture of the trade-offs possible. With such a requirement in mind, we next consider Problem 2. The corresponding solutions from Problem 1 are shown in Table 7.8. The two solutions

Table 7.1: Security Controls Obtained for Problem 1 with Different α and β.

α	Optimum security controls	RD	SCC
0	*null*	(31777)	0
0.05, 0.1	SC_2, SC_{13}, SC_{15}	(3700)	625
0.2 to 0.35	SC_1, SC_6,SC_{13},SC_{15}	(1094)	930
0.4 to 0.5	SC_1,SC_6, SC_9, SC_{13}, SC_{15}	(612)	1135
0.55,1.0	SC_1, SC_6, SC_{12}, SC_{15}	(566)	1370

corresponding to $\alpha = 0.2$ and 0.35, including any other solutions in the vicinity, are likely candidates for a decision makers choice as it reduces risk down to an acceptable level with a reasonable cost. Unlike the single objective approach, where determining such vicinal solutions could be difficult, the multi-objective optimization approach clearly revealed the existence of at least one such solution.

Figure 7.5 shows the solutions obtained from a single run of NSGA-II on Problem 2. NSGA-II reported all the solutions obtained from multiple runs of SGA, as well as 12 more solutions. Interestingly, there exists no solution in the intermediate range of (0, 600] for security control cost. This inclination of solutions towards the extremities of the residual damage could be indicative of the non-existence of much variety in the security controls under consideration. Most of the security controls for the example network involve either the disabling or patching of a service, resulting in a sparse coverage matrix.

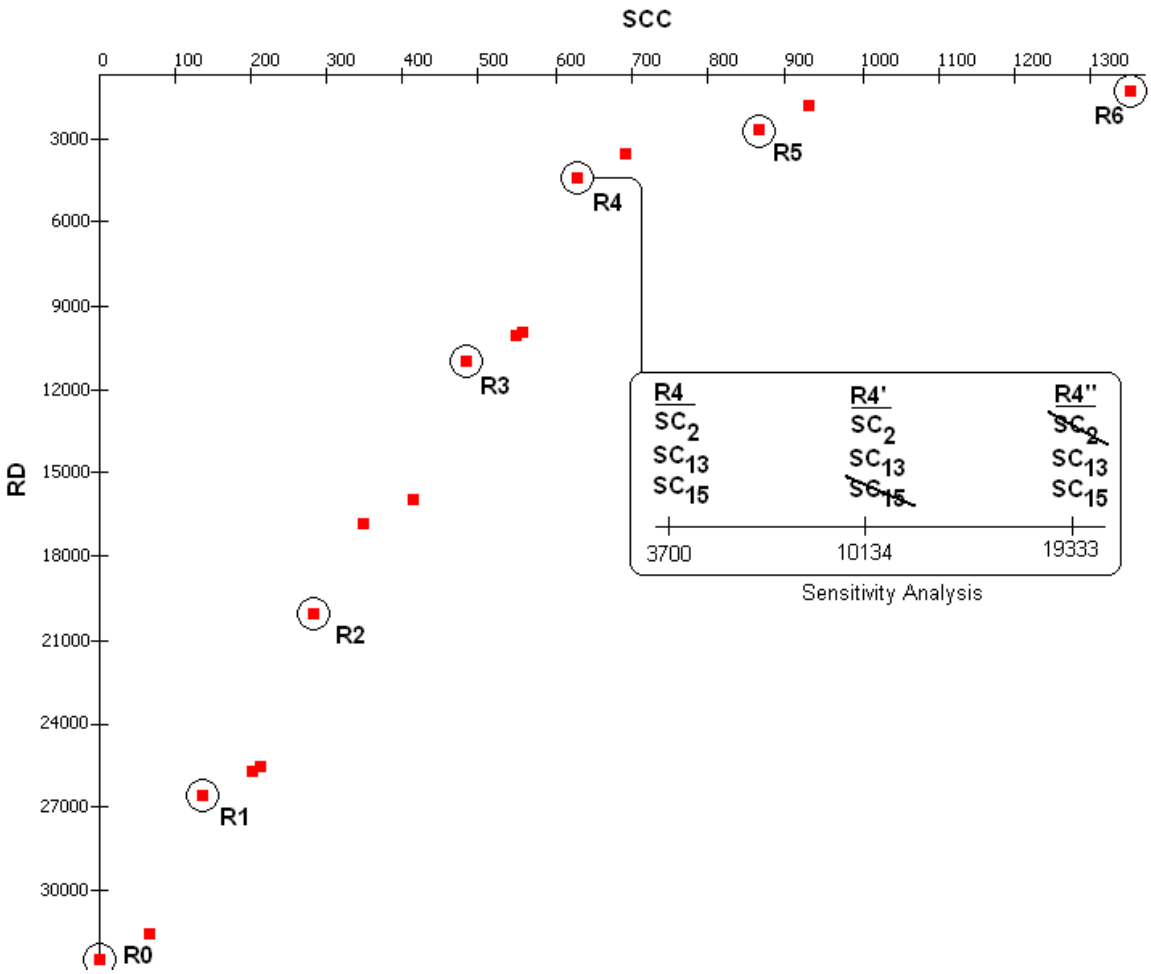


Figure 7.5: NSGA-II Solutions to Problem 2 and Sensitivity of a Solution to Optimum Settings.

For a more "continuous" Pareto-front, it is required to have security controls of comparative costs and capable of covering multiple services. A real-world scenario would likely have a good mixture of both local and global security controls, in which case, such gaps in the Pareto-front will be unlikely. Once the decision maker has a better perspective of the solutions possible, further analysis of the solutions may be carried out in terms of their sensitivity to security control failures. Such sensitivity analysis is helpful in not only reducing valuable decision making time, but also to guarantee some level of fault tolerance in the network. Figure 7.5 shows the sensitivity of the solutions R4 to a failure

in one of the security controls corresponding to the solution. This solution, with security controls SC_2, SC_{13} and SC_{15}, will incur a high residual damage in the event of a failure of SC_2. Thus, a decision maker may choose to perform a sensitivity analysis on each of the solutions and incorporate the results thereof in making the final choice. However, the decision maker then has no control on how much of additional residual damage would be incurred in the event of failure. Problem 3 serves the requirements of this decision stage by allowing the decision maker to specify the maximum allowed perturbation in the residual damage. It is possible to specify the scope of failure the radius r within which the decision maker is interested in analyzing the robustness of the solutions. Because of the sparse nature of the coverage matrix, we set the perturbation radius r to 1. Also, we are mostly interested in obtaining solutions that are fully robust, meaning the residual damage should not increase. However, we can not found alternate solution unless we allow the residual damage tolerance > 500, hence we set D to 1000. Figure 7.6 shows the solutions obtained for this problem. The solutions to Problem 3 reveals that none of the optimum solutions, except the trivial zero SCC solution, previously obtained is fully robust, even for a single security control failure. Such insight could be of much value for a decision maker when making a final choice. Table 7.8 shows the security controls corresponding to the selected robust solutions.

Table 7.2: Robust Solutions Obtained by NSGA-II with r = 1.

	Robust-optimum security controls	RD	SCC
R'0	*null*	(31777)	0
R1	SC_3,SC_6	(30666)	145
R2	SC_8, SC_9,SC_{14}, SC_{15}	(24992)	460
R3	SC_1, SC_{16}	(14428)	730
R4	SC_2, SC_6, SC_{14}, SC_{15}	(9359)	805
R5	SC_2, SC_6, SC_{12}, SC_{13}, SC_{14}	(2282)	1275
R6	SC_1, SC_6, SC_{12}, SC_{13}, SC_{14}	(566)	1515

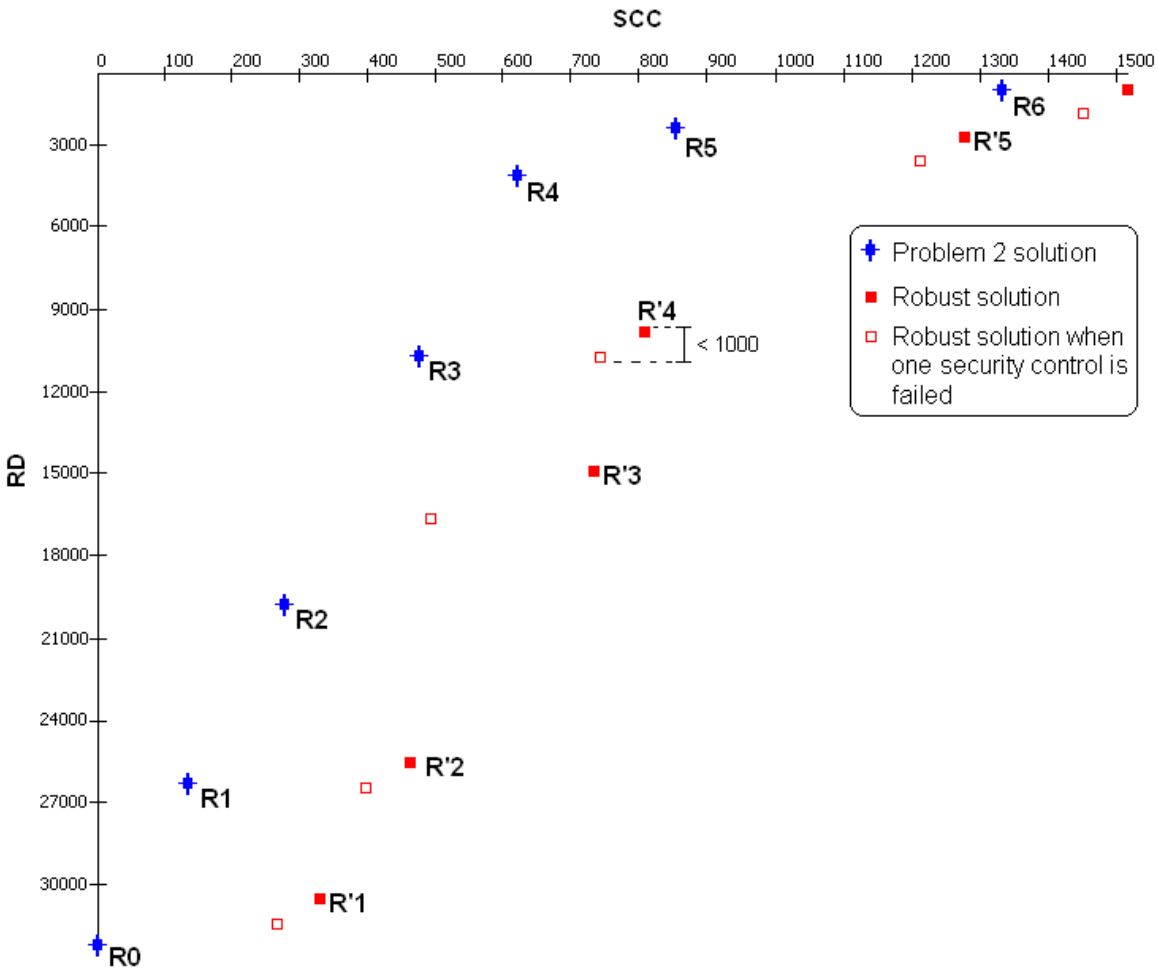


Figure 7.6: NSGA-II Solutions to Problem 3 with D = 1000 and r = 1. Problem 2 solutions are also shown for comparison.

7.9 Chapter Summary

In this paper, we addressed the system administrators dilemma, namely, how to select, when needed, a subset of security hardening measures from a given set so that the total cost of implementing these measures is not only minimized but also within budget and, at the same time, the cost of residual damage is also minimized. We formulated the problem as a multi-objective optimization problem and used non-dominated sorting genetic algorithm to solve it. One important contribution of our solution is the use of an attack tree model of the network to drive the solution. Without the attack tree modeling, the optimization problems would be mostly flat in structure. The choice of security controls would then be determined by the number of coverage they provide and the cost to do so. By using an attack tree in the problem, we were able to better guide the optimization process by providing the knowledge about the attributes that make an attack possible. Quite often, a policy that disables a single attribute is enough to forbid an attack scenario. Vulnerability management would not be of much practical use without integrating such attack modeling approaches in the optimization procedures. Further, a systematic analysis enabled us to approach the problem in a modular fashion, providing added information to a decision maker to form a concrete opinion about the quality of the different trade-off solutions possible.

The cost model that we adopted in this paper is somewhat simplistic. We assumed that from a cost of implementation perspective the security measures are independent of each other when in real life they may not be so. In addition, we have assumed that the system administrators decision is in no way influenced by an understanding of the cost to break the system. Finally, there is a dynamic aspect to the system administrators dilemma. During run time the system administrator may need to revise her decision based on emerging security conditions. In future we plan to refine our models to incorporate these scenarios.

Chapter 8

DYNAMIC SECURITY RISK ASSESSMENT AND MITIGATION USING BAYESIAN ATTACK GRAPHS

Security risk assessment and mitigation are two vital processes that need to be executed to maintain a productive IT infrastructure. On one hand, models such as attack graphs and attack trees have been proposed to assess the cause-consequence relationships between various network states, while on the other hand, different decision problems have been explored to identify the minimum-cost hardening measures. However, these risk models do not help reason about the causal dependencies between network states. Further, the optimization formulations ignore the issue of resource availability while analyzing a risk model. In this paper, we propose a risk assessment framework using Bayesian networks that enable a system administrator to quantify the chances of network compromise at various levels, and show how to use this information to develop a security mitigation and management plan. This risk model lends itself to dynamic analysis during the deployed phase of the network. A multi-objective optimization platform provides the administrator with all trade-off information required to make decisions in a resource constrained environment.

8.1 Introduction

Traditionally, information security planning and management for an organization begins with risk assessment that determines threats to critical resources and the corresponding loss expectancy. A number of researchers have proposed risk assessment methods by building security models of network systems, using paradigms like attack graphs

[12, 38, 50, 62, 65] and attack trees [25, 43, 52, 60], and then finding attack paths in these models to determine scenarios that could lead to damage. However, a majority of these models fail to consider the attacker's capabilities and, consequently, the likelihood of a particular attack being executed. Without these considerations, threats and their impact can be easily misjudged.

To alleviate such drawbacks, Dantu et al. [24] propose a probabilistic model to assess network risks. They model network vulnerabilities using attack graphs and applied Bayesian logic to perform risk analysis. Liu and Man [41] use Bayesian networks to model potential attack paths in a system, and develop algorithms to compute an optimal subset of attack paths based on background knowledge of attackers and attack mechanisms. In both Dantu et al.'s and Liu and Man's works, nodes in the attack graph are assigned a probability value that describes the likelihood of attack on a node. They compute the likelihood of system compromise by chaining Bayesian belief rules on top of the assigned probabilities. The organizational risk is then computed as the product of the likelihood of system compromise and the value of expected loss. The major problem with both these works is that they do not specify how the conditional probability value of an attack on each node is computed. Further, these works do not consider the problem of optimal risk mitigation.

System administrators are often interested in assessing the risk to their systems and determining the best possible way to defend their network in terms of an enumerated set of hardening options. Risk assessment methods such as those discussed earlier have been adopted by researchers to determine a set of potential safeguards, and related security control installation costs. Noel et al. uses exploit dependency graphs to compute minimum-cost hardening measures [47]. Given a set of initial conditions in the graph, they compute boolean assignments to these conditions, enforced by some hardening measures, to minimize the total cost. Jha et al. [38] determine the minimal set of attacks critical for reaching a goal and then find the minimal set of security measures that cover this set of attacks.

While such cost analysis is useful, they miss out on one major issue. The system administrator often has to work within a given set of budget constraints which may preclude her from implementing all possible hardening measures or even measures that cover all the weak spots. Thus, the system administrator needs to find a trade-off between the cost of implementing a subset of security hardening measures (from a set of measures that can potentially close all attack paths) and the damage that can be potentially inflicted on the system after the security decision has been made (the residual damage). Dewri et al. [30] first formulated this so-called "system administrators' dilemma" (discussed above) as a series of multi-objective optimization problems. The solutions to these problems allow one to select a subset of hardening measures so that the total cost of implementing them is not only minimized but also within a fixed budget and, at the same time, the residual damage is minimized. One of the significant contributions of this work is the development of an attack tree model of network risks in order to drive the solution methodology. The attack tree model is able to better guide the optimization process by providing knowledge about the attributes that make an attack possible. While this work makes significant contribution towards appreciating the security planning process as something beyond simple risk assessment, it has one significant shortcoming. The authors' modeling of the problem is a static one. There is however a dynamic aspect to the security planning process. For every attack, there is a certain probability of occurrence that can change during the life time of a system depending on what the contributing factors for the attack are and how they are changing. During run time, the system administrator may need to revise her decision based on such emerging security conditions. Dewri et al.'s attack tree model does not allow such dynamic security planning. In order to address these limitations, the current work makes five major contributions.

- We propose an alternative method of security risk assessment that we call *Bayesian Attack Graph*s (BAGs). In particular, we adapt the notion of Bayesian belief networks so as to encode the contribution of different security conditions during sys-

tem compromise. Our model incorporates the usual cause-consequence relationships between different network states (as in attack graphs and attack trees) and, in addition, takes into account the likelihoods of exploiting such relationships.

- We propose a method to estimate an organization's risk from different vulnerability exploitations based on the metrics defined in the Common Vulnerability Scoring System (CVSS) [59]. CVSS is designed to be an open and standardized method to rate IT vulnerabilities based on their base, temporal and environmental properties.

- We develop a model to quantify the expected return on investment based on a user specified cost model and likelihoods of system compromise.

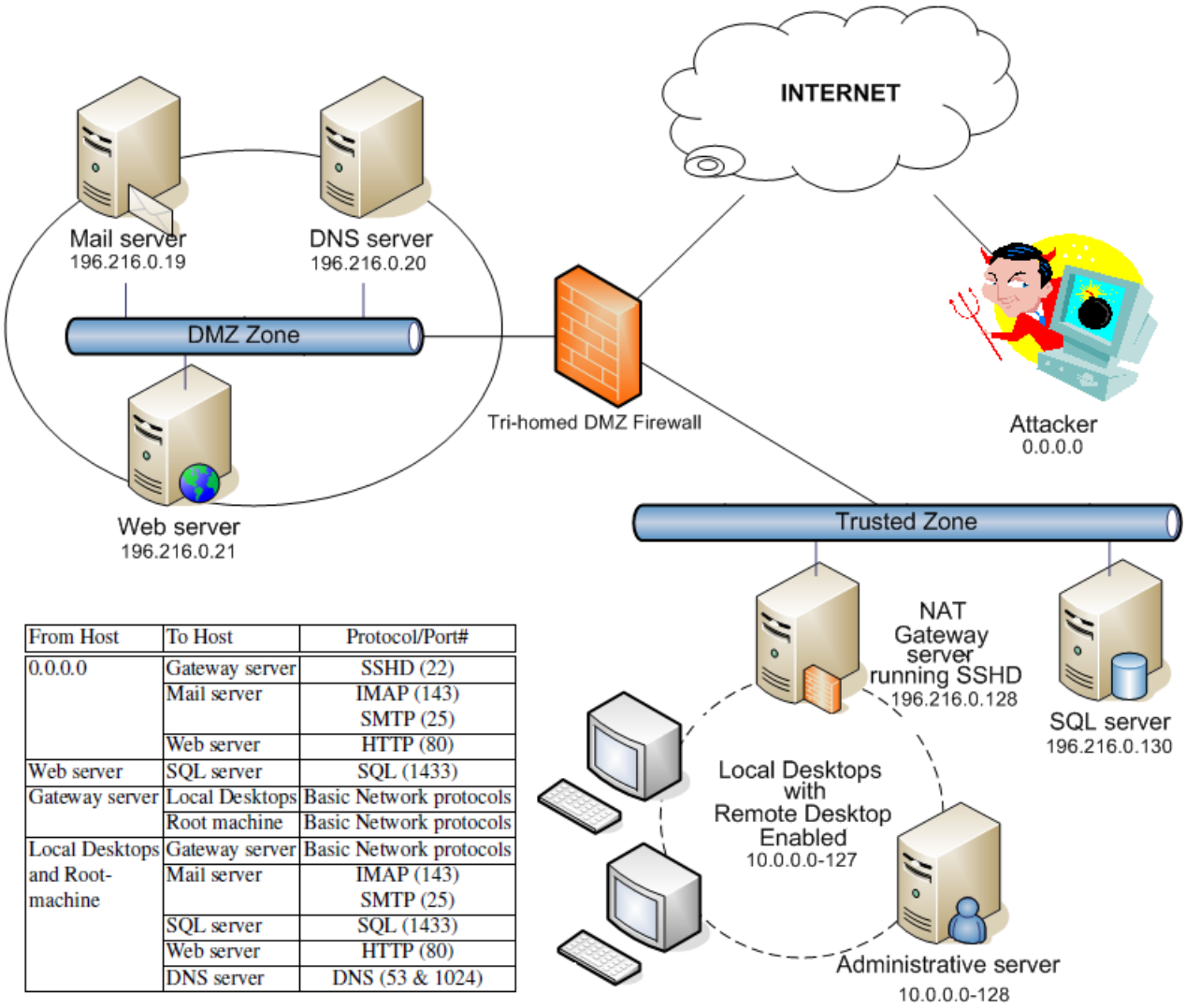
- We model the risk mitigation stage as a discrete reasoning problem and propose a genetic algorithm to solve it. The algorithm can identify optimal mitigation plans in the context of both single and multi-objective analysis.

- Last, but not the least, we discuss how the above contributions collectively provide a platform for static and dynamic analysis of risks in networked systems.

The rest of the paper is organized as follows. The test network used to illustrate our problem is shown in Section 4.1. Section 8.3 presents the formalism for a Bayesian Attack Graph model. The likelihood estimation method in static and dynamic scenarios is discussed in Section 8.4. The risk mitigation process along with the expected cost computations is presented in Section 8.5. Results and discussion are presented in Section 8.6. Finally, we conclude the paper in Section 8.7.

8.2 A Test Network

We revisit the test-bed network previously discussed in the Chapter 4. Figure 8.1 recaptures the test network. The network consists of eight hosts located within two subnets. A DMZ tri-homed firewall is installed with preset policies to ensure that the Web



From Host	To Host	Protocol/Port#
0.0.0.0	Gateway server	SSHD (22)
	Mail server	IMAP (143) SMTP (25)
	Web server	HTTP (80)
Web server	SQL server	SQL (1433)
Gateway server	Local Desktops	Basic Network protocols
	Root machine	Basic Network protocols
Local Desktops and Root-machine	Gateway server	Basic Network protocols
	Mail server	IMAP (143) SMTP (25)
	SQL server	SQL (1433)
	Web server	HTTP (80)
	DNS server	DNS (53 & 1024)

Figure 8.1: Test-bed Network Model.

server, Mail server and the DNS server, located in the DMZ network, are separated from the local network so that the damage will only be limited to the DMZ zone if one of these servers is compromised. The firewall has a strong set of policies (shown in the inset table) to prevent remote access to the internal hosts. In particular, all machines in the DMZ zone passively receive service requests and only respond to the sender as needed. However, in order to accommodate Web service's transactions, the Web server is allowed to send SQL queries to the SQL server located in the trusted zone on a designated channel. Local machines are located behind a NAT firewall so that all communications to external parties are delivered through the Gateway server. In addition, all local desktops, including the administrator machine, have remote desktop enabled to facilitate remote operations for company employees working from remote sites. The remote connections are monitored by SSHD installed in the Gateway server.

