Rochester Institute of Technology

Development of a Cyber Attack Simulator for Network Modeling and Cyber Security Analysis

A Thesis

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Industrial Engineering

in the

Department of Industrial & Systems Engineering

Kate Gleason College of Engineering

by

Kevin C. Costantini

Rochester Institute of Technology, 2007

October, 2007

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Kevin C. Costantini

has been examined and approved by the

thesis committee as satisfactory for the

thesis requirement for the

Master of Science degree

Approved by:

_____

Dr. Michael E. Kuhl, Thesis Advisor

_____

Dr. Moises Sudit

Acknowledgements

I would like to thank my advisor, Dr. Michael Kuhl, for his guidance and support throughout the development of this thesis. I would also like to thank Dr. Moises Sudit for his insight and expertise.

I would like to thank Jason Kistner for providing me with an understanding of the cyber domain and the initial cyber attack simulation research performed for his thesis.

I would like to thank Katie McConky for helping with the initial development of the java-based simulation model and user interface. I would also like to thank Greg Tauer for helping with some of the modeling and interface features.

This research is funded through the National Center for Multisource Information Fusion.

Abstract

Computer networks are now relied on more than ever before for gathering information and performing essential business functions. In addition, cyber crime is frequently used as a means of exploiting these networks to obtain useful and private information. Although intrusion detection tools are available to assist in detecting malicious activity within a network, these tools often lack the ability to clearly identify cyber attacks. This limitation makes the development of effective tools an imperative task to assist in both detecting and taking action against cyber attacks as they occur. In developing such tools, reliable test data must be provided that accurately represents the activities of networks and attackers without the large overhead of setting up physical networks and cyber attacks. The intent of this thesis is to use operation research and simulation techniques to provide both data and data-generation tools representative of real-world computer networks, cyber attacks, and security intrusion detection systems. A simulation model is developed to represent the structure of networks, the unique details of network devices, and the aspects of intrusion detection systems used within networks. The simulation is also capable of generating representative cyber attacks that accurately portray the capabilities of attackers and the intrusion detection alerts associated with the attacks. To ensure that the data provided is reliable, the simulation model is verified by evaluating the structure of the networks, cyber attacks, and sensor alerts, and validated by evaluating the accuracy of the data generated with respect to what occurs in a real network. By providing accurate data with respect to network structure, attack structure, and intrusion detection alerts, the simulation methods used offer considerable support in developing tools that can accurately detect and take action against attacks.

Table of Contents

1 Introduction

Computer networks have developed significantly over recent years. From college networks to business networks, nearly all aspects of their functionality have improved with regard to both speed and capacity. Because of these improvements, increases in the size and complexity of such networks have been common. This growing size and complexity is making the tasks of accurately monitoring network activity and maintaining all of the desired security standards more difficult for network administrators.

The reliability that people and organizations expect from networks has also increased dramatically over recent years. There is an increasing dependence on networks to store needed information and provide for many of the operations that individuals and organizations rely on. For some organizations, the information and operations provided through a network are essential to the functioning of the organization. From banking firms to communication companies to healthcare, there is a strong dependency on computer networks to provide the information and functionality that these industries require (DeLooze, Graig, McKean, and Mostow, 2004).

Due to the large size of, complexity of, and dependency on computer networks, the security concerns are escalating. Cyber attacks on such computer networks are becoming more common, and the threats of these attacks increases as organizations continue to store and access critical information through computer networks. Internet-based attacks especially are a major concern, and they are believed to account for about 70% of all malicious attacks on organizations. Furthermore, the occurrence of internal and remote attacks is becoming less common due to both the increased internet activity

that organizations have and the ease of attacking through internet connections. Also, as traffic increases within the network itself, the specific actions of an attack become further buried within this traffic (Fuchsberger, 2005). This can add to the difficulty faced by administrators and sensing devices in effectively locating these attacks.

Cyber attacks are becoming easier for less-trained hackers due to the abundance of useful applications and the availability of organization and network-related information that can be retrieved even from simple Google searches. As organizations leave sensitive information available to users outside of their network, search engines such as Google are able to find this information (through the use of bots) and display the information to any interested hacker that can come up with the proper search criteria (Kurtz, McClure, and Scambray, 2005). Also, tools and applications used for intrusions are becoming more powerful, requiring less knowledge from the attackers using them. For example, password guessing operations that had previously been manual are now being automated with a variety of tools. Such tools are becoming more available to the general hacking community, so attackers no longer need to be experts on the subject matter (Fuchsberger, 2005).

These vulnerable computer networks have not been left defenseless, though. Intrusion Detection Systems (IDSs) have taken a role in helping network administrators find and follow network attacks. Intrusion detection offers a means by which network traffic can be monitored by a device, or sensor, in addition to being monitored by an administrator. This is done by the sensor determining if a particular packet of network traffic could perform or allow for a potentially malicious action. If so, an alert is generated. An IDS can provide warnings to indicate when malicious or suspicious

activity is detected, and these warnings can help the system and the administrators defend against an attack. Furthermore, an IDS can verify that changes to the security configurations of a network do in fact provide additional security (Allen, Christie, and McHugh, 2000). Several different types of intrusion detection systems are available today, ranging from free open-source software to costly, specialized, and highly-integrated proprietary systems.

Although sensor alerts do indicate when something suspicious is happening, there still exists the need for individuals to look through these alerts (or some of these alerts) and try to determine what is really happening in the network. This process is where the idea of information fusion has recently been introduced into the cyber domain. Information fusion is a technique where data from many different sensors is combined with related databases of information to provide inferences that could not be determined from looking only at individual sensors (Hall and Llinas, 1998). Multi-sensor data fusion in the cyber domain is a fairly new concept that involves combining data from multiple IDS sensors to gain an understanding of network activities and events. In the cyber domain, determining the intent and threat of an attacker by looking at actions individually is a difficult task. Using information fusion as a tool allows for many actions to be analyzed together to gain a high-level understanding of the current network situation. The high IDS false-alarm rates (and the problems associated with them) as well as the lacking ability of most IDSs to gain a full understanding of a network's current situation stress the need for improvements in the way these sensor alerts are handled. By utilizing information fusion techniques, there is potential for a new generation of robust intrusion

detection systems to be developed that will provide situational information in addition to the information regarding individual actions (Bass, 2000).

Even with the development of such intrusion detection systems using information fusion techniques, there is still the difficulty of testing and verifying that the systems provide the desired accuracy and functionality. Obtaining significant amounts of IDS sensor data is difficult, and setting up physical networks to perform such tests requires significant time and money. Also, the variability of networks as well as the value of information stored on such networks can make the process of accounting for the different network structures nearly impossible to handle using a physical setup. The development of intrusion detection systems utilizing information fusion needs to be aided with a significant amount of reliable data with which to verify and validate the systems' performance.

2 Problem and Scope

Providing security in a private network is one of the leading challenges that organizations are facing with respect to the management of information. Companies often keep proprietary and personal information stored on a private network, and these companies do this with the intent of the information only being accessed by reliable individuals. Frequently, though, hackers (with either individual motives or affiliation with competing organizations) will find such information of great value and manage to pry their way into the private network to either obtain or corrupt the desired information. Most private networks have some form of internet connection, which allows for web-based attacks to be a threat. Hackers will track down such networks and determine the vulnerabilities and the exploits that will assist them in gaining access into such networks. Although security measures within private networks are improving, finding all of the vulnerabilities and keeping the various security measures up to date is a challenging task for company network administrators, and those planning to attack a private network are counting on just this. Furthermore, new exploits will always be found before security measures are made available to deal with them (Kurtz, McClure, and Scambray, 2005). Due to the difficulty of maintaining security in the world of private networks, the importance that private networks and the information stored on them have to companies and organizations around the world, and the frequency at which hackers will attack such networks to obtain the desirable information, the scope of this research will deal with security in private networks.

The development of network security tools that make use of information fusion techniques are a significant step forward in reliably securing a private network.

However, as noted previously, there is still much to be done in this realm due to the difficulty of developing robust information fusion systems and verifying that they accurately recognize attacks. In fusing together information from multiple IDS sensors, a critical factor is to have an understanding of whether the fusion engine being developed sufficiently detects the attacks taking place. Attack and sensor data that is provided during the development stages of information fusion engines can significantly help the developers confirm their functionality. To verify that attacks are sufficiently detected, a network could be setup where known attacks are performed and sensors are collecting related alerts from the actions involved. However, this option creates a substantial expense for the developers, and such a setup also does not account for the great variability in the intent and styles of attacks that could possibly occur. For these reasons, the use of simulation modeling is suggested to accurately portray both the flow of an attack through a network and the IDS alerts associated with that attack (Holender, Stotz, and Sudit, 2006). Simulating the attacks and the generation of IDS alerts allows for significant amounts of attack and sensor related data to be generated quickly with little or no cost. Furthermore, this simulation will allow for networks of varying sizes and structures to be easily modeled and used in developing the attack and sensor data. By providing simulated cyber attack data, information fusion system developers will have the potential to test their fusion engines across a wide range of scenarios.

A simulation model and methodology has been developed by Kuhl and Kistner (2005) to provide attack and sensor data for use by fusion engines. This simulation model allows for varying network sizes and configurations, but there are considerable limitations in accurately portraying private networks and attacks on those networks. The

goal of this thesis is to provide a simulation model and methodology that addresses the limitations and issues with the existing model and improves the accuracy and performance of the model in generating valid attack and sensor data. This thesis will expand upon the simulation model developed by Kuhl and Kistner (2005).

There are many important criteria in developing a valid attack simulation model with high accuracy and strong performance. These, broadly, include allowing attack scenarios to be run and modified easily, providing significant control of the modeled network, producing valid and detailed attack and IDS related data, and allowing for multiple applications in modeling intrusion detection systems. With respect to the ease of modifying and running scenarios, the simulation model should provide the user with intuitive controls, give options with regard to how output data should be generated and dealt with, and graphically display the aspects of the network and the attacks. Accounting for all of these considerations, there will be less possibility of the resulting model being improperly used or generating data that the user did not intend to generate. In providing significant control of the modeled network, the simulation should allow for a wide spectrum of details to be at the discretion of the user. This includes both the network topology and the individual device attributes. For instance, the simulation should provide the ability to specify the vulnerabilities of a network device that will reflect what attack actions are available against the device. Also, the simulation should allow for IDS sensors and IPS tools to be easily added to the network devices. Having these options will allow the model to better reflect the slight differences that exist between actual network devices. With respect to producing valid and detailed network data, the attacks created and the alerts generated should be accurately portrayed. The IDS

sensor data should be representative of the output that actual sensors provide. The attacks that the simulator creates should follow a logical attack step progression based on existing modeled attack templates. To allow for the simulation model to be valuable to multiple applications in modeling intrusion detection systems, the model should have parameters setup to handle a variety of user defined information and methods established by which new (or different) IDS sensors can be added and used in the model.

3 Background

The methodologies presented in this thesis require an understanding of basic networking principles. This chapter reviews some of the basic networking principles referenced in the network and attack simulation methodology to give the reader a clearer understanding of what is being accomplished. First, an overview of network architecture is provided in section 3.1. Section 3.2 gives a more detailed description of the network devices. Section 3.3 describes network packets and presents the functionality of the packets. Section 3.4 gives an overview of the exploits that hackers used to perform malicious activity on a network. Lastly, section 3.5 discusses the development of intrusion detection systems (IDSs) and the role these systems play in providing a level of security in a network.

3.1 Computer Networks

A network generically represents any set of devices that are connected together through some means in order to communicate and share information. A computer network focuses on computers being interconnected for this purpose. Networks can range from a simple single connection between two computers to a vast network of cables and routers that connect thousands or even millions of computers (such as the Internet). This networking review, though, will focus specifically on local area networks (LANs) where all of the network devices are located in roughly the same geographic location. In a typical LAN, the devices are managed by the same group or company and are connected at high speeds. This section provides a brief overview of the devices used in a computer

network, the connections made between these devices, and the typical structure or topology of LANs (Barrett and King, 2005).

The primary devices in a network are computers. With respect to a company network, these computers are used to manage data and perform essential and/or beneficial business functions. For these purposes, two broad categories of computers are used: servers and clients. A server is a computer specifically designed to store and share resources among other computers in the network or even in a different network through connection to the Internet. A client (or host) is a computer that can function individually or use resources provided by servers. Each computer in a network is provided with a unique MAC (machine access control) address that is used to reference the computer (Barrett and King, 2005).

Although direct connections can be made between computers, some additional network hardware is typically provided to allow multiple (and possibly more complex) connections to a computer. This hardware is in the form of a routing device, which can be connected to several machines as well as several other routing devices. Figure 3.1 displays how these routing devices are used to connect computers in a network. In this figure, both Computer A and Computer B are able to communicate with the fileserver. Depending on the routing device used, a certain level of functionality will be provided that can include logic used to manage the communication capabilities and limitations between computers in the network (Barrett and King, 2005).

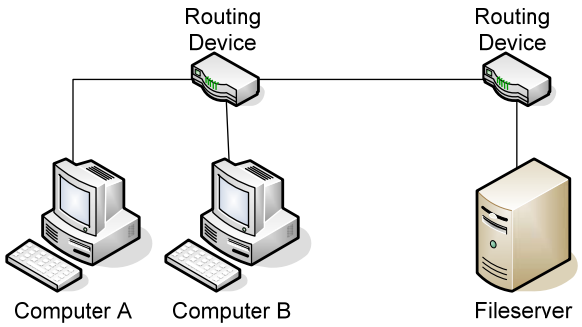

Figure 3.1: Communication between Multiple Computers

Different types of routing devices include hubs, switches, bridges, and routers. Hubs provide the least functionality and simply retransmit all received information through every other connection made with the hub. A bridge is used to send information between different LANs, and a bridge includes some functionality for identifying the destination of the information. Switches send information to the specific MAC address (computer) in the network that the information is intended for. A switch also includes some functionality to filter out certain types of communications. Routers are used to send information to a specific computer in the network or to the Internet. A router includes tables that store which routes can be used in allowing one computer to communicate with another computer. A router is also integrated with the functionality for filtering out certain types of communication (Barrett and King, 2005).

Keeping an organized structure for the device connections made within a computer network is an important part of setting up and managing a network. Computers with both a similar functionality and physical location are typically connected with the same routing device. The illustration created by Barrett and King (2005) and displayed in Figure 3.2 shows a common method used to connect computers in this way (known as a star topology). In this type of topology, each computer has one segment connected to the

routing device. With this method, an interruption in one segment will not affect the other segments. The routing device can then be connected to other routing devices within the network to provide communication between these computers and another group of computers (Barrett and King, 2005).

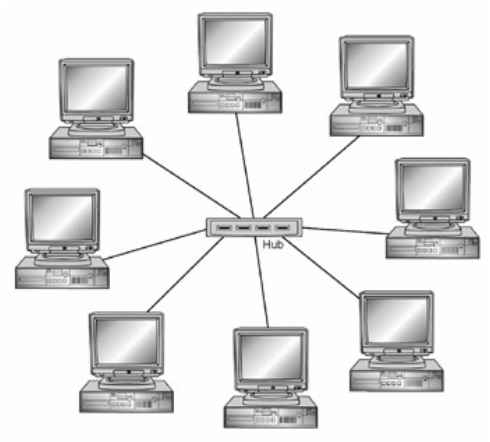

Figure 3.2: A Star Topology for Connecting Computers (Barrett and King, 2005)

Another common approach used in setting up a computer network is to separate the externally accessed servers from the computers and servers intended for internal company functions. For example, an organization may have a web server that provides the company's website and an FTP server that allows files to be transferred to different organization locations or even different organizations. These two servers can be kept on a separate portion of the network in order to provide a more secure setting for the remaining internal computers and servers. Figure 3.3 displays this type of configuration. The two external servers are connected to a router that can directly communicate with the Internet. The remainder of the network is connected to internal routers that provide some security through the means of firewalls, which will be explained in section 3.2.

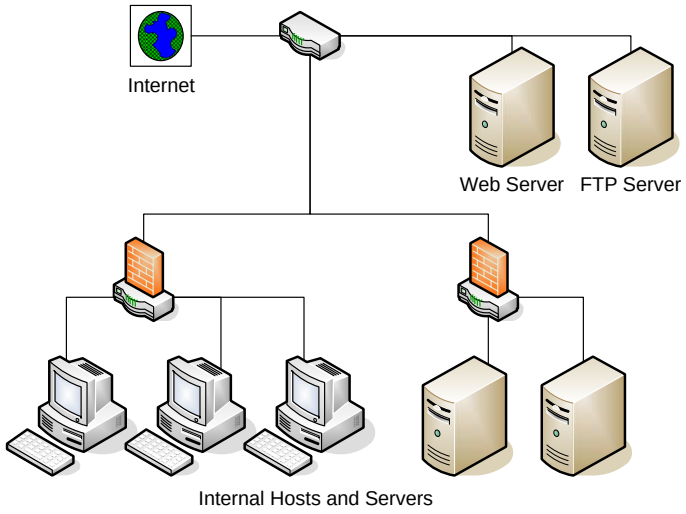

Figure 3.3: Example Network Configuration

3.2 Device Details

Both computers and routing devices have certain characteristics that effectively guide what type of communication can occur. For computers, such characteristics include the type of computer, the operating system used on the computer, and the services running on the computer. For routing devices, the primary characteristic is the set of firewall permissions. This section provides more details regarding these four device characteristics.

The type of computer refers to whether the computer represents a host (client) or a server. Hosts and servers have differences in the way that they communicate, and these differences need to be recognized and considered when developing communication rules. The operating system used by the computer indicates the underlying software used to run applications on the computer. Different operating systems, such as Windows and Linux, typically use a special set of protocols (or communication mechanisms) to transfer information. Services represent the specialized software running on a machine to provide

a particular functionality that is not possible with the operating system alone. These services help the operating system communicate with applications running on the computer. For example, an FTP service allows specific files to be uploaded to and downloaded from a file server. Many services have specific protocols that must be used in order for communication to occur with the service (Barrett and King, 2005).

The firewall permissions stored on some routing devices indicate what type of communication is allowed along a certain communication path. Typically, these permissions are set up to either allow only certain types of communication or block only certain types of communication. For each specific path through a router, the firewall permissions will contain a listing of what protocols and computer ports are allowed or banned. A computer port represents a specific mechanism used to establish a connection with another computer (or data source). Each computer has thousands of ports that can be used to establish a communication path. This wide range of ports allows for multiple communication paths to be established at the same time on a computer. Firewall permissions are typically setup such that only a small subset of ports is available to be used. The specific port used in sending a particular piece of information will commonly depend on the service associated with the information (unless the information is not associated with a service) (Barrett and King, 2005).

3.3 Network Packets

The components, or devices, within a computer network communicate and send data through the use of “packets”. Packets contain a pre-defined amount of data with additional headers that indicate how the data will be handled. Simple communication

between two machines may only require a single packet while transferring a large file between two machines may take hundreds of packets. When more than one packet is used, the data within the packets must be pieced back together to form the original data stream (file). The packets flowing through a computer network are also referred to as the network traffic.

The functionality for handling packets in a computer network is provided by a set of protocols known as TCP/IP. This set has four layers of protocols that are all an essential part of allowing computers to communicate with each other. A breakdown of the protocols is shown in Figure 3.4. The lowest (physical) layer is the Ethernet, which provides the physical connection between computers and the electrical signals that represent packets. The remaining three layers each have a specific header, or set of instructions, in the network packets (Crothers, 2003).

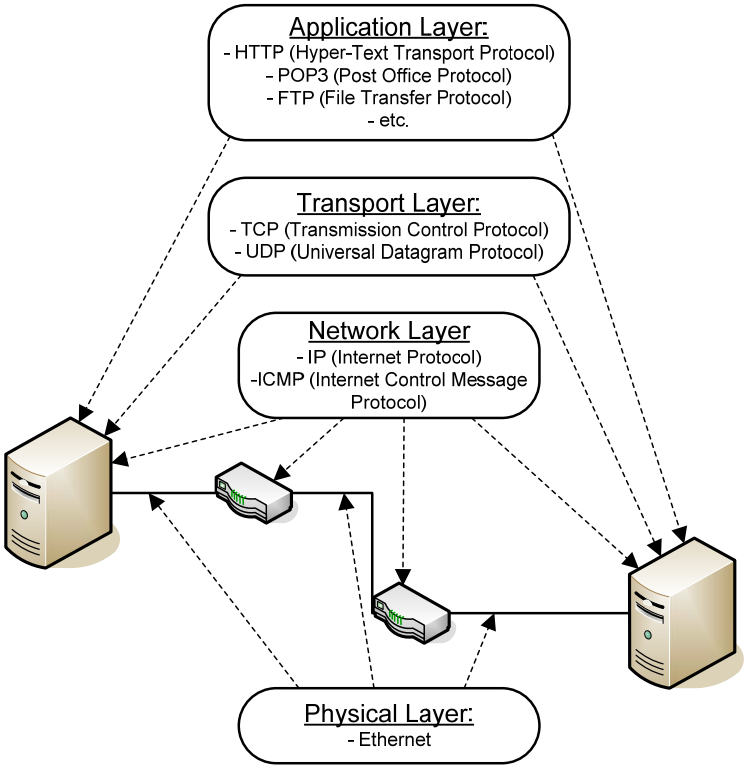

Figure 3.4: TCP/IP Protocol Suite

The Network layer, using IP and ICMP, provides the routing of the packets, which is essentially handling how the packets get from the source computer to the destination computer through the available Ethernet connections in the network. This layer makes use of the IP addresses of computers in order to route the packets. The IP header includes the source and destination computer for a particular packet, and each network linking device (router, hub, etc.) uses this information to send the packet through the appropriate link in the network (Crothers, 2003).

The transport layer, using the TCP and UDP, provides the packet handling functionality. This functionality includes breaking a file or stream of data into packets at the source computer and restructuring or re-ordering the packet data to form a full data stream at the destination computer. Both TCP and UDP have a wide range of ports that packets can be sent or received by. Usually, though, a specific set of source and destination ports (TCP or UDP) is used for a packet relating to the machine service responsible for the packet. The transport layer header included in the packet indicates which type of protocol (TCP or UDP) and which port number to use (Crothers, 2003).

The application layer is used to interpret and allocate the data to a specific machine service and service protocol, such as a web server (HTTP), a mail server (POP3), a file server (FTP), or some other service. The service can then make use of the information provided by the packet data. The application layer header indicates which type of service-specific protocol to use (Crothers, 2003).

At the source computer, the packets are assembled starting with a particular service. The service pulls together all of the data that needs to be sent, known as the packet payload, and adds the application layer header. The transport layer then breaks

this lumped data into several packets of equal (or near equal) size and includes an additional header pertaining to the protocol and port combination to use along with the order of the packets. The network layer adds another header to each packet indicating the source and destination IP addresses that the packet needs to be routed between. Lastly, the physical layer provides a dynamic header that indicates the source and destination MAC addresses for each packet at each link along the path taken, meaning that the header will change each time the packet moves through a different link (Crothers, 2003).

3.4 Exploits

Exploits represent the actions that hackers perform to attack a network. While an attack is considered an attempt to bypass some sort of computer security measures, an exploit is considered a step within an attack used to take advantage of a specific flaw or vulnerability in the network. An attack can consist of many different exploits performed in sequence. A successful attack, that violates the security policy of a system, is considered an intrusion.

Weaknesses in the network that are subject to being exploited are considered vulnerabilities. Vulnerabilities commonly exploited by hackers can be broken down into three categories: development and design problems, management problems, and trust abuse. Development and design problems refer to software coding errors or architectural application issues that allow for an application (or the system running the application) to be used in a means that was not intended. This type of vulnerability is the most commonly exploited vulnerability in network attacks. Management problems refer to improper network configurations or policies that allow hackers to gain access to points in the network believed to be protected. This type of vulnerability is common with

inexperienced network administrators. Lastly, trust abuse refers to the misuse of privileges given to an individual. This type of vulnerability requires that the hacker already has some level of allowed access to the network. Internal-based attacks (as opposed to internet-based attacks) will commonly exploit the trusted user abilities (Crothers, 2003).

Both development/design problems and management problems are typically exploited by breaking (or at least bending) the rules of the TCP/IP protocols. Such exploits can be in the payload (data) of the packet or in one of the packet headers. Exploits associated with software (or service) design vulnerabilities tend to either include malicious code within the packet data or take advantage of the application header and the application protocols. Exploits associated with the network configuration vulnerabilities will typically take advantage of the transport layer protocols or the IP layer protocols. For example, an internet-based hacker can spoof a packet's source IP using an IP address of the internal network. This activity effectively takes advantage of the IP layer to make the packet appear to be coming from a trusted internal source, and the destination computer will most likely accept the packet data (Crothers, 2003).

3.5 Development of IDS

An intrusion detection system (known as an IDS) offers a way for network administrators to monitor what type of actions are occurring within a network and attempt to take action against attempted intrusions (attacks). This system works by correlating a set of actions with known vulnerability exploits.

The modern techniques used to identify and correlate cyber attacks are significant improvements over the original techniques. Initially, the only real method to locate

malicious activity involved a dedicated administrator that would sit in front of a console and monitor the network activities looking for suspicious actions. The increases in network size and activity drove this method of detection to become far too difficult for detecting attacks in real-time. Thus, the use of audit logs became prevalent to keep track of and store the network activity without the need for individuals to continuously monitor the activity. This method, though, still resulted in a vast amount of information for administrators to manually sort through. Often, this information would only be used as a forensic tool when an actual incident had occurred and the administrators wished to pinpoint what had caused the incident. Therefore, the method of direct observation and response has become less effective over time (Kemmer and Vigna, 2002).