A list of initial vulnerabilities/attack templates in this test network is listed in Table 8.1. Further scrutiny of this initial list using a vulnerability database reveals that eight malicious outcomes are possible in this network. However, the list of vulnerabilities alone cannot suggest the course of actions that lead to these outcomes, or accurately assess the casualty of each outcome as it may have involved other damages along the way. These vulnerabilities produce more than 20 attack scenarios with different outcomes, ranging from information leakage to system compromise. Moreover, two of these scenarios use machines in the DMZ zone to compromise a local machine in the trusted zone.

8.3 Modeling Network Attacks

We use a Bayesian belief network to model network vulnerabilities. We extend the notion of Bayesian networks as presented by Liu and Man [41] to encode the contributions of different security conditions during a system compromise. We term such a Bayesian network as a *Bayesian Attack Graph* (BAG). Different properties of the network effectuate different ways for an attacker to compromise a system. We first define an

Host	Vulnerability	CVE#	Attack Template
Local desktops (10.0.0.1-127)	Remote login LICQ Buffer Overflow (BOF) MS Video ActiveX Stack BOF	CA 1996-83 CVE 2001-0439 CVE 2009-0015	remote-2-user remote-2-user remote-2-root
Admin machine (10.0.0.128)	MS SMV service Stack BOF	CVE 2008-4050	local-2-root
Gateway server (196.216.0.128)	OpenSSL uses predicable random Heap corruption in OpenSSH Improper cookies handler in OpenSSH	CVE 2008-0166 CVE 2003-0693 CVE 2007-4752	information leakage local-2-root authentication bypass
SQL Server (196.216.0.130)	SQL Injection	CVE 2008-5416	remote-2-root
Mail Server (196.216.0.19)	Remote code execution in SMTP Error message information leakage Squid port scan vulnerability	CVE 2004-0840 CVE 2008-3060 CVE 2001-1030	remote-2-root account information theft information leakage
DNS Server (196.216.0.20)	DNS Cache Poisoning	CVE 2008-1447	integrity
Web Server (196.216.0.20)	IIS vulnerability in WebDAV service	CVE 2009-1535	remote-2-local authentication bypass

Table 8.1: Initial List of Vulnerabilities in Test-bed Network

attribute-template that allows us to categorize these network properties for further analysis.

Definition 20 ATTRIBUTE-TEMPLATE

An attribute-template is a generic property of the network which includes, but not limited to, the following:

1. *system vulnerabilities (which are often reported in vulnerability databases such as BugTraq, CERT/CC, or netcat),*

2. *(insecure) system properties such as unsafe security policy, corrupted file/memory access permission, or read-write access in file structure,*

3. *(insecure) network properties such as unsafe network condition, unsafe firewall properties, unsafe device/peripheral access permission, and*

4. *access privilege such as user account, guest account, or root account.*

An attribute-template helps us categorize most of the atomic properties of the network that may be useful to an attacker. For example, *"SSH buffer overflow vulnerability in FTP server"* can be considered as an instance of the system vulnerabilities template. Similarly, *"user access on Local machine*" is an instance of the access privilege template. Such templates also let us specify the properties as random variables. We define an *attribute* with such a concept in mind.

Definition 21 ATTRIBUTE

An attribute is a Bernoulli random variable representing the state of an instance of an attribute-template.

An attribute S is therefore associated with a state – *True* ($S = 1/T$) or *False* ($S = 0/F$) – and a probability $Pr(S)$. The state signifies the truth value of the proposition underlined by the instance of the attribute template. For example, the instance S :"*user access on Local machine*" is an attribute when associated with a truth value signifying whether an attacker has user access on the local machine. We shall also use the term "compromised" to indicate the *true* (or $S = 1$) state of an attribute. Further, $Pr(S)$ is the probability of the attribute being in state $S = 1$. Consequently, $Pr(\neg S) = 1 - Pr(S)$ is the probability of the state being $S = 0$. The success or failure of an attacker reaching its goal depends mostly on the states of the attributes in a network. It also lays the foundations for a security manager to analyze the effects of forcing some attributes to the *false* state using security measures. We formally define a BAG to capture the cause-consequence relationships between such attributes.

Definition 22 ATOMIC ATTACK

Let S be a set of attributes. We define $\mathcal{A}$, a conditional dependency between a pair of attributes, as a mapping $\mathcal{A} : S \times S \rightarrow [0,1]$. Then, given $s_{pre}, s_{post} \in S$, $a : s_{pre} \mapsto s_{post}$ is called an atomic attack if

1. $s_{pre} \neq s_{post}$,

2. *given* $s_{pre} = 1$, $s_{post} = 1$ *with probability* $\mathcal{A}(s_{pre}, s_{post}) > 0$, *and*

3. $\nexists s_1, \ldots, s_j \in S - \{s_{pre}, s_{post}\}$ *such that* $\mathcal{A}(s_{pre}, s_1) > 0, \mathcal{A}(s_1, s_2) > 0, \ldots,$ *and* $\mathcal{A}(s_j, s_{post}) > 0$.

Therefore, an atomic attack allows an attacker to compromise the attribute s_{post} from s_{pre} with a non-zero probability of success. Although, given a compromised attribute, another attribute can be compromised with positive probability using a chain of other attributes; the third condition in the definition does not allow such instances to be considered as atomic attacks. Instead, each step in such a chain is an atomic attack. Informally, an attack is associated with a vulnerability exploitation, denoted by e_i, which takes the attacker from one network state (s_{pre}) to another (s_{post}). The probability of an exploitation, $Pr(e_i)$, states the ease with which an attacker can perform the exploitation. Hence, we say that $\mathcal{A}(s_{pre}, s_{post}) = Pr(e_i)$, and s_{pre} and s_{post} are respectively called a *precondition* and *postcondition* of the attack a, denoted by *pre(a)* and *post(a)* respectively. An attack relates the states of two different attributes so as to embed a cause-consequence relationship between the two. For example, for the attributes s_{pre} = *"sshd BOF vulnerability on machine A"* and s_{post} = *"root access privilege on machine A"*, the attack $s_{pre} \mapsto s_{post}$ is associated with the e_i = *"sshd buffer overflow"* exploit. Using this exploit, an attacker can achieve root privilege on a machine provided the machine has the sshd BOF vulnerability. $\mathcal{A}(s_{pre}, s_{post})$ is the probability of success of the exploit, i.e. $\mathcal{A}(s_{pre}, s_{post}) = Pr(e_i)$.

Definition 23 BAYESIAN ATTACK GRAPH

Let S be a set of attributes and A be the set of atomic attacks defined on S. A Bayesian Attack Graph is a tuple $BAG = (S, \tau, \varepsilon, \mathcal{P})$, *where*

1. $S = N_{internal} \cup N_{external} \cup N_{terminal}$. $N_{external}$ *denotes the set of attributes* s_i *for which* $\nexists a \in A | s_i = post(a)$. $N_{internal}$ *denotes the set of attributes* s_j *for which* $\exists a_1, a_2 \in A | [s_j = pre(a_1)$ *and* $s_j = post(a_2)]$. $N_{terminal}$ *denotes the set of attributes* S_k *for which* $\nexists a \in A | S_k = pre(a)$.

2. $\tau \subseteq S \times S$. *An ordered pair* $(s_{pre}, s_{post}) \in \tau$ *if* $s_{pre} \mapsto s_{post} \in A$. *Further, for* $s_i \in S$, *the set* $Pa[s_i] = \{s_j \in S | (s_j, s_i) \in \tau\}$ *is called the parent set of* s_i.

3. ε *is a set of decomposition tuples of the form* $\langle s_j, d_j \rangle$ *defined for all* $s_j \in N_{internal} \cup N_{terminal}$ *and* $d_j \in \{AND, OR\}$. d_j *is AND if* $s_j = 1 \Rightarrow \forall s_i \in Pa[s_j], s_i = 1$. d_j *is OR if* $s_j = 1 \Rightarrow \exists s_i \in Pa[s_j], s_i = 1$.

4. $\mathcal{P}$ *is a set of discrete conditional probability distribution functions. Each attribute* $s_j \in N_{internal} \cup N_{terminal}$ *has a discrete local conditional probability distribution (LCPD) representing the values of* $Pr(s_j \mid Pa[s_j])$.

Figure 8.2 shows the BAG for our test network. We have developed an in-house tool to generate such BAGs. The tool takes as input an initial vulnerability table, generated by a vulnerability scanner, and the network topology (currently provided manually to the tool). Using a sequence of SQL queries on a vulnerability exposure database, the tool creates consequence attributes for the graph until no further implications can be derived. The BAG in Figure 8.2 depicts a clear picture of 20 different attack scenarios. Each node is a Bernoulli random variable (s_i) representing the state variable of the attribute. The set $N_{external}$ represents the entry points of the graph. These nodes reflect an attacker's capability as discovered in a threat-source model. $N_{terminal}$ resemble the end points in the graph. These nodes reflect casualty at the end of each attack scenario. The set of ordered pair, τ, reflects the edges in the graph. The existence of an edge between two nodes imply that there is a causal dependency between their states, signified by the decomposition at each node. AND-decomposition signifies that the compromised state of a node implies that all nodes in its parent set have also been compromised. Similarly, OR-decomposition signifies that at least one parent node is in the *true* state. Note that these decompositions are uni-directional. For instance, under AND-decomposition, compromising all nodes in the parent set does not necessarily imply the node itself has been compromised. This is because the attacks relating the node with its parents can have varying levels of difficulty,

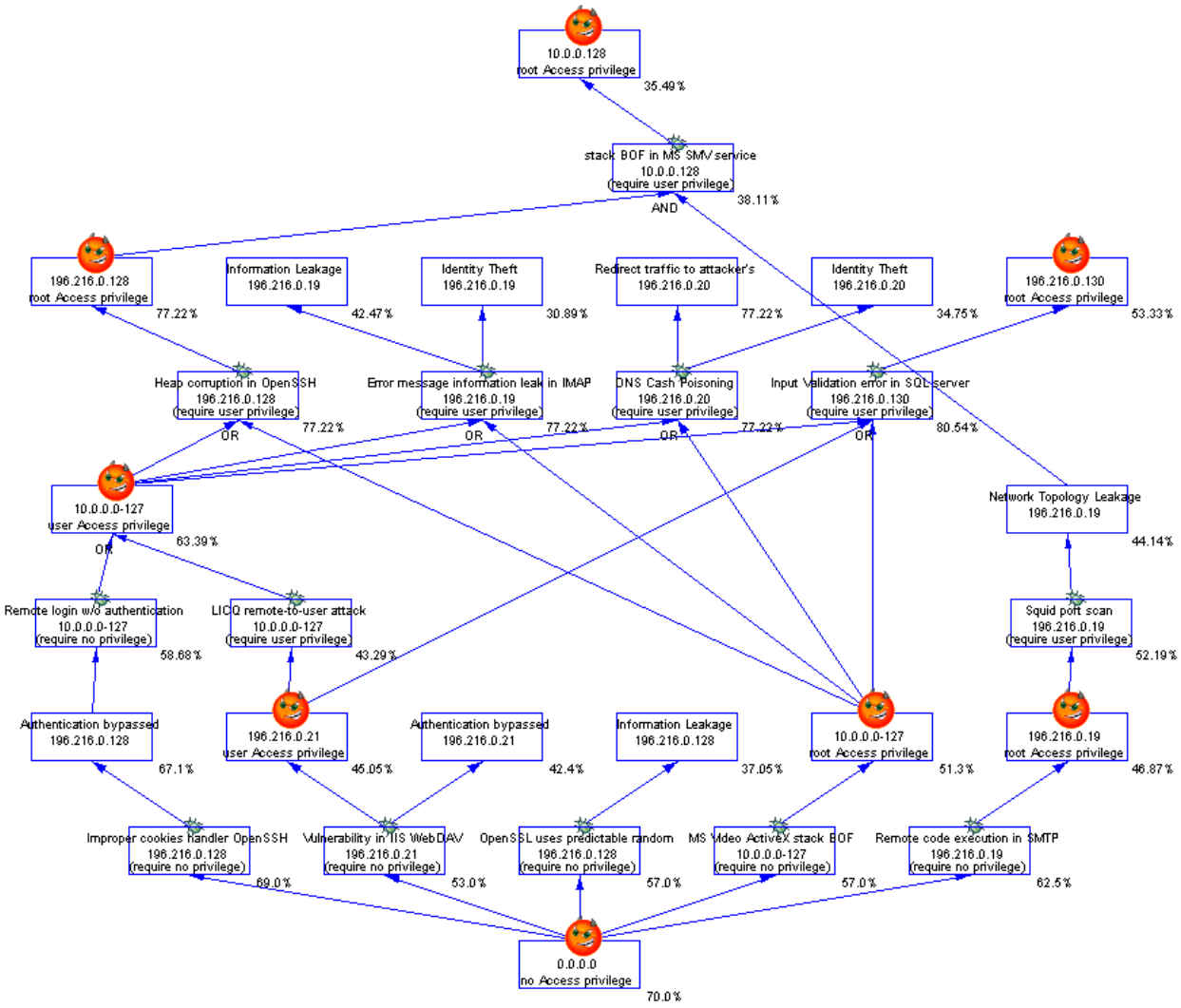


Figure 8.2: BAG of Test-bed Network with Unconditional Probabilities

or in other words, different probabilities of success. Hence, although the preconditions of the attacks have been met, there can still be a non-zero probability that the attacker is unable to carry out all the exploits successfully. The existence of this probability is what primarily differentiates a BAG from a classical attack graph. The probabilities are captured in the local conditional probability distribution of the node. The LCPD is a set of probability values specifying the chances of the node being compromised, given different combination of states of its parents.

Definition 24 LOCAL CONDITIONAL PROBABILITY DISTRIBUTION

Let BAG $= (S, \tau, \varepsilon, \mathcal{P})$ *be a BAG and* $s_j \in N_{internal} \cup N_{terminal}$. *For* $s_i \in Pa[s_j]$, *let* e_i *be the vulnerability exploitation associated with the attack* $s_i \mapsto s_j$. *A local conditional probability distribution (LCPD) function of* s_j, *mathematically equivalent to* $Pr(s_j \mid Pa[s_j])$, *is defined as follows.*

1. $d_j = AND$

$$Pr(s_j \mid Pa[s_j]) = \begin{cases} 0, & \exists s_i \in Pa[s_j] \mid s_i = 0 \\ Pr(\bigcap_{s_i=1} e_i), & otherwise \end{cases}$$

2. $d_j = OR$

$$Pr(s_j \mid Pa[s_j]) = \begin{cases} 0, & \forall s_i \in Pa[s_j], s_i = 0 \\ Pr(\bigcup_{s_i=1} e_i), & otherwise \end{cases}$$

To compute the local conditional probabilities when multiple exploits are involved, we proceed as follows. For AND-decomposition, each vulnerability exploitation is a distinct event. The chance of compromising the target node depends on the success of each individual exploit. Therefore, we use the product rule in probability to derive $Pr(\bigcap_{s_i=1} e_i)$ as

$$Pr(\bigcap_{s_i=1} e_i) = \prod_{s_i=1} Pr(e_i). \tag{8.1}$$

For OR-decomposition, Liu et al. observed that the joint probability is equivalent to the noisy-OR operator [41], given as

$$Pr(\bigcup_{s_i=1} e_i) = 1 - \prod_{s_i=1} [1 - Pr(e_i)]. \tag{8.2}$$

8.4 Security Risk Assessment with BAG

Security risk management consists of threat analysis, risk assessment, loss expectancy, potential safeguards and risk mitigation analysis. A BAG positions itself between threat analysis and risk assessment. Threat sources and the list of initial vulnerabilities are used to build the BAG threat model. Once the graph is built, the administrator can expect better results in risk assessment and risk mitigation analysis as follows.

1. *Static Risk Assessment*: Risk assessment begins with the identification of system characteristics, potential threat sources and attacker capabilities. Threat sources are represented as the external nodes in a BAGs, along with their impact on other network attributes. One set of attributes act as preconditions to an exploit, which when successfully executed by an attacker, can make the network state favorable for subsequent exploits. Estimating the amount of risk at each node therefore requires some judgment on attacker capabilities. Often this judgment is indirectly stated as the system administrator's subjective belief on the likelihood of a threat source becoming active and the difficulty of an exploit. The former is represented by the probabilities $Pr(s_i)$ for all $s_i \in N_{external}$, also called the *prior probabilities*, and is subjectively assigned by the administrator. The latter is incorporated into an internal node's LCPD. Thereafter, given the prior probability values and the LCPDs, we can compute the *unconditional probability* $Pr(s_j)$ for any node $s_j \in N_{internal} \cup N_{terminal}$. These risk estimates can be used to help locate weak spots in the system design and operations.

2. *Dynamic Risk Assessment*: A deployed system may experience first hand attack incidents during its life cycle. Formally, an attack incident is evidence that an attribute is in the *true* state. A security administrator may then want to investigate how these incidents impact the risk estimates initially derived solely based on subjective beliefs. Knowledge about attack incidents is therefore used to update the probabilities using the Bayesian inference techniques of forward and backward propagation. Forward propagation updates the probability on successor attributes that are directly influenced by the evidences. Backward propagation corrects/adjusts the initial hypothesis on all prior attributes. Thereafter, the *posterior probabilities* (updated unconditional probabilities) reflect the likelihoods of other potential outcomes under the light of detected events.

CVSS metrics group	CVSS attributes	category	score
base metrics	access vector(B_AV)	local(L)	0.395
		adjacent network(A)	0.646
		network(N)	1.0
	attack complexity(B_AC)	high(H)	0.35
		medium(M)	0.61
		low(L)	0.71
	authentication instance(B_AU)	multiple(M)	0.45
		single(S)	0.56
		none(N)	0.704
temporal metrics	exploitability (tools & techniques) (T_E)	unproved(U)	0.85
		proof-of-concept(POC)	0.9
		functional(F)	0.95
		high(H)	1.0
	remediation level(T_RL)	official fix(OF)	0.87
		temporary fix(TF)	0.90
		workaround(W)	0.95
		unavailable(U)	1.0
	report confidence (T_RC)	unconfirmed(UC)	0.90
		uncorroborative(UR)	0.95
		confirmed(C)	1.0

Table 8.2: CVSS Attributes Used for Estimation of Attack Likelihood.

3. *Risk Mitigation Analysis*: Risk assessment paves the way for efficient decision making targeted at countering risks either in a proactive or reactive manner. Given a set of security measures (e.g. firewall, access control policy, cryptography, etc.), we can design the security plan which is the most resource efficient in terms of reducing risk levels in the system. This can be done before the deployment (static mitigation) or in response to attack incidents (dynamic mitigation).

8.4.1 Probability of Vulnerability Exploitation

In order to compute the local conditional probability distribution (LCPD) of an attribute, the administrator needs to estimate the probability of success while an attacker exploits a known vulnerability exploitation. We propose a method to estimate this attack likelihood using publicly available risk exposure data sources. In particular, we are interested in deriving attack likelihoods using the metrics defined in NIST's Common Vulnerability Scoring System (CVSS) [59]. A CVSS score is a decimal number on a

scale of 0 to 10. It is composed of three groups – *base*, *temporal* and *environmental*. The base metrics quantify the intrinsic characteristics of a vulnerability with two sub-scores – (i) the exploitability sub-score, composed of the access vector (B_AV), access complexity (B_AC) and authentication instances (B_AU), and (ii) the impact sub-score, expressing the potential damage on confidentiality (B_C), integrity (B_I) and availability (B_A). The temporal metrics quantify dynamic aspects of a vulnerability on the environment around the organization. These metrics take into account the availability of exploitable tools and techniques (T_E), remediation level (T_RL) and report confidence (T_RC). The environmental metrics quantify two aspects of impact that are dependent on the environment surrounding the organization. More details on CVSS metrics and their scoring computation can be found in the CVSS guide [59]. In this study, we are interested in likelihood estimation and hence the impact sub-score and environmental metrics are ignored in the analysis. A summary of the metrics used here is shown in Table 8.2. We refine Houmb's Misuse Frequency model [36] to estimate the probability of success in vulnerability exploitation. Given the vulnerability exposure information (CVSS attributes), the probability of success $Pr(e_i)$ while executing a given vulnerability exploitation e_i is computed by the following equations.

$$Pr(e_i) = (1-\mu)MF_{init} + \mu MF_{uFac}, \text{ where} \tag{8.3}$$

$$\begin{aligned} & 0 \leq \mu \leq 0.5 \\ MF_{init} &= \frac{B\_AV \times B\_AC \times B\_AU}{0.49984} \\ MF_{uFac} &= T\_E \times T\_RL \times T\_RC. \end{aligned}$$

The constant μ represents the evaluator's preference weight on temporal knowledge of the exploitation. In the case where the vulnerability exploitation is unknown to the evaluator, the estimation should rely on the base score by setting μ to zero. In the case where the evaluator or organization has experienced the vulnerability exploitation, or there is an ongoing concern about the exploitation, the evaluator may set the value of μ to a specific value. However, we bound μ to a maximum value of 0.5 in order to restrict

likelihood estimates based solely on temporal factors. Nonetheless, temporal metrics help capture the uncertainties in relatively new exploits. For instance, at the time of writing this article, CVE announced a vulnerability in Acrobat Reader(VU#905281) where the only workaround is to disable JavaScript in Acrobat Reader. In such a case, temporal metrics often influence security decisions because of immediate needs. We design μ to capture such an aspect.

Our empirical estimation in Eq. (8.3) preserves the CVSS design characteristics and extends the range of possible values in Houmb's model [36] from $[0.53, 0.83]$ to $[0.12, 1.00]$.

8.4.2 Local Conditional Probability Distributions

Refer to the BAG in Figure 8.3. Nodes A:*"root/FTP server"*, B:*"Matu FTP BOF"* and C:*"remote BOF on ssh"* are internal attributes, while node D:*"remote attacker"* is an external attribute. A is the successor of B and C which in turn are successors of D. The values on the edges reflect the probability of success of the associated vulnerability exploitation, computed by following the procedure described in the previous section. We begin by assigning a prior probability of $Pr(D) = 0.7$ to the external attribute D. This probability represents the administrator's subjective belief on the chances of a remote attack. For the nodes A, B and C, we calculate LCPDs by the equations previously defined in Definition 24. For example, for node A, there are 2^2 marginal cases given the two parents B and C. The decomposition at the node dictates the rule to follow while computing the local conditional probability for each case.

8.4.3 Unconditional Probability to Assess Security Risk

Once the LCPDs have been assigned to all attributes in the BAG, we can merge the marginal cases at each node to obtain the unconditional probability at the node. This is commonly known as *marginalization*. Further, given a set of Bernoulli random variables

$S = \{s_1, ..., s_n\}$ in a Bayesian belief network, the joint probability of all the variables is given by the *chain rule* as

$$Pr(s_1, ..., s_n) = \prod_{i=1}^{n} Pr(s_i \mid Pa[s_i]). \tag{8.4}$$

In Figure 8.3, the unconditional probability at node A is derived as the joint probability

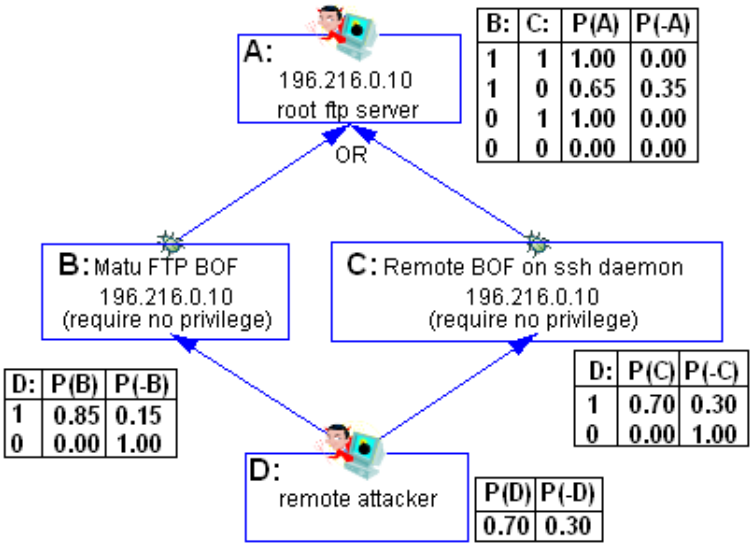


Figure 8.3: Simple BAG Illustrating Probability Computations.

of A along with all nodes that influence its outcome, which is essentially all ancestors of A. Hence we have,

$$\begin{aligned}
Pr(A) &= Pr(A,B,C,D) \\
&= Pr(A \mid B,C) \times Pr(B \mid D) \times Pr(C \mid D) \times \\
&\quad Pr(D) \\
&= \sum_{B,C,D \in \{T,F\}} [Pr(A \mid B,C) \times Pr(B \mid D) \times \\
&\quad Pr(C \mid D) \times Pr(D)] \\
&= (1.0 \times 0.85 \times 0.7 \times 0.7)_{TTT} + \\
&\quad (0.65 \times 0.85 \times 0.3 \times 0.7)_{TFT} + \\
&\quad (1.0 \times 0.15 \times 0.7 \times 0.7)_{FTT} \\
&= 0.6060 \approx 61\%.
\end{aligned}$$

Similarly, unconditional probabilities at nodes B and C can be computed by considering the sub-network rooted at the corresponding nodes. The unconditional probabilities are shown under the LCPD table of each node. Figure 4.2 shows the unconditional probabilities of the nodes in our test network. It exposes the weak spots of the system where the

likelihood of attack is higher than others. The security administrator can use this threat model to prioritize risk and derive an effective security hardening plan so as to reduce the risk to a certain level (e.g. $< 50\%$) before deploying the system. The model can also be used to assess what-if scenarios, for e.g. while deploying new machines, services, or operations of interest.

8.4.4 Posterior Probability with Attack Evidence

The BAG can also be used to address dynamic aspects of the security planning process. Every network state has a certain probability of occurrence. This probability can change during the life time of the system due to emerging security conditions, changes in contributing factors or the occurrence of attack incidents. The BAG can then be used to calculate the posterior probabilities in order to evaluate the risk from such emerging conditions. Let $S = \{s_1, ..., s_n\}$ be the set of attributes in a BAG and $E = \{s'_1, ..., s'_m\} \subset S$ be a set of attributes where some evidence of exploit have been observed. We can say that attributes in E are in the *true* state, i.e. $s'_i = 1$ for all $s'_i \in E$. Let $s_j \in S - E$ be an attribute whose posterior probability has to be determined. The probability we are interested in is $Pr(s_j \mid E)$ and can be obtained by using the *Bayes Theorem*, given as

$$Pr(s_j \mid E) = Pr(E \mid s_j) \times Pr(s_j)/Pr(E). \tag{8.5}$$

$Pr(E \mid s_j)$ is the conditional probability of joint occurrence of $s'_1, ..., s'_m$ given the states of s_j. $Pr(E)$ and $Pr(s_j)$ are the prior unconditional probability values of the corresponding attributes. Since evidence attributes in E are mutually independent, $Pr(E \mid s_j) = \prod_i Pr(s'_i \mid s_j)$ and $Pr(E) = \prod_i Pr(s'_i)$. For example, in Figure 8.3, assume that the system administrator detects an attack incident on A (attacker compromises FTP server). The posterior

probability of C is then computed as follows.

$$\begin{aligned} Pr(C \mid A) &= Pr(A \mid C)Pr(C)/Pr(A) \\ &= 0.81 \text{ where} \\ Pr(A \mid C) &= \textstyle\sum_{B \in \{T,F\}} [Pr(A \mid B, C = T)Pr(B)] \\ &= (1.0 \times 0.6)_T + (1.0 \times 0.4)_F \\ &= 1.0 \\ Pr(A) &= 0.61 \\ Pr(C) &= 0.49 \end{aligned}$$

Similarly, the posterior probability at node B can be computed in the same manner. Note that the unconditional probability of node C was originally 0.49. After taking into account the attack incident at node A, the posterior probability becomes 0.81. Further, computation of posterior probabilities for successor nodes of A (forward propagation) remains the same as described in the previous sub-section, with the change that the LCPDs at those nodes only account for the $A = 1$ case while marginalization. In this manner, the security administrator can revise the probability of occurrence of every node of the graph in response to an emerging attack incident. Figure 8.4 shows the posterior probabilities in response to two hypothetical evidences (denoted by the label Ⓔ) in the Mail server of our test network. Note that the parent ("*root access @ 196.216.0.19*") of the evidence node "*squid port scan*" has a posterior probability of less than 1.0. Ideally, given the evidence that the port scan has been executed, the attacker must have had root access on the machine. Hence, the parent node should also have an updated probability of 1.0. However, this inference assumes that the squid port scan is only executable after gaining root access on the machine. The system administrator may decide to relax such an assumption in order to account for uncertainties (e.g. zero-day attacks), achieved by replacing the zero values in Def. 24 with non-zero values. Such a relaxation will reduce the impact of the evidence nodes on their parents.

As can be seen in Figure 8.4, most of the unconditional probabilities increase after the attack incidents, but not at the same rate. It is possible to have nodes with decreased probabilities as well. In this specific scenario, there is a significant increase in the chance

that the administrator machine is targeted by an attacker. This observation shows that the attacker is likely to execute an attack to compromise the root machine. Hence, sufficient measures should be taken to protect it. Moreover, it is also possible that the mitigation plan designed earlier in static analysis may no longer be appropriate under the light of the emerging events. We will formally address this problem in the next section.

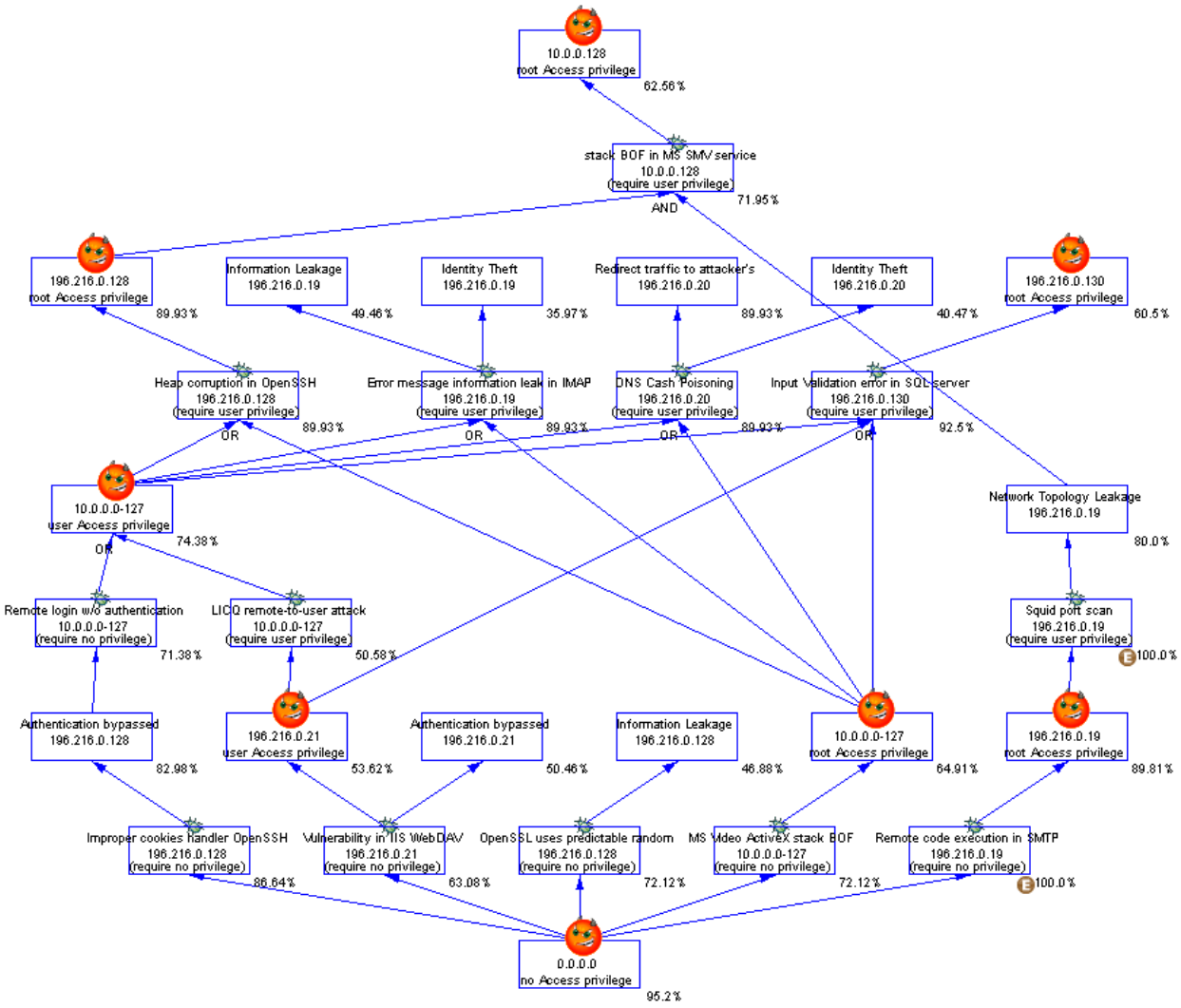


Figure 8.4: Posterior Probabilities in Test-bed Network After Attack Incidents (marked by Ⓔ).

8.5 Security Risk Mitigation with BAG

Although many researchers have studied risk assessment schemes, including the NIST, the methodologies used to estimate loss varies from organization to organization. Loss can be measured in terms of monetary units, relative magnitudes [13, 16, 40, 64] or multi-units [17, 18, 30]. In a BAG, the security manager can choose to evaluate the risks by considering an expected loss/gain quantity. The expected loss/gain is computed from organization specific factors such as potential loss or gain associated with an attribute's states. It usually reflects the impact of attack likelihoods on the economic turnout of an organization. We will describe this scheme later in the section. We begin with the notion of a security control in the context of the BAG.

8.5.1 Assessing Security Controls

Definition 25 SECURITY CONTROL

Given a BAG $(S,\tau,\varepsilon,\mathcal{P})$*, a Bernoulli random variable* M_i *is a security control if* $\exists s_j \in N_{internal} \cup N_{external}$ *such that* $Pr(s_j \mid Pa[s_j], M_i = T) < Pr(s_j \mid Pa[s_j], M_i = F)$ *for at least one assignment of states to* $Pa[s_j]$*. Further,* $Pr(M_i) = 1.0$ *if* $M_i = T$*; otherwise zero.*

In other words, a security control is a preventive measure that minimizes or eliminates the likelihood of attack on one or more attributes so as to prevent an attacker from reaching its goal. We define the security measure as a Bernoulli random variable with the *true* state signifying that the control is enforced and *false* signifying that the control is known but not enforced. Enforcement of a control directly influences the LCPD of the associated attribute and indirectly impacts the unconditional probabilities of other attributes. For example, the probability of the node A in Figure 8.3 is initially $Pr(A \mid B,C)$. Assume that a security measure M_0*:"local access control"* can influence the outcome at A. The probability distribution therefore becomes $Pr(A \mid B,C,M_0)$ and the LCPD of the node is hypothetically expanded as shown in Table 8.3. The probabilities when $M_0 = 0$ are directly taken from the original LCPD. However, probabilities for $M_0 = 1$ are assigned based on certain subjective belief on the security measure's capacity to prevent

the attribute's compromise. The modified LCPDs are used to compute the unconditional probability of nodes in the graph. It is not difficult to see that the unconditional probability of a node (and its successors) influenced by a control will reduce when the control is enforced. Note that, by definition, the unconditional probability of the control itself is 1.0 if its state is *true*.

B	C	$Pr(A)$	$Pr(\neg A)$
1	1	1.00	0.00
1	0	0.65	0.35
0	1	1.00	0.00
0	0	0.00	0.00

$\longrightarrow$

B	C	M_0	$Pr(A)$	$Pr(\neg A)$
1	1	1	0.50	0.50
1	1	0	1.00	0.00
1	0	1	0.65	0.35
1	0	0	0.65	0.35
0	1	1	0.75	0.25
0	1	0	1.00	0.00
0	0	1	0.00	1.00
0	0	0	0.00	1.00

Table 8.3: Expanded LCPD of Node A (in Figure 8.3) under the Presence of Security Control M_0.

Definition 26 SECURITY MITIGATION PLAN

Given set $M = \{M_1, \ldots, M_m\}$ of m security controls, a security mitigation plan is represented by a boolean vector $\vec{T} = (T_1, T_2, \ldots, T_m)$ where $M_i = T_i$.

Therefore, the mitigation plan is a specification of which controls have been chosen for enforcement as part of the hardening process. Further, for a given control M_i, consider the set I of all $s_j \in N_{internal} \cup N_{terminal}$ such that $Pr(s_j \mid Pa[s_j], M_i = T) < Pr(s_j \mid Pa[s_j], M_i = F)$ (for some assignment of states to $Pa[s_j]$). Then, the subset $\{s_k \in I | Pa[s_k] \cap I = \phi\}$ is called the *coverage* of M_i. With reference to Figure 8.3, a control such as M_0*:"disconnect from Internet"* directly changes the probability $Pr(D)$ (equal to zero if $M_0 = 1$). This in effect changes the LCPD tables at nodes B, C and D. Therefore, the set I contains all four nodes for M_0. However, only node D is in the coverage of M_0 since, for all other nodes, one or more parent nodes are also present in I. Intuitively, the coverage nodes are those whose LCPDs are directly affected by a security control, rather than by indirect inference. For a given security mitigation plan $\vec{T}$, we can define the plan coverage by collecting the

coverage of each enforced control in the plan. Each control M_i also has an associated cost C_i of implementation, giving us the total plan cost as

$$SCC(\vec{T}) = \sum_{i=1}^{m} (T_i C_i). \tag{8.6}$$

8.5.2 Assessing Security Outcomes

When using a BAG, a better quantitative representation of the loss/gain is obtained by considering the expected loss/gain once a set of security measures have been implemented. Hence, we augment the BAG with a value signifying the amount of potential loss/gain at each node, and account for the security decision during evaluation.

Definition 27 AUGMENTED-BAYESIAN ATTACK GRAPH

Let $BAG = (S, \tau, \varepsilon, \mathcal{P})$ be a Bayesian attack graph. An augmented-Bayesian attack graph (augmented-BAG) $BAG_{aug} = BAG|(M, \gamma, V)$ is obtained by adding a node set M, edge set γ and by associating a value V_j to each $s_j \in S$, where

1. *M is the set of security controls.*
2. *$\gamma \subseteq M \times S$. An ordered pair $(M_i, s_j) \in \gamma$ if s_j is in the coverage of M_i.*
3. *V_j is the expected loss/gain associated with the attribute $s_j \in S$.*

The set M extends the BAG with additional nodes representing hardening measures. The set γ represents the new edges between the controls and attributes of the graph. A new edge is inserted if a control directly influences the state of an attribute. In this work, all external attributes are given a zero cost, i.e. $V_j = 0$ for all $s_j \in N_{external}$. The value associated with $s_j \in N_{internal} \cup N_{terminal}$ is computed as

$$V_j = \left[1 - Pr(s_j)\right] \times G_j - Pr(s_j) \times L_j, \tag{8.7}$$

where L_j is the potential loss representing the damage value that an organization might have to pay when the attribute s_j is compromised, G_j is the potential gain if s_j is not

compromised and $Pr(s_j)$ is the unconditional probability of s_j. If there exists $(M_i, s_j) \in \gamma$, $Pr(s_j)$ is computed as a conditional probability $Pr(s_j \mid M_i)$ where the state of M_i depends on the security plan $\vec{T} = (T_i)$. The expected loss/gain w.r.t. the security plan $\vec{T}$, denoted by $LG(\vec{T})$, is computed as the cumulative sum of all node values, i.e.

$$LG(\vec{T}) = \sum_{s_j \in S} V_j. \tag{8.8}$$

A positive value of LG signifies gain, while a negative value signifies loss. Note that we do not assume any particular cost model in our formulations, both for control cost and loss/gain evaluation. The cost model is usually subjective to organizational policies and hence can differ from one institution to another. The cost factors considered here (security control cost, potential loss and potential gain) are standard quantities that any organization must be able to determine in order to perform risk analysis.

8.5.3 Assessing the Security Mitigation Plan

In order to defend against the attacks possible, a security manager (as a decision maker) can choose to implement a variety of safeguard technologies, each of which comes with different cost and coverage. For example, to defend against the "*ftp/.rhost*" exploit, one might choose to apply a security patch, firewall, or simply disable the FTP service. Each choice of action has a different cost and different outcome. A security administrator has to assess the technologies and make a decision towards maximum resource utilization. The two objectives we consider in this study are the total security control cost and the expected loss/gain. The single-objective problem is the most likely approach to be taken by a decision maker. The problem is stated as follows.

Definition 28 THE SINGLE-OBJECTIVE OPTIMIZATION PROBLEM (SOOP)
Given an augmented-BAG $(S, \tau, \varepsilon, \mathcal{P}) \mid (M, \gamma, V)$*, find a vector* $\vec{T}^* = (T_i^*)$*,* $T_i^* \in \{0,1\}; 1 \leq i \leq |M|$*, which maximizes the function* $\alpha LG(\vec{T}^*) - \beta SCC(\vec{T}^*)$*, where* α *and* β *are preference weights for the expected loss/gain and the total cost of security control respectively,* $0 \leq \alpha, \beta \leq 1$ *and* $\alpha + \beta = 1$.

The method for assessing a security plan is as follows. First, the security analyst chooses a subset of security controls to construct a security plan $\vec{T}^*$. She then updates the unconditional probability of all attributes using the plan coverage information. She computes the expected loss/gain associated with every attribute $s_j \in S$ using Eq. (8.7). Finally, the total expected loss/gain of the entire graph is taken as an assessment of the security plan's outcome. The best security plan is the one that maximizes the function $\alpha LG(\vec{T}^*) - \beta SCC(\vec{T}^*)$.

Given only two objectives, a preference based single-objective approach might seem to provide a solution in accordance with general intuition. However, the quality of the solution obtained using this process can be quite sensitive to the assignment of the weights. In addition, security administrators often have to work within a limited budget that may be less than the minimum cost of system hardening. The objective in such a case is to design a security plan that maximizes the organization's financial throughput. The next level of sophistication is added by formulating the optimization as a multi-objective problem. The multi-objective approach alleviates the requirement to specify any weight parameters and hence a better global picture of the solutions can be obtained.

Definition 29 THE MULTI-OBJECTIVE OPTIMIZATION PROBLEM (MOOP)

Given an augmented-BAG $(S,\tau,\varepsilon,\mathcal{P}) \mid (M,\gamma,V)$*, find a vector* $\vec{T}^* = (T_i^*)$, $T_i^* \varepsilon \{0,1\}; 1 \leq i \leq |M|$*, which minimizes SCC and maximizes LG.*

Multi-objective optimization differs from single-objective ones in the cardinality of the optimal set of solutions. Single-objective optimization techniques are aimed towards finding the global optima; whereas, in the case of multi-objective optimization, there is no such concept of a single optimum solution. This is due to the fact that a solution that optimizes one of the objectives may not have the desired effect on the others. Solutions to a multi-objective problem are therefore characterized by the concept of Pareto-optimality. In our case, a security plan $\vec{T}_1$ is *Pareto-optimal* if there is no other plan $\vec{T}_2$ such that

1. $SCC(\vec{T}_2) < SCC(\vec{T}_1)$ and $LG(\vec{T}_2) \geq LG(\vec{T}_1)$, or

2. $SCC(\vec{T}_2) \leq SCC(\vec{T}_1)$ and $LG(\vec{T}_2) > LG(\vec{T}_1)$.

If any of these conditions hold, then $\vec{T}_2$ is said to *dominate* $\vec{T}_1$. $\vec{T}_1$ and $\vec{T}_2$ are *non-dominated* w.r.t. each other if none dominates the other. Pareto-optimal solutions are non-dominated w.r.t. all other solutions. A multi-objective optimizer identifies (or approximates) the set of Pareto-optimal solutions and reveals the trade-off relations between the underlying objectives. Choice of a final solution from this set is at the discretion of the decision maker, often decided by the cost to benefit ratio. For the BAG shown in Figure 8.6, we have identified that 13 different security controls are possible. These controls are represented by an 'eclipse' in the figure. These security controls produce 2^{13} candidate security plans. A genetic algorithm based approach is presented next to search through these candidate plans in an efficient manner.

8.5.4 Genetic Algorithm

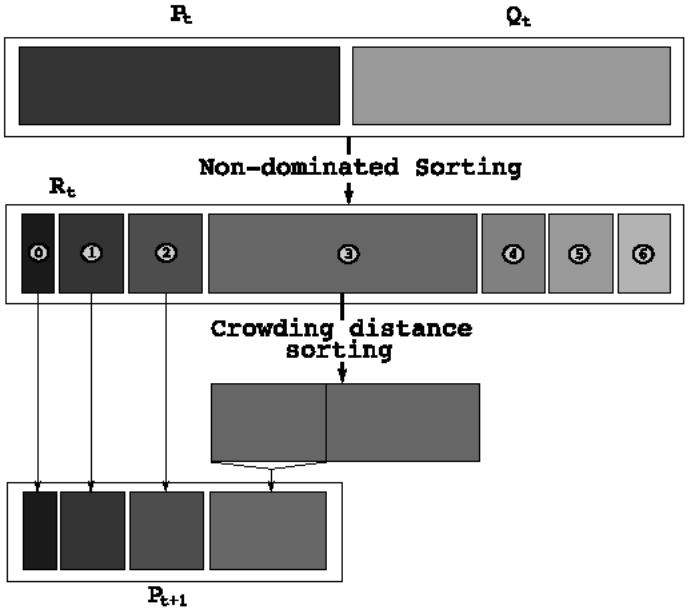


Figure 8.5: Schematic of the Genetic Algorithm Used to Solve SOOP and MOOP.