As a next step in identifying cyber attacks, intrusion detection systems (IDSs) have been developed to automatically detect suspicious-looking network traffic. These systems offered a relief from the increasing amount of network traffic that needed to be looked through. Intrusion detection systems (specifically, host-based IDSs) started out by reviewing audit data as the data was produced in order to provide network administrators with only information that could potentially be related to an attack, thus filtering out information regarding the typical, harmless network traffic (Kemmer and Vigna, 2002). As the IT security industry developed over the years, the use of intrusion detection systems significantly increased, and the processing rate of these systems improved as well. Network-based IDSs were introduced to monitor the actual network traffic instead of the log files produced by a specific host. These systems identify attack actions by matching attack patterns in the TCP and IP packet stream, and they have a distinct advantage over host-based IDSs in that they are not associated with a device that

could potentially be a target itself. As processing power has improved, these systems have become capable of monitoring the network traffic in "real-time" and triggering alerts as suspicious activity is identified. Market demands further drive this industry to develop full-fledged software corresponding to the IDSs, with a wide array of capabilities in monitoring and organizing the alerts generated by the IDS sensors (Fuchsberger, 2005).

Intrusion detection systems, even with improved accuracy and processing, still only account for part of the methodology needed to actually defend a network against the attacks. These systems can determine the cause of an attack (and possibly the intent), but the IDSs alone do not take the measures to stop such an attack. To perform this function, intrusion prevention systems (IPSs) have been developed in an attempt to react to the detected malicious activity. Since these IPSs are actually capable of thwarting some attacks, the demand for such systems is becoming stronger than the demand for simple IDSs (Fuchsberger, 2005). However, the majority of current IPSs available do little in terms of actually identifying an overall attack. Instead, a methodology similar to the IDSs' methodology is used where the network traffic is inspected to look at individual packets and "reflex"-type actions are performed when suspicious activity is identified. Therefore, many common intrusion prevention systems are really only capable of reacting to individual attack steps rather than entire attacks.

4 Literature Review

There are three primary areas of research related to the problem being addressed, which include the modeling and classification of cyber attacks, the simulating of computer networks, and the use of information fusion techniques in the cyber domain. Although there is some overlap between these topics, the combination of all three has only recently gained attention. Therefore, there is a great deal of opportunity in identifying some of the leading issues across these fields and providing a methodology that effectively addresses these issues. This section reviews some key research that has already been performed in these fields. Apart from these three areas of research, the development of object-oriented simulation models is discussed to provide some preliminary concepts that helps establish an effective object-oriented simulation model for the modeling required in this thesis. This discussion includes several different modeling approaches explored by researchers. Also, this section concludes by discussing both the details and limitations of an existing cyber attack and intrusion detection simulator developed by Kuhl and Kistner (2005). This existing simulator is important because the research performed in developing the simulator has served as a basis for the initial work accomplished by this thesis.

4.1 Modeling and Classification of Cyber Attacks

An effective means to understanding cyber attacks is through classifying the attacks and modeling the attack progression. This involves carefully observing the actions associated with a known attack to gain an understanding of the attacker behavior. Fortunately, there has been significant progress made in modeling attacks.

Dougherty and Gonslaves (2006), for instance, developed an adaptive cyber-attack modeling system to assist in testing software protection. Previously, a team of subject matter experts (SMEs) were required to mimic the actions typically associated with hacker attacks in order to evaluate and validate their software protection methodology. This method, though, adds significantly to the cost and time required for such a project, and the method is not really appropriate for the large numbers of scenarios which need to be thoroughly tested. The research performed by Dougherty and Gonslaves found that developing accurate models of cyber attacks had the potential to substantially reduce the cost and time required. Through this modeling, three primary categories of attacks were identified: web-based application attacks (considered the most vulnerable due to the substantial number of tools at the hacker's discretion), client-server application attacks, and stand-alone system attacks (which are the hardest to implement). Also, the models implemented a Bayesian belief network approach to mimic the reasoning made by hackers. Although the modeling done through this research only extended to that which was needed for testing the software protection, the methodology is a good representation of what is becoming necessary for any type of security tools.

Other researchers have looked into modeling attacks by analyzing isolated IDS alerts associated with attack steps. For example, Cheung, Fong, and Lindqvist (2003) developed a project called "Correlated Attack Modeling" (CAM) where attack scenarios are modeled by observing actual IDS alerts. These IDS alerts can generally be associated with a specific attack step (usually an exploit of some sort). Although this allows for actual attacks to be used in developing attack patterns, many IDS alerts could be false positives and thus affect the modeling process. Furthermore, some attack steps may be

missed altogether or could be temporally distributed enough that piecing together an attack (let alone add to an attack pattern) becomes a difficult task.

Also, higher-level research has been done to identify the typical sequence of the different exploits that a hacker can perform. Holdender, Stotz, and Sudit (2005) developed a graph-based template that makes use of graph theory techniques to designate what types of attack actions or exploits are necessary before other certain types can be performed. This development began by first grouping known exploits into categories or "stages" based on what type of activity was necessary before these exploits occurred and what type of activity could be performed afterward. An adjacency matrix was then developed to correlate the different stages, indicating what stages can occur after an exploit from a given stage has occurred for all of the stages in the template. A simplified directed graph of this attack exploit template is displayed in Figure 4.1. The nodes containing S0 through S9 represent the ten stages of attack exploits, while the edges (arrows) represent the precedence of stages. For instance, an intrusion-type exploit from stage 1 (where the attacker gains user access on a device) will only occur after a reconnaissance-type exploit from stage 0 has been performed (where the attacker obtains useful information about the device). By categorizing possible exploits of known vulnerabilities into one of the stages, the process of modeling an attack becomes simplified in that only the stages of the attack truly need to be modeled as opposed to the vast amount of exploits that would need to be considered otherwise. Due to these advantages, this graph-based template was used in the simulation model developed by Kistner (2006) to allow for attacks to be automatically simulated based on a set of parameters.

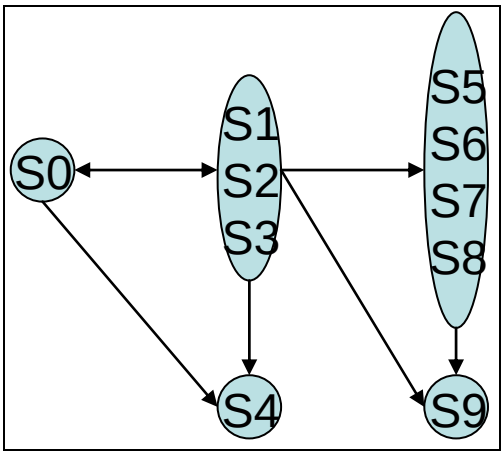

Figure 4.1. Directed Graph of Stage Precedence (Holender, Stotz, and Sudit 2005)

4.2 Simulation of Computer Networks

When simulating the progression of attacks through a network and in generating typical IDS alerts within a network, providing a simulated network structure to work with is a necessity. Even though these processes could be established on a physical network, the large overhead costs and long testing times make such an arrangement highly undesirable. Furthermore, covering the large variety of network setups and sizes with physical means is nearly impossible. Simulation modeling allows for numerous setups and significant structural changes to be made quickly. Although other simulation models have been developed to portray the cyber attack process, the generation of IDS alerts, or both of these functions, these models lack the functionality and modularity necessary to generate the proper input needed for development of information fusion tools for the cyber-attack realm.

Garg, Kwiat, and Upadhyaya (2006) developed a framework (known as SimCo) for measuring the capabilities of security mechanisms in detecting attacks. Inaccuracies among intrusion detection systems and other security systems can have a significant impact on organizations; thus, identifying where flaws lie within such systems is crucial.

The framework developed included a platform for simulating complex attacks along with templates for both security detection mechanisms and for attacks. However, even though IDSs were included, this project was focused more on evaluating differences among the security systems rather than providing IDS alert data. The IDSs and other security systems were modeled in much more complexity than is really necessary for a simulation specifically requiring alert data. In most cases, the entire functionality of the IDS/security system was modeled. This functionality focused mostly on the internal workings of the sensors, which included modeling the means by which attack steps were identified and correlated to alerts. For the simulation methodology needed in this thesis for strictly generating IDS alerts, the means by which the alerts are generated is irrelevant as long as the alerts are representative of the corresponding IDS alert format.

DeLooze et al. (2004) have also developed a simulation methodology to model the combination of cyber attacks and security systems. They developed a simulation model called "The Virtual Network Simulation" to assist in education and training courses for individuals pursuing careers in network security. The simulation model developed includes a vast amount of network devices (including IDSs) that can be setup at the user's discretion, and the model interjects simulated attacks into these networks. However, since this system is setup as a training tool, the model requires consistent interaction with an individual to monitor, make adjustments to, and handle problems occurring within the simulated network. Therefore, this tool is also not well designed for data generation associated with the attacks and IDS alerts.

Significant work has also been accomplished in developing a simulation model for both generating attacks and producing associated IDS alerts. Kuhl and Kistner (2005),

through the use of the commercial simulation package ARENA, developed a simulation structure that allows for computer networks to be modeled and cyber attacks to occur within the networks and produce alerts for corresponding IDS sensors included in the modeled computer network. Kistner (2006) further expanded upon this work to develop more detailed attributes for the network devices and provide a methodology for automatically generating attacks based on a set of parameters. Section 4.5 provides a more detailed discussion of the research performed by Kuhl and Kistner (2005) and Kistner (2006).

4.3 Information Fusion Techniques Applied to the Cyber Domain

Information fusion techniques, as recently applied to cyber security issues, have the potential to identify useful trends and a significant amount of information about hacker attacks based on data provided by IDS sensors. In general, multi-sensor data fusion refers to the techniques used to combine data from multiple sensors along with related information from databases and templates to determine certain information about the system of interest. The use of multiple sensors allows for more accurate information that accounts for a wider spectrum of the system, in addition to the simple statistical advantage of having more data points to base a decision on. This information fusion technique has previously been used in surveillance mechanisms, guidance and control of vehicles, monitoring machinery, medical diagnosis, and other military and non-military purposes (Hall and Llinas, 1998).

The generic process of fusing data has five specific levels of functionality defined by the Joint Directors of Laboratories (JDL). Figure 4.2 depicts these levels with respect to a variety of multi-sensor data fusion applications. Level 0 refers to the pre-processing

of data, where data most pertinent to the current situation is filtered from a large set of raw data. Level 1 represents Object Refinement, where data is transformed for consistency and several attributes of the objects are identified and classified. Level 2 represents Situation Refinement, where relationships are developed between objects and events through incorporation of environmental information and previous (*a priori*) knowledge. Level 3 represents Threat Refinement, where the current situation is projected into the future to determine potential threats and vulnerabilities to the system, as well as opportunities available. Level 4 represents Process Refinement, where the long-term data fusion performance is monitored, improvement opportunities are identified, and adjustments are made to the system to achieve some desired effects (Hall and Llinas, 1998).

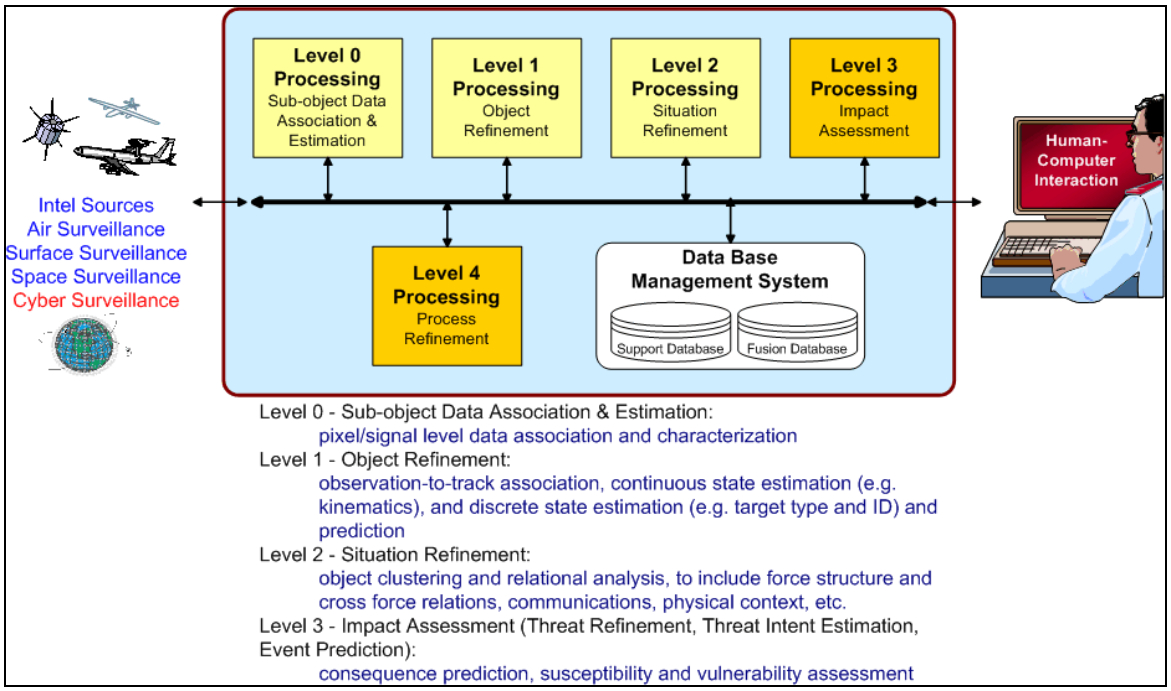

Figure 4.2: JDL Fusion Levels (Hall and Llinas, 1998)

Holender, Stotz, and Sudit (2005) have developed an "Information Fusion Engine for Real-time Decision Making" (called INFERD) with the capabilities necessary to be

used with IDS alert data in the cyber-domain. Essentially, the alerts produced by IDS sensors are treated as data points that INFERD can accept as input in order to piece together the information to identify attacks or malicious activity occurring in the network. Therefore, INFERD can provide some awareness with regard to the current state of the network (in real-time) that alert data is received from. With respect to the JDL fusion levels, INFERD accounts for a combination of Level 1 and Level 2 fusion. In addition to the data input into the engine, INFERD also requires attack templates along with some information regarding the network topology (which refers primarily to the structure of the network being assessed). Although the intent of this engine is to correlate actual alerts to determine attacks, the process is still in development and needs extensive testing.

Since actual IDS alert data is challenging to obtain, as discussed previously, and since there is no faultless method to evaluate whether the engine is perfectly identifying the situation, the use of simulation to provide representative IDS alert data is very beneficial in this spectrum of the fusion process. Also, by assisting the development of INFERD, the use of simulation can also provide benefits for the higher levels within this fusion process. For example, TANDI, a fusion engine used within Level 3 of the fusion process, provides predictions of the attacker's next moves (essentially, the threat imposed by the hacker) (Holender et al., 2006). With a simulation model, we will know exactly what the attacker's next moves are, and this knowledge can be compared with the output that TANDI provides to give a measure of TANDI's effectiveness. The simulation could even go beyond this to allow an engine or methodology focused on process refinement (Level 4 fusion) to update the simulation model in real-time and evaluate how well the entire fusion system is functioning within a network.

4.4 Object-Oriented Simulation Modeling

The use of an object-oriented approach to simulation modeling has many advantages over a procedural approach. The ability to re-use and extend objects is a central feature and advantage to developing an object-oriented framework. Also, since objects are modular, the information associated with each instantiation of an object is held in only one location, with references to the object's location when needed (Bischack and Roberts, 1991). Furthermore, with an object-oriented simulation focusing on representing objects, the computations and modeling logic can be divided among objects (classes). This delegation of modeling functions provides a more organized structure than would otherwise be possible with procedure-based modeling (Joines and Roberts, 1998).

The first step in developing an object-oriented simulation model is to devise the appropriate software architecture concept. This includes a broad definition of how the model and the simulation aspects and controls will interact at a high-level. Sarjougian and Singh (2003) recommend the use of three separate frames within the simulation architecture to provide the needed interaction. These include a model, control, and view frame. The model frame in this situation composes the application functionality and handles the states of objects being represented. The control links user actions to model changes and selects specific "views" or states of the system to present to the user. The view represents the user side of the architecture, where the model is rendered and user gestures are provided. In developing a discrete-event simulation architecture, Brunner and Schriber (2005) indicate the importance of providing a "transaction-flow world view," where a discrete amount of traffic or transactions are moving from point-to-point (or object to object) within a system in order to change the state of the system. Numerous

approaches can be taken to provide this type of interaction, but the key is to establish a framework that will clearly define this transaction-flow.

The structure for the classes in the desired simulation framework is another major consideration. Darmont (2000) proposes the use of a centralized simulation class that ties together all aspects of the system. The class includes a link to a scheduler class that handles the various discrete events that occur during the simulation. An event manager class is used to provide implementation of the different events within the simulation model. Also, the resources involved are handled through a resource class linked with the simulation class. This approach allows for the simulation class specifically to handle the transactions involved in the simulation; although, there is some redundancy with having both a scheduler and event manager class. Joines and Roberts (1998) provide a detailed structure by which to setup a simulation class hierarchy. The top of this structure includes an abstract class with properties and methods available to all simulation components. Children (sub-classes) of this class include a random class that provides random number generation and distributions, a statistics class to calculate model statistics based on the attributes accessible to the abstract class, and a simulation element to model and run a simulation. The simulation element has five primary child classes that represent the main modeling components. An events class is used to organize and implement the events defined during the simulation run. A process class is used to handle specific resource manipulating processes. An entities class is used to represent the different transactions that move through a system. A nodes class represents specific points or objects in a system that the entities enter and exit. Lastly, a choices class handles the logic needed to manipulate transactions in the system. This overall approach

is broad enough to account for numerous types of systems that could be modeled through discrete-event simulation. Furthermore, the inheritance structure allows for the common simulation properties to be easily accessible.

The real functionality of the simulation model itself, though, is still left to the objects. The simulation structure and class hierarchy are only useful if the individual objects involved are given the appropriate methods and attributes needed to effectively interact and change the model state. Baezner and Lomow focus on three basic criteria to provide the general needs of a simulation: an entities class, an events class, and a property to keep the simulation time. The entities class is used to model any of the physical objects or processes being modeled, and the events class is used to schedule and trigger the actions of all entities. The simulation time is appropriately updated through the events. Joines and Roberts (1998) further extend their class hierarchy recommendations by providing specific information about the classes involved. Three classes of interest in the hierarchy are the events, entities, and nodes classes. The entities class provides the elements that are to be moved throughout the system to invoke changes. The entity class includes methods to obtain the entity's creation time, status, current location, and unique identifier (ID). The nodes class is used to model different nodes or locations within the modeled system. This class allows for a network of nodes to be established and provides methods to obtain the type of node, identify entities at the current node, list nodes within the system or network, and identify the node ID. There are also methods triggered when an entity enters a node or leaves a node. The events class contains methods to set and get the event time, process the event, and manage the

events and what they are associated with. The next section will provide more details regarding to the handling of events.

The majority of discrete-event simulation approaches rely on methods for handling a series of defined events. Typically, events are added to an ordered list in which each event is handled separately based on the time. In this setup, a simulation clock keeps track of the time and only advances in discrete amounts when all of the events for the current time are finished processing (Brunner and Schriber, 2005). In the structure proposed by Joins and Roberts (1998), events are pulled off of a calendar and trigger appropriate transactions in the model. These transactions (entities) trigger node events and, thus, allow for a fairly decentralized method of logically handling each event. Lin, Sheu, and Yeh (1996) propose an event handling method that focuses on sending each current event to a specific event handling routine (either a class or method) to handle the logic associated with that event. Although this can simplify and organize the responses to events, the logic required to check the routine could be avoided when utilizing the overloading features of class inheritance.

Numerous applications have been developed to illustrate and utilize the power that object-oriented simulation has to offer. One such application, YANSL, was developed by Joines and Roberts (1998) as an illustration of their proposed object-oriented simulation architecture. YANSL includes both transaction and resource entities moving through a network of node objects. The node objects consist of a variety of both departure and destination node types with different functionality, including a source node (to create entities), queue node (to hold entities), activity node (to modify entities), and sink node (to remove entities). The event handling triggers methods in the departure and

destination nodes to be appropriately triggered and adjust the movement and attributes of entities in the model. YANSL still has significant modeling and statistical limits, but the extensive use of class inheritance effectively reduces the amount of logical statements and decisions to be made, and allows for decentralized control of the simulation.

4.5 Cyber Attack Simulator Developed by Kuhl and Kistner

The simulation model developed by Kuhl and Kistner (2005), known as the Cyber Attack Simulator, was created using ARENA, a commercial simulation modeling software developed by Rockwell Software. Within this software, a custom template is provided to represent the network devices that needed to be modeled. This template includes the connectors (routers or switches) and the machines. Also, the ability to connect these devices allows a user to graphically setup a simple model of a computer network. Figure 4.3 displays a sample network setup in this interface.

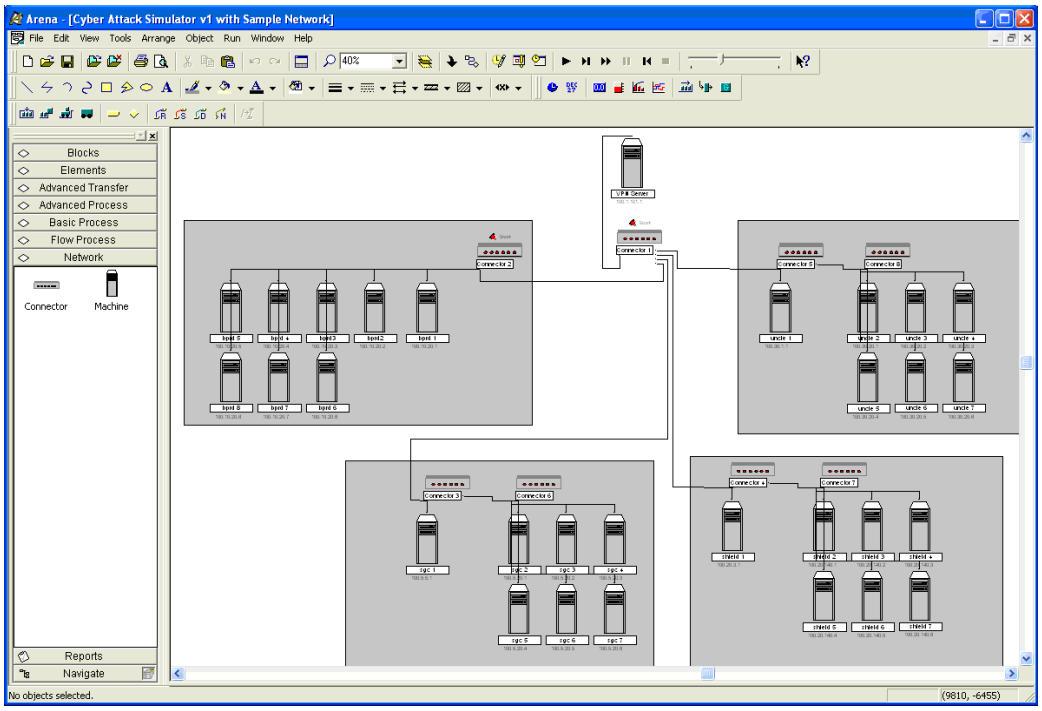

Figure 4.3: Sample Computer Network

The machines within the simulated network are provided with attributes, such as an IP address, the type of access, and the use of an IDS sensor, to assist in the attack modeling. The IP address is used as a unique identifier, the type of access (internet access) controls whether attacks could start at a machine, and use of an IDS sensor dictates whether the machine would generate appropriate IDS alerts (for the corresponding sensor type) based on the traffic to/from the sensor. Kistner (2006) also expanded upon these attributes to add an indication of the type of machine (whether it was a PC or a server) and the operating system. These attributes are important because they narrow down the type of attack exploits that can be performed on a machine. However, there are still more attributes that should be considered in effectively modeling what exploits can occur, especially with respect to a machine's software and services. Additionally, the ability to represent a group (or subnet) of machines can simplify the process of modeling networks.

The connectors within the simulated network are also provided with the use of an IDS sensor, which functions in the same way as with the machines. In reality, though, there exist differences between these two types of sensors (network-based and host-based sensors) that should be considered in the simulation model. The IDS alerts are generated by matching the attack (or noise) traffic with a sensor alert through a file containing direct correlations between the attack signatures and the alert messages.

The Cyber Attack Simulator, through integration with VBA, provides forms for setting up attacks on the modeled networks. Cyber attacks, initially, have to be specified by the user. A source (attacking) machine and a target (attacked) machine are specified for each step, along with what exploit or specific category of exploits is going to be used.

Also, the model is limited to allow only ten attacks and twenty-five steps per attack. This process puts stringent limitations on how much IDS alert data can be generated, and introduces significant user bias to the system. For these reasons, Kistner (2006) developed an automatic attack methodology that generates a set of attack steps based on several attack parameters provided by the user. These parameters primarily include the target machine, goal type, efficiency of the attack, stealth of the attack, and the average step time. The efficiency relates to the directness of the attack and how successful the steps are, while the stealth relates to how hidden the attack is from being detected by avoiding certain categories of exploits. Although the methodology provided a good basis for developing unbiased attacks in a short amount of time, there are still significant limitations to the types of attacks possible. Considerations regarding the parameters provided and the logic within the methodology can lead to significant improvements in the attack generation process.