Figure 8.5 depicts the structure of the genetic algorithm designed for this study. The algorithm begins with a population P_0 of N randomly generated security plans. A generation index $t = 0, 1, \ldots, Gen_{MAX}$ keeps track of the number of iterations of the algorithm. Each iteration proceeds as follows. The *SCC* and *LG* values of every plan in P_t are calculated. $N/2$ plans are then selected from P_t to form a mating pool M_t. The selection process is different for SOOP and MOOP, and discussed later. An offspring population Q_t (containing $N/2$ plans) is generated from the mating pool by using the standard single-point binary crossover and mutation operators [33]. The process is then repeated with $P_{t+1} = Q_t \cup M_t$ until $t = Gen_{MAX}$.

8.5.4.1 Solving SOOP

The selection process to solve SOOP is based on the objective function $\alpha LG(\vec{T}) - \beta SCC(\vec{T})$ and uses the process of binary tournament. In this process, two plans are randomly selected (with replacement) from P_t and the one with the higher objective function value is inserted into the mating pool. This process is repeated until the mating pool is full. The solution to SOOP is the plan with the highest objective value across all iterations of the algorithm.

8.5.4.2 Solving MOOP

Simple objective value comparison is not possible in the presence of more than one objective function. Hence, a different selection scheme is required for MOOP. The scheme used here is based on *non-dominance ranking*, a popular concept in evolutionary multi-objective optimization. Under this process, all non-dominated solutions in P_t (solutions not dominated by any other solution in P_t) are identified and assigned a rank 1. The rank 1 solutions are then removed from P_t and the non-dominated solutions in the remaining population form rank 2 solutions. The process is repeated until all solutions are assigned a rank. Selection of $N/2$ solutions for the mating pool is then performed according to increasing rank. A *crowding distance metric* [26] is used if the number of

solutions required to fill the mating pool is lower than the available solutions in the rank being considered. The crowding distance of a solution is the perimeter of the rectangle formed by its two neighbors of the same rank; the distance is infinite for points having less than two neighbors (e.g. extreme points). Choice of solutions within a rank is done in decreasing order of crowding distance, thereby giving preference to solutions that are not at close proximity to others. The set of solutions to MOOP are the rank 1 solutions of $P_{Gen_{MAX}}$.

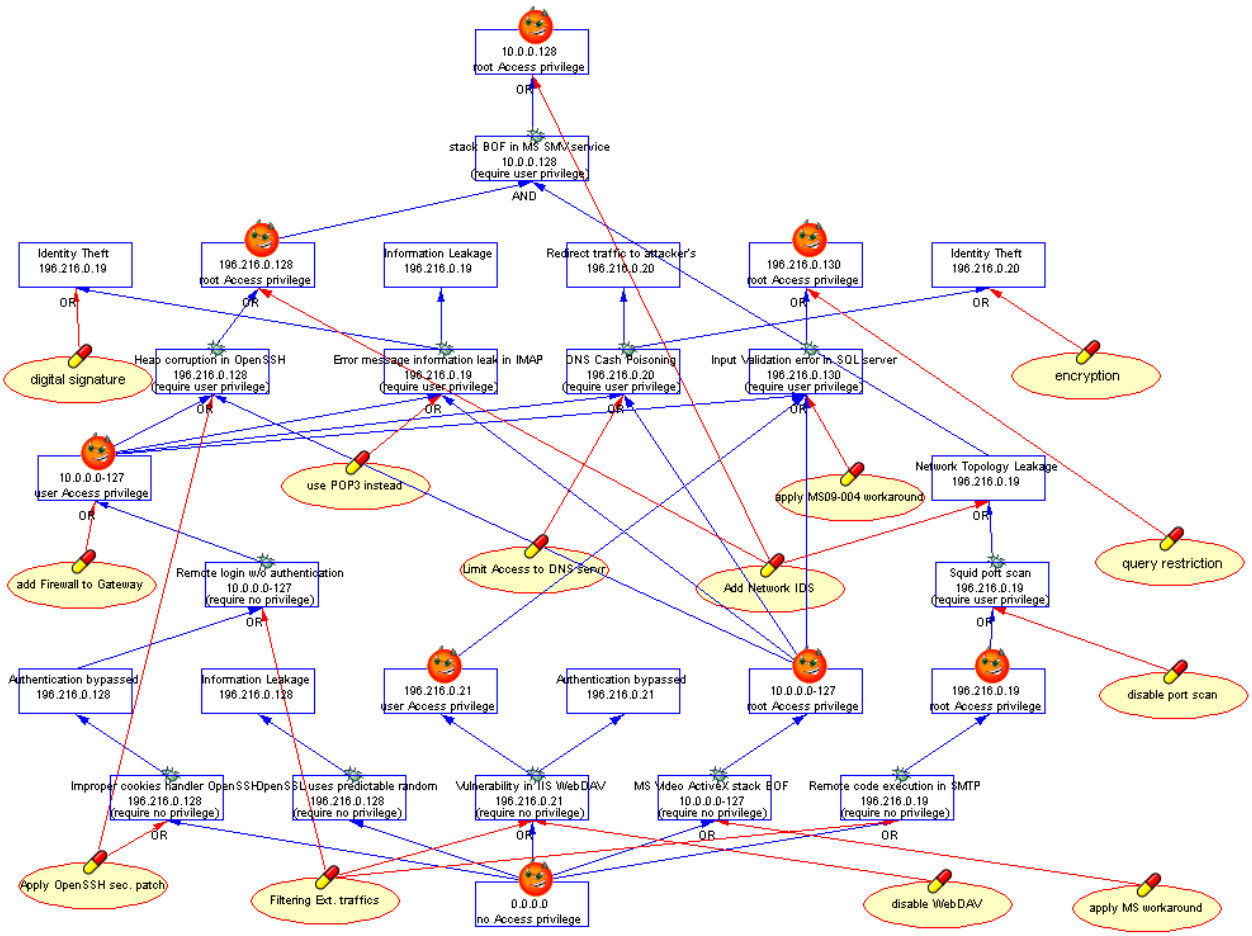


Figure 8.6: Augmented-BAG of Test-bed Network with 13 Security Controls.

8.6 Empirical Results

In the preparation phase, we conduct risk assessment analysis to initially compute the static risk. Figure 4.2 shows the unconditional probabilities at the nodes of the test network. We identify 13 security controls that can be used to reduce the risk. We assign a security control cost to each individual control and link each control to the attribute(s) in the BAG that are covered by it. The augmented-BAG resulting from this process is shown in Figure 8.6. Next, we assign different damage costs and revenue to every attribute in the graph. Although we do not assume any particular cost model and values are assigned hypothetically for the purpose of demonstration, we did try to maintain some relative difference in magnitude to account for the relative importance of different services.

In the first experiment, we assess the expected loss/gain on top of the static risk analysis results (Figure 4.2) using Eq. (8.7). When using no security control, i.e. a mitigation plan signified by the zero vector, we have an overall expected gain of 622.0 units. Then we assess the cost on the dynamic environment where we assume that two attack incidents have been detected. Figure 8.4 and Figure 8.7 show the posterior probabilities and the expected loss/gain at the nodes under this situation. Note that these attack incidents quickly change the business scenario. The total expected loss/gain (LG) changes from 622.0 to -398.17 units. We also notice a change in the momentum of risk. In particular, the posterior probabilities indicate a significant change in risk level at the Administrative server owing to the two attack incidents. This change influences the priority of risks identified earlier during static analysis, and highlights the importance of dynamic risk analysis.
Next, we conduct several tests to assess the outcome of using a security control. The base case where no individual control is used yields an expected gain of 622.0 units. Table 8.4 shows the net benefit of using each control individually. At this point, the security administrator may want to rank the security outcomes and build a security mitigation plan from the top-ranked controls. Such a methodology has two drawbacks.

First, the ranking procedure itself is not straight forward because of reciprocal relationships between control cost and expected outcome. For example, "*disable portscan*"

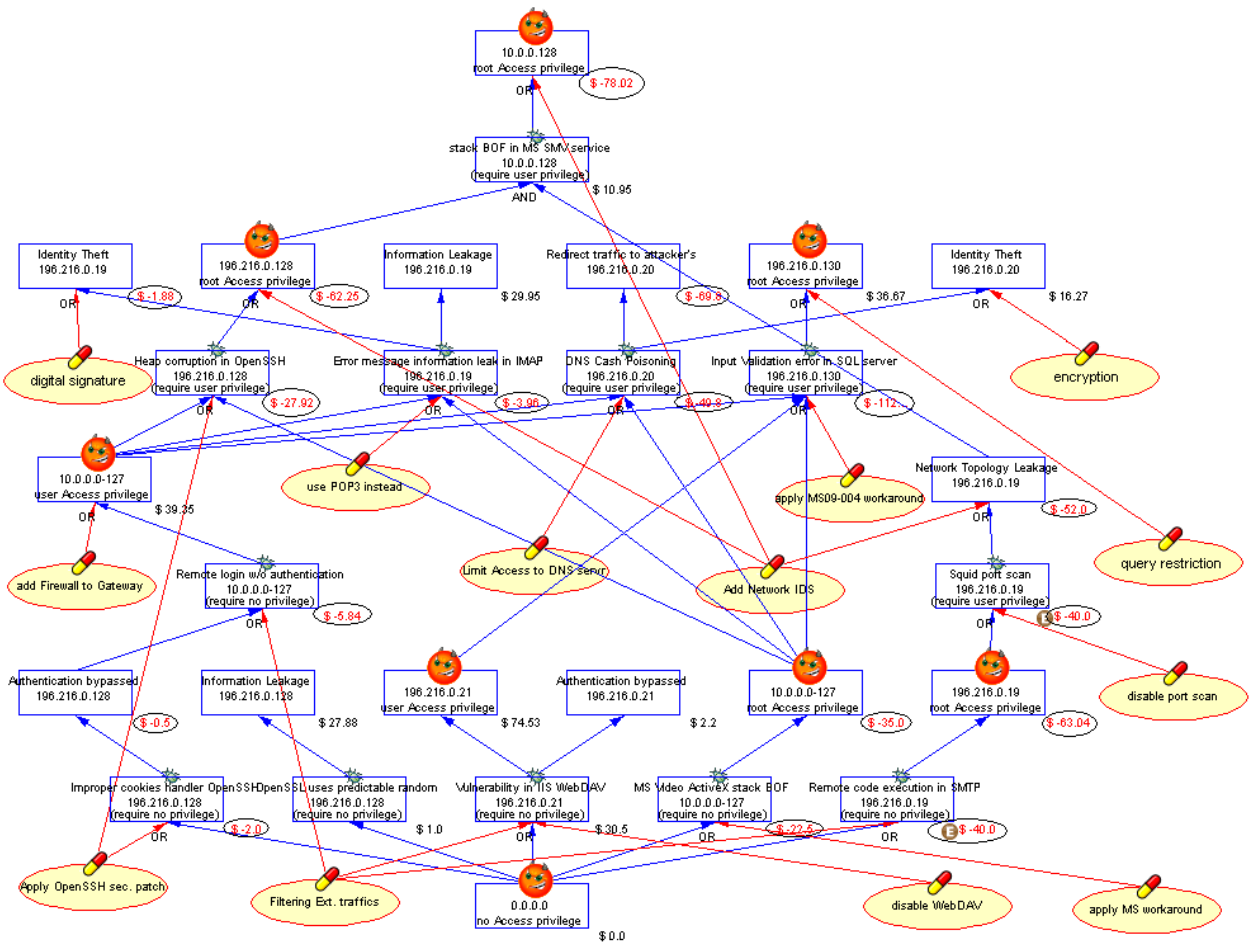


Figure 8.7: Augmented-BAG of Test-bed Network Under Two Attack Incidents (The expected loss/gain (V_j) is shown at each node)

and "*filtering external traffic*" when applied alone raises the expected gain from 622.0 units to 875.44 (an increase of 253 units) and 1208.84 units (an increase of 587 units) respectively. The combined outcome when applying both is 1351.27 units (less than $622 + 253 + 587$). On the other hand, combining "*add Firewall*" (individual increase from 622.0 to 881.15 units) and "*apply MS work around*" (individual increase from 622.0 to 1202.45 units) can raise the outcome to 1735.6 units (greater than $622 + 259 + 580$). The latter two are better choices based on expected outcome, but the former two incurs a lower cost of implementation. This makes the ranking of controls, based on a specific cost factor, a difficult process. Second, even if a ranking has been established, greedy

ID	Security Control	Cost ($\mathcal{A}$)	Outcome ($\mathcal{B}$)	Net benefit ($\mathcal{B}-\mathcal{A}-622.0$)
SC_1	apply OpenSSH security patch	63	1407.89	722.89
SC_2	apply MS work around	14	1202.45	566.45
SC_3	filtering external traffic	70	1208.84	516.84
SC_4	limit DNS access	53	1000.65	325.65
SC_5	disable portscan	11	875.44	242.44
SC_6	disable WebDAV	250	1095.90	223.90
SC_7	apply MS09-004 work around	31	861.10	208.10
SC_8	add Network IDS	102	858.91	134.91
SC_9	add Firewall	205	881.15	54.15
SC_{10}	encryption	34	681.75	25.75
SC_{11}	digital signature	33	673.28	18.28
SC_{12}	query restriction	84	681.00	-25.00
SC_{13}	use POP3 instead	153	704.67	-70.33

Table 8.4: Security Outcome Assessment for Each Control in Augmented-BAG of Testbed Network.

selection can lead to sub-optimal plans. Assume that controls are ranked based on the net benefit they incur individually. The security controls are ordered in this manner in Table 8.4. Given a control cost constraint of, say, 200.0 units and a selection scheme based on the ranks, an administrator will choose the first four controls in the table. These controls have a combined cost of 200.0 units and results in an expected gain of 2673.96 units (a net benefit of 2473.96 units collectively). However, selecting the 5th and the 7th controls, instead of the 4th one, effectuates an expected gain of 2809.28 units at the cost of 189.0 units (a net benefit of 2620.28 units). This shows that the security administrator should not choose the security controls based on their individual outcomes or by greedy selection. Instead, a more sophisticated decision making platform is required. This motivates the next three experiments with single and multi-objective optimization.

We conduct three risk mitigation analysis experiments on the test network. The genetic algorithm discussed in Section 8.5.4 is used for this analysis. The algorithm parameters are set as follows: population size $N = 100$, $Gen_{MAX} = 50$, crossover probability $= 0.8$, and mutation probability $= 0.01$. We ran each instance of the algorithm five times to check for any sensitivity of the solutions obtained from different initial populations. We also check if running the algorithm for a higher number of iterations (upto

200 generations) results in any improved convergence. However, since the solutions always converged to the same optima (or set of optima), we dismiss the presence of such sensitivity.

In single-objective cost analysis, we run multiple instances of SOOP using different combination of values for α and β. α is varied in the range of $[0,1]$ in steps of 0.05. β is always set to $1-\alpha$. Figure 8.8 shows the solutions obtained from this process. In general, a decision maker may want to assign equal weights ($\alpha = 0.5$) to both objective functions – security control cost and total expected loss/gain. It is clear from the figure that such an assignment does not necessarily provide the desired balance between the two objectives. Furthermore, the solutions are quite sensitive to the weights and they are not uniformly distributed across different ranges of α. Since the weights do not always influence the objectives in the desired manner, understanding their effect is not a trivial task for the administrator. It is also not always possible to perform an exhaustive analysis of the affect of the weights on the objectives. Given such situations, the decision maker should consider obtaining a global picture of the trade-offs possible. With such a problem in mind, we next consider the multi-objective variant. Figure 8.9 shows the non-dominated solutions (in $P_{Gen_{MAX}}$) obtained in the multi-objective analysis. Further, all mitigation plans explored by the genetic algorithm during the iterations are highlighted. The algorithm reported all solutions generated for SOOP (using multiple α), as well as several others, specifically solutions in the range where the security control cost is between 200.0 and 700.0 units. These new solutions provide much better flexibility in the decision making process. Moreover, performing the multi-objective analysis is much faster than solving SOOP. This is because the security administrator has to solve SOOP with multiple parameter settings in order to identify the plan with the desired outcomes, whereas by solving MOOP, one can generate a good overview of multiple plans in one single run.

We also compare our experiment with the result from minimization analysis [35, 47, 68]. We first derive logic expression out of the Bayesian attack graph model given in

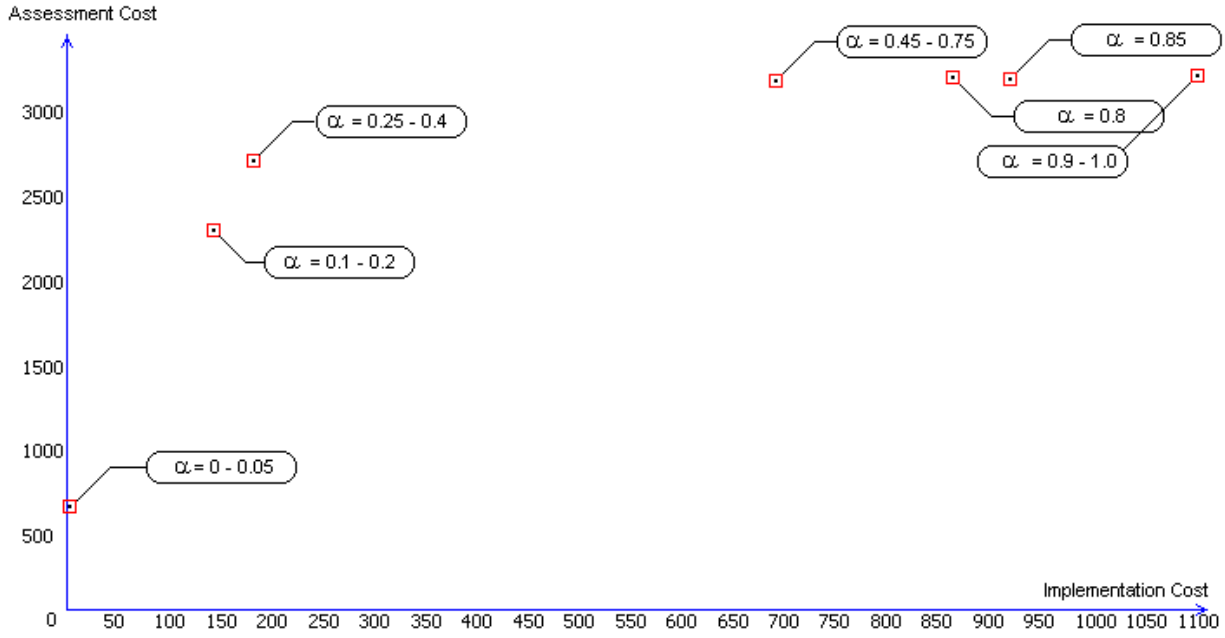


Figure 8.8: Genetic Algorithm Solutions to Single Objective Problem Obtained by Using Different Weights

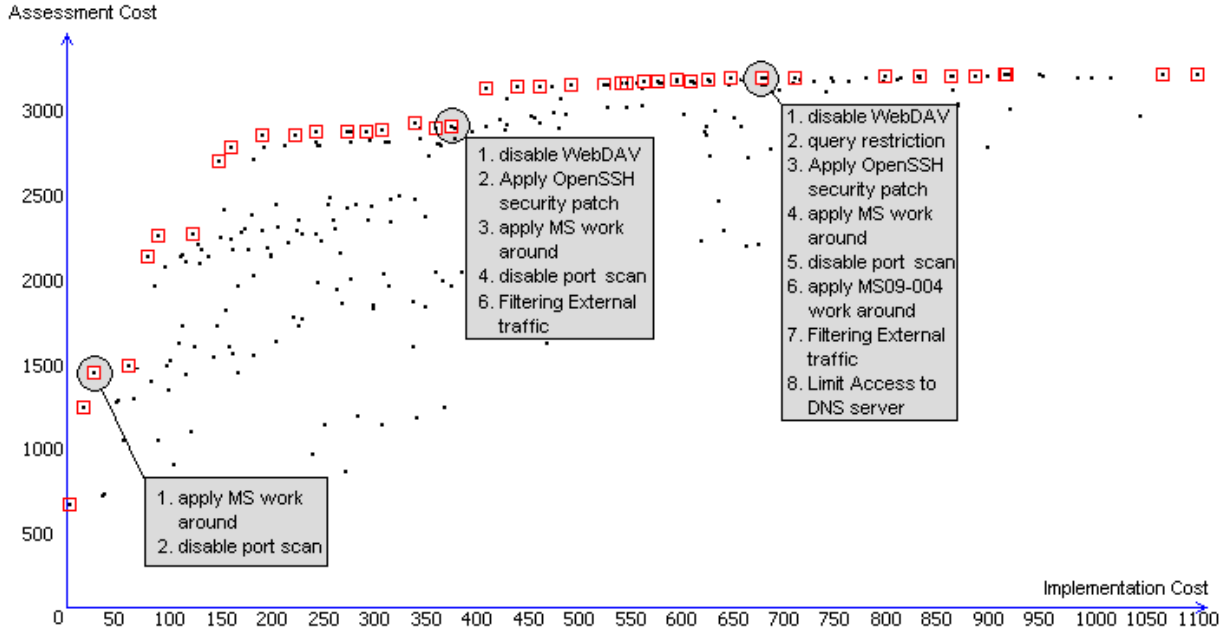


Figure 8.9: Genetic Algorithm Solutions to Multi-Objective Problem with Static Risk Assessment

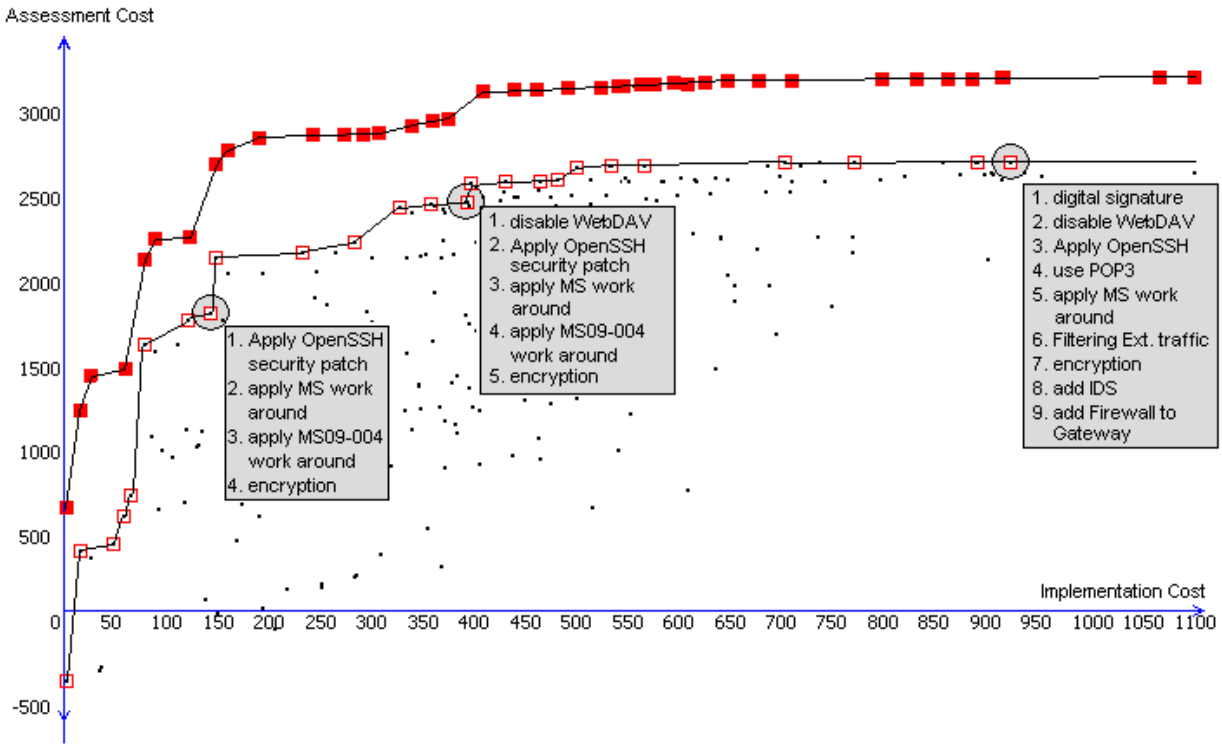


Figure 8.10: Genetic Algorithm Solutions to MOOP with Dynamic Risk Assessment

figure 8.2. The logic expression derived in this stage represents the critical factors that accommodate the security violation. This expression then converts into its disjunctive normal form (DNF) to give the choices of protective measures. To preserve the meaning of security objective, we derive DNF from all possible outcomes (all terminal nodes in the graph) and then union the results so as to take into account the whole system like what multi-objective analysis does. Similarly, we have to adjust the LCPD to take either true or false so as to have our Bayesian attack graph representing propositional logic like what Exploit Dependency graph [47] does. Figure 8.11 shows the comparison test with Noel's minimal-cost analysis. The result returns from the derivation is thus $T = \neg(((\Omega \vee \Gamma) \wedge \vartheta) \vee (\Omega \vee \Psi \vee \Gamma) \vee \Xi \vee \Psi)$. The DNF derivation returns the security hardening plan as $\neg\Omega \wedge \neg\Gamma \wedge \neg\Psi \wedge \neg\Xi \wedge \neg\, \vartheta$.

Given the security control as shown in the table 8.4, the least-cost security hardening plan includes SC_1, SC_2, and SC_3 with the total security cost of 147.0 (see table 8.4 and figure 8.6). The experimental result in Bayesian attack graph is also shown in the inlet figure. The graph is much steeper than figure 8.9 due to the adjustment in LCPD. As we expected, the minimum-cost hardening solution is on the Pareto front. In particular, it locates in the right-most of the graph. However, the graph does not show significant benefit of using Bayesian attack graph over minimization analysis but there always exists a condition where minimum-cost analysis gives a solution with the cost higher than the organization budget. In such a case, system administrator may decide to tolerate some risks in order to control the expense. This is where our Bayesian attack graph contributes. Consider the following situation, if node 'G' represents a much lower security risk than root compromise. It is tempting to not including SC_1 in the plan and manage to tolerate a slice amount of risk with a cheaper cost. Such solutions can be found in our multi-objective analysis. In the last experiment, we use the genetic algorithm to assess the choice of security hardening in a dynamic environment. Figure 8.10 shows the choices of mitigation plans in response to two emerging attack incidents, previously shown in Figure

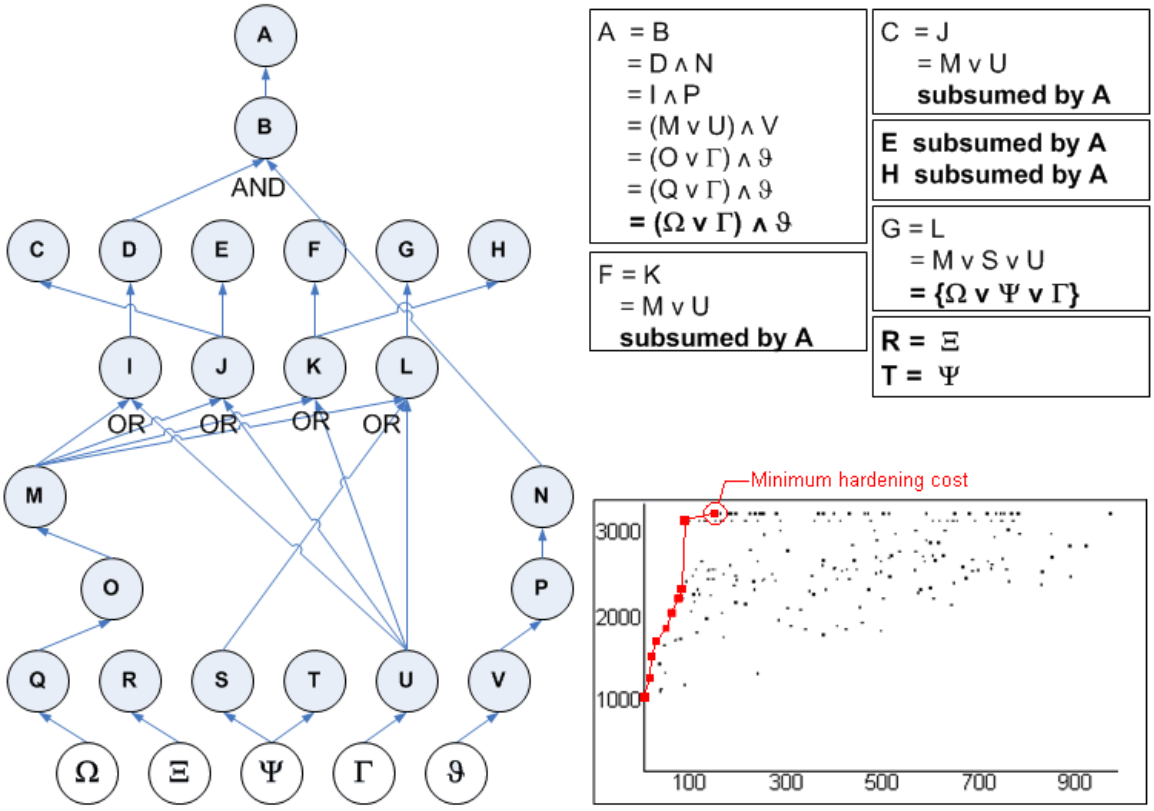


Figure 8.11: BAG Results Compare with Noel's Minimum-cost Analysis

8.4. In this plot, we compare the dynamic results with the static ones. Not surprisingly, the plans in this case effectuate lower gains owing to the damage already caused by the attacker when (and at which point) the incidents are detected. Despite this difference, the mitigation plans with similar costs are not so different between the static and dynamic solutions. The three plans highlighted in the figure are very similar to those shown in Figure 8.9. Such minimal changes in plan characteristics can be considered a positive outcome since the security administrator is not required to revise the entire plan chosen during static analysis. Instead, she can exploit the commonalities for efficient usage of already invested resources. Results from the dynamic analysis also highlight the requirement for pro-active action in security management. Note that although not implementing any controls still results in a positive gain, the appearance of two attack incidents quickly transform this into a case with negative expected outcome.

8.7 Chapter Summary

In this paper, we address the system administrators' dilemma, namely, how to assess the risk in a network system and select security hardening measures from a given set of controls so as to maximize resource utilization. One important contribution of our solution methodology is the use of a BAG model of the network to drive the decision process. We have provided formal definitions for network characteristics, attacks and security measures under this model. We also show that by using a BAG, we are able to better understand the causal relationships between preconditions, vulnerability exploitations and post conditions. This is facilitated by computing the likelihoods of different outcomes possible as a result of the cause-consequence relationships. We have demonstrated how the BAG can be used to revise these likelihoods in the event of attack incidents. Using empirical results on a test network, we show that such a dynamic risk analysis helps the system administrator identify evolving weak spots in a network. We also provide the necessary optimization formulations required to build a mitigation plan that reduces the risk levels. Towards this end, we propose a genetic algorithm capable of performing both single

and multi-objective optimization of the administrator's objectives. While single objective analysis uses administrator preferences to identify the optimal plan, multi-objective analysis provides a complete trade-off information before a final plan is chosen. Results are shown to demonstrate the effectiveness of the algorithm in both static and dynamic risk mitigation.

As immediate future work, we shall work on improving the efficiency of our evaluation algorithm. The evaluation algorithm is used to compute the unconditional probabilities and is currently implemented using brute force DFS traversal. Posterior probability computation is expensive using this implementation and therefore impacts the decision making time in a dynamic scenario. In particular, we wish to revise the evaluation algorithm to include heuristic based update mechanisms in order to reduce the time required to complete the mitigation analysis, without scarifying the quality of results obtainable. Furthermore, the mitigation process in dynamic situations needs to be improved so that a security administrator can quickly identify the best security response that accounts for all former investments made as part of the static analysis stage. It is worth mentioning that some security controls have been found to be commonly included in the optimal solutions. It is possible that security hardening is more critical in certain areas of the attack graph. Such areas could be nodes that has multiple fan outs. In other words, these critical areas are at-risk junctions that can be used by an attacker to cause multiple outcomes. Security controls that can reduce risk in such areas are likely to be parts of the optimal solutions. Therefore, it is worth investigating how such controls can be identified efficiently so as to reduce the search space for the optimization algorithm.

Chapter 9

USING ATTACK TREES IN INTRUSION DETECTION SYSTEM TO DETECT MALICIOUS INTENT

A major concern for computer systems security is the threat from malicious individuals who can execute perfectly legitimate operations to compromise system security. Unfortunately, most currently available intrusion detection systems (which include anomaly and misuse detection systems) fail to address this problem in a comprehensive manner. An attack tree has been proposed to identify malicious activities from users. We develop algorithms to generate minimal forms of attack tree customized for each user such that it can be used efficiently to monitor the users activities. If the users activities progress sufficiently up along the branches of the attack tree towards the goal of system compromise, we generate an alarm. Attack tree is not intended to replace existing intrusion detection and prevention technology, but rather is intended to complement current and future technology.

9.1 Introduction

Intrusion detection systems can be broadly classified into two groups knowledge-based systems and behavior-based systems [28]. Knowledge-based detectors [42, 56, 61] are the most popular techniques. Almost all commercially available intrusion detection tools are knowledge-based [10]. These tools operate by processing system audit data for signatures of known attacks and/or specific outcomes of interest. The result of this processing is compared against a knowledge-base of signatures of specific attacks and vulnerabilities. A positive match signals an intrusion.

Knowledge-based detectors tend to be fairly accurate in the sense that they have low rates of false positives. However, they are limited by their inability to detect new attacks for which there are no known signatures. Any action that is not recognized as an attack is considered acceptable. In addition, knowledge-based detectors are specific system (operating system, software tools etc.) dependent. Thus their wide adoption is limited.

Behavior-based detectors, on the other hand, take the paranoid approach everything that has not been witnessed before is considered dangerous. To operate, these systems compare the observed behavior of the system and/or the users against a model of normal (or expected) behavior. Any deviations from the normal behavior is considered an intrusion. Behavior-based systems are often considered complete, that is, all attacks (even previously not known attacks) can be caught. However, the accuracy of these systems are often low. Behavior-based systems need to undergo extensive training sessions to determine what constitutes normal behavior. During this phase these systems tend to generate false alarms at a very high rate. In addition, such systems require periodic on-line retraining. This results in either unavailability of the system or generation of false alarms till such times as the system is retrained.

It is not possible to make a sound and objective judgment as to which type of intrusion detection system is better. Thus hybrid intrusion detection systems have also been proposed. They combine knowledge-based detectors with behavior based detectors. However, they still do not provide an ideal detector. An ideal intrusion detection system is the most difficult to build because it needs to address a set of rather tough and often contradictory requirements. It should be deployable in a heterogeneous and distributed environment. It needs to have a low latency of detection by rapid decision making. It should have both, a very low false positive rate as well as a very low false negative rate. It should be able to scale to large environments. Last but not the least, it should provide strong deterrence to attacks by active real-time monitoring [21].

Instead of trying to build an ideal intrusion detection system we would like to develop tools that complement existing systems. We note that existing intrusion detection systems suffer from two shortcomings. First, not many of them do a good job in handling threats from malicious insiders. These attacks, which are often considered to cause the majority of security breaches, can arise in one of two ways: (i) A user uses perfectly legitimate operations (those that the user is authorized to do) to exploit known system vulnerabilities and launch an attack. (ii) A user uses information and resources that do not fall directly under the category of computer system resources, and launches attack. The latter category is considerably more difficult to prevent, detect or deter than the first category. Addressing such threats is beyond the scope of this work. We are more interested in addressing the first line of attack.

A second concern with intrusion detection systems is that they generate alerts only after they are able to see the misuse signatures or some deviations from norm. A malicious activity may result from a sequence of perfectly innocuous activities. Intrusion detection systems do not report on these activities mostly to prevent information overload for the system administrator. Thus the intrusion detection system generates an alarm only after the cause for alarm has occurred. In many situations however, this may already be too late. These two factors lead us to propose a new approach that can be used to predict attacks arising from user activities. Our work uses Upadhyaya's intent-analysis system [21, 66, 57, 67] to extract activities and pass in to our prediction system as an input.

9.2 Dynamic Reasoning Based User Intent Driven (DRUID)

Upadhyaya et al. [66, 57, 67] propose DRUID system, a host-based concurrent intrusion detection scheme. The system is based on user work profiling [10]. This technique assumes that if one can encapsulate the intent of a user in a reasonable manner, then it is possible to assess intrusions by monitoring the activities on-line. Figure 9.1 shows the flow diagram of DRUID system. The system works as follows. Sometime prior to

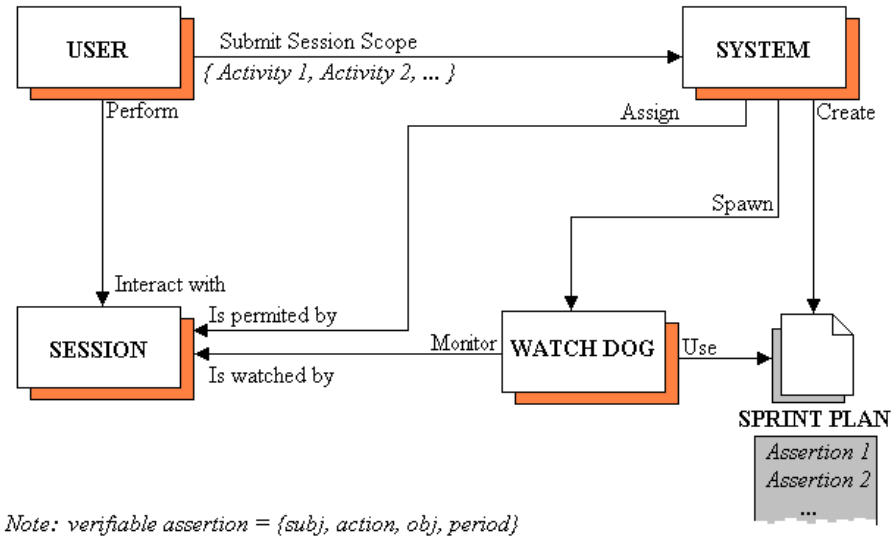


Figure 9.1: Flow Diagram for the DRUID System

login, a user submits a description of his intended system usage. This forms the users session scope. The system converts the scope to a "SPRINT" (Signature Powered Revised Instruction Table) plan which is a list (may be ordered) of quadruples of the form $\langle subject, action, object, period \rangle$. Here "subject" represents a user, "action" is an operation performed by the subject (such as login, logout, read etc.), "object" is the target of an action (such as files, programs, messages, printers etc.), and "period" represents the time interval for the duration of the action. Each quadruple represents a verifiable assertion, a concept that is a generalization of IDESs [22] specification of user characteristics, and can be monitored on-line. When a user is active, a monitor process (called the "Watchdog") monitors the users commands and checks them against the users SPRINT plan. Deviations beyond a certain tolerance limit are considered potential intrusions and DRUID generates alerts for such deviations. In short, DRUID ensures that during a particular session a user remains reasonably within the scope of a previously declared set of activities. Any digression beyond this reasonable limit constitutes a misuse of system and steps are taken to protect against such digressions. However, this approach fails to account for the

fact that a user may remain completely within the scope of a previously declared set of activities and still be able to launch attacks. This is where our approach contributes.

9.3 Attack Tree Base User Intent Intrusion Detection

We propose an alternative method of analyzing users intent by applying attack tree analysis. With this approach, the system only monitors parts of users behaviors which may conceal vicious intents. The advantage of this technique over DRUID is that the system needs not be monitored in all the sessions. This saves the time and reduces the number of resources required by the original SPRINT plan. Our attack prediction system works as follows.

We begin by developing a model of network risks. We augment the notion of attack trees as previously described in the Chapter 5 for this purpose. We then assign the quantitative metric "attack probability $\mathbb{P}$" as node property in the attack tree. Next we iteratively apply each users SPRINT plan to the augmented attack tree, to generate a trimmed attack tree for each user. We call such an attack tree the minimal cut of an attack tree with respect to the user intent. Branches of this trimmed attack tree represent, in a concise manner, all the different ways by which a user can use his assigned job privileges to launch an attack on the system. In the event such a trimmed attack tree does not exist for a particular user, we can safely claim that the users current job description does not pose a threat to the system. This does not necessarily mean, however, that we can cease to monitor this users activities. If we allow a user to deviate from her/his SPRINT plan as is done in the original work [21] then we should continue monitoring the user as proposed in that work. For this work we will assume that the user we are planning to monitor are the ones who, by virtue of their work definition, are able to launch attacks against the system. In the following sections we describe each component of our system in details. We begin with the notion of augmented attack trees to model network risks.

9.3.1 Augmented Attack Tree

Definition 30 AUGMENTED-ATTACK TREE

Let A be a set of attacks (see Definition 3), including the ϕ*–attack. An* Augmented Attack Tree *is a tuple* $AAT = (s_{root}, S, \tau, \varepsilon, \mathbb{P})$*, where*

1. s_{root} *is an attribute which the attacker want to become* true. s_{root} *denotes an attribute* s_k *such that for which* $\nexists\, a \in A |\ s_k \in pre(a)$.

2. $S = N_{internal} \cup N_{external} \cup \{s_{root}\}$ *is a multiset of attributes.* $N_{external}$ *denotes the multiset of attributes* s_i *for which* $\nexists a \in A |\ s_i \in post(a)$. $N_{internal}$ *denotes the multiset of attributes* s_j *for which* $\exists a_1, a_2 \in A |\ [s_j \in pre(a_1) \wedge s_j \in post(a_2)]$.

3. $\tau \subseteq S \times S$. *An ordered pair* $(s_{pre}, s_{post}) \in \tau$ *if* $\exists\ a \in A |\ [s_{pre} \in pre(a) \wedge s_{post} \in post(a)]$. *Further, if* $s_i \in S$ *has multiplicity n, then* $\exists\ s_1, s_2, \ldots, s_n \in S |\ (s_1, s_i)$ $, (s_2, s_i), \ldots, (s_n, s_i) \in \tau$. *A set* $\{s_1, s_2, \ldots, s_n\}$ *becomes a parent set of* s_i*, denoted by* $Pa(s_i)$.

4. ε *is a set of decomposition tuples of the form* $\langle s_j, d_j \rangle$ *defined for all* $s_j \in N_{internal} \cup s_{root}$ *and* $d_j \in \{AND, OR\}$. d_j *is AND when* $\forall a \in A,\ post(a) = s_j \mid Pr(pre(a)) = 1.0 \leftrightarrow Pr(s_j) = 1.0$ *and OR when* $\exists a \in A, post(a) = S_j \mid Pr(pre(a)) = 1.0 \leftrightarrow Pr(s_j) = 1.0$ *(a true state).*

5. $\mathbb{P}$ *is a set of attack probability given by the tuple* $\langle n, m \rangle$ *where m and n are positive integers greater than 0 with* $n \leq m$. *For all* $s_i \in N_{internal} \cup s_{root}$ *there exists a set of attack probability satisfy* $Pr(s_i) \in \mathbb{P}$. *The value of n for the node* $s_i \in S$ *can change over a period of time;however the value of m is fixed for the node* s_i. *The item m is termed the* least effort to compromise subgoal s_i *while the item n is termed the* number of currently compromised subgoals under s_i.

An instance of an augmented attack tree for a hypothetical system is shown in Figure 9.2. We will use this attack tree as our running example. Note, in the figure we have

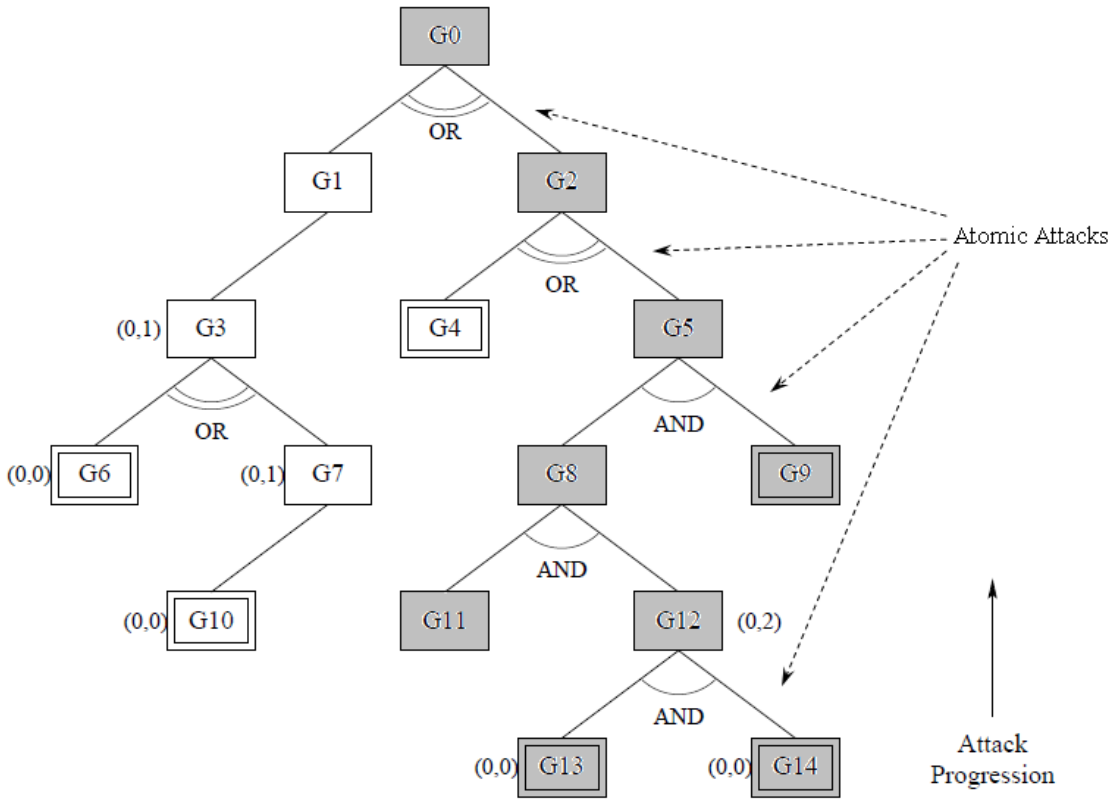


Figure 9.2: Attack Tree Corresponding to a Hypothetical System

shown only some of the labels on the edges to keep the figure simple. In realty, all edges will have their corresponding attack probability labels. The values m and n in the attack probability label of the root node s_{root} are of particular interest to us. The ratio $\frac{n}{m}$ at any given time provides a measure of how far an attacker has progressed towards the ultimate goal in terms of the "least effort" along the most advanced attack path that he has been through. Thus this ratio provides the probability of the system getting compromised at that time. The values m and n corresponding to the root node are computed based on the corresponding values for the other nodes. At this time we show how to compute the value of m for any given node. Note that this is a one time effort that is done during system initialization.