5 Simulation Model Structure

Prior to this thesis, the cyber attack simulator was initially developed using the ARENA software. This development resulted in a model with basic network editing capabilities and a reasonable level of attack specification capabilities. Although this ARENA model has proved that the concept of simulating cyber attacks is plausible, the new features desired for the model, such as working with XML data, are beyond the limitations of ARENA. The desire to overcome such limitations has since motivated the development of an independent simulation model that can be appropriately configured to implement features that are not possible with the ARENA simulator. This new model includes a customized interface specific to the development of network structures and attack scenarios, a series of input and output options, and more detailed features in defining the networks and attack scenarios.

The model was developed using the object-oriented programming language Java. Java has many desirable characteristics in modeling realistic environments. Java is also a cross-platform programming language that only requires the JRE (Java runtime environment) to run a java program on any platform. The remainder of this section discusses the primary intentions of the model, provides a background of the development environment used, and presents the structure of the modeling elements.

5.1 Modeling Intentions and Model Overview

The overall goal of developing this cyber attack simulation model is to create an application that can generate valid intrusion detection system alerts in a virtual network representative of a real private network. With the cyber attack simulator, a user can

create or load a specific network topology, specify the vulnerabilities of the network, create and run attack scenarios, and view sensor alert data produced. Several inputs and outputs are necessary to the functionality of the simulator. A diagram depicting the types of inputs and outputs is displayed in Figure 5.1.

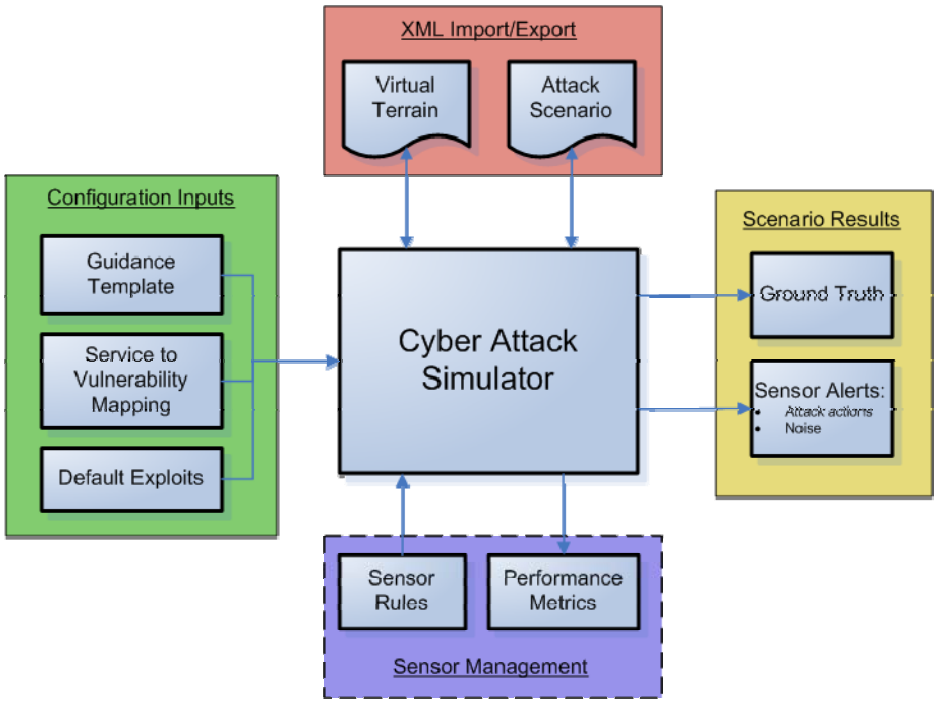

Figure 5.1: Cyber Attack Simulator Functionality

As is shown in Figure 5.1, the simulator consists of three primary categories of inputs and outputs: configuration inputs, XML imports and exports, and scenario results. A sensor management add-on, developed by McConky (2007) is also shown.

The configuration inputs include data that is loaded from files as the simulator is opened. The guidance template file is a directed graph that indicates what sequence of stages can be used in an attack and what categories of exploits are included in each stage. This information is used when attacks are generated based on a set of parameters. Another file includes the service to vulnerability mappings, which maps a machine service to a set of vulnerabilities through the use of service IDs and vulnerability IDs.

This effectively indicates what exploits can be executed on a machine that is running a certain service. The default exploits file loads a database of known exploits (also referred to as available actions) that the simulator can choose and filter from. This database of actions is used by the simulator when creating the individual steps of an attack as well as when creating the noise (or false-positives) that occurs during an attack scenario.

The XML imports and exports allow for structured XML documents to be created by or interpreted by the simulator. These documents can also be created by and interpreted by other applications, providing a means by which the simulator can interface with these applications. A document that is commonly used by the information fusion tools in development is the “Virtual Terrain” XML document. This document depicts the detailed structure of a network, whether the network is real or virtual. Figure 5.2 shows how the virtual terrain document is used among different applications. The cyber attack simulator specifically can read in a network model from the virtual terrain or create a virtual terrain document from a network that has been modeled. Therefore, applications that make use of the alert data generated by the simulator can also be provided with the entire structure of the network used to generate the data. Also, an additional XML document depicting the set of attacks in an attack scenario can be created by or read in by the simulator.

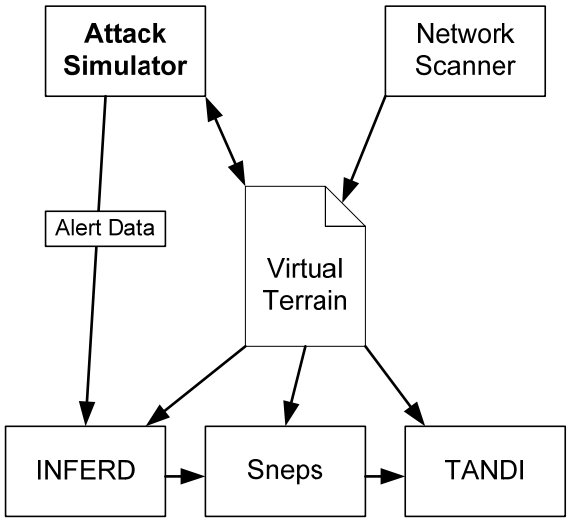

Figure 5.2: Integration of Virtual Terrain

The scenario results include data that is generated during an attack scenario run and output into text files. A ground truth file lists all of the attack-based actions that occur, along with details such as the attack that the action belonged to, the source and destination of the attack, and the success or failure of the action. A set of sensor alert files list all of the IDS alerts generated by sensors placed in the network model, with a separate file for each sensor. These IDS alerts correspond to both attack-based actions and noise-based actions.

Lastly, the sensor management add-on includes the appropriate functionality to define rules for the manner in which sensors handle the alerts generated. A fusion engine queue is modeled, and alerts are sent to this queue as defined by the sensor rules. Performance metrics are generated for purposes of comparing the sensor rules. For more information regarding the sensor management add-on and the underlying methodology, refer to McConky (2007).

5.2 Java Background

Before delving into the details of the program's structure, the reader needs to have a basic understanding of Java's object-oriented approach. This section describes some of the basic features and benefits of the Java programming language. Those familiar with the basic structure of Java (or a similar object-oriented language) can proceed to the next section.

The Java language is implemented as an object-oriented programming language. The primary approach in object-oriented programming is to represent real-world objects, such as people or machines, as programming classes with unique attributes and functions (or methods) that the objects can provide. Simulations in general attempt to provide an accurate model of some real-world situation that can be manipulated and studied to evaluate existing or proposed systems. Therefore, the use of object-oriented programming in developing a simulation framework has many benefits when considering that simulation models tend to represent the interactions between real objects (Bischack and Roberts, 1991). For example, in the case of the cyber attack simulator, a *Sensor* class is used to represent a real-world IDS sensor. Each sensor included in a modeled network represents one instance of this class. A sensor instance is then given unique attributes, which include the location, type, alerts produced, etc. The class also includes methods that each instance can use, such as adding a new alert to the sensor's list of alerts or outputting all of the alerts produced to a text file.

In addition to the use of classes in representing objects, Java has other key features that help in providing the modularity and reusability to make a simple and flexible

programming language. These features include inheritance, encapsulation, and polymorphism.

Inheritance allows a class or object to be treated as a child of another class. All of the attributes and methods associated with the parent class are inherited in child classes. Additionally, child classes can declare new classes and attributes that make them unique from the parent. Also, a child class can overload the methods used in parent classes in order to provide a different functionality (Bailey et al., 2005). Referring back to the sensor class example, two other classes are used to specifically represent network-based sensors and host-based sensors. Since these classes are slightly more specific versions of the sensor class, they use the inheritance property to acquire all of the same attributes and methods as a generic sensor.

Encapsulation allows for the attributes and methods specific to a class to be shielded from other classes if desired. By making the attributes and methods private in this manner, external classes must access the attributes and methods through a type of interface to the class. This interface is in the form of public methods where the programmer can clearly define and limit how a class's attributes can viewed and changed and how methods can be used. The use of packages in Java is another form of encapsulation. Packages allow several related classes to be grouped together with fewer restrictions on class interactions.

Polymorphism allows multiple classes to utilize the same behavior by implementing shared methods. This also includes the ability to use different constructors in initializing objects (Joines and Roberts, 1998). For example, in the cyber attack simulator, the *Machine* class uses a different constructor for defining a single host

machine than the class uses for defining a machine that belongs to a host cluster (or subnet).

In review, the ability to re-use and extend objects through the features described earlier is a central characteristic and advantage to developing a modeling framework in Java. Also, since objects are modular, the information associated with each instantiation of an object is held in only one location, with references to the object's location when needed (Bischack and Roberts, 1991). Furthermore, the simulation modeling logic can be appropriately divided among classes. This delegation of modeling functions provides a more organized structure than would otherwise be possible with procedure-based modeling (Joines and Roberts, 1998).

5.3 Package Structure

The simulator has its classes organized into six packages based on the interactions of the classes. These include a simulation package, network package, attack package, visual package, VT (virtual terrain) interface package, and a sensor management package. The diagram in Figure 5.3 illustrates this structure and lists some of the primary class types and/or features of each package.

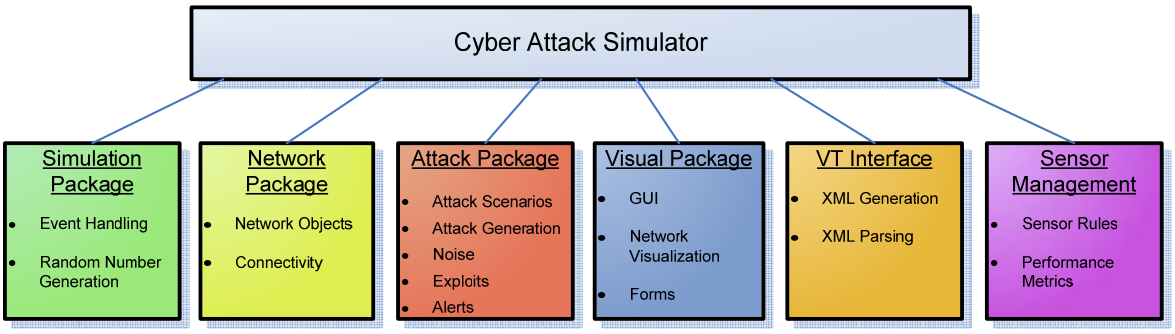

Figure 5.3: Cyber Attack Simulator Package Structure

The simulation package includes classes to maintain and simulate through an ordered list of events. The package also includes classes to provide random number generation. Section 5.4 discusses the simulation package in further detail.

The network package includes all of the classes used in defining the network topology and device attributes. Also, the connectivity of network devices is implemented through methods within the classes. Section 5.5 discusses the network package in further detail.

The attack package includes the classes necessary to create attack scenarios, which consist of a series of attacks on a network along with network noise. The package includes classes and methods used to automatically generate a series of attacks when only attack parameters are specified. Also, the attack package contains classes that act as a database for available attack actions (or exploits) and sensor alerts. Section 5.6 discusses the attack package in further detail.

The visual package includes classes used to provide an interface to the user of the simulator. These classes help makeup the program graphical user interface (GUI), the network visualization, the data entry forms, and the result displays. Section 5.7 discusses the visual package in further detail.

The VT interface package contains four classes used to parse and create XML documents with virtual terrain data and attack data. Two of the classes are static classes that serve only the purpose of providing a series of class variables to represent XML elements and attributes. The remaining two classes are used to read and write XML documents, with one class devoted to the virtual terrain data and the other class devoted to the attack data. These two classes utilize the JDOM package, which is a separate

publicly available package used to write and interpret XML documents when provided with the structure of the document. The JDOM package is not included with the basic Java Development Kit (JDK).

The sensor management package includes classes to define and implement a set of sensor rules. The package also includes a fusion engine class and specific sensor classes that inherit from the sensor classes defined in the network package. More information regarding the structure of the sensor management package can be acquired by referring to McConky's thesis (2007).

5.4 Simulation Package

One of the crucial components of the model is the underlying simulation functionality. This functionality is provided by using objects and methods that are commonly found in an object-oriented simulation framework. In the model, the simulation package encompasses a set of custom-made classes that emulate the typical workings of a simulation framework. Figure 5.4 illustrates the simulation package class structure as a simplified UML diagram.

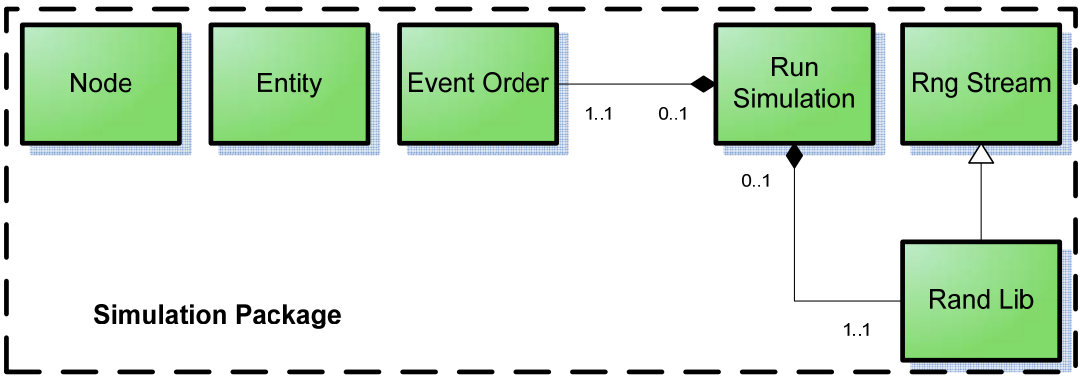

Figure 5.4: Simulation Package Class Structure

The structure used in Figure 5.4 depicts labeled blocks as classes, while lines between two blocks represent a relationship between classes. The directed arrow indicates inheritance; for instance, the *Rand Lib* class inherits from the *Rng Stream* class. The diamond-ended connection indicates composition; for example, an instance of the *Event Order* class is treated as an attribute of the *Run Simulation* class. Table 5.1 provides details regarding each class in the package.

Table 5.1: Simulation Package Class Details

Class Name	**Type**	**Description**
Node	Abstract	Represents a physical location that entities enter and exit. This class is not directly instantiated, but instead acts as a super-class for the classes in the Network Package representing network devices.
Entity	Abstract	Represents objects that move between nodes. The class also includes attributes that can be changed or used by the nodes. This class is also not directly instantiated, but instead acts as a super-class for specific classes in the Attack Package representing network traffic.
Event Order	Concrete	The event order class is essentially a list that keeps track of the events to occur in the simulation, in chronological order. An instance of this class is treated as an attribute of the run simulation class.
Run Simulation	Concrete	Executes a discrete event simulation of the attack scenario by pulling events from (and adding events to) the event order class. The run simulation class also keeps track of the simulation time, which is based on the time of the current event.
Rng Stream	Concrete	The RNG (random number generator) stream class provides the functionality to generate random numbers when provided with a starting seed. The RNG stream is a publicly available class created by L'Ecuyer et al. (2002)
Rand Lib	Concrete	The rand lib class inherits from the rng stream and provides additional functionality to allow for multiple streams to be defined. The class also provides the functionality to sample numbers from a specific distribution (such as Uniform and Exponential).

The six classes included in the simulation package provide the basis for running a discrete event simulation in the cyber attack simulator; however, the implementation of

these simulation elements is substantially integrated with both the network and attack package. Therefore, a clear understanding of both of these packages is necessary before providing more details about the simulation. Section 7 of this thesis (attack simulation methodology) provides a more in-depth description of the functionality of the simulation elements and how they are integrated with the other packages.

5.5 Network Package

The model includes a set of classes used to define a virtual computer network. The network package encompasses this set of classes, which range from physical network devices to software settings on these devices. Figure 5.5 illustrates the network package class structure, while Table 5.2 provides details about each class included in the package.

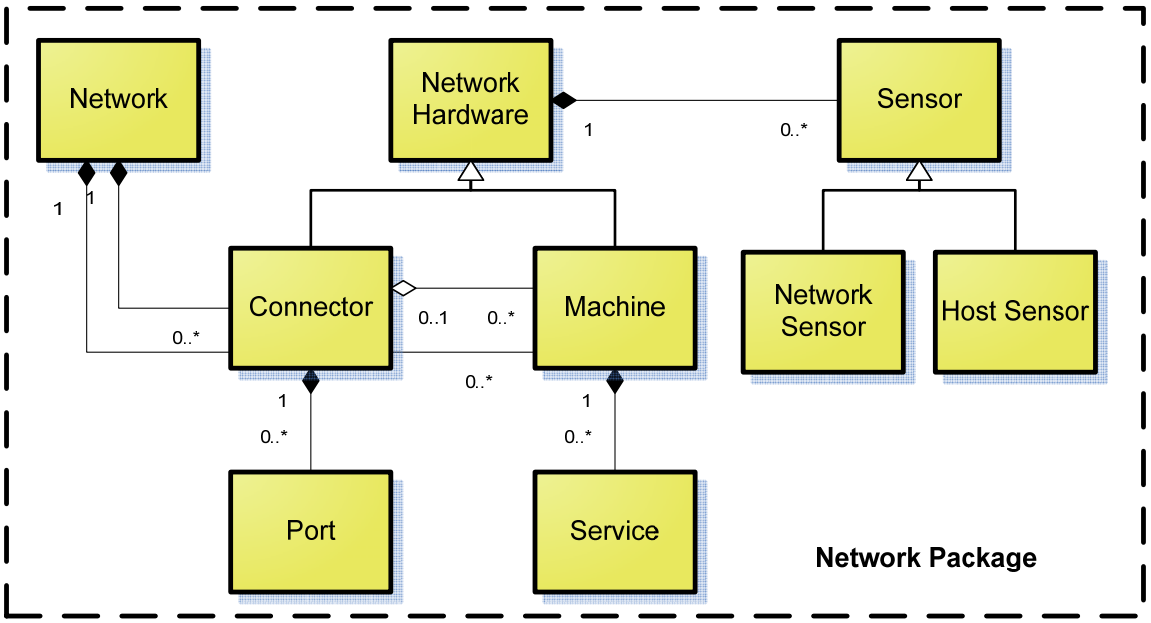

Figure 5.5: Network Package Class Structure

Table 5.2: Network Package Class Details

Class Name	Type	Description
Network	Concrete	Represents a virtual network, and the class consists of a set of connector and machine objects. The class also contains attributes such as the network name, the starting IP address, and the attack scenario objects created for the network (from the attack package).
Network Hardware	Abstract	Represents the physical devices that are linked together to form the network. This class itself is not instantiated, but both the connector and machine class are sub-classes of the network hardware class. Key attributes include the name, unique id, and IDS used (if any).
Connector	Concrete	Generically represents a network linking device, such as a hub, router, or switch. Some key attributes of the connector class include the machines and other connectors linked to the connector and the firewall restrictions (allowed/banned ports). Also, the class inherits attributes from the network hardware class.
Machine	Concrete	Generically represents a host computer or server located in a network. This class has attributes such as the type of machine, the group ID, the operating system, the services running, the connector that the machine is linked with, and the IP address. The machine class inherits attributes from the network hardware class.
Sensor	Concrete	Represents an IDS sensor, which can be added to devices in a network. The sensor class includes attributes such as the type of sensor, the location of the sensor, and a list of alerts generated by the sensor.
Network Sensor	Concrete	The network sensor class is a sub-class of the sensor class representing an NIDS (network IDS). This type of sensor is placed on connectors only, and monitors network traffic.
Host Sensor	Concrete	The host sensor class is a sub-class of the sensor class representing and HIDS (host IDS). This type of sensor is placed on machines only, and monitors logs (which essentially represent successful network traffic).
Port	Concrete	Represents a specific port that is either allowed or banned for traffic moving along a particular path through the network. The class includes attributes such as the port number, the protocol, and whether it is allowed or not.
Service	Concrete	Represents a service (or type of software) that a machine is running. This class has attributes such as the service name, unique ID, and ports used for the service.

As mentioned previously, the abstract *Node* class in the simulation package is used as a super-class of the physical network components. This is accomplished by the network hardware class inheriting from the node class. Therefore, both connectors and machine objects represent nodes that entities can move between. This is representative of a real network, as packets of information do move through defined paths between network devices. Also, although there is no specific subnet (or host cluster) object, a subnet is defined by providing a series of machine objects with the same group ID. Therefore, each member of the subnet still functions independently, yet can be controlled collectively. Chapter 6 of this thesis discusses the methodology used in providing the network functionality and creating virtual networks that are representative of real private networks.

5.6 Attack Package

With the stationary aspects of the cyber attack model defined in the network package, the attack package is intended to incorporate the “mobile” components of the model (or at least mobile with respect to the movement of information). The attack package essentially includes classes for modeling the attacks that can progress through a computer network, along with other non-attack related traffic as well. Figure 5.6 illustrates the class structure of the attack package.

The dashed arrows in Figure 5.6 represent a dependence relationship. In this relationship, the class is dependent on another class for either its methods or attributes; however, neither inheritance nor composition is used. For instance, the *Attack* class accesses the class attributes in the abstract *Guidance Template* class even though the two

classes are not physically linked in any manner. Table 5.3 provides details about each class included in the attack package.

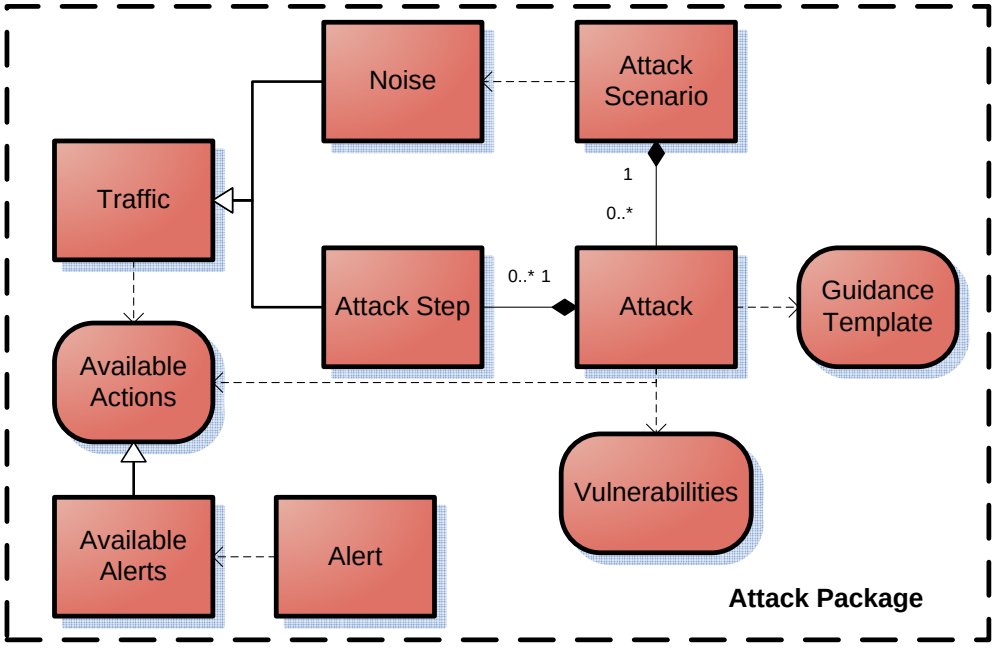

Figure 5.6: Attack Package Class Structure

In addition, the traffic class inherits from the abstract *Entity* class defined in the simulation package. Defining the traffic as a sub-class of the entity class allows for traffic to be routed between nodes in the network structure just as real network traffic would be routed through devices in a real network. Additional methodology is provided to prohibit or allow the routing of traffic between two nodes and to generate an alert if a sensor is present at a node. Chapter 7 of this thesis discusses the methodology used to manage the flow of network traffic as well as the methodology associated with generating attacks and attack scenarios.