Let us assume without loss of generality that the attacker uses one unit of *effort* to perform one atomic attack that furthers his goal. In other words, each hop along one edge

of the attack tree takes one unit of effort to get through. The *least effort to compromise a subgoal* is the minimum effort the attacker needs to compromise the given subgoal. In general, if a given goal s has an OR-decomposition, the least effort is computed as the minimum least efforts of its child nodes plus one unit effort needed to advance to s from the child node. If the goal s has an AND-decomposition then the least effort is the sum of the least efforts of the child nodes plus all additional unit efforts, one for each child node to go to s. These rules apply to all except ϕ - attacks. For ϕ - attacks, there is no effort need to taken in order to advance the progress. The following definition captures the steps to compute the least effort for a subgoal s.

Definition 31 *Given a subgoal s and its parent set $Pa(s)$ (locate below s in the tree), the* least effort to compromise s*, m_s, is defined as follows.*

1. *If s is a leaf node of the attack tree, the least effort is 0.*

2. *if s is some interior node and is an AND-decomposition, then*
$$m_s = \sum_{s_i \in Pa(s)} \left(m_{s_i} + \begin{cases} 0 & if\ (s_i, s)\ is\ a\ \phi - attack \\ 1 & if\ otherwise \end{cases} \right)$$

3. *if s is some interior node and is an OR-decomposition, then*
$$m_s = Min(m_{s_i}) + \begin{cases} 0 & if\ Min(m_{s_i})\ is\ \phi - attack \\ 1 & if\ otherwise \end{cases}$$

Henceforth we will use the terms attack tree and augmented attack tree interchangeably to mean the latter.

9.3.2 Minimal Cut of Attack Trees w.r.t. User Intent

An augmented attack tree can be used to model system vulnerabilities in a very effective manner. The attack tree describes all possible ways in which a particular attack can be launched. If there are more than one attack against a system that we are concerned about, we can generate separate attack trees for each. However, there are a few drawbacks of the attack tree defined as it is now. First, for a complex system the attack tree can

become quite deep and spread out. Thus it will become difficult to manage. Second, it is possible that a number of users are executing the same set of operations albeit at different paces. In this case, the cumulative effects of these users actions will be reflected on the attack tree. If the users are not colluding this does not give the true picture of the state of the attack. For example, let a user has initially launched an attack and has compromised up to subgoal s_1 of S in an attack tree. Another user has compromised unto subgoal s_2. If the node S is an AND-decompostion of s_1 and s_2, the model will indicate that subgoal S is compromised. However, if the two users are not co-operating, then this is not the case. Thus, we want to refine the concept of attack tree so that we are able to monitor each individual users activities. If we believe there is possibility of collusion among attackers we will maintain the system-wide attack tree as generated so far in addition to the per-user attack tree that we are now ready to define.

That a per-user attack tree is relevant is further strengthened by the following observation. For any attack, we may not always need to know all possible ways the attack can be launched, but rather the practical ways. In the case of attacks from insiders, for example, we are interested only in the activities of authorized users in the system. Thus, we want to determine if the operations that a user executes can lead to an attack. This implies that for a particular user, only a portion of an attack tree is relevant. This leads us to propose the notion of a minimal cut of an attack tree with respect to a user intent. We begin with the following definitions.

Definition 32 *Given an augmented attack tree, AAT, an* attack scenario, AS *of AAT is defined to be a sub-tree of AAT that is rooted at the root of AAT, and follows one or more branches through the tree to end at one or more leaf nodes of AAT such that*

1. *if the subtree has a node that is an AND-decomposition then the subtree must contain all the children of this node, and*

2. *the sub-tree represents one and only one of the many attacks described by AAT.*

Figure 9.2 also represents one possible attack-scenario corresponding to the attack tree, with the shaded boxes constituting the nodes in the attack scenario. Referring to Definition 3, attacks are represented as edges (s_{pre}, s_{post}) in the tree where s_{pre} is the pre-condition of an attack and s_{post} is post condition of the attack. In this figure, some of the attacks are pointed to by dashed arrows. Note that, to successfully execute an attack, the attacker must execute some operations that exploit one or more vulnerabilities in the system. Once a vulnerability has been exploited the attacker executes a set of "attacking operations" that achieve the goal of an attack. Thus,

Definition 33 *A suspicious operations set, SO^a, corresponding to an attack $a \in A$, is a set of operations on specific objects that may potentially lead to the culmination of the attack a. SO^a is a set of tuples of the form $\langle action, object \rangle$. If a is $\phi - attack$, SO^a is an empty set.*

We can identify two different types of operations in a suspicious operations set, SO^a. The first subset of operations is the set *Vul* of vulnerable operations. At least one of the operations in the vulnerable set needs to be executed to exploit a vulnerability. An atomic attack can be launched by exploiting one or more vulnerabilities. Similarly each vulnerability can be exploited by executing one or more vulnerable operations. The second subset of operations is the set *Ao* of attacking operations. All of these needs to be executed to accomplish the atomic attack.

We would like to point out here that we specifically omit the use of the term "sequence" from the definition of suspicious operations set. It is quite possible that only a particular order of execution of the operations will lead to an attack. Since we are interested in estimating the probability of an attack and not just reporting on the attack if and when it is launched, we are interested in all the operations in the set and not just the operations in some particular order.

Definition 34 *The set of intended operations of a particular user, IOs, is a projection of the SPRINT plan for the user over attributes action and object.*

We need to be worried about a users SPRINT plan if some members of the corresponding intended operations set includes a suspicious operations set. We define the intended operations to abet an attack subgoal as follows.

Definition 35 *Given, an attack subgoal s_i (that is a node in an attack scenario) decomposed as the set of atomic attacks a_i, the suspicious operations sets corresponding to each atomic attack, SO_i^a , and a set of intended operations, IOs for a user, we say that the intended operations abet the attack subgoal if and only if one of the following conditions holds true.*

1. *If s_i is an AND-decomposition with m edges then $\forall$ i, $1 \leq i \leq m$; $SO_i^a \subseteq IOs$*
2. *If s_i is an OR-decomposition with m edges then $\exists$ i, $1 \leq i \leq m$; $SO_i^a \subseteq IOs$*

The intended operations *abet* an attack scenario if the intended operations abet all attack subgoal in that attack scenario. Recall that one of our objectives is to determine if a users activities in a system can lead to an attack on the system. A related objective is to determine the exact way in which an attack can be launched with the users intended operations. Thus, given an attack scenario, AS, consisting of subgoals (s_1, s_2,..., s_n), we need to determine for each users intended operations, if the intended operations abet every subgoal $s_i \in AS$. If the intended operations do not abet every subgoal s_i in AS, it implies that this particular attack scenario cannot arise from the users activities. However, this does not mean that another attack scenario cannot arise from the same intended operations. What we need to identity, therefore, is the maximal set of attack scenarios that can arise from a given set of intended operations.

Definition 36 *The minimal cut, MCIOS, of an attack tree, AAT with respect to a particular user intent, IOs, is the minimal subtree of AAT which is rooted at the root of AAT and whose leaf nodes are a subset of the leaf nodes of AAT, such that the subtree includes the maximal set and only the maximal set of attack scenarios that can arise from IOs.*

Algorithm 2 Pruning Algorithm

{**Description:**This algorithm takes an attack tree and a set of intended operations for a particular user and generates a confined version of the attack tree. The confined tree includes those and only those attack scenarios that are in the minimal cut and maybe some more attack subgoals. The original attack tree is represented as a tree structure in which each node except the leaf nodes, contain an array of adjacency lists. Each element in the array represents an attack subgoal. For each attack subgoal,s, $Adj[s]$ contains a reference to a pre-condition subgoal. Each edge in the set τ of edges refer to an atomic attack.
The algorithm assumes the existence of a procedure called OPERATIONS that takes an edge (s_i, s_j) corresponding to a state transition in the attack tree and returns the set of operations that result in the state transition. It also assumes a second procedure called PARENT that takes a node s and returns the parent of s in the tree. The algorithm uses three temporary queues called Explore-List, $\tau\prime$ and $S\prime$ with operation ENQUEUE and DEQUEUE defined. Finally, the algorithm assumes a procedure DRAW_TREE that builds a tree given a set of nodes and edges. }
{**Input:** The attack tree AAT, s_{root}, and the set of intended operations for the user, IOs}
{**Output:** The pruned attack tree containing the minimal cut of attack tree with respect to the user intent.}

```
BEGIN
ENQUEUE(Explore-List, s_root)
τ' ← ϕ
S' ← ϕ
while Explore-List ≠ ϕ do
    s ← DEQUEUE(Explore-List)
    ANY_MET ← false
    for all s_i ∈ Adj[s] do
        ENQUEUE(Explore-List, s_i)
        if OPERATIONS((s_i,s)) ⊆ IOs then
            ENQUEUE(τ',(s_i,s))
            ANY_MET ← true
        end if
    end for
    if (Adj[s] = ϕ && (PARENT(s),s) ∈ τ' ) then
        ENQUEUE(S', s)
    end if
    if ANY_MET = true then
        ENQUEUE(S', s)
    end if
end while
DRAW_TREE(S', τ')
END
```

We now give two algorithms (Algorithm 2 and Algorithm 3) applying which in sequence gives us a minimal cut of an attack tree with respect to a given user intent. We call the

first algorithm the *pruning algorithm* and the second algorithm the *trimming algorithm*. The first algorithm takes an attack tree and generates a subtree rooted at the root of the attack tree such that the subtree contains the desired minimal cut. It removes from the original tree any attack scenarios whose attack subgoals are not abetted by the intended operations. The second algorithm further reduces the subtree produced by the pruning algorithm to produce the minimal cut.

The following theorems hold on the pruning algorithm.

Theorem 1 *Let $AAT = (s_{root}, S, \tau, \varepsilon, \mathbb{P})$ be an augmented attack tree and assume that the pruning algorithm is executed on ATT starting with the root node $s_{root} \in S$. If the user's actions abet an attack then that attack subgoal will be presented in the pruned attack tree generated by the pruning algorithm.*

Proof: To prove soundness of the algorithm we need to prove that during its execution the pruning algorithm explores every state s that forms an attack subgoal for the attacker and includes it in the pruned attack tree. To prove completeness, we must prove that if a node s is included in the pruned attack tree it must form an attack subgoal for the attacker.

First let assume that at the termination of the pruning algorithm, an attack subgoal $s \in S$ which can lead to the root of the attack tree exists such that it is not enqueued by the pruning algorithm. The pruning algorithm starts by exploring the roots adjacent nodes and then iteratively explores the adjacent nodes of these. Thus, if there is an unexplored subgoal s left at the termination of the pruning algorithm, it must be the case that that subgoal s is not be reachable from the root. Then according to Definition 30, subgoal $s \notin S$ which contradicts the assumption.

We prove completeness of the algorithm as follows. Let us assume that at the termination of the algorithm there exist a state transition $(s_i, s_j) \in \tau$ such that $SO_{(s_i,s_j)} \subseteq IOs$ but $(s_i, s_j) \notin \tau\prime$. Since all states in an attack tree have been explored, then (s_i, s_j) must have been explored. By Definition 35, if user intent IOs abet an attack which causes the state transition (s_i, s_j), it must be explored and included in $\tau\prime$. This results in a contradiction.

The pruned attack tree generated by the pruning algorithm may include atomic attacks that however can never materialize from a users activities. This is because the preconditions to these attacks are never satisfied by the user actions. For example, a users intent may abet a remote login attack but may not abet the user to perform an ftp/.rhost attack on the target machine. In this case, the attacker cannot perform these atomic attacks at least till such time as they are not permitted to modify their intent. The next algorithm called the trimming algorithm removes these attack scenarios and produces the minimal cut of attack tree.

Algorithm 3 Trimming Algorithm

{**Description:**This algorithm takes the pruned attack tree generated by the pruning algorithm, and removes attack goals that the user can never reach. }
{**Input:** The set of nodes from the pruned attack tree ordered by traversing the tree in breadth first order and stored in an array $S\prime$, and the corresponding $\tau\prime$}
{**Output:** Minimal cut of an attack tree with respect to user intent}
BEGIN
$\mathcal{K} \leftarrow$ SIZEOF($S\prime$)
while $\mathcal{K} > 0$ **do**
  $s_i \leftarrow S\prime[\mathcal{K}]$
  $\mathcal{K} \leftarrow \mathcal{K} - 1$
  $valid \leftarrow false$
  if $Adj[s_i] = \phi$ **then**
    $valid \leftarrow true$
  else
    for all $s_j \in Adj[s_i]$ **do**
      if $s_j \in S\prime$ **then**
        $valid \leftarrow true$
      else
        remove (s_i, s_j) from $\tau\prime$
      end if
    end for
  end if
  if $\neg valid$ **then**
    remove s_i from $\tau\prime$
  end if
end while
DRAW_TREE($S\prime, \tau\prime$)
END

Theorem 2 *If the trimmed attack tree generated from a pruned attack tree by the application of a users intended operation contains an attack scenario, then the intended operations abet that particular attack scenario.*

Proof: Let assume that there exist subgoal s (which its preconditions are met by the user intents) mistakenly removed by the trimming algorithm. According to the semantic of the trimming algorithm, the *while loop* in the trimming algorithm explores every node in the input pruned tree and the *for loop* trying to discover all possible paths from the leaf nodes to a subgoal currently explored by the *while loop* s iteration. Then the remove instruction removes a subgoal if and only if the previous for loop could not find such a path to the leaf node. We will split subgoal s in 2 cases.

Case 1: If subgoal $s \in S\prime$ is an initial subgoal, this case could not happened since the if statement of *line 10* in the procedure detect the leaf node. Then the initial subgoal $s \notin$ Minimal cut if and only if $s \notin S\prime$ which contradicts the previous assumption.

Case 2: If subgoal $s \in S\prime$ is an intermediate subgoal, s will be removed by the trimming algorithm if and only if s can not be reached from any initial subgoal. This mean the preconditions of an intermediate subgoal s are not met which contradict to the previous assumption.

9.4 Compute Probability of Attack from User Activities

We now use the minimal cut of an attack tree with respect to a user intent to determine the probability of an attack originating from that user. Algorithm 4 computes the attack probability label of a subgoal at any given time t. By applying this algorithm on the root node of the minimal cut of an attack tree for a user, we get the attack probability label

Algorithm 4 Risk Evaluation Algorithm

{**Description:**This algorithm takes an attack tree subgoal A and returns the attack probability label (n,m) for that goal. Here n refers to the number of nodes that have been compromised on the most advanced attack paths and m refer to the least-effort needed to compromise A on that path.}
{**Input:** A subgoal of an attack tree}
{**Output:**

1. Number of subgoals that have been compromised along the most advanced attack path.
2. Least-effort needed to compromise the subgoal along the most advanced attack path.

}
Let n = number of currently compromised subgoal under A on the most advanced attacking path.
Let m = least-effort needed to compromise A on the most advanced path.

BEGIN
if A is a leaf node **then**
  return (0,0)
end if
if A is an AND-Decomposition **then**
  $(n,m) \leftarrow \sum(Risk-Evaluation(A_i) \mid \forall A_i\ Childnodes\ of\ A)$
  $k \leftarrow \sum \begin{cases} 0 & if\ A_i\ is\ \phi-attack \\ 1 & if\ otherwise \end{cases}$
else
  $(n,m) \leftarrow MAX(\frac{n}{m})\ from\ Risk-Evaluation(A_i) \mid \forall A_i\ Childnodes\ of\ A)$
  $k \leftarrow \begin{cases} 0 & if\ MAX(\frac{n}{m})\ is\ \phi-attack \\ 1 & if\ otherwise \end{cases}$
end if
if A is compromised **then**
  return(n+k,m+k)
else
  return(n,m+k)
end if
END

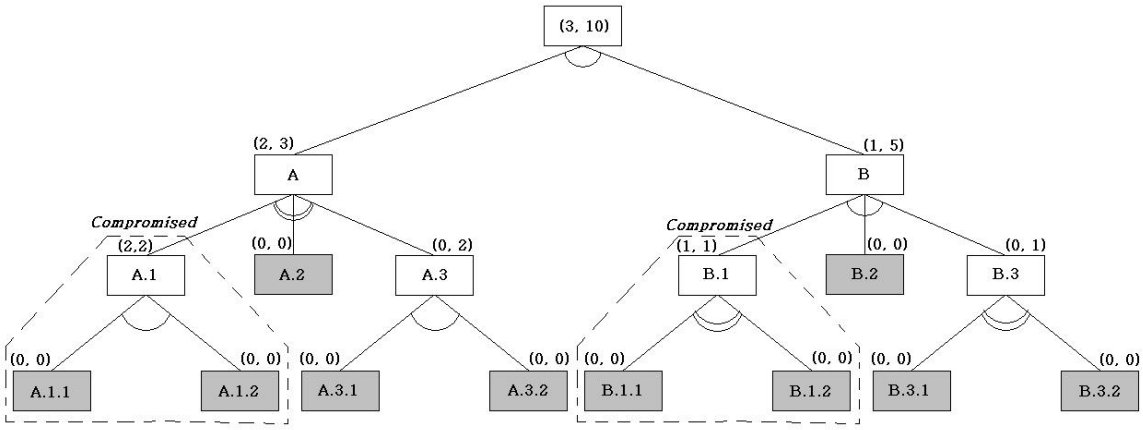


Figure 9.3: Attack Probability on Minimal Cut of an Attack Tree

corresponding to the root at time t. The ratio $\frac{n}{m}$ at time t gives the probability of the users attack succeeding at time t.

To illustrate this algorithm, Figure 9.3 shows an example trace for Algorithm 4. Assume that at time t a malicious user compromises two subgoals as shown in the figure. The algorithm computes the $\langle n, m \rangle$ value for the root of the tree as follows. All leaf-nodes return value (0, 0). Node A.1 and A.3 have the summation equal to (0, 0) since their immediate child nodes are all leaf nodes. When A.1 is compromised the procedure returns (2, 2). Similarly for A.3 the procedure returns (0, 2). For B.1 and B.3 the values are (1, 1) and (0, 1) respectively. At this point the value of (2, 2) tells us that it takes 2 units effort to compromise A.1. The attacker has already compromised A.1 but no damage has been done on A.3. Next we calculate value on A. Eventually, since the root is an AND-decomposition on two branches A and B the least effort is 8 + 2 = 10 and the number of compromised nodes is 3. This yields a probability of attack value of $\frac{3}{10}$ at time t.

9.5 Chapter Summary

In summary, this section applies the augmented attack tree with the quantitative framework to the user intent. The intent-based intrusion detection has been originally

proposed by Upadhyaya et al. In DRUID framework, the user declares the intended activities prior to the login. Based on the users intention, the system generates the SPRINT plan. Then the system detects the misbehavior by comparing real-time activities with the SPRINT plan. The deviation from the SPRINT plan shows the sign of intrusion. However, an intrusion can be launched from the activities which do not deviate from the SPRINT plan. Our work addresses this problem.

Our attack prediction takes the users SPRINT plan as an input. This plan is submitted to the augmented attack tree to generate the trimmed attack tree. We call this the minimal cut of an attack tree with respect to the user intent. Branches of this minimal cut represent all the different ways by which an attacker can use his/her assigned jobs to launch an attack. In the event such a minimal cut does not exist, we can safely claim that the users jobs do not pose a threat to the system.

Chapter 10

A SYSTEMATIC APPROACH FOR INVESTIGATING COMPUTER ATTACKS USING ATTACK TREES

System log files play a major role in the investigation of computer attacks. However, system log files almost always have a flat, sequential structure. They grow very large in size and contain significant amount of information that are not relevant to the specific attack. Thus, it is extremely difficult and time consuming to extract evidence of attack from the log file. In this paper, we propose an automated approach for filtering out irrelevant information from a system log file and creating a shorter log that contains sufficient information about a computer attack.

10.1 Introduction

Following a large scale computer attack an investigator (system administrator or law enforcement official) often needs to make a reasoned determination of who caused the attack, when, and what the exact sequence of events were that led to the attack. The system log that contains records of all events that occur in the system, is used for this purpose. However, the process of this investigation is almost always manual, frequently prone to errors and often inconclusive. There are three major contributing factors to this.

1. There is currently no standardized model for log file organization. The log file is usually a simple, flat structured text file (see Figure 10.1). There is no minimum requirement for information that needs to be stored in the log file.

```
1697 05/04/1998 08:51:19 00:00:06 172.016.113.084 172.016.113.064/28  cp
2525 05/04/1998 09:12:42 00:00:01 172.016.114.158 172.016.114.128/28  nessus
2538 05/04/1998 09:13:21 00:00:07 172.016.114.159 172.016.114.128/28  nessus
2701 05/04/1998 09:16:02 02:42:20 135.013.216.191 135.013.216.10/24   mv
2731 05/04/1998 09:17:09 00:05:00 135.013.216.182 135.013.216.10/24   telnet
3014 05/04/1998 09:23:44 00:00:07 172.016.114.158 172.016.114.128/28  ftp
3028 05/04/1998 09:24:54 00:00:07 172.016.114.159 172.016.114.128/28  cp
3461 05/04/1998 09:37:14 00:00:13 172.016.114.159 172.016.114.128/28  telnet
4598 05/04/1998 10:01:51 00:00:01 196.037.075.158 196.037.075.10/24   finger
4612 05/04/1998 10:02:37 00:00:01 196.037.075.050 196.037.075.10/24   xterm
4834 05/04/1998 10:09:39 00:00:01 172.016.114.158 172.016.114.128/28  rlogin
4489 05/04/1998 10:10:22 00:00:01 195.073.151.050 195.073.151.10/24   smtp
4859 05/04/1998 10:10:33 00:00:01 195.073.151.150 195.073.151.10/24   smtp
4930 05/04/1998 10:11:36 00:00:06 172.016.114.158 172.016.114.128/28  telnet
5014 05/04/1998 10:13:55 00:00:06 172.016.114.148 172.016.114.128/28  bind
5092 05/04/1998 10:14:59 00:00:10 172.016.114.159 172.016.114.128/28  suid
5308 05/04/1998 10:21:38 00:00:01 194.027.251.021 194.027.251.021/24  smtp
5323 05/04/1998 10:23:11 00:00:01 196.037.075.158 196.037.075.5/24    ssh
5456 05/04/1998 10:28:50 00:00:01 194.027.251.021 194.027.251.021/24  ftp
5467 05/04/1998 10:29:26 00:00:01 196.037.075.158 196.037.075.10/24   sftp
5730 05/04/1998 10:36:58 00:00:03 135.008.060.182 135.008.060.10/24   ssh
7270 05/04/1998 11:09:08 00:00:02 135.008.060.182 135.008.060.10/24   mv
8098 05/04/1998 11:33:26 00:00:13 172.016.114.158 172.016.114.128/28  telnet
9057 05/04/1998 11:57:00 00:00:01 172.016.112.207 172.016.112.10/24   smtp
9113 05/04/1998 11:58:26 00:00:08 172.016.114.148 172.016.114.128/28  telnet
9352 05/04/1998 12:48:01 00:00:01 172.016.113.078 172.016.113.64/28   cp
```

Figure 10.1: Flat Structured System Audit Log

2. There are currently no established procedure for filtering and retrieving information from the log file other than a sequential backward scan from the most recent entry. System administrators use ad-hoc regular expression searching commands to extract information from the log file.

3. When a security incident occurs, the system administrator does not usually have any information about what are the things to look for in the system log. Most of the time the person has to rely on her experience or intuition for this purpose.

To address some these concerns, we propose an attack tree based approach for filtering log files. An attack tree model is proposed that allows one to capture all the different ways a particular system can be attacked based on currently available knowledge of system vulnerabilities and exploits. The filtering approach then selects a set of records from a log file that are relevant to the current attack being investigated by matching against the

attack tree. Subsequently, other SQL queries can be used to extract evidence from this table in an automated manner.

The rest of this paper is organized as follows. In section 10.3 we present the attack tree model for computer attacks. Section 1.3 describes our approach of filtering log files. Finally section 1.4 concludes the paper.

10.2 Attack Correlation Modeling

Attack trees have been previously proposed [25, 43, 60] as a systematic method for specifying system security based on varying attacks. They help organize intrusion and/or misuse scenarios by

1. utilizing known vulnerabilities and/or weak spots in the system, and

2. analyzing system dependencies and weak links and representing these dependencies in the form of an And-Or tree.

For every system that needs to be defended there is a different attack tree. The nodes of the tree are used to represent different stages (milestones) of an attack. The root node of the tree represents the attackers ultimate goal, namely, cause damage to the system. The interior nodes, including leaf-nodes, represent possible system states during the execution of an attack. System states can include level of compromise by the attacker (such as successful access to a web page or successful acquisition of root privileges), altering the system configuration (such as a modification of trust or access control or escalation of user privilege) or state changes achieved on specific system components (such as implantation of Trojan Horses) and other sub-goals that will ultimately lead to the final goal (such as sequence of vulnerabilities exploited). Branches represent a change of state caused by one or more action taken by the attacker. Change in state is represented by either AND-branches or OR-branches. Nodes may be decomposed as:

1. a sequence of events (exploits) all of which must be achieved for this sub-goal to succeed. An example of this can be the changing of the file mode of /proc/self/ files and the execution of suid command that cause the root to be compromised (CVE-2006-3626). This is represented by the events being combined by AND branches at the node.

2. a set of events (exploits), any one of which when occurring will result in the sub-goal succeeding. An example of this is the stack buffer over- flow that exploits the libtiff library in SUSE v10.0 (CVE-2006-3459) or the SQL injection in Bugzilla v2.16.3(CVE-2003-1043) which cause the root compromised (assume both service run in the same machine). This is represented by the events being combined by OR branches at the node.

The notion of attacks trees is related to the notion of attack graphs that have been proposed by other researchers [12, 38, 47, 62] for network vulnerability analysis. The difference is in the representation of states and actions. Attack graphs model system vulnerabilities in terms of all possible sequence of attack operations. As pointed out by Ritchey and Ammann [53] a major shortcoming of this approach is its scalability. On the other hand, attack trees model system vulnerabilities in terms of cause and effect. Sequential ordering of events does not have to be captured in attack trees. Thus constructing an attack tree is significantly less complex than attack graphs. An often cited criticism of attack trees (vis-a-vis attack graphs) is that they are not able to model cycles. However, we believe that this criticism is valid only in cases where attack trees are used to represent sequence of operations leading to attacks, not when they are used to represent the dependency of states reached. A second criticism of using attack tree to model attack scenarios is that they tend to get unwieldy. We assume that a technique is available to generate an attack tree corresponding to the network system we are attempting to defend. We use the following running example to describe how an attack tree is used to represent system

vulnerabilities. Figure 10.2 shows the configuration of the network for a (hypothetical) small company. The company has installed a firewall to protect itself from the Internet. In the de-militarized zone (DMZ) there is the company web server. Other relevant machines are on the local area network behind the firewall. The companys system administrator has configured the firewall to block port scans and flooding type attacks. The firewall allows incoming connections only via port 25 and 80 (for smtp and http respectively). We assume John Doe, a disgruntled employee plans to attack his own company system. He performs a vulnerability scan of company network using his insider knowledge and determines that he needs to obtain "root privilege on the Web server" to achieve this objective.

John Doe identifies that there are two alternative ways to gaining root privilege his ultimate goal. One way is via launching the *FTP/.RHOST* attack. In this attack, the .rhost file on the Web server will be first overwritten by a .rhost file of John Does choosing (namely the .rhost file on his own machine) by exploiting a known vulnerability. At this stage the Web server will begin to trust John Does machine. This allows John Doe to remotely login on the Web server from his machine without providing a password. Once John Doe is a user on the Web server he will conduct the well known setuid buffer overflow attack and gain root privilege.

A second way of attacking the Web server is via buffer overflow attack on the local DNS server. John Doe knows that the system administrator uses an old unpatched version of the BIND DNS application program. John can perform the BIND buffer overflow attack on the local DNS server and takes control of this machine. He can then install a network sniffer on this DNS server to observe sessions across the entire network. Eventually he can hijack the system administrators telnet session to the Web server and gain root privilege there. The above attacks can be concisely represented in the form of the simple attack tree shown in Figure 10.3. The interesting feature of the simple attack tree is that it captures in a precise manner all the possible known ways in which a system can be breached. While it does not capture unknown or zero day attacks, we believe that the

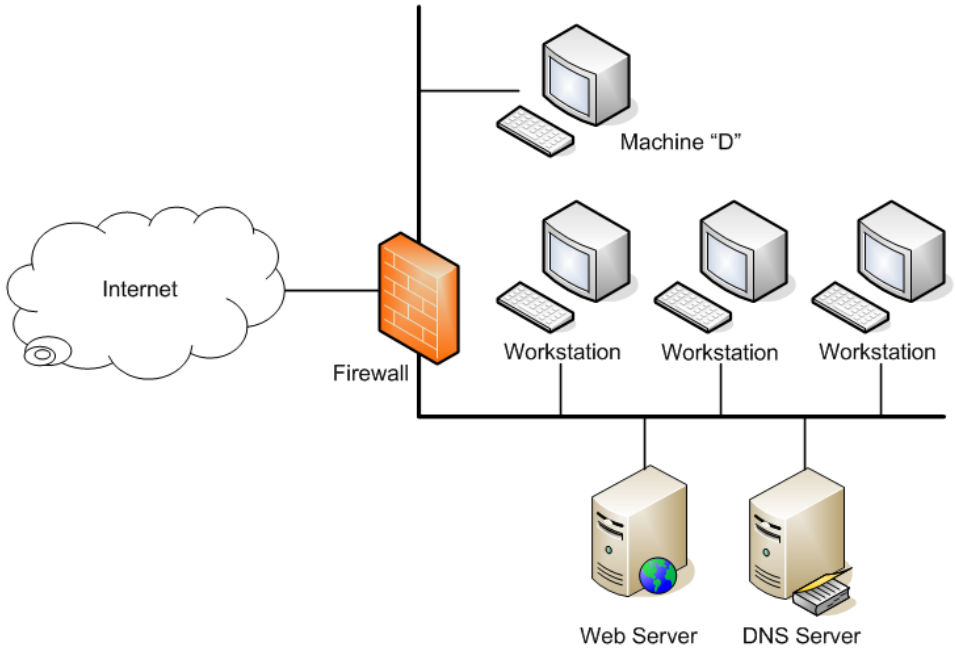


Figure 10.2: Small Company Network Being Analyzed

attack tree model can be effectively used for helping log file analysis in a vast majority of the cases. Following an attack, we just need to look out only for those operations that lie in sequence along the paths leading to the attack. With the attack tree of Figure 10.3, for example, if we determine that root privilege at the web server has been compromised we need to look for just the sequence of operations given by the left branch or the right branch. Moreover, these operations need to be in the exact temporal order as given by the nodes going down in the tree. Any other order is not relevant for this particular incident. In fact, if the log file filtering and analysis approach discussed next, does not show up a sequence of events leading to a specific attack, we know that the attack in question is a zero day attack.

10.3 Signature Embedded Attack Tree

To use an attack tree in forensic investigation, we extend the notion of the attack tree (Def. 4) discussed in the chapter 5 by associating each branch with a sequence of

malicious operations that could have been used in the attack. We call such a tree an signature embedded attack tree. We also formalize additional notions as the following.

Definition 37 *An* atomic event *is an ordered pair* $\langle operation, target \rangle$.

Definition 38 *An atomic event is an* incident *if it's execution contributes towards a system compromise.*

Definition 39 *An* incident-choice *is a group of related incidents, the occurrence of any one of which can contribute towards the state transition in the attack tree.*

Definition 40 *An attack Signature* $SIG_{pre,post}$ *is a sequence of incident-choices (IC)* $\langle IC_1, IC_2, \ldots, IC_n \rangle$ *such that the sequence* $(incident_i \in IC_1, incident_j \in IC_2, \ldots, incident_m \in IC_n)$ *constitute an attack.*

The attack signature corresponding to the attack discussed in CVE-1999-1562 involving an execution of wuftp in a target machine (let's call machine A) and resulting in clear text password disclosure will be represented by: ((ftp, A),(debug, A),(open localhost, A), (''user name root", A), (''password xxx", A), (quote user root, A),(quote pass root, A)).

Definition 41 SIGNATURE EMBEDDED ATTACK TREE

Let A be a set of attacks (see Definition 3). An Signature Embedded Attack Tree *is a tuple* $AAT = (s_{root}, S, \tau, \varepsilon, SIG)$, *where*

1. s_{root} *is an attribute which the attacker want to become true.* s_{root} *denotes an attribute* s_k *such that for which* $\nexists\, a \in A |\, s_k \in pre(a)$.

2. $S = N_{internal} \cup N_{external} \cup \{s_{root}\}$ *is a multi-set of attributes.* $N_{external}$ *denotes the multi-set of attributes* s_i *for which* $\nexists\, a \in A |\, s_i \in post(a)$. $N_{internal}$ *denotes the multi-set of attributes* s_j *for which* $\exists\, a_1, a_2 \in A |\, [s_j \in pre(a_1) \wedge s_j \in post(a_2)]$.

3. $\tau \subseteq S \times S$. *An ordered pair* $(s_{pre}, S_{post}) \in \tau$ *if* $\exists\ a \in A |\ [s_{pre} \in pre(a) \wedge s_{post} \in post(a)]$. *Further, if* $s_i \in S$ *has multiplicity n, then* $\exists\ s_1, s_2, \ldots, s_n \in S |\ (s_1, s_i), (s_2, s_i), \ldots, (s_n, s_i) \in \tau$. *A set* $\{s_1, s_2, \ldots, s_n\}$ *becomes a parent set of* s_i*, denoted by* $Pa(s_i)$.

4. ε *is a set of decomposition tuples of the form* $\langle s_j, d_j \rangle$ *defined for all* $s_j \in N_{internal} \cup \{s_{root}\}$ *and* $d_j \in \{AND, OR\}$. d_j *is AND when* $\bigwedge_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ *is* true*, and OR when* $\bigvee_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ *is* true.

5. $SIG_{pre,post} \in SIG$ *is an attack signature associates to the edge* $(s_{pre}, S_{post}) \in \tau$.

An AND-decomposition node s_v, means that each subgoal of s_v represented by a child of s_v needs to be reached in order to reach s_v. An OR-decomposition means that the goal s_v can be reach only if any one of the subgoals is reached. Note that reaching a child goal is only a necessary condition for reaching the parent goal and not a sufficient condition.

10.4 Forensic Investigation with a Signature Embedded Attack Tree

We now show how to use an augmented attack tree to support forensic investigation. The attack tree will be used to simplify the process of looking for activities that could have potentially caused the attack. The process works briefly as follows. First the augmented attack tree is used to prepare the set of incidents for all attack signatures. It is then used to filter out suspicious activities from non-suspicious ones. All suspicious activities from this stage is next written to a relational database for further investigation. We propose the following database structure to store the filtered log file. The log file table includes five fields $id, timestamp, source, source-group, operation, target$, and $duration$. The source field stores the IP address of the connection originator and source-group field contains the network address of the originator, if available. The target field similarly stores the destination address of the network connection. (Note that if the investigation policy dictates using other information from a log file then those can also be included in the table.)

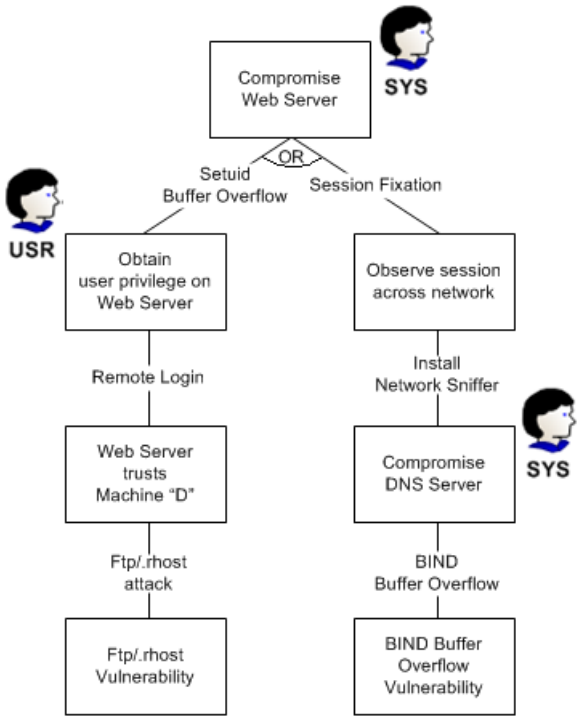


Figure 10.3: Attack Tree for Network of Figure

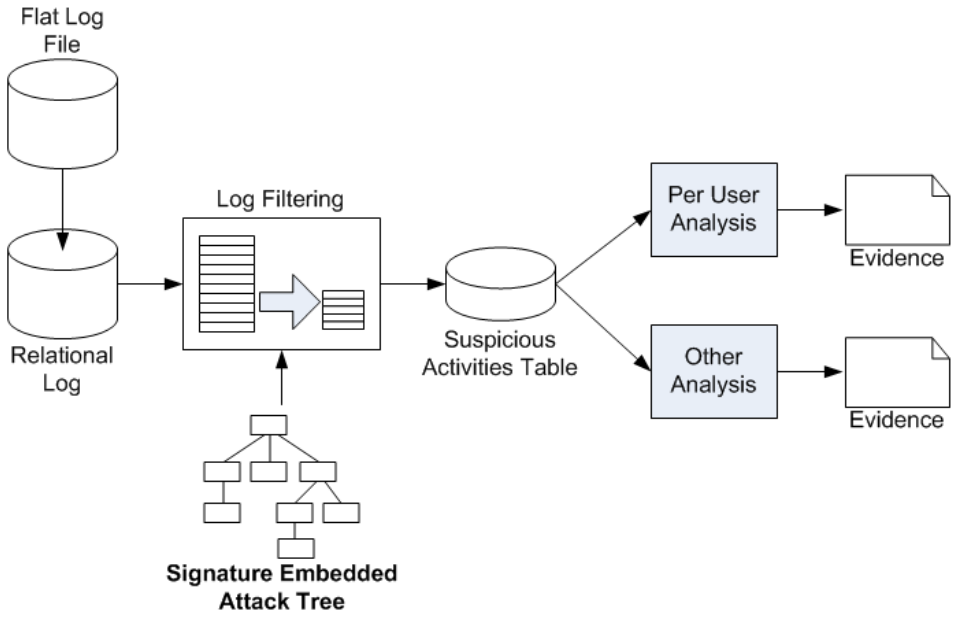


Figure 10.4: Log File Investigation Process

10.4.1 Use of Signature Embedded Attack Tree in Log File Filtering

Algorithm 5 Investigate(Node s_u, Table *systemlog*)

{**Description:**This algorithm begins at the root of an augmented attack tree. It recursively calls itself during the subtree traversal under a node s_u in depth-first manner to visit all edges (exploits). For each edge, the algorithm extract the suspicious incidents from edge visits. Finally, the algorithm returns a set of evidence from the log file as appearing in an attack tree.}
{**Input:** Node s_u (initial from root), Table *Suspected Activities*}
{**Output:** The list of all suspicious activities.}
BEGIN
if s_u is a leaf node **then**
return ϕ
else
for all $s_v \in$ Adj[s_u] **do**
$SIG_{s_u,s_v} \leftarrow getSignature((s_u, s_v))$
for all $incident_i \in SIG_{s_u,s_v}$ **do**
RESULT ← SELECT *logRecord*
FROM *System Log*
WHERE (operation,target) = $incident_i$ AND timestamp < u.timestamp
if *RESULT* $\neq \phi$ **then**
INSERT *RESULT* INTO TABLE *Suspected Activities*
end if
end for
v.timestamp ← the earliest timestamp of all *RESULT* in the previous loop
Suspected Activities ← Investigate(s_v,*systemlog*)
Mark (s_u, s_v) if s_v is compromised
end for
if $s_u.\varepsilon$ = "AND" && $\forall s_v \in Adj[s_u] \mid (s_u, s_v)$ is marked **then**
Mark s_u.compromised = *true*
end if
if $s_u.\varepsilon$ = "OR" && $\exists s_v \in Adj[s_u] \mid (s_u, s_v)$ is marked **then**
Mark s_u.compromised = *true*
end if
return *Suspected Activities*
end if
END

The signature embedded attack tree is used to prepare the set of incidents for all attack signatures. The set of incidents is then used to filter out suspicious activities from clean activities. The log file filtering algorithm sequentially executes SQL queries to extract suspicious activities from the original log file database. The results from this algorithm is written to a separate table called the suspicious activities table for further

investigation. This table has the same schema as the log file table but the size of these table is much smaller than the size of whole log file. Algorithm 5 to filter the log file based on the augmented attack tree.

The algorithm works as follows. It commences the investigation at the root node of the attack tree. It traverses every edge incident to the root node. For each edge, the algorithm extracts the attack signature SIG_{s_u,s_v} given by the label of the edge. As mentioned earlier, the attack signature is the sequence of steps where an attacker may, or may not, have a choice of incidents (operation on a particular machine/target) to execute. For each step in the attack signature, the algorithm searches the log file for matching operations. An incident in the table is said to match the signature if the operation is executed on the particular machine or against the particular target as indicated in the attack signature. Moreover, only matched incidents that were executed prior to the time that the root node was compromised are suspected. Next, the suspected incidents are recorded into the suspicious activities table by the selection procedure. Once the algorithm finishes the exploration on a particular edge (s_u, s_v), it sets a time threshold for node s_v by selecting from the earliest incidents in (s_u, s_v). This threshold is assumed to be the time instance at which node s_v has been compromised. Thus there is no need to suspect any incidents in the subtree(s) under s_v that executed after this time. The next step in the algorithm recursively calls itself to investigate the subtree under s_v from which the edge (s_u, s_v) emerged. All subtrees under the node are explored in this recursive manner. The recursive calls ensure that the algorithm has thoroughly explored all subtrees. Once all subtrees under the root node or any intermediate node s_u have been explored, the algorithm marks an edge (s_u, s_v) if it finds evidence that shows that all steps in the attack signature SIG_{s_u,s_v} have been executed. If node s_u has an AND-decomposition, node s_u is considered compromised when all exploits (represented by edge (s_u, s_v)) that are incident to s_u together with the state s_v that the exploit has emerged from, are marked. If node s_u has OR-decomposition, node s_u is compromised when any one of its branch together with the state s_v is marked by the

recursive call. Finally, the algorithm returns the augmented attack tree where nodes have been marked compromised together with the suspicious activities table. The latter stores the evidence of attack for the further analysis.

10.4.2 Use of Augmented Attack Tree in Identifying Possible Suspect

Algorithm 6 Investigate(Node s_u, specific-source, Table *suspicious activities*)

{**Description:**This algorithm takes the root of an attack tree as an initial input. It recursively calls itself during the subtree traversal under s_u in depth-first manner to visit all edges (exploits). For each exploit, the algorithm extracts the suspicious incidents from edge visits. Finally, the algorithm returns the set of evidence as appearing in an attack tree.}
{Input: node s_u (initial from root), specific-source, suspicious activities table}
{Output: evidence-log-for(specific-source)}
BEGIN
if s_u is a leaf node **then**
  return ϕ
else
  for all $s_v \in$ Adj[s_u] **do**
    $SIG_{s_u,s_v} \leftarrow getSignature((s_u, s_v))$
    for all $incident_i \in SIG_{s_u,s_v}$ **do**
      RESULT ← SELECT *logRecord*
                 FROM *specific-source*
                 WHERE (operation,target) = $incident_i$ AND timestamp $< s_u$.timestamp
      if *RESULT* $\neq \phi$ **then**
        INSERT *RESULT* INTO TABLE evidence-log-for(*specific-source*)
        Mark *RESULT* with the exploit from edge (s_u, s_v)
      end if
    end for
    s_v.timestamp ← the earliest timestamp of all *RESULT* in the previous loop
    evidence-log-for(*specific-source*) ← Investigate(s_v,specific-source,*suspicious activities*)

    Mark (s_u, s_v) if s_v is compromised
  end for
  if $s_u.\varepsilon$ = "AND" && $\forall s_v \in Adj[s_u] \mid (s_u, s_v)$ is marked **then**
    Mark s_u.compromised = *true*
  end if
  if $s_u.\varepsilon$ = "OR" && $\exists s_v \in Adj[s_u] \mid (s_u, s_v)$ is marked **then**
    Mark s_u.compromised = *true*
  end if
  return evidence-log-for(*specific-source*)
end if
END

The input to this stage is the suspicious activities table resulting from the log-file filtering process. Hopefully, this table is more manageable than the original log file. At this point, if the investigator sorts the suspicious activities table by source aggregated by source-group, she will have a list of candidate sources for the attack. From this list, we can conduct further investigation on a per source basis either to reinforce or discard our belief about the specific source. Algorithm 6 is intended for this purpose. The output of this algorithm will be a table named evidence-log-for(source) where "source" is the identity of the source being investigated. This table has almost the same schema as the suspicious activities table. The only difference is that this table has an extra column called exploit. This field holds the exploit label corresponding to a relevant edge of the attack tree. If the algorithm returns a non-empty table it indicates that the source is guilty. On the other hand, if the algorithm returns an empty table no conclusion can be reached regarding the guilty verdict of the source. This is because of the possibility of zero-day attacks. Thus, we would like to emphasize that Algorithm 6 should not influence the decision as it only marks evidence of activities that were possibly involved in an attack.

This algorithm is similar to the log-file filtering algorithm. The difference is the SQL queries that are executed on a per source basis for sources in the suspicious activities table. The algorithm marks the suspected record with the exploit label to accommodate the final decision. The investigator may use these labels to map the evidences back to an exploit in attack tree.

The table evidence-log-for(source) holds the activities that are believed to be responsible for an attack against the system. The records are ordered chronologically. Typically if there exist an internal node that is marked by Algorithm 6, then the suspect is almost certainly responsible for the attack.