Table 5.3: Attack Package Class Details

Class Name	Type	Description
Traffic	Abstract	Generically represents the information moving between network devices. Some attributes of the traffic class include the time, source machine, destination machine, action performed.
Attack Step	Concrete	Represents traffic that is associated with a cyber attack. The class includes attributes indicating the attack that the step belongs to and the sequence of the step.
Noise	Concrete	Represents traffic that is not associated with an attack (although the traffic can still be considered malicious by sensors).
Attack Scenario	Concrete	Represents a series of attacks and a set of noise parameters, which effectively define a cyber attack scenario that can be run as a simulation. Attributes for this class include a set of attack objects, noise-related parameters, the network the scenario corresponds with, and a start time for the scenario.
Attack	Concrete	Represents an individual attack, which includes a series of steps progressing through the network toward a specific goal. The attack class also includes parameters that can be used to set up an automatically generated attack (rather than specifying the steps). The attack class attributes include a set of attack steps, the start time of the attack, the total attack time (or, alternatively, the average step time), the final target machine, the final goal of the attack, and the auto-attack parameters.
Guidance Template	Abstract	Used to establish the structure of the guidance template input. This class is utilized by the auto-attack generation to determine the appropriate stages that the attack will progress through.
Vulnerabilities	Abstract	Used to provide the mapping of service IDs to a set of exploits that the service is vulnerable to through the use of vulnerability IDs.
Available Actions	Abstract	Represents a database of actions (or potential exploits) that are available to be used when generating the network traffic. Each action has a unique index that its attributes are referenced by. These attributes include the stage category, the vulnerability ID, the type of machine required, the operating system required, and the default port used.
Available Alerts	Concrete	This class inherits from the available actions class to provide additional sensor-related alert information about specific actions. One instance is created for each unique sensor type.
Alert	Concrete	Represents an alert generated by a sensor. Attributes of this class include the time of the alert, the action detected, and the path of the traffic that produced the alert.

5.7 Visual Package

An important aspect pertaining to the use of the program is the graphical user interface (GUI). The GUI for the cyber attack simulator is provided through the many classes contained in the visual package. Figure 5.7 illustrates some of the primary classes (and interfaces) that make up this package.

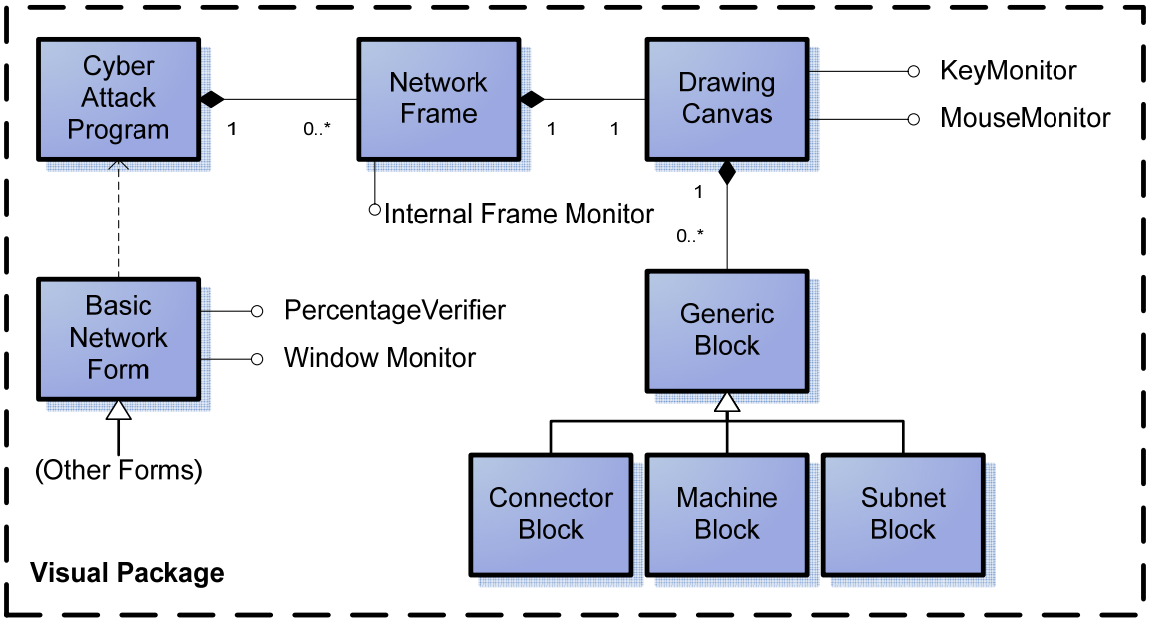

Figure 5.7: Visual Package Class Structure

The small circles attached to the classes in Figure 5.7 represent custom-made interfaces that are implemented by those classes. An interface in Java reacts to certain actions (user actions in this case) to invoke methods in another class. For instance, the *Mouse Monitor* detects the clicking and movement of the mouse by the user. The visual package also contains over twenty other forms not displayed in the figure. These forms are either sub-classes of the *Basic Network Form* or subclasses of some other form, but the forms all inherit the properties and methods of the basic network form. These additional forms are used for both the entry and viewing of data for the simulator.

Table 5.4 provides details about the primary classes that make up the visual package. An overview of all of the forms used in the visual package is provided in Appendix A.

Table 5.4: Visual Package Class Details

Class Name	**Type**	**Description**
Cyber Attack Program	Concrete	The main class and primary interface for working with the program. This class provides the user with menus, toolbars, and a work area for managing multiple networks.
Basic Network Form	Abstract	Represents the basic structure of the program's data entry forms. The class provides details about the structure and placement of a form and keeps track of the active window.
Network Frame	Concrete	Provides the interaction with an individual network. This frame is placed in the program's working area and includes a drawing canvas object along with components used to setup attack scenarios.
Drawing Canvas	Concrete	Provides a canvas that is used for constructing the network topology. Different network components are added as "blocks" to this drawing area.
Generic Block	Abstract	Represents the basic structure for a block component that will be placed in the drawing canvas to visually represent a device (or set of devices) in the network.
Connector Block	Concrete	A sub-class of the generic block class that represents a connector in the network.
Machine Block	Concrete	A sub-class of the generic block class that represents an individual machine in the network, with a different image for hosts and servers.
Subnet Block	Concrete	A sub-class of the generic block class that represents a group or subnet of multiple machines in the network.
Percentage Verifier	Interface	Provides some forms with the ability to monitor the values entered by the user and update other values accordingly.
Window Monitor	Interface	Used with each form to monitor when the form is closed and switch the user's "focus" to the appropriate window (or form).
Internal Frame Monitor	Interface	Monitors when a network frame is added or removed from the program workspace to appropriately update the menu options of the cyber attack program class.
Key Monitor	Interface	Monitor for keys pressed by the user while drawing the network, such as the "esc" or "delete" key.
Mouse Monitor	Interface	Monitors for mouse actions performed by the user, which allow the movement and editing of blocks in the canvas.

Upon running the model, the user is initially provided with an instance of the cyber attack program class, along with all default inputs loaded. This interface allows the user to load networks, create networks, or alter the program preferences. More menus and options become available once a network model is actively displayed. Figure 5.8 displays a screenshot of the program with two example networks displayed. The figure also indicates some of the key features of the program (red boxes/arrows).

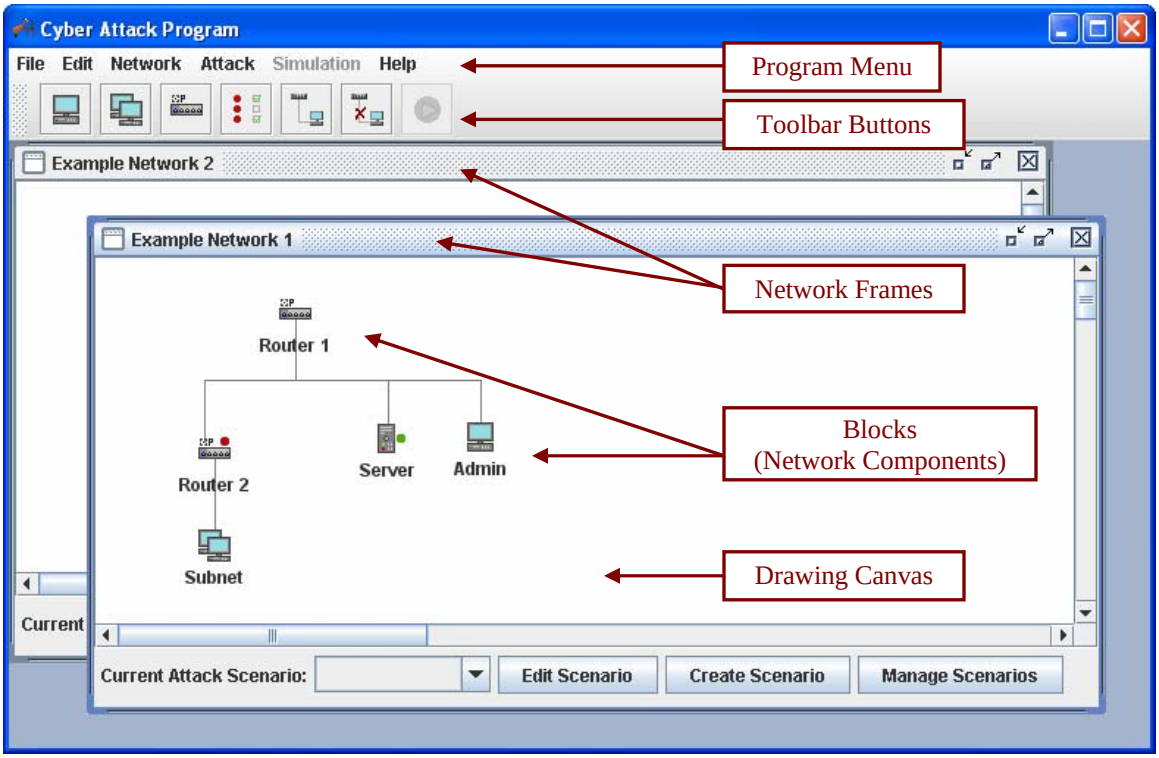

Figure 5.8: Screenshot of Cyber Attack Program

In Figure 5.8, for the network shown in the foreground, the objects drawn represent the visual network components. Router 1 and Router 2 are instances of the connector block class, which visually represent instances of the connector class (in the network package). Server and Admin are instances of the machine block class, which represent machine objects. Subnet is an instance of the subnet block class and represents a set of machine objects. The network canvas class also provides methodology to draw

connections between objects in the network that are physically linked. The figure displays that Router 2, Server and Admin are all connected with Router 1, and that Subnet is connected with Router 2. The blocks in the canvas can be moved around, and double-clicking on a particular block will invoke a form to edit the object that the block represents. Additional blocks, connections, and even attack scenarios can be setup using the appropriate buttons provided in the network frame, program toolbar, and program menus. Also, the program provides an interface of forms used to both run a simulation and display results once both a network and an attack scenario for the network have been created.

6 Network Methodology

The ability to define a well structured and detailed virtual network is an important aspect of providing realistic Intrusion Detection System alerts. In this application, the modeled network must be representative of a real private network, and network-related attributes that the model provides must reflect real network attributes that can potentially have an effect on the IDS sensors and alerts. This chapter details the methodology used to supply the simulator with network modeling capabilities that fulfill both of these requirements. By modeling the network to meet these requirements, a network structure is provided that can both effectively and efficiently be used in constructing cyber attacks and generating IDS sensor alerts.

The chapter starts by discussing the architecture of a typical private network and how the model takes this architecture into consideration when modeling a network topology. Next, the approach taken to model machines (or computers) in the network is discussed. Then, the connectivity and permissions associated with the connectors in a network is explained. Additionally, the placement and functionality of both NIDS and HIDS sensors is discussed. Lastly, the process and critical details of reading and writing a "virtual terrain" document are reviewed.

6.1 Network Architecture

The structure of a private computer network can take on a variety of forms. In the simplest case, all of the network devices (host computers, servers, printers, and modems) can be connected to a single central hub (connector) that allows all devices to communicate directly with each other. This setup, though, provides limited security once

any device in the network becomes compromised. The modern approach to setting up a network is to create several different levels or tiers within the network. Each of these levels consists of a specific set of network devices, and the levels deeper into the network are more difficult to get access to from an external source. Figure 6.1 illustrates this concept with an example three-level network structure.

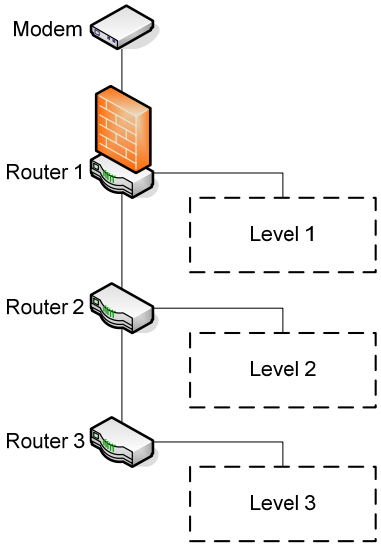

Figure 6.1: Network Levels Concept

For the network structure in Figure 6.1, Level 1 typically represents a set of web servers and ftp servers that are setup in the network. These servers are connected directly to Router 1, which provides a firewall that allows only a specific subset of traffic to the servers. Level 2 commonly represents a set of internal servers and administrator host machines. Internal servers cannot be directly accessed from an external source, while the administrator hosts are used to configure both the internal and external servers. Level 3 typically represents a subnet of host machines used by many individuals. Such host machines are often necessary in providing essential functions for an organization, and the machines most likely have access to retrieve and alter information on the internal and

external servers. These machines are therefore connected at the deepest level within the network.

The cyber attack simulator is designed to model a variety of network configurations; however, the focus is on more intelligently configured networks that provide at least some breakdown of the network into multiple levels. Although simplistic, centralized-hub networks can still be modeled, the simulator contains several useful features to assist in defining multi-level networks. One of the primary features is the use of a “parent-child” relationship in modeling the links between connector objects. This type of setup is a tree structure where each connector linked with another connector can either be treated as a “parent” or “child” of that connector. A parent connector refers to a connector at a higher or shallower level of the network (closer to the external portion); while a child connector refers to a connector deeper in the network structure. To keep the network structures reasonably simple and avoid cases where the network levels are not clearly defined, a connector is limited to having one parent connection. Although some real networks may allow multiple connections to a higher portion of the network, this is a redundant approach that forms in a complete loop within the network. Having a complete loop results in multiple paths possible between devices and complicates the routing of traffic through the network.

A visual example of the connector linking approach used in the simulator is displayed in Figure 6.2. The connector blocks feature a special connection point to indicate which link from the connector represents that connector’s parent. All other links are treated as child connections. When referring to Router 2 in the figure, Router 1 represents the parent connector, while Router 3 and Router 4 are the child connectors.

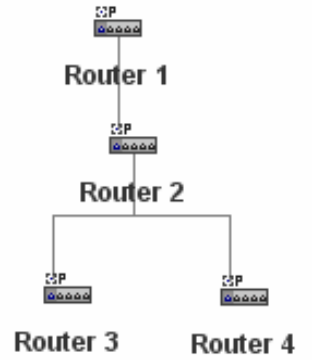

Figure 6.2: Parent-Child Relationship

The simulator prohibits the network structure from containing lateral connections. For instance, in Figure 6.2, Router 3 and Router 4 cannot be linked in any additional manner (other than through Router 2) since such a link would result in a complete loop. Although this limitation can restrict the connectivity of the network, the simulator provides a way around this limitation, which is discussed in section 6.3.1.

6.2 Machines

The network modeling features of the cyber attack simulator use machine objects to represent host computers and servers within the network. These two types of devices are responsible for both information and functionality within a network, and they are the targets that attackers seek to exploit. Each machine can have a substantial number of attributes that reflect both the functionality and accessibility of the machine. Modeling all of these attributes would be far too meticulous for this application. Therefore, the model focuses on key machine attributes that are associated with or have an important impact on the attacks and intrusion detection alerts generated.

Machines can either be added to the modeled network individually or in a group, which is also referred to as a subnet or cluster. When adding an individual machine to the modeled network, the user is presented with a form that allows for the attributes of that

machine to be specified. Figure 6.3 displays this form with the attributes assigned. As is shown in the figure, the attributes of interest in modeling a machine object are the name, ID, connector linked to, IP address, access, machine type, IDS used (if any), operating system, services running, IPS used (not currently implemented), and the relative importance of the machine in the network. Machine subnets (or clusters) are also provided with this same set of attributes. Refer to section 6.2.1 for information on how machine subnets/clusters are modeled and created.

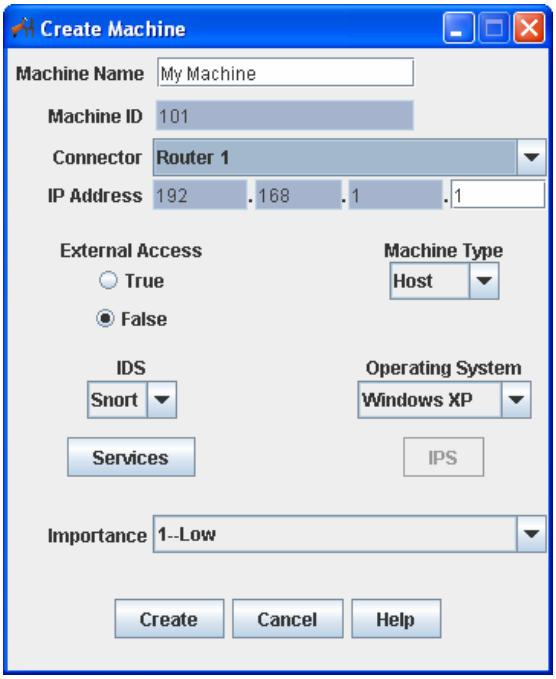

Figure 6.3: Form for Creating a Machine

The machine name is primarily used for display purposes and is shown underneath the machine block that is placed in the network's canvas. The machine ID represents a unique (and auto-generated) device ID that is used to reference a specific machine in the virtual terrain document. The connector refers to which connector the machine is directly linked with. A machine can only be linked with one connector, and

this will be shown with a line drawn between the machine and connector in the network's canvas.

The IP address is another unique identifier of the machine. In a network, machines communicate by using the IP address. The source and destination of a particular packet of network traffic refers to the IP addresses of the machine. Also, the sensor alert output will refer to a machine by the IP address. By default, the first two components for all IP addresses of machines in the network are specified in the network setup. The third component of the IP address is based on the unique ID of the connector that the machine is linked with. The fourth component, by default, is the next available unique number (up to a value of 255) when considering IP addresses matching the other three components.

The external access option indicates whether a machine has direct external access (meaning that it can be accessed from the internet). Web servers and FTP servers are examples of machines that will typically have external access. This is an important attribute, since attackers typically use computers with external access as a stepping stone into the rest of the network.

The machine type refers to how the machine functions. The options for this attribute are host, server, and generic. By this point, hosts and servers have already been described. Generic, though, indicates that the machine has both the features of a host and a server (or there is no certainty as to which type the machine is). Similarly, the operating system attribute indicates the underlying software running on the machine. The options for this attribute currently include Windows XP, Windows 2000, Linux, and UNIX. Both

the machine type and operating system are important attributes in determining the type of vulnerabilities that attackers can exploit on a machine.

The IDS option allows for a sensor to be added to the machine. This sensor represents an HIDS (host-based intrusion detection sensor), which is a sensor that exists as software running on the machine monitoring the system logs. More information about the sensors be provided in section 6.4.

The services attribute of a machine indicates which types of software-based services are running on that particular machine. Referring back to Figure 6.3, the services button provided opens a separate window that allows for the set of services used to be specified. Section 6.2.2 provides more depth regarding the services and the role that the services play in modeling a network.

6.2.1 Subnets/Clusters

A subnet (or host cluster) is not modeled as an individual object, but rather as a set of machine objects. The machine objects represented in a subnet have only a slight variation from the structure of individual machine objects. For this reason, the same class is used for both types (with a different constructor). The primary differences include an additional attribute for a group ID as well as an attribute for defining an IP range.

The group ID defines which group (essentially, subnet) a machine object belongs in. Each machine that is part of a specific subnet will be given the same group ID. With this attribute, the machine objects in a subnet can still be tracked back to the subnet even when the objects have different names, IDs and IPs. The group ID is kept unique in the same manner that the device ID is kept unique.

In addition to the group ID feature, machine objects within a subnet can also specify the fourth component of the IP address as a range of IPs (such as “0-255”). This feature allows for a machine object to represent up to 256 machines in a subnet. Therefore, a subnet can consist of one object that represents all of the IP addresses that are part of the subnet. In modeling large networks, minimizing the number of objects used to define a network has several performance benefits, and the option of listing a range of IPs to define a subnet can help in that respect.

The use of subnets also simplifies the display of and interaction with the modeled network. In the network’s canvas, a single subnet block represents all of the machines within a particular subnet. Editing the attributes of this block, such as the machine type, operating system, and services, will affect the attributes of all machine objects that are represented by the block.

6.2.2 Services

Services refer to specific software-based applications used by a machine. Services are another important aspect in determining a machine’s vulnerabilities (which will be discussed in section 6.2.3). Each type of service has a different set of ports and protocols that can be used for communication with the service. The set of services considered in the model are listed in Table 6.1 along with the ports and protocols that are used by the service. This information is provided by Crothers (2003). These services are chosen because each service is commonly used and generally requires communication with other machines either in the network or on the internet.

Table 6.1: Service Details

Service Name	Description	Protocol	Ports
FTP	File transfer protocol	TCP	20, 21, 989, 990
HTTP (Web)	Web server	TCP	80, 443
TFTP		TCP	69
Samba		TCP	901
Apache	Apache web server	TCP	80, 8000
SSH	Secure Shell	TCP	22
IMAP		TCP	143, 993
Oracle		TCP	1521
SQL	MySQL databases	TCP	118, 156, 1433, 1434
ICMP	Internet Control Message Protocol		

The services attribute used by machine objects allows for a list of services to be defined. By default, the ten services displayed in Table 6.1 are the services that can initially be selected from. Additional services can be part of this attribute list through the virtual terrain XML input. This capability is possible because the virtual terrain provides a direct vulnerability mapping of the services as well as a list of which protocols and ports the service uses.

In the model, each service is represented as a service object. The service class defines some basic attributes and methods for the services, and the class also includes class variables that store the default set of services and ports. These defaults are read in from a text file upon starting the application.

6.2.3 Vulnerabilities

As mentioned previously, one of the key vulnerabilities in a network relates to the coding errors and design issues present in certain software applications and services. Therefore, the vulnerabilities of a machine in the model correspond with the set of machine attributes that indicate what type of software/service exploits can be performed.

These attributes include the services running, operating system, and the type of machine. The concept used for filtering the potential list of exploits based on a machine's vulnerabilities is illustrated in Figure 6.4.

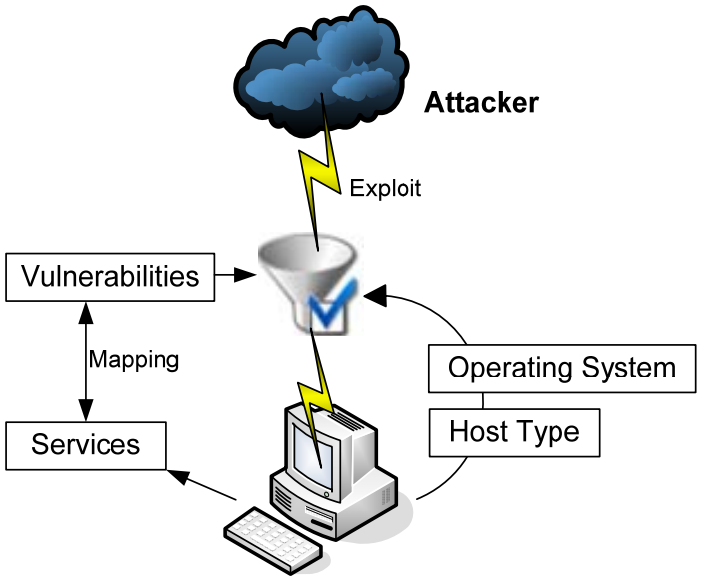

Figure 6.4: Machine Attributes Used to Filter Potential Exploits

Both the operating system and machine type directly filter out many of the potential exploits that are included in the model's database. This filtering occurs because exploits that may be possible using a Windows-based operating system may not work (using the same method) with a UNIX system.

The services running on a machine use a different approach to filtering potential exploits. For each operating system and machine type, a service running on a machine typically has several exploits that can be successfully performed against the machine. Therefore, one of the inputs to the model is a file mapping the services to a set of vulnerabilities. Each exploit in the database has a specific vulnerability ID that indicates what vulnerability is being exploited. Therefore, the set of services included on a

machine will indicate what exploits can be performed by referring to both the service to vulnerability mapping and the vulnerability ID of the exploits.

6.3 Connectivity

A connector object in the model generically represents any type of network linking device, such as a router, switch, hub, etc. However, Java allows for sub-classes of the connector class to be easily developed to specifically represent one type of connector. Therefore, a router or switch class (for example) could eventually be created that inherits the attributes and methods from the connector class. This thesis, though, will focus on generic connector functionality. By using this approach, only the connector attributes associated with modeling the device connectivity, network traffic flow, and alert generation need to be included.

Connectors are added to the model individually, and the user is presented with a form to specify the connector's attributes. Figure 6.5 displays the form that is presented. The attributes of interest for a connector object include the name, unique ID, relative importance of the connector in the network, parent connector, child connectors, IDS sensors used (if any), firewall rules, child connector links, and the bandwidth allowed for security (IDS) purposes.

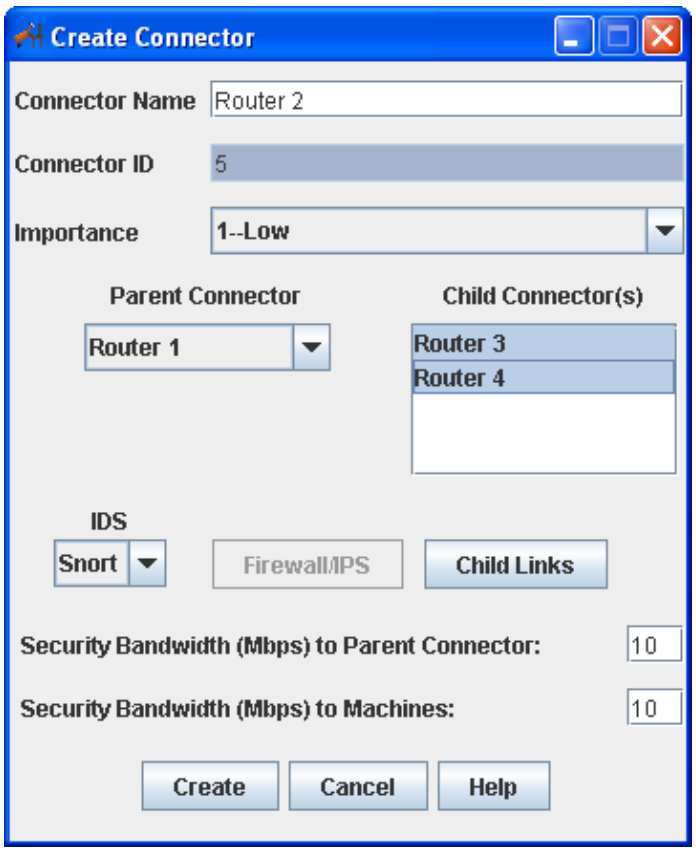

Figure 6.5: Create Connector Form

The connector name, like the machine name, is primarily used for display purposes and is shown underneath the connector block that is placed in the network's canvas. The connector ID represents a unique auto-generated device ID that is used to both reference the connector in a virtual terrain document and generate the default third IP component of machines connected to the connector.

The parent connector represents what the connector is connected to at a higher (shallower) level in the network, while the selected child connectors represent what the connector is connected to at a lower (deeper) level in the network. A connector can only have one parent connection in order to maintain the tree structure. There is no limit on the number of child connectors selected; however, the candidates for child connectors (that can be selected from) must not yet have a parent connection themselves. The parent and

child connections are all represented with lines drawn between the connector blocks in the network's canvas. A specific parent connection point is used to differentiate a parent connection from a child connection.

The IDS option allows for a sensor to be added to the connector. This sensor represents an NIDS (network-based intrusion detection sensor), which is a separate hardware device that monitors the network traffic. More information about the sensors is provided in section 6.4.

A connector's firewall rules can only be established once the parent connector and/or child connectors are specified. These rules essentially allow or block specific communication ports in the network traffic depending on the immediate source and destination of the traffic. Section 6.3.2 discusses how these rules work and how they are implemented.

The child connector links attribute indicates which of the lower level connectors can communicate with each other through the current connector. This feature is necessary since lateral (and possibly looping) connections are not allowed in the model. Section 6.3.1 discusses how the model allows for and makes use of these types of links.

6.3.1 Links

Consider a setup like the one displayed in Figure 6.6. The computers connected to Router 2 and Router 3 can both communicate with the fileserver on Router 1. Another server is also connected to Router 3, but the model's basic connectivity limitations prohibit any of the machines on Router 2 from communicating with this other server. This type of communication is commonly found in computer networks where

administrators wish to keep a crucial server at a deeper and more secure location in the network structure while still allowing different subnets to access the server.

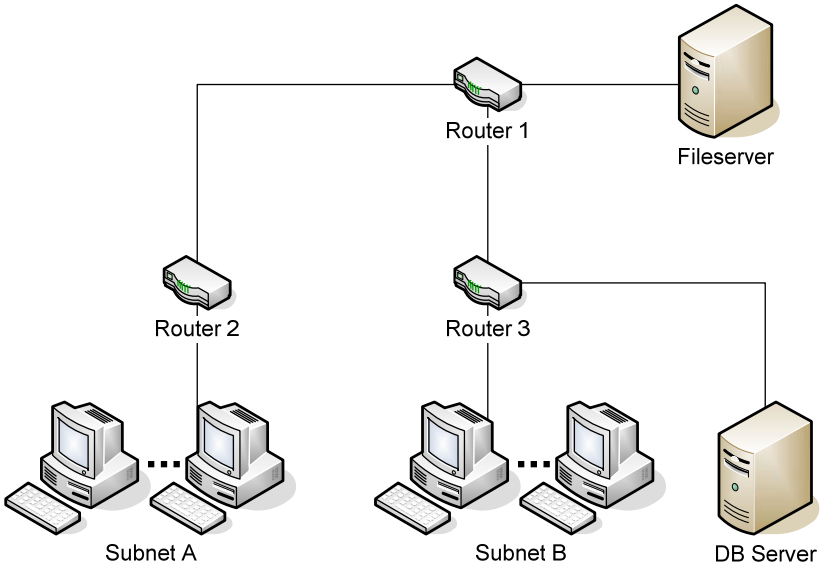

Figure 6.6: Child Connector Setup Example

As a way to provide this type of lateral communication between machines along the same network level (but on different connectors), the child links connector attribute is used. This attribute is essentially a listing showing which of a connector's child connectors can communicate with each other (through the connector). The attribute stores a set of source and destination communication links, since a network administrator could only want one-way communication to be allowed. The form for inputting these links is displayed in Figure 6.7. Through this form, the user can select both the source and destination connector from the list of available connectors (specifically, the list of child connectors).

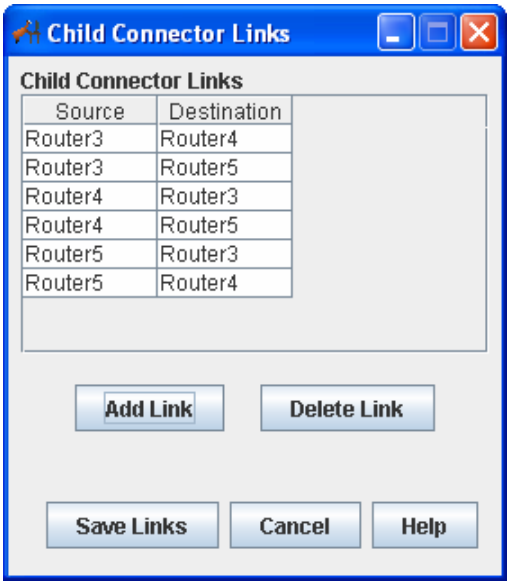

Figure 6.7: Child Links Form

6.3.2 Firewalls/Port Permissions

The firewall permissions attribute allows a user to indicate what type of network traffic can be processed, by the connector, through a specific path (or link in the network). This is the approach used in many network configurations to filter out both unwanted and unnecessary traffic in certain parts of the network. For example, a server that works strictly as a Web server will typically not be able to process application protocols other than HTTP. Also, keeping all of the server's ports open may leave the server susceptible to a wide array of exploits that could have easily been avoided by filtering out network packets by the port used. Figure 6.8 displays a simple network and example firewall setup that could be used to reduce both the network traffic and vulnerabilities. In this type of setup, both the immediate source and destination of a packet are factors in determining whether a connector finds the packet permissible. In the figure, the blue dashed lines and text boxes indicate what type of traffic is allowed through Router 1 and Router 2 between a specific source and destination.

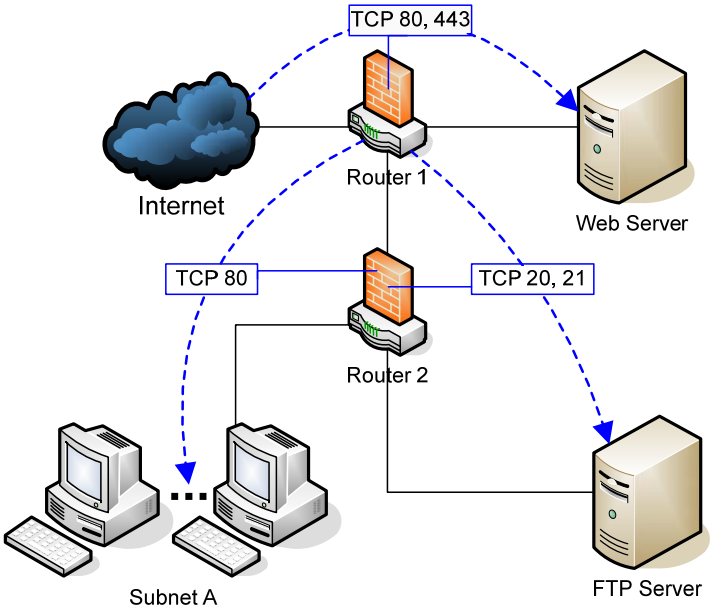

Figure 6.8: Sample Network with Firewall Permissions

The model implements such firewall permissions through a permission list array, which is an attribute of connector objects. This list specifies, for a source and destination device, what type of ports and protocols are allowed through the connector. The available set of source and destination devices include all nodes linked with the connector, such as machines, child connectors, and the parent connector. For a specific set of source and destination nodes, there are four different firewall setups that can be used, which include: allowing all traffic, allowing traffic using a port/protocol combination that is contained on a list of allowed port/protocol combinations, blocking all traffic, and blocking traffic using a port/protocol combination that is contained on a list of banned port/protocol combinations. The lists of allowed or banned port and protocol combinations are modeled as an array of port objects. The port class defines attributes for the port number, protocol, and a Boolean indication of whether the port is allowed or banned. To simplify the process of checking a firewall, the port objects in a permission list must be all allowed or

all banned for a specific source/destination pairing. The form used to setup a permission list for a connector is displayed in Figure 6.9. With this form, a user can add a list of either allowed ports or banned ports for specific source and destination pair.

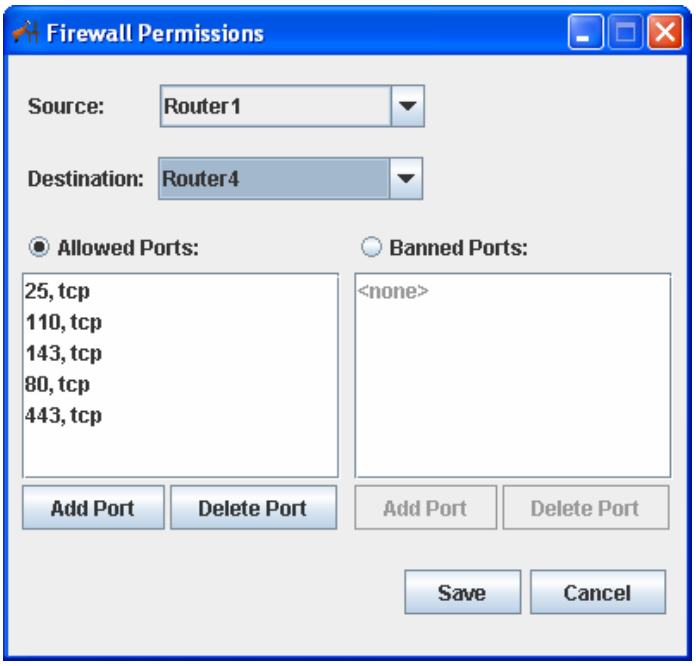

Figure 6.9: Permission List Form

A port/protocol permission list can be established for any two different devices (essentially, nodes) linked with the connector, including child links. A sample modeled network using different firewall permission list configurations is displayed in Figure 6.10. Router 2 and Router 3 in this figure allow all traffic for each source/destination combination. Router 1, though, has several different permission lists for the possible source/destination combinations. The set of permission lists used by this connector is illustrated in Table 6.1. The ports used in the permission lists reflect the services running on the servers and subnets in the network.

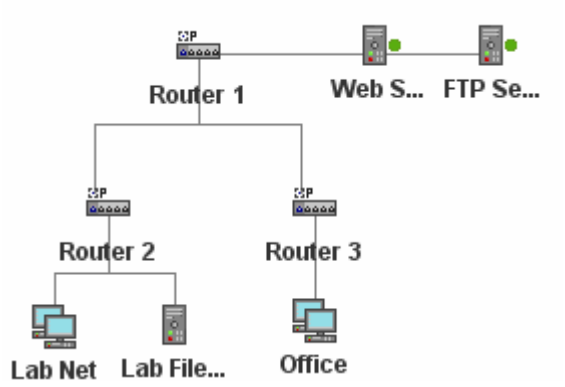

Figure 6.10: Sample Modeled Network with Firewall Permissions

Table 6.2: Permission Lists for Router 1.

Source Device (Node)	Destination Device (Node)	Allowed / Banned	Port List
(Internet)	Web Server	Allowed	TCP: 80, 143, 443
(Internet)	FTP Server	Allowed	TCP: 20, 21, 143
(Internet)	Router 2	Allowed	TCP: 25, 110, 143, 80, 443
(Internet)	Router 3	Allowed	TCP: 25, 110, 143, 80, 443
Web Server	FTP Server	Allowed	(All Ports)
Web Server	Router 2	Allowed	TCP: 53, 80, 135, 139, 443, 445 UDP: 67
Web Server	Router 3	Allowed	TCP: 53, 80, 135, 139, 443, 445 UDP: 67
FTP Server	Web Server	Allowed	(All Ports)
FTP Server	Router 2	Allowed	TCP: 53, 80, 135, 139, 443, 445 UDP: 67
FTP Server	Router 3	Allowed	TCP: 53, 80, 135, 139, 443, 445 UDP: 67
Router 2	Router 3	Allowed	(All Ports)
Router 3	Router 2	Allowed	(All Ports)

6.4 Sensors

Sensors are a necessary component for modeling the generation of IDS alerts, and these IDS sensors are represented as sensor objects in the model. Each sensor object is created as an attribute of either a machine or a connector object, and multiple sensors can exist on a machine or connector (through storing an array of sensor objects). These sensors will essentially monitor what is occurring on a network device (connector or

machine) and compare the activity's "signature" to a database of alert signatures to determine if an IDS alert should be generated. The database of alert signatures is one of the inputs to the model, and a separate database is used for each sensor type.

A generic sensor class provides most of the functionality of the sensor objects. The attributes of the sensor class include a unique ID, the type of the sensor, the output location, an on/off toggle, an array of alerts produced, the available alert database to use, and the network device that the sensor is placed on. This class also provides the methodology for writing all of the generated alerts out to a text file. Two other classes, a network sensor class and host sensor class, inherit from the sensor class to indicate the specific application of the sensor. The network sensor class represents a NIDS, while the host sensor class represents a HIDS.

NIDS refers to a network-based intrusion detection sensor. In a real network, this type of sensor works by monitoring activity in the network traffic. Also, this type of sensor is a physical device that must be somehow attached to a connector in order to function. In some cases, the sensor will be attached just as a machine is typically attached, and all of the network traffic will be sent to the sensor in addition to being sent to the actual destination. Another method is to use a network tap, where one or more specific Ethernet links are interrupted by a device that duplicates the traffic and sends the duplicates to the sensor (Crothers, 2003). The model can be considered to use either of these approaches. A sensor placed on a connector object will effectively monitor all of the traffic that moves through the connector.

HIDS refers to a host-based intrusion detection sensor. This type of sensor exists in a real network as software running on a machine that monitors the machine's activity

logs. This type of sensor is typically not as reliable as a NIDS since the machine running the software could be compromised during an attack, forfeiting the validity of the logs. One of the advantages of a HIDS, though, is that fewer false alerts (false positives) will be generated than with NIDS. Host based sensors accomplish this because only successful actions will be reported in a system log (Crothers, 2003). The model takes this trait into consideration also, and alerts are only generated by a sensor on a machine object if the action triggering the alert is successful.

As briefly mentioned, the model mimics the generation of alerts through signature-based detection. A signature represents a pattern found in one or more network packets that can be matched to a particular type of activity. Intrusion detection sensors focus specifically on malicious activity. The general approach of signature-based detection is to look through network traffic (packets) and cross-reference the findings with a library of signatures to find a match. If a signature match is found, the sensor generates an alert, which is analogous to tripping an alarm. Each brand (or type) of sensor has a unique library of signatures to compare network traffic with (Crothers, 2003). For each default sensor type available, an available alerts object is created to act as a library of signatures and alert messages. More information about the available alerts class will be presented in chapter 7.

Currently, the only sensor type implemented in the model is Snort. Snort IDS sensors are open source, and the organization offers all of the information necessary to establish a separate library of the signatures and associated alert information. This information is included in an alert definitions file that is used as one of the modeling inputs (for the Snort sensors specifically). Additional sensors can easily be added to the

list of available sensor types by providing both a name for the sensor and a file containing the alert definitions to be used by the sensor. The preferences menu of the application provides a means to add (or remove) sensor types.

6.5 Virtual Terrain

The virtual terrain XML document is essentially used to represent a detailed network structure. This type of document can be created and used by the cyber attack simulator presented in this thesis. A virtual terrain document can also be created by a device known as a network scanner that scans a physical network and generates a XML file document depicting the detected network topology and device details. The document is intended to be used by INFERD and other information fusion tools as one of the inputs in determining the progression (and threat) of an attack.

As mentioned previously, the VT interface package provides the functionality for both reading and generating a virtual terrain document. This package relies on the use of a publicly available package known as JDOM to both read and write the XML document. When reading in a XML file, JDOM will parse all of the necessary XML tags and create an object with appropriate elements and attributes. Also, when writing out a XML file, JDOM will create the XML tags and structured formatting when provided with a set of elements and attributes that make up the document.

Since this document is created and used by multiple applications, a common schema is necessary to ensure that each application keeps the XML format consistent. This schema, developed by Argauer (2007), is portrayed in Figure 6.11. The virtual terrain element is considered the root element of the document, and all other elements branch off of this root element.

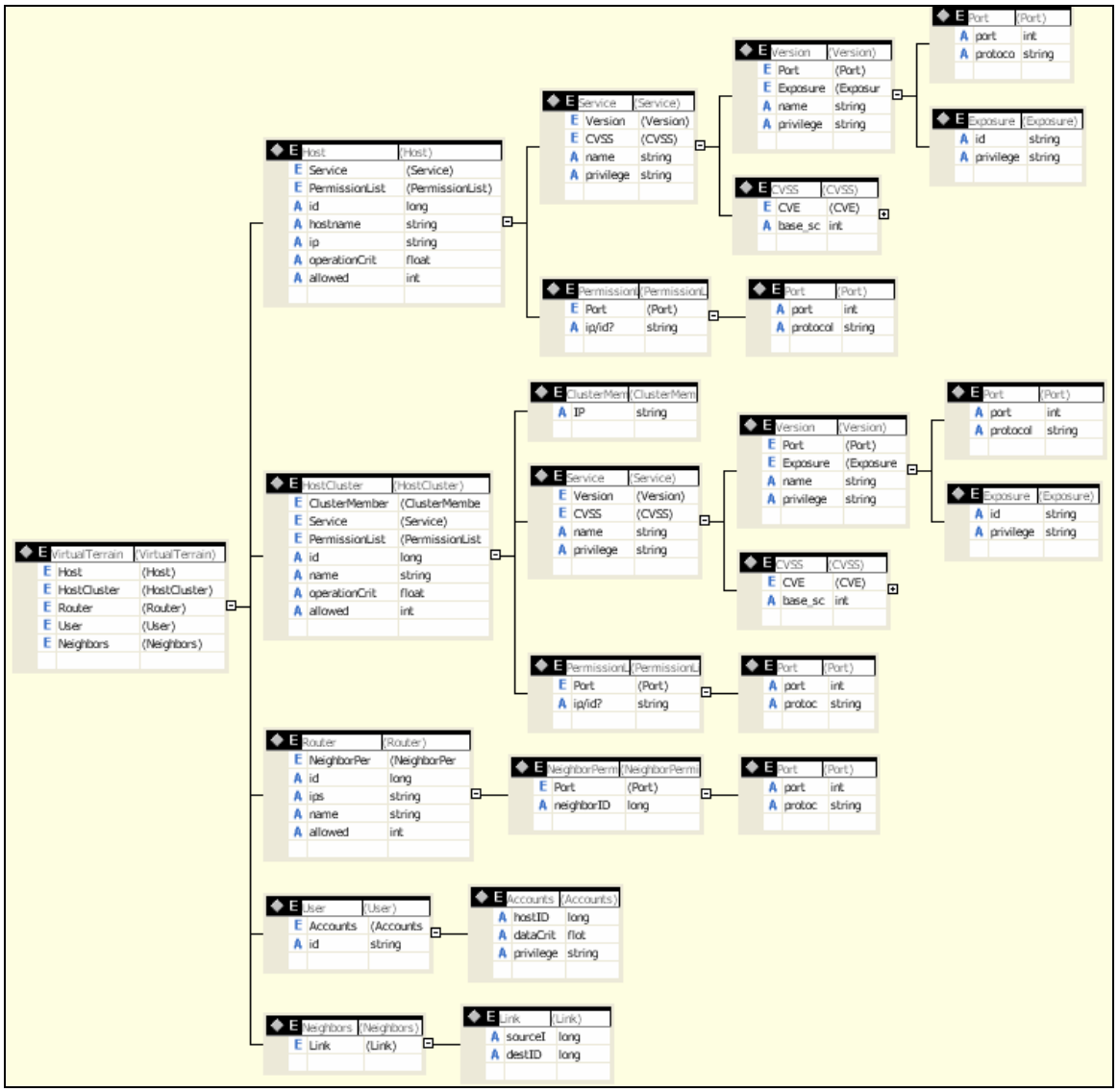

Figure 6.11: Virtual Terrain XML Schema

The VT parser class is responsible for importing and exporting XML documents. One of the advantages of using an XML document is that the model does not need to use all of the information included in the document. Only the elements and attributes associated with desired network details are parsed by the VT parser class when reading in the virtual terrain. For instance, the virtual terrain includes a structure for defining user accounts and privileges within a network. However, the model does not contain or use this information, so it is not considered when parsing a file.

The first step in reading in a virtual terrain document is to create the network object. If a name tag is provided, the network will use that name. Otherwise, the default "virtual terrain" will be used. A connector object is created for each router element defined in the document, and the appropriate firewall (port) permissions are included as connector attributes. Next, machine objects are created for each host and host cluster element defined in the document. Aside from basic attributes, the hosts and host clusters also include service elements (and possibly a list of exploit elements for each service). The machine objects created include each of the listed services as an attribute, even if the service is not one of the defaults available in the simulator. The service objects created will use the ports provided in the XML document for the port attribute. Also, if the XML document defines a set of exploits associated with each service, the service object will store these exploits, by vulnerability ID, as an attribute array that overrides the built in exploits available through the service-to-vulnerability mapping. The sensors included in a network are separate elements that have an ID in reference to a device in the network. The reference ID is synonymous with the device (connector or machine) ID, and a sensor object is created and added to the appropriate connector or machine object. Lastly, the neighbors element of the XML document includes all of the links within the network (by device ID), and the connector and machine (or subnet) objects are updated to contain the appropriate links.

With a fully detailed network object in place, the following task is to draw the network in the canvas. The methodology for drawing the network uses an algorithm that specifies row and column variables that refer to specific increments of space vertically and horizontally in the canvas. Each consecutive level of the network is created in a new

row below any existing rows. Figure 6.12 illustrates a network read in from a virtual terrain document and drawn in the network canvas.

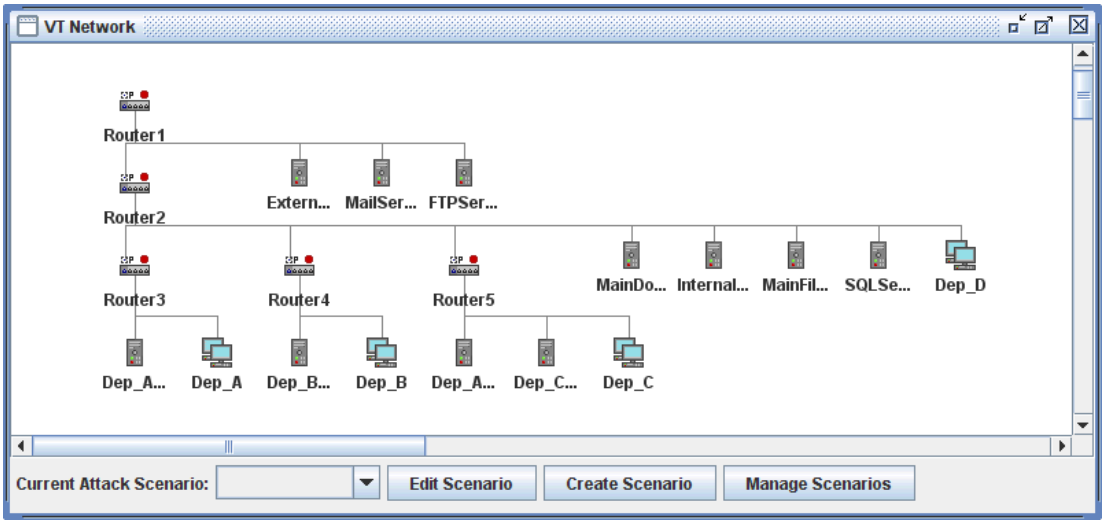

Figure 6.12: Network Model Imported from Virtual Terrain

A network is exported to a virtual terrain document by first creating element and attribute objects that represent the various network objects and attributes. These elements and attributes are used by JDOM to write the XML document. The structure of the document, indicating which elements belong to which other elements, is also defined during the process. When writing the document, the VT parser class goes through all of the devices within the network (connectors and machines) and creates the appropriate elements and attributes for each object. The type of elements and attributes created while exporting the network are the same as those parsed when importing a network from the virtual terrain. A network structure created and exported to an XML document using the simulator can be imported from the XML document to produce the same network structure.