10.5 Chapter Summary

System log files play a major role in the investigation of computer attacks. If properly maintained a system log contains complete information about all pertinent events culminating in an attack. However, these files are difficult and time consuming to process. This is partly because the files are almost always flat structured; this requires a sequential scan for extracting information. They grow very large in size thus becoming unmanageable. Last but not the least they contain an enormous amount of information a major portion of which is not related to the attack being investigated. For facilitating investigation, an automated approach to log file filtering is needed. In this paper, we propose such a filtering approach. Our approach is capable of eliminating irrelevant information from a log file and creating a shorter file that contains most of the times, sufficient information about a computer attack.

Our approach is based on the observation that in order to attack a system an attacker must exploit certain vulnerabilities that exist in the system in a particular sequence. We can represent these exploits in the form of an attack tree, the root of which gives the final attacked state of the system. While unknown or zero day attacks cannot be captured in this manner, a vast majority of attacks are not zero days. Thus this representation of attacks against a particular system is useful. We propose an algorithm that filters a log file by correlating it with the attack tree that is relevant for the particular attack in question. The filtered log is much smaller than the original file as it contains only relevant information. This filtered log file can form the basis of further investigation. A lot of work remains to be done. Apart from not being able to capture information about zero day attacks, our protocol assumes that the log file available is trusted. That is, the attacker has not tampered with the log file to remove evidence of attack. How to ensure this in real life is an open challenge. We are currently investigating this problem. We also assume that a central log file is available that records all events occurring on the network. In reality, each machine on a network maintains its own log. Thus, there needs to be a protocol to merge these log files into one comprehensive whole. This remains part of our future work.

Chapter 11

CONCLUSION AND FUTURE WORKS

The enterprise attack model serves as a basis building block in the risk management, intrusion detection, and forensic investigation. In conclusion, we pioneer three area of contributions in the problem domain of information security.

11.1 Contributions to Risk Analysis and Risk Mitigation Analysis

An attack tree can help the security manager analyze the effectiveness of security controls and prioritize risk. What's always facing the security risk management is the difficulty in assessing the risk and selecting security controls to mitigate the risk.

In the first challenge, although there are available methodologies to compute the risk, but the actual risk is also tied to the factors out of reach by the current assessment methods. We have not seen an assessment model that takes into account the propagated effect. That is even a low risk vulnerability can lead to the catastrophic damage to the system if an attacker can use that risk as an entry point to attack. Beside, the risk is also tied to its locations and how much it can propagate to make other undesirable events. Without the knowledge about cause and effect of the possible attacks, the security administrator cannot truly assess the damage from a given list of vulnerabilities alone. An attack tree / attack graph assesses the risk in terms of dependencies between preconditions, attacks, and outcomes which, in turns, become the part of preconditions of other attacks. Assessing the damage on each outcome individually, the security analysts can simulate the possible outcomes that capture all propagated effects if a given attack is executed. As-

sessing the risk with an attack tree or attack graph is a systematic method. It provides more accurate and reasonable results.

The second challenge is even harder, assuming that the risk assessment can be evaluated by other means; it is still difficult to assess the security controls the system should employ given a number of countermeasures currently available. It is a difficult task because there is no known security control that can address all vulnerabilities. Even worse, some countermeasures address one problem but worsen or open other vulnerabilities (e.g. cryptography allows secure communication but weaken the system against the denial-of-service attack). System administrators have to deal with the trade-off between costs and benefits of deploying the security controls. An attack model also allows security analysts to consider many what-if scenarios during the risk mitigation analysis. For example, the analyst can easily simulate the effects of adding a new Firewall in a certain location in the network. Similarly, a security administrator can identify potential damage if a certain control (or even a certain set of controls) is removed. Therefore, evaluating security controls can be easily made.

11.2 Contributions to Intrusion Detection and Incident Response

In the area of Intrusion Detection System (IDS), we realize that such systems rely heavily on configuring the sensors sensitivity to detect certain symptoms. Setting up these sensors usually involves the trade-off between sensitivity and false alarms. When the sensors are set to report all suspicious events, the sensors frequently issue alerts on benign events. This often results in annoying and functionality drawbacks. On the other hand, decreasing the sensitivity reduces the ability to detect malicious events.

System administrators can use attack tree to deal with this problem by having an attack tree correlates alerts and by applying probabilistic risk analysis on correlated alerts. Evidencing alerts with an attack tree will also give an ability to predict future attacks, reduce volume of information to be analyzed, and increase an accuracy of IDS.

Moreover, an attack tree can help administrating other kinds of analysis in the area of intrusion detection. For example, marking the paths in the attack tree when an IDS detects an attack incident allows systematic and real-time intrusion alert, determining where the new IDS components should be deployed for the best coverage, exploring trade-offs between different security policies and between different software/hardware configurations, and identifying the worst-case scenarios and prioritizing the defense accordingly.

11.3 Contributions to Forensic Investigation

After a break-in, forensic analysis is used to find a trace of attackers actions to assess the actual damage and identify how to reinstate back to the last normal state. In addition, if a legal action is required, analysts seek evidences to prove that a sequence of certain activities is, indeed, a comprising of the coherent attacks and not just a series of benign events. Typical forensic processes involve investigating the system log for the known trace of attack patterns. This task relies heavily on the analyst's experience and thoroughness. The task becomes even harder when intruders obfuscate attack steps by slowing down the pace of the attack or varying specific steps. An attack tree is proposed to help improving the accuracy and speed of the forensic process by submitting the data extracted from IDS logs to a formal reference model based attack tree.

11.4 Future Works

This dissertation innovates an enterprise risk management analysis. In this dissertation, we propose a formalism of enterprise risk model to help analyst understand the enterprise risk model so as to be able to apply the model on different risk analysis methods. We demonstrate the feasibility by implementing the comprehensive attack tree analysis tool that discovers all possible paths in which the system can be compromised given vulnerability and network topology. We also pioneer the works toward risk mitigation analysis. In particular, the Multi-Objective analysis extends the problem domain so as

to cover the use of external security controls and discover the optimal security hardening plan. Thus, it gives a more comprehensive solution than the traditional risk mitigation analysis.

Our work, however, is still in its early state. There are many works that need to be done. In the virtue of formalism, more practical studies of attack analysis are needed. In particular, the attack generation tool needs to have more rigorous tests. The challenge here is to study the use of the model analysis on different network environments such as mid-size network, large scale network, virtual network, and network on cloud. The results from these studies will be used to improve the model formalism, model generation algorithm, node insertion/deletion, and other maintenance issues.

In the virtue of tool automation, we need to integrate the tool with the vulnerability parser and topology parser. In the current implementation, we rely on the information about attack templates, network topology, and vulnerabilities presented in the network system from external data sources. This information is written in a formatted language such as Html or XML. Hence, the tool can be significantly improved in terms of performance and error reduction by the automatic program that extracts this information and fill the database for reference.

In terms of the risk management analysis, we have conducted several experiments on risk mitigation analysis. These experiments cover both the static and dynamic aspects of the risk mitigation analysis. It is worth mentioning that some security controls have been found to be commonly included in the optimal solutions. Hence, it is possible that security hardening is more critical in certain areas of the attack graph. Such areas could be nodes that have multiple fan outs. In other words, these critical areas are at-risk junctions that can be used by an attacker to cause multiple outcomes. Security controls that can reduce risk in such areas are likely to be parts of the optimal solutions. Therefore, it is worth investigating the "critical path analysis" to understand how such controls can be identified efficiently so as to reduce the search space for the optimization algorithm.

Last but not least, risk assessment and risk mitigation analysis are the two instances of problems where we can use causal dependency model to solve the problem. There are other instances of problems where the causal dependency model can be used as well. Chapters 9 and 10 are two small examples of applying an attack tree to solve problems in intrusion detection and forensic investigation domains.

It is worth to note that there is a key information to determine if a graph-base model is applicable on the specific problem. That is the analyzer must be able to model the problem as an acyclic dependency graph. In our case, we successfully apply *Monotonicity* constraint to prevent a graph cycle. Monotonicity implies that attack progress never backtracks. This constraint is valid when we view causal dependency as "Why a given node can be compromised" since backtracking path will become trivial. However, this constraint may not be applicable to other problem domains.

Chapter 12

GLOSSARY

1. **Vulnerability** – Vulnerability is a weakness in the system allowing an attacker to violate the integrity, confidentiality, access control, availability, consistency or audit mechanism of the system or the data and applications it hosts. Typically, vulnerabilities often result from the carelessness of a programmer, though they may have other causes. Vulnerability allows an attacker to misuse an application to cause it in (for example) bypassing access control checks or executing commands on the system hosting the application. Some vulnerabilities arise from un-sanitized user input, often allowing the direct execution of commands or SQL statements (known as code injection and SQL injection). Others arise from the programmer's failure to check the size of data buffers, which can then be overflowed, causing corruption of the stack or heap areas of memory. The method of disclosing vulnerabilities is a topic for debate in the computer security community. Some advocate the complete disclosure of information about vulnerabilities once they are discovered. Others argue for limiting disclosure to the users placed at greatest risk, and only releasing full details after those notified have fixed the problem by developing and applying patches, but may also increase the risk to those not privy to full details. This type of vulnerability is often called *zero day attack* [58].

2. **Vulnerability Exploitation** – Vulnerability exploitation (or simply 'exploit') is an execution that takes advantage of a bug, glitch or *vulnerability* in order to:

(a) Gain control of a computer system, or

(b) Allow privilege escalation, or

(c) Bypassing the security validation, or

(d) Perform a denial of service attack, or

(e) Reconnoiter for the credential information, or

(f) Change the system configuration in such a way that it prepares for the execution of future exploit.

As an example, the Buffer overflow in Collaboration Data Objects (CDO) used in Microsoft Windows and Microsoft Exchange Server (CVE-2005-1987) allows remote attackers execute arbitrary codes when CDOSYS or CDOEX processes an e-mail message with a large header name. Another example of an exploit that prepare for another attack is the Denial of Service in MyDNS Server (CVE-2006-0351). An attacker might executes this exploit to temporary disable the local DNS Server, redirects DNS requests to an attacker provided address and then fools the user to get sensitive information such as the login name or password.

3. **Attack Pattern** – Exploits can also be classified by the characteristic of vulnerability they attack. The buffer overflow, heap overflow, integer overflow are attacks that exploit the vulnerability of the victim program and cause the corruption of the stack or heap areas of the memory allowing an attacker to execute arbitrary commands on the behalf of the application's privilege. The code injection is a technique to introduce (or "inject") code into a computer program or system by taking advantage of the unenforced and unchecked assumptions the system makes about its inputs. The consequence of this type of attack can allow an attacker to compromise the victim machine. An SQL injection is another code injection type of an attack that occurs in the database layer of an application. The vulnerability is presented when the user

input is either incorrectly filtered for string literal escape characters embedded in SQL statements or the user input is not strongly typed. A cross-site scripting attack (XSS) is an attack allowing malicious Web server send executable code to a client-browser. If the client site has an improper input validation, a script from one page could be allowed to access data from another page or object. As a result, malicious Web site could steal sensitive information or even bypass the access control.

We employ classification taxonomy and schema to classify attack patterns from the Common Attack Pattern Enumeration and Classification (CAPEC) [2]. CAPEC is sponsored by the Department of Homeland Security as part of the Software Assurance strategic initiative of the National Cyber Security Division. The objective of this effort is to provide a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy. CAPEC defines a standard schema for representing attack patterns and to describe in adequate detail the meaning and intent of each constituent schema element. An attack pattern is the mechanism to capture and communicate the attacker's perspective. It is a description of common methods for exploiting software. The CAPEC web site is hosted by the MITRE Corporation and can be assessed at url:http://capec.mitre.org/.

Based on these available information we can form the inter connection between a precondition, vulnerability exploitation, and consequence that uniquely identify an attack. We define an *attack template* in the following section.

4. **Attack Template** – An attack template is an atomic transformation between pre-conditions, exploitation, and post conditions. Both precondition and post conditions are represented by attributes. Note that an information necessary for establishing an attack-template model can be obtained from the Vulnerability Bulletin. To illustrate one of these attack-templates, let consider the Microsoft RPC vulnerability as reported in the Microsoft Security Bulletin (MS03-026). MS03-026 is a buffer

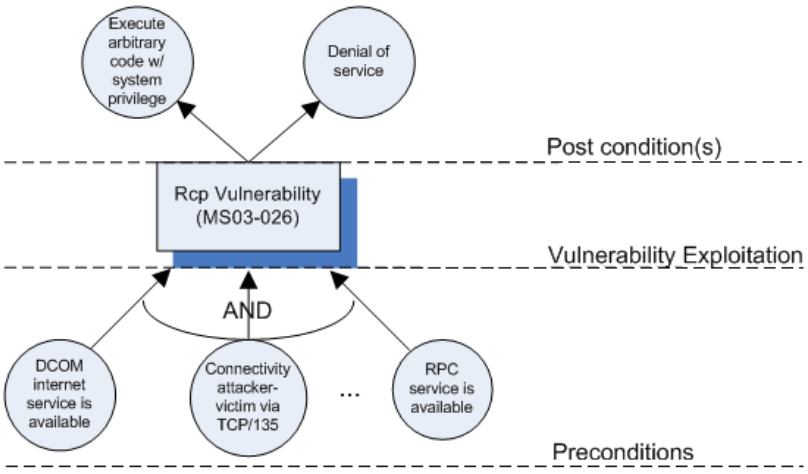


Figure 12.1: An Attack Template for Microsoft's RPC Vulnerability

Overflow attack discovered in the Microsoft's RPC service. The vulnerability exists in the part of the RPC that deals with message exchange over the TCP/IP. This vulnerability affects a Distributed Component Object Model (DCOM) interface with RPC, which listens on the TCP/IP port 135. The vulnerability allows an attacker to execute arbitrary code with the system privileges or cause a denial of service. Judging from this information, we can construct an attack-template model for the RPC buffer Overflow attack (MS03-026) as shown in Figure 12.1.

Bibliography

[1] CERT liveview. Available at http://liveview.sourceforge.net/.

[2] Common Attack Pattern Enumeration and Classification (CAPEC). Available at http://capec.mitre.org, Copyright 2009 by MITRE Corporation.

[3] Dacier, M., Deswart, Y., and Kaaniche, M., Quantitative Assessment of Optimal Security: Models and Tools. LAAS Research Report 96493, May 1996.

[4] EBIOS: An Official document describing EBIOS published by secretariat general de la defense nationale rpublique francaise. Available at http://www.ssi.gouv.fr/en/confidence/ebiospresentation.html.

[5] GNU netcat. Available at http://netcat.sourceforge.net.

[6] Metasploit, Available at http://www.metasploit.com .

[7] Microsoft Baseline Security Analyzer (MBSA). Available at http://www.technet.microsoft.com/enus/security/cc184924.aspx.

[8] Nmap. Available at http://nmap.org/.

[9] OMB Circular A-130: Management of Federal Information Resources, Washington, D.C., Executive Office of the President, Office of Management and Budget, November 2000.

[10] The SANS Institute. Intrusion detection faq. Available at http://www.sans.org/resources/idfaq, April 2004.

[11] W. Venema. Memdump. Available at http://www.tssc.de/products/tools/memdump/.

[12] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the Ninth Conference on Computer and Communications Security*, pages 217–224, Washington, DC, USA, 2002.

[13] A. Arora, D. Hall, C.A. Piato, D. Ramsey, and R. Telang. Measuring the risk-based value of IT security solutions. *IT Professional*, 6(6):35–42, 2004.

[14] B.D. Carrier and E.H. Spafford. An Event-based Digital Forensic Investigation framework. In *In Proceedings of the 2004 Digital Forensic Research Workshop*, 2004.

[15] B.D. Carrier and E.H. Spafford. Automated Digital Evidence Target Definition Using Outlier Analysis and Existing Evidence. In *Proceedings of the 5th Annual Digital Forensic Research Workshop, DFRWS 2005, Astor Crowne Plaza, New Orleans, Louisiana, USA, August 17-19, 2005*, 2005.

[16] B. Berger. Data-Centric Quantitative Computer Security Risk Assessment. *Information Security Reading Room, SANS*, 2003.

[17] S.A. Butler. Security attribute evaluation method: A cost-benefit approach. In *ICSE 2002: Proceedings of the 24rd International Conference on Software Engineering*, pages 232–240, Orlando, FL, USA, 2002.

[18] S.A. Butler and P. Fischbeck. Multi-Attribute Risk Assessment. In *Proceedings of SREIS02 in conjunction of 10th IEEE International Requirements Engineering Conference*, Raleigh, NC, USA, 2002.

[19] C.A. Coello. An updated survey of ga-based multiobjective optimization techniques.

[20] C.A. Phillips. The network inhibition problem. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 776–785, New York, NY, USA, 1993. ACM.

[21] R. Chinchani, S. Upadhyaya, and K. Kwiat. Towards the scalable implementation of a user level anomaly detection system. In *MILCOM 2002. Proceedings*, volume 2, pages 1503–1508 vol.2, Oct. 2002.

[22] D. Denning and P. Neumann. Requirements and model for IDEA - A real-time intrusion detection expert system. Technical report, Technical Report, Computer Science Laboratory, SRI International, 1985.

[23] G.C. Dalton, R.F. Mills, J.M. Colombi, and R.A. Raines. Analyzing Attack Trees using Generalized Stochastic Petri Nets. In *Information Assurance Workshop, 2006 IEEE*, pages 116–123, June 2006.

[24] R. Dantu, K. Loper, and P. Kolan. Risk management using behavior based attack graphs. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, volume 1, 2004.

[25] J. Dawkins, C. Campbell, and J. Hale. Modeling network attacks: Extending the attack tree paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection, Baltimore, MD*. Johns Hopkins University, June 2002.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[28] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Comput. Netw.*, 31(9):805–822, 1999.

[29] R. Deraison et al. The nessus project. *Available at http://www. nessus.org*, 2003.

[30] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 204–213. ACM New York, NY, USA, 2007.

[31] C. Fung, Y.L. Chen, X. Wang, J. Lee, R. Tarquini, M. Anderson, and R. Linger. Survivability analysis of distributed systems using attack tree methodology. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 583–589 Vol. 1, October 2005.

[32] G. Malkin. Traceroute Using an IP Option, RFC 1393, Internet Engineering Task Force, January 1993.

[33] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[34] M. Gupta, J. Rees, A. Chaturvedi, and J. Chi. Matching information security vulnerabilities to organizational security policies: A genetic algorithm approach. *Decision Support Systems*, 41(3):592–603, 2006.

[35] J. Homer and X. Ou. Sat-solving approaches to context-aware enterprise network security management. *IEEE J.Sel. A. Commun.*, 27(3):315–322, 2009.

[36] S.H. Houmb and V. Franqueira. Estimating toe risk level using cvss. In *Proceedings of the Fourth International Conference on Availability, Reliability and Security (ARES 2009 The International Dependability Conference)*. IEEE Computer Society Press. ISSN 1077-2626, March 2009.

[37] S. Jha, O. Sheyner, and J.M. Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63, 2002.

[38] S. Jha, O. Sheyner, and J.M. Wing. Two formal analysis of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 49–63, Cape Breton, Nova Scotia, Canada, 2002.

[39] K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms.* John Wiley and Sons Inc., 2001.

[40] W. Lee. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1):5–22, 2002.

[41] Y. Liu and H. Man. Network vulnerability assessment using Bayesian networks. In *Proceedings of SPIE*, volume 5812, page 61. SPIE, 2005.

[42] T.F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, and S. Listgarten. Knowledge-based intrusion detection. In *AI Systems in Government Conference, 1989.,Proceedings of the Annual*, pages 102–107, Mar 1989.

[43] A.P. Moore, R.J. Ellison, and R.C. Linger. Attack modeling for information survivability. Technical Note CMU/SEI-2001-TN-001, Carnegie Melon University / Software Engineering Institute, March 2001.

[44] R.H. Moses and I. Glover. The CCTA Risk Analysis and Management Methodology-Risk Management Model. In *Proceedings of the Second Computer Security Risk Management Model Builders Workshop. June*, pages 20–22, 1989.

[45] N. Provos. nttlscan, Available at http://linux.die.net/man/1/nttlscan.

[46] P. Ning, Y. Cui, and D. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, D.C.*, 2002.

[47] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 86–95, Las Vegas, NV, USA, 2003.

[48] A. Pal, K. Shanmugasundaram, and N. Memon. Automated reassembly of fragmented images. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, volume 1, pages I–625–8 vol.1, July 2003.

[49] G. Palmer. A road map for digital forensics research. Technical Report DTR-T0010-01, Report from First Digital Forensic Research Workshop (DFRWS), November 2001.

[50] C. Phillips and L.P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 New Security Paradigms Workshop, Chicago, IL*, pages 71–79, January 1998.

[51] R. S.C. Leong. FORZA Digital Forensic Investigation Framework that Incorporate Legal Issues. In *In Proceedings of the 6th Digital Forensic Research Workshop, Digital Investigation, Volume 2*, pages 29–36, 2006.

[52] I. Ray and N. Poolsappasit. Using attack trees to identify malicious attacks from authorized insiders. In *ESORICS 2005*, pages 231–246, Milan, Italy, 2005.

[53] R.W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Computer Society Symposium on Security and Privacy, Oakland, CA*, pages 156–165, 2000.

[54] M. Roesch. Snort–Lightweight Intrusion Detection for Networks.

[55] S. Rubin, S. Jha, and B.P. Miller. Automatic generation and analysis of NIDS attacks. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 28–38, December 2004.

[56] S. Kumar and E.H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *In Proceedings of the 17th National Computer Security Conference*, pages 11–21, 1994.

[57] S. Upadhyaya and K. Kwiat. A Distributed Concurrent Intrusion Detection Scheme Based on Assertions. *the SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems*.

[58] M. Sachs and R. Shein. *Zero-Day Exploit*. Syngress Publishing, 2004.

[59] M. Schiffman. Common Vulnerability Scoring System (CVSS). *URL http://www. first. org/cvss/cvss-guide. html*.

[60] B. Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.

[61] S.E. Smaha and J. Winslow. Misuse detection tools. *Computer Security Journal*, 10(1):39–49, 1994.

[62] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *SP 2002: Proceedings of the IEEE Symposium on Security and Privacy*, pages 273–284, Oakland, CA, USA, 2002.

[63] S.J. Templeton and K. Levitt. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM.

[64] G. Stoneburner, A. Goguen, and A. Feringa. Risk Management Guide for Information Technology Systems. *NIST Special Publication*, pages 800–30, 2002.

[65] L.P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II*, pages 307–321, June 2001.

[66] S. Upadhyaya, R. Chinchani, and K. Kwiat. An Analytical Framework for Reasoning about Intrusions. In *Symposium on Reliable Distributed Systems (SRDS) 2001*, pages 99–108, 2001.

[67] S. Upadhyaya, K. Kwiat, and R. Chinchani. A Comprehensive Reasoning Framework for Information Survivability. In *the 2nd Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop*, page 148155, West Point, NY, USA, June 2001.

[68] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Comput. Commun.*, 29(18):3812–3824, 2006.

[69] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: a basis for building self-protecting servers. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 213–222, New York, NY, USA, 2005. ACM.

[70] D. Zamboni. SAINT: A Security Analysis Integration Tool. In *Proceedings of the Systems Administration, Networking and Security Conference*, 1996.