7 Attack Simulation Methodology

After defining a detailed network structure using the components from the network package, the network can be used as a basis for both setting up and simulating a series of cyber attacks. The attack package provides the means for detailed attacks to be specified and simulated in the application. The package also provides a major portion of the functionality required to produce sensor alerts. This chapter discusses the methodology used in providing network attacks and sensor alerts that are representative of real cyber attacks and associated IDS alerts.

Since this thesis focuses on external-based attacks, the traffic entering and moving through the network needs to be effectively modeled. Section 7.1 provides the details and reasoning behind modeling the network traffic. Section 7.2 presents the database of actions (or exploits) that is referenced when generating the network traffic.

The primary means for simulating a set of attacks over a period of time in the modeled network is to establish an attack scenario for that network. Section 7.2 provides the reasoning for using attack scenarios and the details contained in the scenario. Section 7.3 then explains the methodology used for modeling attacks and describes the attributes of attacks.

When an attack scenario is fully defined, the application can then proceed to run a discrete event simulation of the attacks progressing through the network and the sensors generating alerts. This simulation process occurs through the handling of events, which is discussed in section 7.4. The generation of simulation results, such as alert files, is explained in section 7.5. Also, the event simulation process provides the option to

generate an XML document containing the attack details for all of the attacks in a scenario. Section 7.6 provides the details for generating this XML document.

7.1 Modeling Traffic

In a real computer network, the network traffic represents all of the packets moving between devices in the network. Modeling all of the possible network traffic is a tedious process that would also substantially reduce the performance of the simulation. Also, modeling this traffic does not add any additional value to the simulation. Therefore, the model only includes network traffic that is involved in the attack progression or the intrusion detection processes.

The model is concerned with three types of network traffic: traffic associated with an attack step (or action) that matches an alert definition, traffic associated with an attack step that does not match an alert definition, and traffic that is not associated with an attack step but still has an alert definition. The last type refers to noise (or false-positives) that represent perfectly normal activity that can be mistaken as being malicious. Traffic that is neither associated with an attack step nor has an alert definition is not modeled because such traffic will not have an impact on the attacks or the IDS sensors.

The model uses traffic objects to define each traffic entity that moves between network devices. These objects do not, however, represent individual packets. Instead, the objects represent the set of packets required to perform a specific action. Representing the traffic in this manner reduces the amount of information that must be routed between the nodes of the network model. The primary attributes defined in the traffic class include the source machine, target machine, action index, action category, and time created. The

traffic class also includes a set of class variables that indicate the possible primary and secondary categories for actions.

The source machine represents the machine that generates the traffic and indicates the node in the network model that the traffic object is initially located at during the simulation run. The destination machine represents the machine that the traffic's action is intended for. Upon reaching this destination node, the traffic is removed from the simulation. The process used to route the traffic entity between these nodes is described in section 7.4.

The action index is an integer that refers to the action that is being performed by the traffic. This action could be a malicious step within an attack or just common noise. The details of the action are contained in the available actions class and referenced by the action index. Section 7.2 presents the details of the actions that are available in the model and explain how the action index is used. Additionally, the traffic object includes the action category to indicate what broad category the action is associated with.

The possible primary and secondary categories listed in the traffic class variables represent the classification system used for the actions. This classification is accomplished using a set of five primary categories that each has a subset of secondary categories. The category breakdown is displayed in Figure 7.1. Each action performed (and thus each traffic entity) is considered a member of one of the categories listed in this figure.

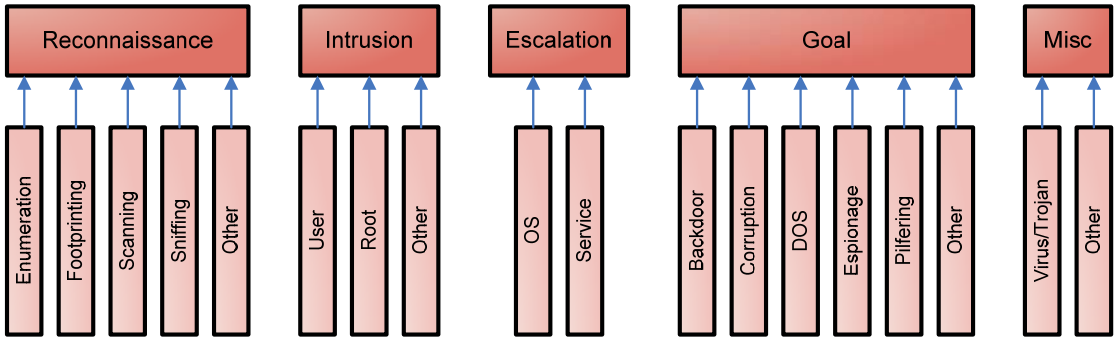

Figure 7.1: Breakdown of Action Categories

Although the traffic class defines a sufficient level of detail for generating network traffic, the traffic class relies on two other subclasses to actually create traffic objects. These two classes are the attack step and noise class, and each class inherits the attributes and methods provided in the traffic class.

The attack step class specifically defines the traffic associated with an individual step of an attack. For an attack step object, the action referenced represents an exploit performed on the destination machine. An attack step object also includes some additional attributes related to an overall attack. An attack attribute indicates the attack object that the step belongs to. Also, a step number attribute indicates where the step occurs in the progression of the attack. Lastly, a success attribute is a Boolean value that indicates whether the attack step was successfully executed.

The noise class defines the traffic associated with typical non-malicious network noise. This type of traffic will still generate IDS alerts, but the action referenced by a noise object will represent a false-positive (where the action did not actually occur or did not occur as part of an attack). The noise class does not define any additional attributes, but the class does provide the methodology for generating random noise traffic.

7.2 Available Actions

One of the key inputs to the model is the database of actions that are available to be used in generating both attack steps and network noise. With respect to attack steps, these actions represent potential exploits that can be used during an attack. With respect to noise, though, these actions represent the potential set of false-positives that can be selected from when generating noise objects.

The available actions class is an abstract class that specifically defines the list of actions that can be used. This class includes attribute arrays to represent the attributes of each action. A unique index, representing the position in an array, is used for each individual action. Using this approach, any of the action's attributes can be retrieved through class methods when providing only the index. The application provides a table in the preferences menu where the default available actions can be viewed. This portion of the preferences menu is displayed in Figure 7.2.

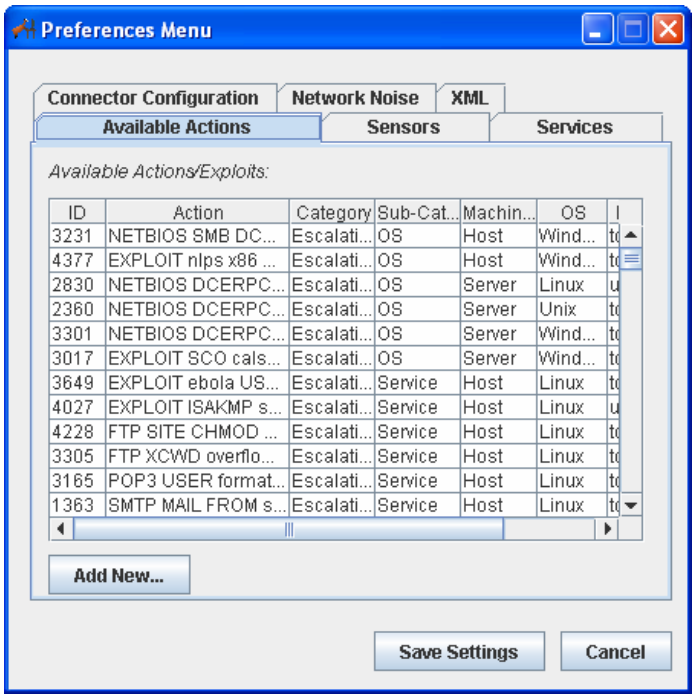

Figure 7.2: Available Actions Displayed in Preferences Menu

The attribute arrays established in the available actions class include the action, primary category, secondary category, machine type, operating system, vulnerability ID, protocol, and port number. These arrays are populated by a text file that lists all of the default actions to be used in the application. This text file is loaded at the start of the application.

The action attribute represents the actually activity performed. This attribute can be considered synonymous with the signature, although the attribute does not represent the specific hexadecimal packet signature. The action attribute is used both as the name of the action and the attribute that IDS sensors look at when comparing network traffic to a database of alert signatures.

The primary and secondary categories classify the action and indicate what type of activity is being performed. This information is used when filtering out the list of potential actions based on the desired category. Also, this information is used when developing the progression of an attack through the use of stages, which is discussed in section 7.4.3 and section 7.4.4.

The machine type, operating system, and vulnerability ID all indicate the type of details necessary on the target machine in order for the action to be successful. For instance, with respect to the machine type and operating system, some actions are only possible on a server or on a machine using Windows 2000. Actions possible on multiple machine types or operating systems must have an entry in the database for each type. The vulnerability ID represents the specific vulnerability that the action can exploit, and this attribute is used when mapping a machine's services to the potential vulnerabilities presented when using the services. Therefore, when provided with a set of services

running on the target machine, a set of potentially successful actions can be retrieved based on the vulnerability IDs.

The protocol and port number attributes indicate an alternative port that the action can use. By default, the action will use one of the service ports of the service that is associated with the action. The success of the action being performed by a traffic object, though, will require that at least one of these service ports is allowed along the entire path that the traffic object takes through the network.

7.3 Attack Scenario

The concept of an attack scenario is used in the model to clearly define what type of activity occurs over the course of a simulation run. This activity includes both threatening attacks and typical network noise. Each modeled network can contain a set of attack scenarios that are defined specifically for the network; however, the simulation only processes one selected scenario at a time. A specific panel is provided in the network frames for managing a network's attack scenarios. This panel is displayed in Figure 7.3. With this panel, a user can select a scenario, create a scenario, edit a scenario, or manage the list of scenarios, which includes removing or copying a scenario).

When creating a new scenario or editing an existing scenario, the user is presented with a form where the scenario attributes can be modified. This form is shown in Figure 7.4. Each attack scenario object has a set of attributes pertaining to the noise generated by the scenario and a set of attributes pertaining to the attacks included in the scenario.

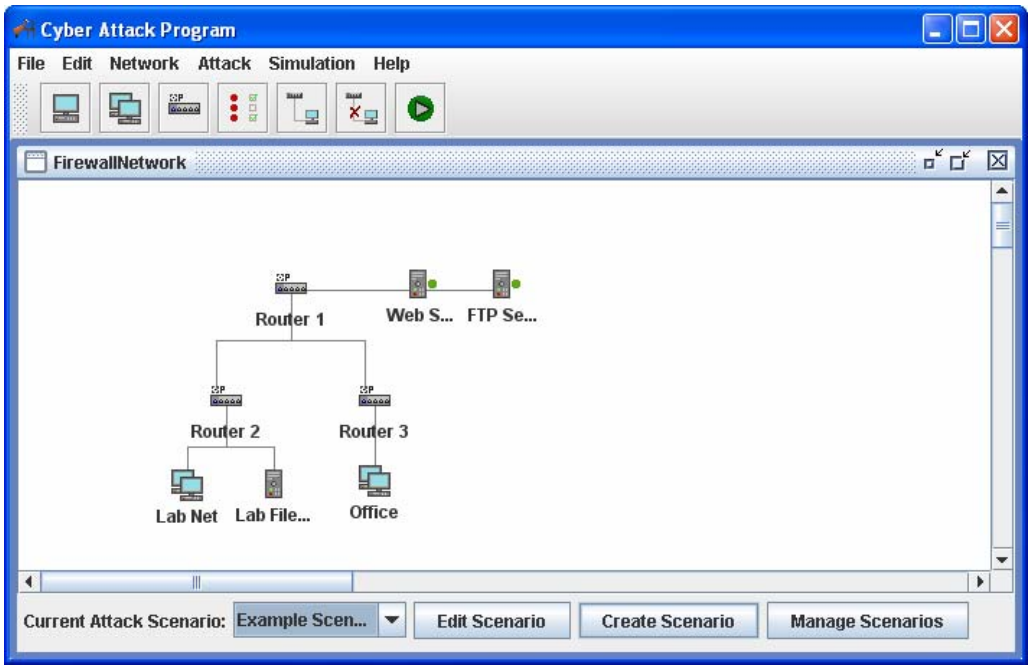

Figure 7.3: Attack Scenario Panel on Network Frame

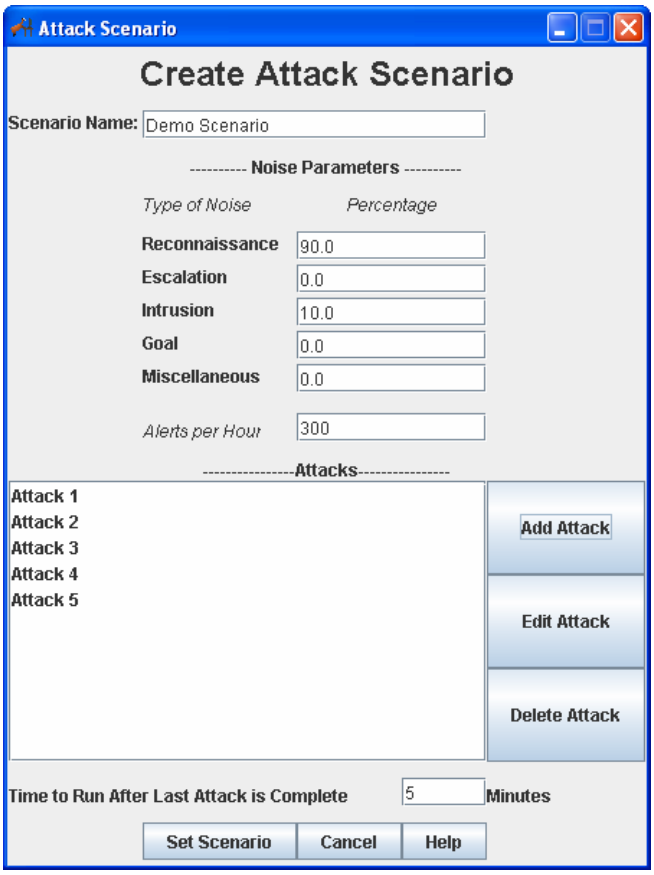

Figure 7.4: Attack Scenario Form

The noise traffic used in an attack scenario is generated during the simulation run. This process substantially reduces the amount of information that needs to be stored in an attack scenario. The generation of this noise is guided by a set of noise parameters, which are attributes of the attack scenario. These parameters include the breakdown (as percentages) of the categories of actions used in the noise and a rate at which the noise is generated. The user can indicate which percentage (out of 100%) each of the five primary action categories will account for in the noise generated. The reconnaissance field provided in the form will auto-adjust so that all of the percentages sum to 100%. The noise rate attribute (referred to as “alerts per hour” in the scenario form) represents approximately how many noise objects will be generated per simulated hour. This rate is exponentially distributed.

The scenario also consists of a set of attacks that will be performed during the simulation. These attacks are defined prior to executing the simulation, and an attack scenario can contain as many attacks as the user desires. The interface provided in the attack scenario form (Figure 7.4) allows for attacks to be added to the scenario, edited, or removed from the scenario. Also, the attack scenario included an attribute representing how long (in simulated time) the simulation should continue to run after the attacks have completed. Without this feature, the last traffic object would always be associated with an attack step. Therefore, providing some additional simulation time allows for more noise to possibly be generated and avoids the bias of having the simulation end with the last attack step.

Another crucial attribute to the attack scenario, though not shown in the form, is the random number generator. The scenario is provided with a rand lib object that is used

to provide all of the random numbers used when initializing the scenario's attacks. If desired, this attribute allows for the same set of random numbers to be used each time the attacks are initialized. This process will be further explained in section 7.4.2.

7.4 Attacks

The attack class is used to define individual (stand-alone) attacks that occur as part of an attack scenario. An attack is considered to be directed at a specific target machine with a certain goal action. The attack also encompasses the set of steps (or individual exploits) performed in order to achieve the final goal action. A simple example attack is portrayed in Figure 7.5. In this attack, the final goal is to install a backdoor on the network's seemingly secure internal file server. The outside hacker first performs reconnaissance actions to scan the servers with direct external (internet) access. The hacker then performs an intrusion step to gain access to the web server. From the web server, the hacker performs an intrusion action and an escalation action in order to gain root access to a machine in the internal subnet. With root access achieved, the attacker is able to intrude into the file server and perform the final backdoor action.

The interface for adding an attack to an attack scenario offers two different methods for specifying an attack. These methods include manually specifying the steps of the attack or automatically generating the steps of the attack. If the user prefers to identify each action used in the progression of an attack, then the manual attack setup can be chosen. However, if the user is only interested in the final goal accomplished by the attack and is not really concerned with the exact steps used to achieve the goal, then the automatic attack setup can be chosen.

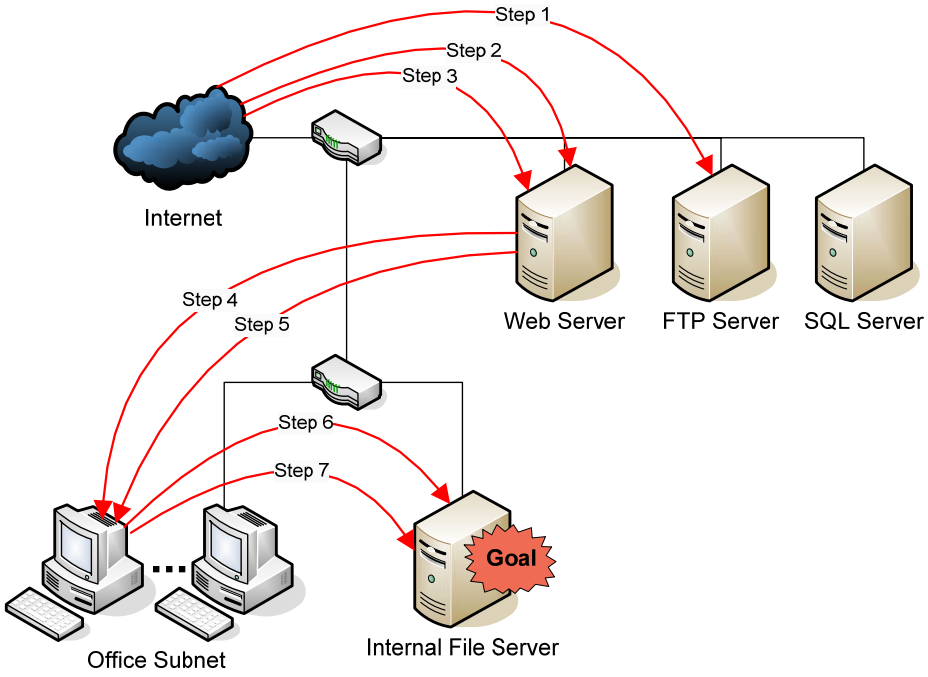

Figure 7.5: Attack Progression Example

The attack class provides the attributes and methods used to create detailed attack objects. Some of the important attributes defined in the attack class include the attack type (manual or automatic), the attack name, the network used, the attack scenario that the attack is included in, the final target machine, the final goal category, the steps in the attack, the start time of the attack, the total time of the attack, an exponential option, an efficiency parameter, a stealth parameter, and a skill parameter. The attack object stores its steps as an array of attack step objects. Also, the attribute for the total time of the attack will only be used if individual step times are not defined. The exponential option is a Boolean attribute that indicate whether the total time of the attack (if defined) is exponentially distributed around a value or uses a constant value. The three parameter attributes (efficiency, stealth, and skill) are used when specifying an automatically generated attack, and these parameters will be discussed in section 7.4.2.

When adding a manual attack to a scenario or when editing a manually defined attack, the user is presented with a form that stores the attack's steps in a table. This manual attack form is displayed in Figure 7.6. The table of steps displays all of the attributes of the attack step objects used in the attack. This form also includes options to indicate the attack name, how the attack's time is determined, and when the attack begins.

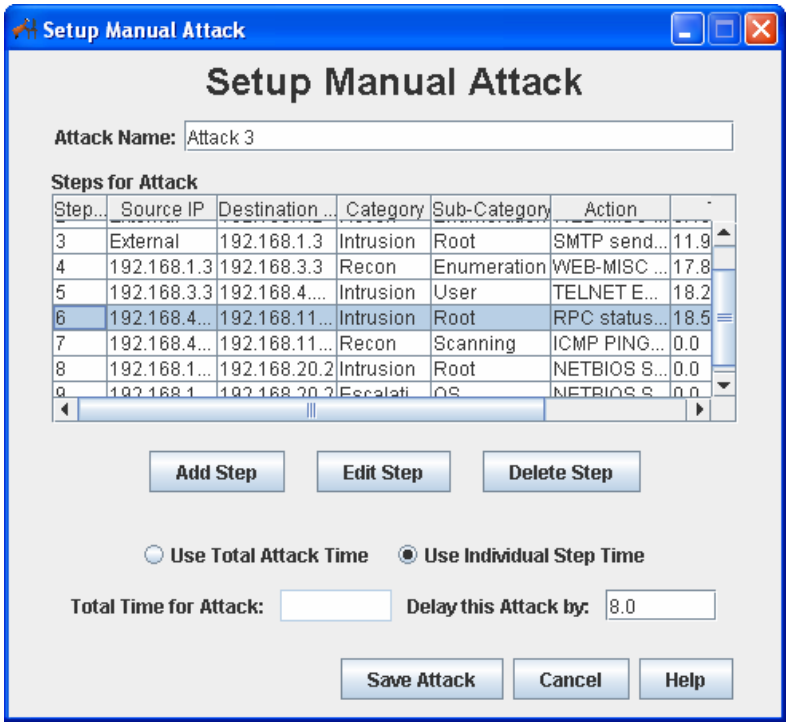

Figure 7.6: Manual Attack Form

Adding a new step or editing an existing step will trigger a new form to be displayed that provides fields for specifying the attributes of the attack step. This attack step form is shown in Figure 7.7. The form essentially allows for the traffic path, the action used, and the time allotted for the attack to be defined.

Figure 7.7: Attack Step Form

In the attack step form, the source IP and target IP fields indicate the IP address of the source and destination machine for the attack step traffic, respectively. Based on the source IP selected, the list of available IP addresses in the target IP field will be filtered to represent only the machines that can be accessed from the machine represented by the source IP. The action is selected by indicating the primary and secondary action categories (the first two fields following the "Action" label) and then choosing from available actions that match these categories. The secondary action category is not required but helps narrow down the list of available actions. If the checkbox for filtering out illogical actions is checked, the target machine selected (by referencing the target IP) acts as another criterion for filtering out the list of actions that can be performed. Section 7.4.1 will explain the methodology for providing only logical actions (specifically, exploits) for the attack steps. Lastly, the time allotted for the step can be specified as a constant or exponentially distributed value. This step time indicates how much simulated time will pass before the next step can be performed.

Setting up an automatically generated attack uses a different form that does not include information about specific steps. The automatic attack form is displayed in Figure

7.8. The form includes the same fields as the manual attack form for setting up the attack name and the time-based information. However, the form includes a set of attack parameters and goal options as a replacement for the individual attack steps. The efficiency, stealth, and skill parameters are values that can be specified with a value between 0 and 1. The meaning of each parameter will be discussed in section 7.4.2, but the basic premise of these parameters is to broadly define the characteristics of the attack steps to be generated. The target field indicates the final target machine of the attack (selected by IP), and the goal type indicates what secondary goal category is to be used for the final action performed in the attack.

Figure 7.8: Automatic Attack Form

All of the parameters defined in the automatic attack form provide enough information for the attack steps of the attack to be automatically generated by the model. Section 7.4.4 will present the methodology used by the model to generate the attack steps from the auto-attack parameters.

7.4.1 Logical Exploits

When specifying the attack steps that occur in an attack, the options available need to reflect what type of activity is actually possible in the modeled network. Features of the network's device, such as services running and firewalls present on connectors, will limit what type of traffic (and, thus, attack steps) can logically occur. Therefore, both the interface for specifying the attack steps and the process for automatically generating attack steps includes the methodology needed to ensure that the actions performed during an attack represent logical exploits.

In the attack step form, this logic is provided by referring to both the attributes of the target machine and the attributes of the connectors that are along the route from the source machine to the target machine. With respect to the target machine, the attributes of interest are the machine type, operating system, and services running. The list of available actions provided in the form will be filtered to include only actions where the machine type matches the target's machine type, the operating system matches the target's operating system, and the vulnerability ID is included in the target machine's mapping of service IDs to vulnerability IDs. With respect to the connectors along the route from the source to target machine, the attribute of interest is the firewall permissions list. The available actions listed in the form will be further filtered to include only actions that will not be blocked by the firewall permissions. This filtering is performed by checking the ports used by the service associated with each action to ensure that at least one of these service-based ports is allowed along the entire route from the source to the target machine. If the associated service does not list any ports (for

example, the “General” service), then the action’s alternate port information, provided through the available actions class, is checked against the firewall permissions.

Although filtering out the illogical actions helps specify attack steps that will be successful, this option can also be disregarded in order to display all actions. The user may also want a specific attack step to include an action that will fail.

The process for generating noise also uses a similar approach for filtering out illogical actions. The target machine of the noise object and the firewall permissions of the connectors along the route from the source to the target machine will provide a means for filtering what actions can be selected during the noise generation process.

7.4.2 Auto-Attack Parameters

The parameters included in specifying an automatic attack are used to guide the generation of attack steps for the attack. These parameters include the efficiency, stealth, and skill, and the parameters are intended to model the behavior of the hacker performing the attack.

The efficiency refers to the directness of the attack. This parameter essentially indicates how well the hacker can progress directly toward the final target. The available values for the parameter are decimal values ranging from 0 to 1, with 0 representing a very inefficient attack and 1 representing a highly efficient, direct attack. Figure 7.9 depicts two different types of attacks in the same network. The attack shown on the left side of the figure represents a highly efficient attack, while the attack shown on the right represents a somewhat inefficient attack. Appropriate efficiency parameter values for these attacks could be 0.9 for the attack on the left and 0.4 for the attack on the right.

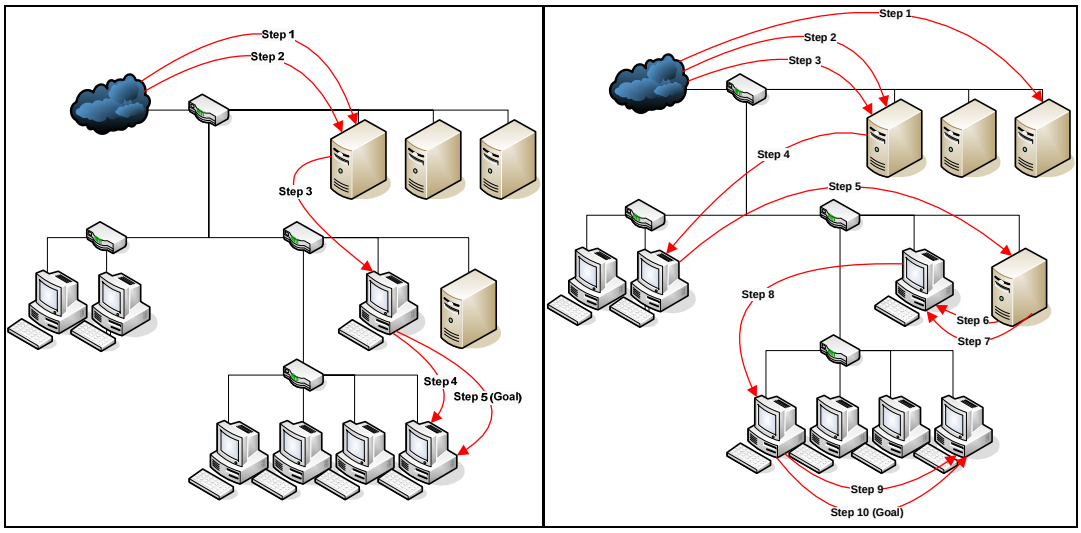

Figure 7.9: Efficiency Example

The efficiency value is referenced during the auto-attack generation process when attempting to progress down through the network toward the target. A random number (between 0 and 1) is sampled from the attack scenario's rand lib object and compared to the efficiency value. If the efficiency is greater than the sampled value, the attack will move to a lower level of the network or to the final target machine if the machine is on the current level. Otherwise, the attack will move to a different machine at the same level.

The stealth refers to how well the attacker avoids intermediate goal steps. Performing many goal steps in addition to the final target goal makes the overall attack much easier for network administrators to notice and for IDSs to detect. Therefore, minimizing the number of additional goal steps helps form a stealthy attack that is better hidden within the network traffic. The stealth parameter also takes on a decimal value between 0 and 1, with 0 representing a very detectable attack that performs goal steps at every machine encountered and 1 representing a stealthy attack that only performs the final goal. The stealth parameter is also checked against a sampled random number at

certain points during the automatic attack generation to determine the category of an attack step.

The skill refers to how successful the hacker is at performing the appropriate steps throughout the progression of an attack. A skillful hacker will have retrieved sufficient information about the network in order to perform actions that will be successful. An unskilled hacker, though, will typically try a random assortment of attacking tools using actions that may or may not succeed based on the details and security of the network. The skill parameter represents the likelihood of the attack failing at some point (where the final goal will not be achieved). The parameter's value can be a decimal number ranging from 0 to1, where 0 represents an attack that will likely fail and 1 represents an attack that will likely succeed. A value of 1, though, does not strictly indicate that the attack will definitely succeed. The network may be configured in such a manner that no exploit will succeed against a particular machine. In this uncommon situation, even a perfectly skilled hacker will not be able to perform a goal step on such a machine. In the automatic attack generation process, the skill parameter (when compared to a sampled value) essentially defines the difference between sampling from only the logical exploits (actions) and sampling from all available actions when choosing an attack step's specific action.

7.4.3 Guidance Template

One of the primary inputs to the automatic attack generation process is the guidance template. The guidance template represents the graph-based template developed by Holdender, Stotz, and Sudit (2005) to indicate what is required in defining the progression of an attack. Refer back to section 4.1 for an explanation of what the guidance template does. A text file is used to store both the stage precedence (for the ten

stages shown in Figure 4.1 in section 4.1) and the action categories available at each stage. The current (default) action categories for each stage are listed in Table 7.1 (indicating both the primary and secondary category).

Table 7.1: Action Categories in Guidance Template Stages

Stage	Action Categories
Stage 0	Intrusion/Other, Reconnaissance/Enumeration, Reconnaissance/Footprinting, Misc/Other
Stage 1	Intrusion/User, Intrusion/Other, Misc/Other
Stage 2	Escalation/Service, Escalation/Other
Stage 3	Intrusion/Root
Stage 4	Goal/Dos, Goal/Backdoor, Goal/Pilfering,
Stage 5	Intrusion/Other, Reconnaissance/Enumeration, Reconnaissance/Footprinting, Misc/Other, Reconnaissance, Scanning
Stage 6	Intrusion/User, Intrusion/Other, Misc/Other
Stage 7	Escalation/Service, Escalation/Other
Stage 8	Intrusion/Root
Stage 9	Goal/Dos, Goal/Backdoor, Goal/Pilfering,

Stage 0 through stage 4 represent the attack progression used to accomplish actions on a machine with external access, while stage 5 through stage 9 represent the attack progression used to accomplish actions on machines in the internal portion of the network. Since these two tasks have many similarities, stages 0 through 4 represent nearly the same action categories as stages 5 through 9, respectively.

The guidance template class acts as an abstract class that stores both the list of action categories per stage along with the precedence required in achieving each stage. Either of these attributes can be retrieved from the guidance template class using the value (index) of the stage. Since the guidance template class is populated through the use of an input file, the model has some flexibility in that the user can alter either the stage precedence or action categories in order to provide different (and possibly more specific) attack progressions.

In performing the automatic attack generation, stage 0 will be used as the first step of the attack since no other stages are accessible until this stage is completed. After accomplishing an action from this stage, stages 1 through 4 are available. Stage 4, though, will only be used if the stealth parameter allows (see section 7.4.4). Once the attack's steps begin targeting an internal machine, the attack generation process will begin using stages 5 through 9. Stage 9, like stage 4, will only be used if the stealth parameter allows; however, the final goal of the attack will also use stage 9.

7.4.4 Auto-Attack Methodology

The attack class includes a set of methods used to automatically generate the attack steps used in the attack. These methods are invoked by the attack's attack scenario class immediately before the simulation run. The attack scenario class will sequentially look through each attack included in the scenario to ensure that the attack's steps have been defined. If an attack does not have a set of steps defined, the automatic attack generation methods will be called. The simulator provides an interface displaying details about this process to the user. This interface is shown in Figure 7.10 (just after the attacks have been initialized). The user has the option to use a common set of random numbers, which will generate the same attacks each time, or a specific random number seed to start with, which will essentially result in different attacks.

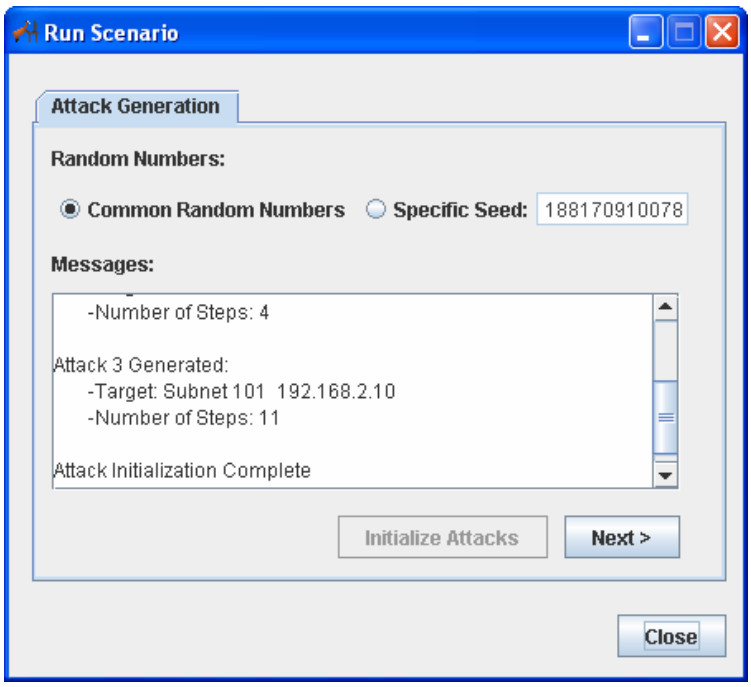

Figure 7.10: Automatic Attack Generation Interface

A few preliminary steps are necessary before the attack steps can be generated. First, the network must be checked to ensure that some point has external (internet) access. Next, the attack scenario's rand lib object resets the random number generation stream with the same initial seed (for common random number) or a specified seed. This stream is only reset once (before the first attack is processed). After the random number stream is reset, the methodology proceeds through each attack individually. The final target machine is located within the network and a critical path is created from the point of external access to this target. This critical path is basically an array of connector objects, and the array is formed by looking at each successive parent connection until a connector is found that contains machines with external access. Figure 7.11 illustrates this process. This critical path is used during the attack generation methodology when an efficient route needs to be selected.

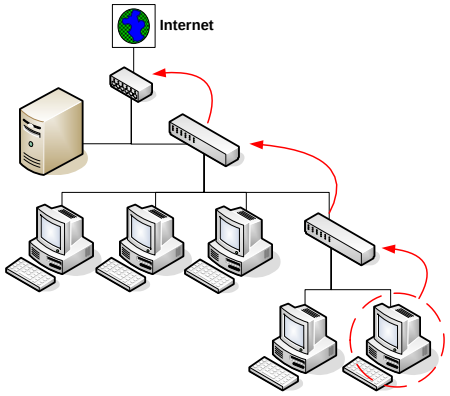

Figure 7.11: Development of Critical Path

The generation of attack steps begins with the external (hacker) location and progresses through the network eventually reaching the final target and goal. Therefore, this process works in the same sequence in which a hacker would perform the attack. Figure 7.12 displays the logic, as a process flow, used in generating both the external and internal attack steps.

The attack generation will begin by locating an initial target that has external access. An appropriate guidance template stage will be selected, and an attack step will be generated using one of the possible action categories for the particular stage. The attack class provides a specific method for generating an individual attack step when providing certain arguments. These arguments include the source and target machine for the step, the stage of the step, the action categories, and the step number. This method will determine what specific set of actions (logical exploits in this case) are available when considering the attributes of the target machine and the firewall permissions on connectors located along the route from the source to the target machine. One of the actions will be selected at random to be used in the attack step. If no actions are available, a different action category (for the same stage) will be selected. If no actions are available for any of the stage's action categories, then a different target machine will be selected.

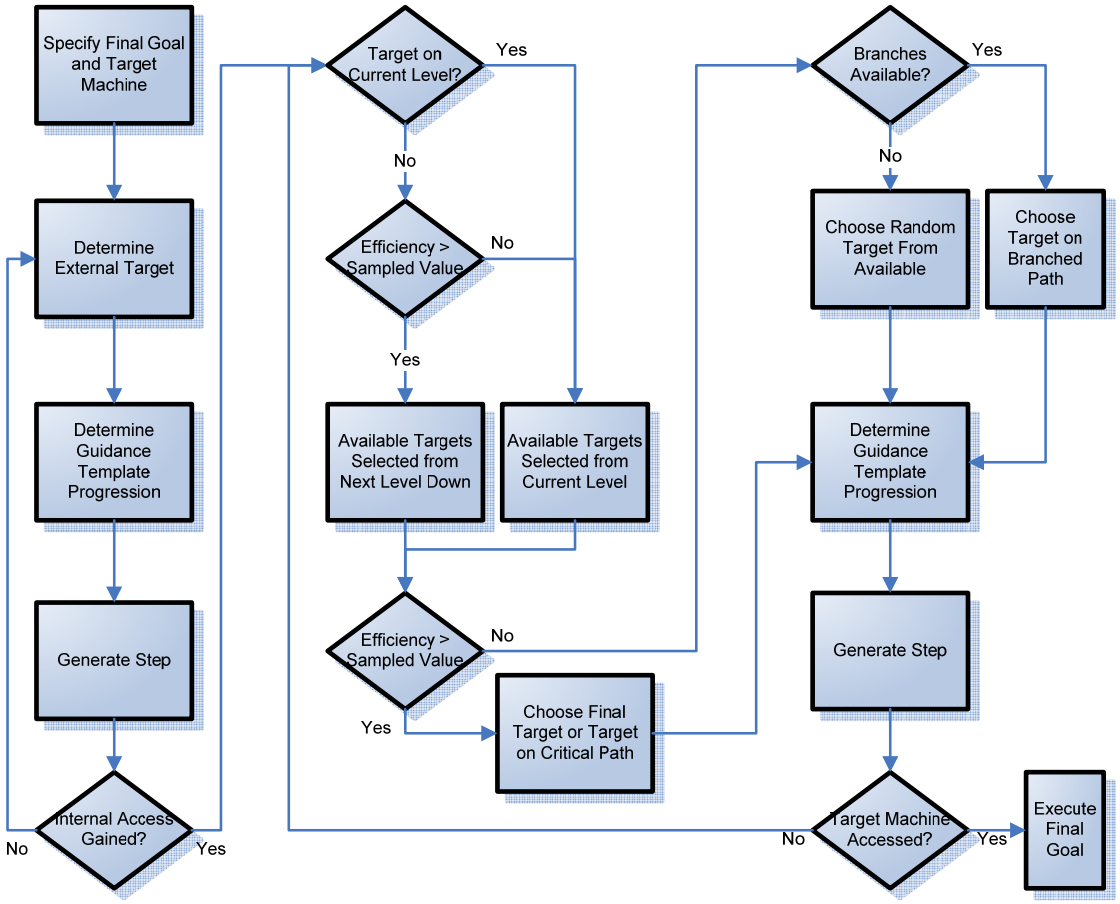

Figure 7.12: Attack Step Generation Logic

Once an attack step is generated where the action successfully gains access to an external machine, then the external portion of the attack is complete. The remainder of the methodology focuses on internal attack steps (where the source machine is a machine in the network). For each internal attack step generated, the logic will first check to determine if the final target is on the same level as the current source machine. If the target is on the same level, then the attack step will remain at that level of the network. Otherwise, the efficiency value is referenced to determine if the attack step will choose a target machine at the next (deeper) level of the network. Also, the efficiency value will be referenced again in determining whether the critical path should be followed or whether the step should branch out by selecting from target machines on a different connector. At

this point, the appropriate guidance template stage is chosen and the attack step is generated (in the same manner as defined in the external step generation). Depending on the value of the stealth parameter, the guidance template stage selected may represent a goal stage (stage 4 or stage 9 in the default guidance template), which will generate an attack step performing an intermediate goal action. Also, the value of the skill parameter will dictate whether a logical exploit is selected as the attack step action or whether a random action will be chosen.

The internal attack steps will continue to be generated in the same fashion until the final target machine becomes the target of an attack step. The next attack step following this step will represent the final attack step of the attack and will include the appropriate goal action.

7.5 Event Handling

The simulation package provided in the simulator is responsible for managing and executing the set of events that make up a simulation run. This section presents more details about the classes included in the simulation package and discusses the methodology used to simulate an attack scenario.

The entity class is devised as an abstract class to represent any sort of transaction moving through the network. In this application, the traffic class inherits from the entity class and represents the only type of entity modeled. The node class is used as a generic representation of network locations, and this class is a super-class of the network hardware class. The node class includes methods that are triggered when entities enter or leave the node. The entity class provides attributes indicating which the node the entity is currently located at, which node the entity was at previously (if any), and which node the

entity will move to next (if any). These attributes are needed when comparing to firewall permissions used by the connector objects. Modeling the traffic as entities moving between nodes in the network provides a more realistic approach in representing network functionality and allows for networking decisions to be de-centralized at the network nodes, where real networking decisions are made.

The task of implementing an attack scenario (through simulation) requires several interactions between classes in both the simulation and attack package. Figure 7.13 displays the interactions between the simulation, event order, node, and entity classes of the simulation package and the attack scenario and traffic classes of the attack package. Each class is displayed in a block along with the key methods used by the class, and the type of interaction between two classes is displayed as a directed arrow.

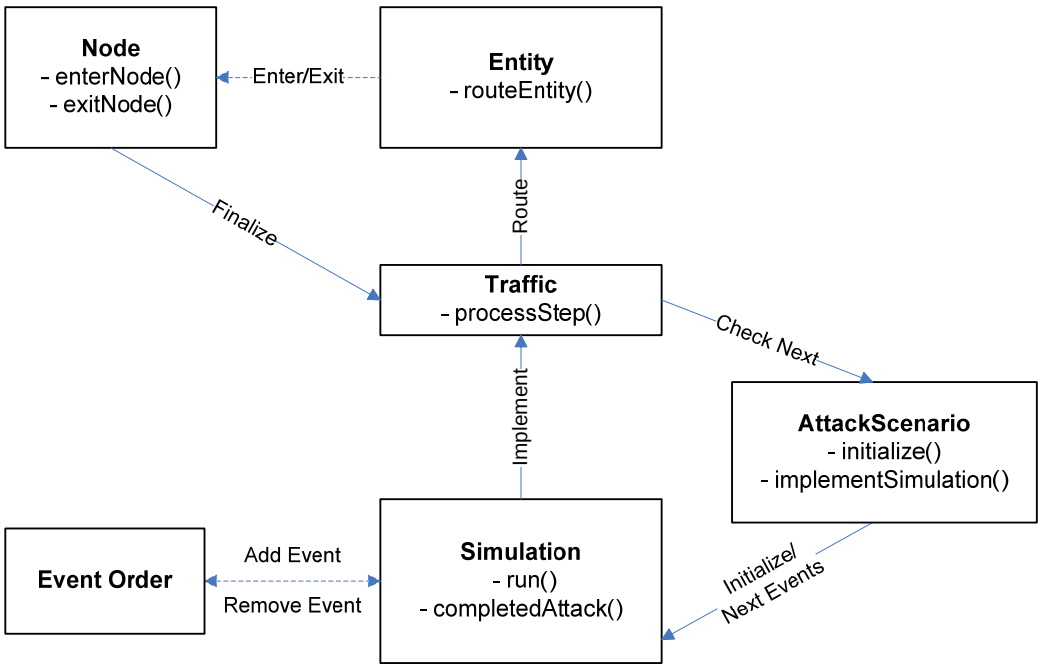

Figure 7.13: Simulation Package Class Interactions

The simulation class is used to run the scenario (a simulation object is created as an attribute of the attack scenario). This class includes a run method that manages the simulation time and processes the events that make up the attack scenario. For each event, a method in the entity class is called to process the event. This method, though, is first overloaded in the traffic class to obtain the details of the traffic. In processing the traffic, the appropriate entry and exit methods of the nodes involved are triggered. This process includes routing the traffic through both machine and connector nodes. Also, if sensors are located on any of the machines or connectors included, alert objects may be generated that are associated with the traffic. For this entire process, both attack step objects and noise objects are casted back into traffic objects to ensure that these different types of network traffic are undistinguishable by the nodes that they pass through. This method ensures that there is no bias in treating the noise objects or the attack step objects since such traffic types would not be directly distinguishable in a real network.

The simulator uses a form of discrete event simulation to execute the attack scenarios. An array of traffic objects, including the attack steps and noise, is used as the event list for the simulation. This array of traffic objects is part of an event order class which contains methods to add and remove events from the array. During the initialization of a simulation run, a new event order object is created which contains an empty array of traffic objects. A noise alert (if required) and the first attack step from each attack in the scenario are placed in the event order array in order of increasing start time. Figure 7.14 presents the modeling logic used to manage the discrete event simulation of an attack scenario.

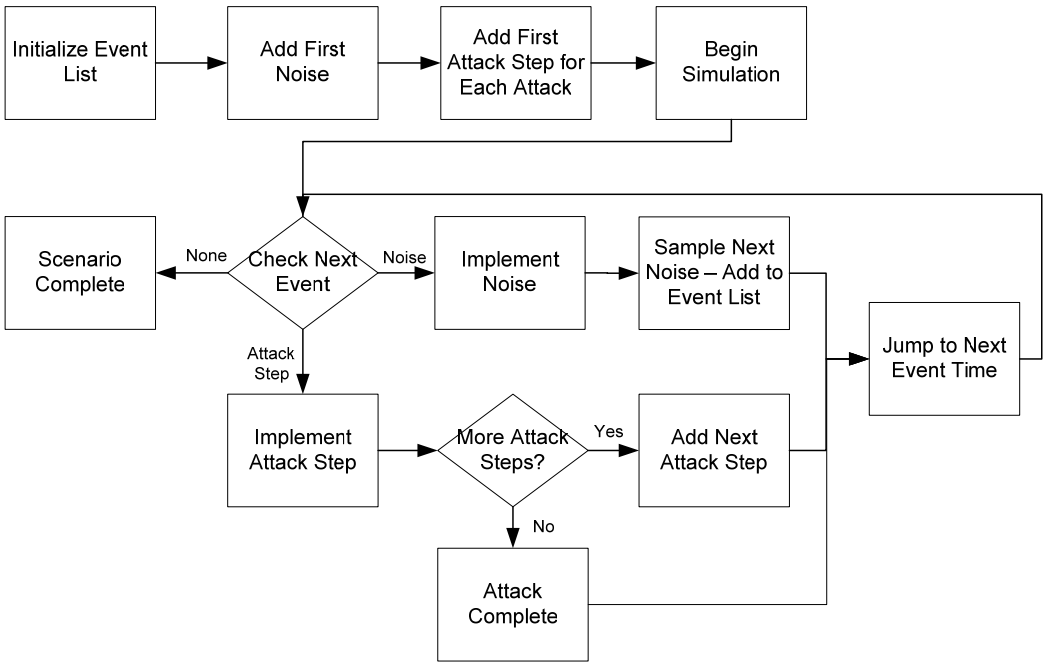

Figure 7.14: Discrete Event Simulation of an Attack Scenario

The simulation is executed by sequentially processing each traffic object (event) contained in the event order array. The simulation time initially starts at 0, and the simulation clock always jumps to the start time of the next event in the event table. When the event is processed, two things may happen: first, the event's traffic object may cause sensors to generate alerts; secondly, the event may cause another event to be added to the event table. For example, if the next event in the event table is an attack step, that step may cause sensors to generate alerts. In addition, the completion of that attack step allows the next step in the attack to be added to the event array. Events are processed in the event array until there are no events left, which indicates that the simulation of the attack scenario is complete.

When running an attack scenario, the user is presented with an interface displaying the attack-based events accomplished during the simulation run. Figure 7.15 shows this interface. In a similar manner as with the automatic attack generation, the user

can specify whether to use a common set of random numbers or a specific random number seed. The random numbers are used in the event simulation when generating noise objects and when identifying the specific start time of events added to the list of events (primarily for traffic that uses an exponentially distributed start time).

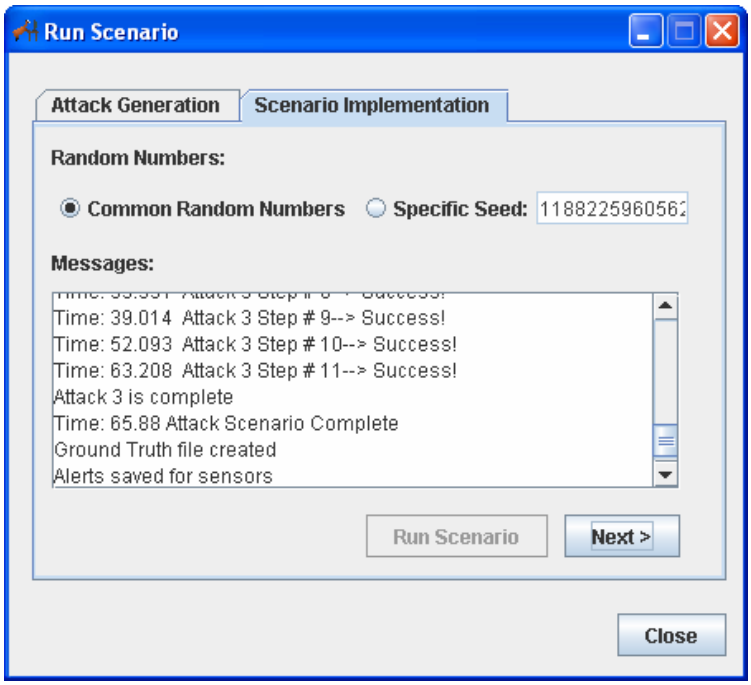

Figure 7.15: Event Simulation Interface

7.6 Scenario Results

Upon completion of the attack scenario implementation, two primary outputs are provided. These outputs include the scenario's ground truth and the sensor alerts. When these outputs are generated, the user is presented with an interface that allows the files associated with the outputs to be viewed. Figure 7.16 displays this interface, including a listing of one output file representing the ground truth and four output files representing the sensor alerts.

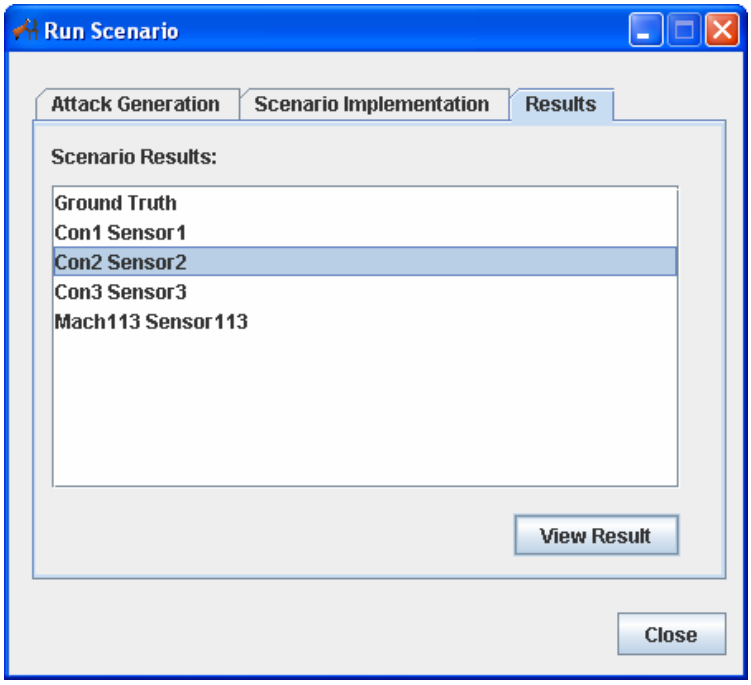

Figure 7.16: Scenario Results Interface

The ground truth output represents what malicious attack-based activity actually occurs during the scenario implementation. This information basically corresponds with what attack steps are generated and processed by the model. Having this information available allows for both the placement of the sensors and the effectiveness of any information fusion tool used to be evaluated. An example ground truth output file is displayed in Figure 7.17. In this file, the attack steps are presented in order of occurrence. The attributes listed include the attack that the step belonged to, the step number in the attack, the action categories, the action name (or message), the path of the step (indicating the source and destination IP), and an indication of whether the step was a success or failure.

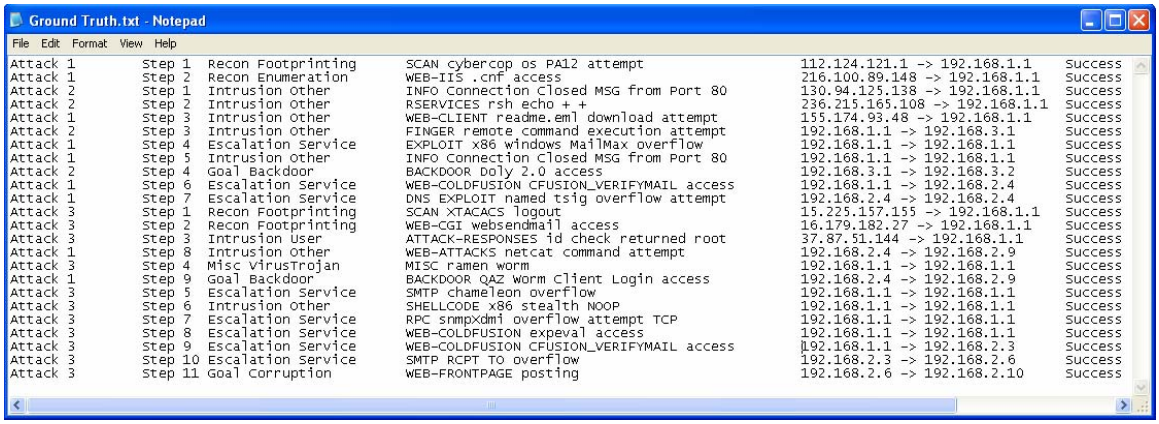

Figure 7.17: Example Ground Truth File

The sensor alert output represents all of the alerts generated by the sensors during the scenario implementation. This output includes alerts generated as a result of both attack-based activity and noise activity. Individual alert objects are used to represent each specific alert generated during the simulation run. These alert objects are created by a network device's sensor when a traffic object (entity) entering the node of that device has an action signature that matches an alert signature in the sensor's database of alert signatures. The alert object is added to an array of alert objects stored as an attribute of the sensor. This array is accessed at the end of the simulation when producing the output files. Also, classes in the sensor management package, if the package is used, will access this array throughout the simulation.

The output files for sensor alerts consist of one file for each sensor used in the network. Each file contains a listing of alerts in the format used by the respective sensor type. Figure 7.18 displays a set of alerts generated by a snort network-based intrusion detection sensor. The alert messages are formatted in the same manner that real snort alert messages are formatted. This message includes the time of detection, the port used, the action/alert message, the classification of the alert, the ranked priority of the alert, the

protocol used, and the source and destination IP address. The classification and priority are used in determining the severity of a particular alert.

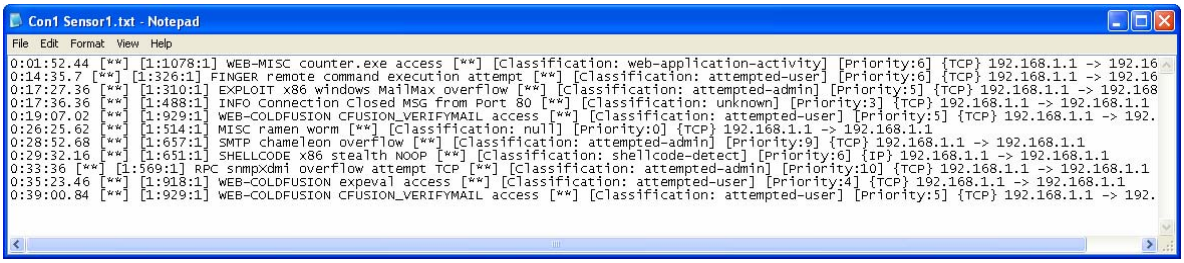

Figure 7.18: Example Sensor Alert File

The various sensor alert files generated are intended to represent the alerts of an actual intrusion detection system. Therefore, the alerts produced provide no indication of whether the alert was associated with an attack or just common network noise. The purpose of these alert files is to supply information fusion tools with alert input that is representative of real network alerts. These tools can then be evaluated using the simulated alerts instead of requiring actual alert data to be obtained. Chapter 8 will explain how this process is accomplished.

7.7 Exporting Attack Scenarios

Another potential output that the simulator can provide is a formatted XML document depicting the attacks included in an attack scenario. An option is provided (in the application's preferences menu) to output such an XML document after the attack generation process. This document includes relevant information about all of the attacks in the scenario along with the full set of attack steps used in the attack. Figure 7.19 shows the XML schema used for this XML document of attack information.

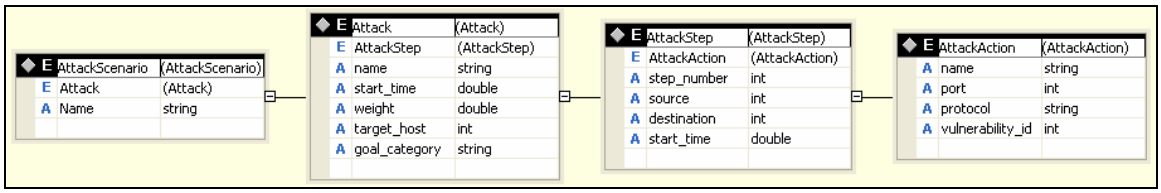

Figure 7.19: Attack Scenario XML Schema

The process for generating this document is similar to the process used for generating the virtual terrain XML document. The associated attack scenario object, attack objects, and attack step objects are all used to generate the elements and attributes included in generating the XML document. Also, this XML document can be imported back into the network model (for the same network) to create an attack scenario with the same attacks and attack steps. This process also provides a means for other applications to set up a set of attacks to be generated in a specific network that is either modeled in the simulator or stored in a virtual terrain document.

8 Evaluation of Simulation Model

The final stage in providing a cyber attack simulation application is to evaluate how well the application performs the functions intended. This chapter will provide a formal verification and validation of the network and attack components used in the model. A discussion of the model's capabilities, limitations, and applications to other related areas of research is also provided.

Both the network model and the attack simulation process need to be properly assessed to ensure that they function according to the methodologies presented in chapters 6 and 7. Section 8.1 discusses the techniques and experiments used to verify that the network modeling features of the application properly define a detailed virtual network. Similarly, section 8.2 discusses the techniques and experiments used to verify that the attack generation process follows the guidance template and automatic attack methodology and that the event simulation actually routes and processes entity events appropriately through the modeled network.

Section 8.3 discusses the process for validating both the attacks and the alerts generated using the model. The value of the output produced by this model relies strongly on the attacks representing real cyber attacks in a network and the IDS alerts representing real sensor alerts produced as a result of both attacks and network noise.

Following the discussion regarding verifying and validating the model, section 8.4 presents the overall capabilities of the model. These capabilities include what the model can be used for as well as what the model can accept as input. Even though the model has numerous capabilities, the model also has several limitations that can be seen as both a barrier to providing more detailed and complex modeling and an indication of what

features should be added to improve the model. Section 8.5 discusses these modeling limitations.

Lastly, the model alone does not provide a significant amount of value to solving problems in the cyber realm. Integration of this application with other applications, such as information fusion tools, is necessary to make notable accomplishments in both modeling and identifying cyber attacks. Section 8.6 indicates some of the other applications that this cyber attack simulator is used with or is intended to be used with.

8.1 Verification of Network Model

The process of verifying the network modeling features involves demonstrating that the results of using such features are the results that one would expect to see. The modeled networks created using the simulator have several features that are directly accessible to the user and several features within the classes that are indirectly used by the model. The modeling features built into the class methods and indirectly accessed are be verified through logical techniques, which are explained in section 8.1.1. The features directly accessible to the user are verified through visually techniques as explained in section 8.1.2.

The use of a specific network structure is helpful in performing both types of verification. Figure 8.1 illustrates an example network that will be used to both logically and visually demonstrate that the network model performs functions as intended. This network includes two external-access servers connected to Router 1, a subnet of 100 machines connected to Router 2, and an administrator machine and internal file server connected to Router 3.

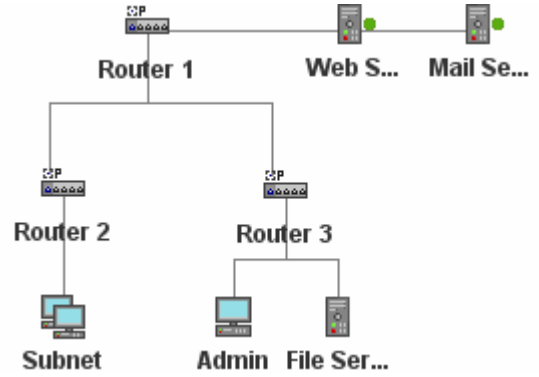

Figure 8.1: Example Network Model for Verification

8.1.1 Logically Verified Features

Using a logical approach to verify that the model functions as designed involves the implementation of conditional statements that will output either a specific attribute or a determined value. Such statements are often temporarily used just for the purpose of verification. Some of the model features that are logically verified include the parent-child connector setup, the implementation of subnets, the services stored on a machine object, and the permission list stored on a connector object.

The parent and child relationships between connectors are necessary in determining what "level" of the network a machine is on. Although a simple indication of a connector's child and parent connections can be verified visually, the actual array values need to be checked to ensure that no redundancy or "ghost" connections exist. A ghost connection represents a connection that had existed at some point and was either changed or removed. To fully test the handling of connections, the network displayed in Figure 8.1 is modified by removing Router 3 and adding Router 4 as a child of Router 2. The machines initially connected with Router 3 are connected to Router 4. This new setup is displayed in Figure 8.2.

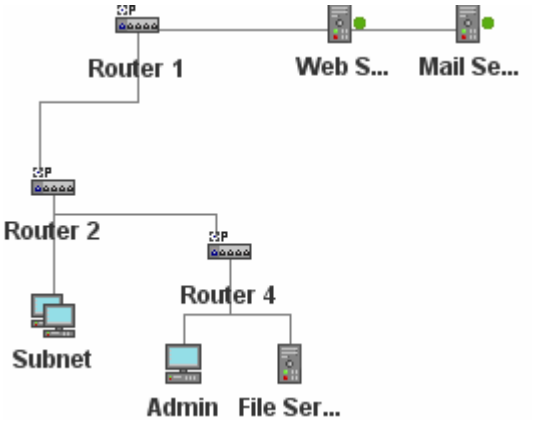

Figure 8.2: Modified Network Connections

The proper handling of these changes is verified by outputting (essentially, displaying string values) that indicate what is contained in the physical array. This output is provided by a statement added to the model code (when viewing the connector's form) to loop through the items in the child connector arrays and produce a string for each item. This same process is used to display the array of machines that are connected to the connector as well. For Router 1 in the network, the statement indicates that Router 2 is the only child connection and that the Web Server and Mail Server are the machines connected. For Router 2 in the network, the statement indicates that Router 4 is the only child connection and that 100 Subnet machines are connected to Router 2. Lastly, Router 4 has no child connections and has the Admin and File Server as connected machines. Also, both of these machines indicate Router 4 as the parent connection. Therefore, this experiment verifies that the process for handling parent and child connections works as intended.

The use of subnets in the model is an important mechanism to group together a set of machines and provide the ability to specify and edit the machine details collectively instead of one-by-one. Since the machines are still represented using individual machine

objects, though, the process of specifying and editing the details of all machine objects included in the subnet needs to be verified. The association of a set of machine objects with a subnet is implemented using a machine object's group ID attribute. The use of this attribute is verified by including a statement in the model code that tallies the number of machine objects matching a particular group ID. For the model displayed in Figure 8.1, the Subnet contains 100 machines, and 100 different machine objects are determined to have a group ID with a value of 103 (which is the ID corresponding with the subnet). Next, the consistency of the machine attributes across all machines represented in the subnet is verified by changing some of the subnet details in the edit subnet form and outputting the attributes of the machine objects through another statement in the model code. The Subnet in Figure 8.1 is changed to contain the details shown in Table 8.1. Each machine object associated with the subnet is verified as having these attributes through observing the output of the added statement.

A key feature of the machine objects is the set of services included as an object attribute. The form for specifying and editing the details of a machine allows for certain services to be selected; however, the service attribute (an array) can also include services not presented to the user. For instance, a Windows-based machine (such as Windows XP or Windows 2000) will include a Windows service. Consider the Web Server and the File Server machines displayed in Figure 8.1. The Web Server uses the Windows XP operating system and is specified as using Web, ICMP, and FTP services. The File Server uses a Linux operating system and is specified as using the TFTP, MySQL, Oracle, and ICMP services. An additional statement is included in the model to retrieve and display the array of services stored as attributes for both machines. This array of services is listed

in Table 8.1. In addition to the services specified, the Web Server includes a Windows service, the File Server includes a Linux Service, and both machines include a Generic service. This list indicates that the set of both desired and default services are properly allocated to a machine object.

Table 8.1: Services Included on Web Server and File Server

Web Server services	File Server services
Generic	Generic
Web	TFTP
ICMP	MySQL
FTP	Oracle
Windows	ICMP
	Linux

A key feature of connector objects is the firewall permission list stored as an attribute. This list indicates the port requirements for each possible path through the connector. Although the set of ports for particular paths can be verified through the forms provided in the application, the entire permissions list needs to be checked to ensure that redundancies and bogus connection paths are not stored within this list. This type of check is performed by including another statement in the model code (associated with editing the connector) that loops through each entry in the permission list attribute to display the connection path and set of ports. The connector used to demonstrate this process is Router 3 from Figure 8.1. This connector is specified with the connector path permissions displayed in Table 8.2. The statement added to the model verifies that these are the only two entries in the firewall permission list. Another connection path is possible from the Admin machine to the File Server machine (and vise versa), but this connection is not restricted in any way and does not require a listing of ports.

Table 8.2: Firewall Permission Specified for Router 2

Source	Destination	Allowed/Banned	Ports
Router 1	Admin	Allowed	TCP: 80, 125, 1234
Router 1	File Server	Allowed	TCP: 69, 118, 156, 1234, 1521

8.1.2 Visually Verified Features

Using visual aspects of the modeling process to verify the model features includes both the display of attribute values and the graphical representation of model objects and features through the interface provided. The model features that are visually verified include the connection graphics, IP addresses, network saving process, and virtual terrain exportation.

The connection graphics include both the blocks used to represent network objects and the lines that display the connections between these objects in the drawing canvas. The use of these graphics is verified by observing the appropriate connection-related attributes established for the objects (displayed in the forms available). For example, when adding a connector (referred to as Router A) and machine (referred to as Host A) to a network model, a line can be created in the canvas representing a connection between Router A and Host B. This effectively sets Router A as the parent connection of Host B. When adding another connector (Router B) to the model with the initial attribute of having Router A as the parent connection, a line will be created in the canvas that represents this connection. This feature is displayed in Figure 8.3. The creation of any additional lines connected to the machine is prohibited since only one connection is allowed. Also, if another connector is added to the model, the creation of lines that cause a looping connection between the three connectors is prohibited to maintain the intended tree structure of network topologies.

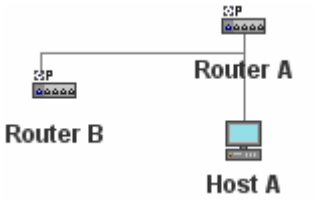

Figure 8.3: Connection Graphics Example

Each of the IP addresses used as machine attributes needs to be uniquely defined in the network model. When adding a machine to a network model, the IP address is determined based on the connector that the machine is linked to and the next available value for the fourth IP component. This process is verified using the network models shown in Figure 8.1 and Figure 8.2. For example, when the Admin machine is added to Router 3 in Figure 8.1, the IP address of the machine becomes 192.168.3.1. The first two components are specific to the network, while the third component represents the connector ID (which is 3 for Router 3) and the next available value when considering the other components (which is 1). When adding the File Server machine to the Router 3, the IP address of that machine becomes 192.168.3.2. Similarly, when the Admin and File Server machines are modified to be connected with Router 4 in Figure 8.2, the IP addresses become 192.168.4.1 and 192.168.4.2, respectively. Additionally, when a subnet of 10 machines is added to Router 4, the IP addresses for machines in this subnet range from 192.168.4.3 through 192.168.4.12. These steps verify that the IP address is created from the appropriate model criteria.

Saving the network is an important model feature that allows users to either view the model or continue working on developing the model at a later point. In saving the network model, though, all of the appropriate network details need to be stored and retrieved properly for the model to be accurately recreated when loading the associated

save file. This process is simply verified with the network model displayed in Figure 8.1 by identifying all of the details associated with the network, saving the network model, reloading the model from the save file, and then observing the same details again. Performing this task recreates the network with the same details as before, which include the same objects, connections, and attributes. Thus, this process verifies the saving operation provided.

Similar to saving a network, exporting the network to a virtual terrain XML document also allows for the network model to be viewed and further developed at a later point in time. In addition, this virtual terrain document allows for other applications to generate models for the simulator and to use models generated by the simulator. The process of exporting a modeled network to a virtual terrain document and importing a modeled network from a virtual terrain document is verified using the same example network shown in Figure 8.1. After first exporting and then importing the network's virtual terrain, the network model created is displayed in Figure 8.4. Although the positions of the block components are different, the respective object attributes are the same as before. This indicates that the virtual terrain is both properly exported and imported.

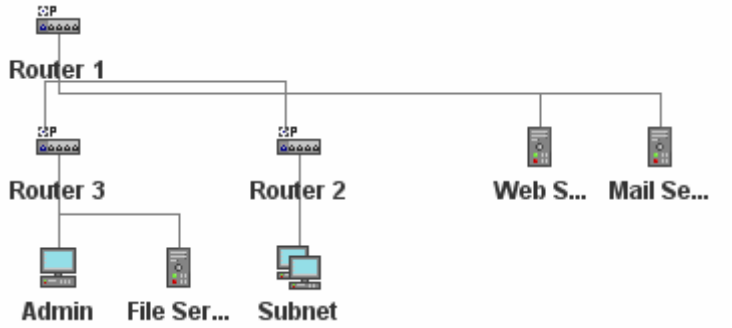

Figure 8.4: Network Model Imported from Virtual Terrain

8.2 Verification of Attack Simulation

Verifying that the attacks are both produced and simulated as intended is necessary to ensure that results produced from a simulation run represent what is designed to be produced based on the input provided by the user. This verification is performed using six different experiments. The first experiment is designed to ensure that the available set of exploits is filtered properly based on the attributes of the machines and connectors involved in performing the exploit. This process is discussed in section 8.2.1. The second experiment traces the stages that an automatically generated attack progresses through to verify that the guidance template is correctly followed. Section 8.2.2 provides a discussion of the experiment that verifies the use of the guidance template. The third experiment evaluates the effects of the three automatic attack parameters to verify that these parameters are used appropriately by the attack generation process. This experiment is presented in section 8.2.3. The fourth experiment verifies that entities are appropriately routed between the intended nodes of the network model, and this experiment is described in section 8.2.4. The sixth experiment verifies that noise is appropriately generated, and this experiment id described in section 8.2.5. The final experiment compares the set of alerts generated by the sensors to the network traffic processed during a simulation run to verify that the proper set of alerts are being generated for specific sensors. Section 8.2.6 provides a discussion of this alert verification process.

These different experiments all utilize the same network model. This network model is developed using the model from Figure 8.1 as a basis, and this new model is displayed in Figure 8.5. This model includes an additional network level as well as IDS

sensors placed on Router 1, Router 3, Router 4, and the File Server. For ease of clarifying the attack step targets used in the experiments, the IP address or IP range of each machine in this network model is listed in Table 8.3

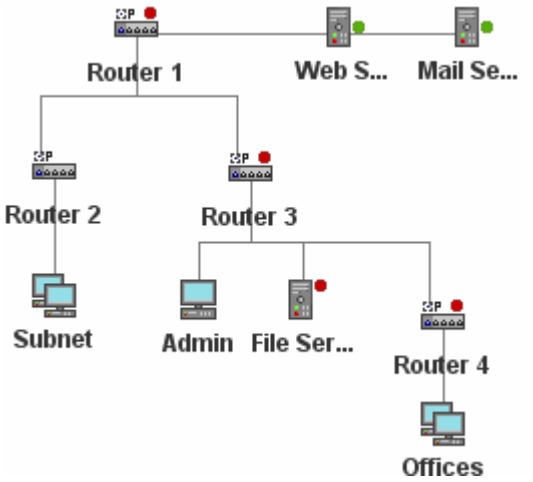

Figure 8.5: Example Network for Attack Verification

Table 8.3: IP Addresses Used in Example Network

Machine	IP Address or Range
Web Server	192.168.1.1
Mail Server	192.168.1.2
Subnet	192.168.2.1 – 192.168.2.100
Admin	192.168.3.1
File Server	192.168.3.2
Offices	192.168.4.1 – 192.168.4.20

8.2.1 Exploit Filtering

When setting up a manual attack step, filtering out illogical exploits is a very beneficial feature. The results of this process must include only exploits that are possible based on the attributes of the target machine and traffic path of the attack step. Verifying that the correct exploits are provided involves checking the details of each exploit against appropriate machine and connector attributes. An experiment is setup using the network model displayed in Figure 8.5 to verify the exploit selection process.

The experiment consists of setting up a manual attack step from the Web Server (with an IP of 192.168.1.1) to the File Server (with an IP of 192.168.3.2) and observing both the total set of exploits listed and the logical set of exploits listed as possible actions for the attack. The details of the attack step are displayed in Figure 8.6. For the Reconnaissance/Scanning action category, a fairly large list of potential exploits is presented. However, when choosing the option to filter out illogical exploits, only eight specific exploits are presented. These eight exploits correspond with the details of the target machine. For example, the operating system required is Linux, the machine type required is a server, and the vulnerability IDs of the exploits are associated with the ICMP service. These three attributes match the attributes of the File Server machine object. Also, the port associated with the ICMP service is allowed by the firewall permissions along the path from the Web Server to the File Server. Therefore, the exploit filtering performs as desired.

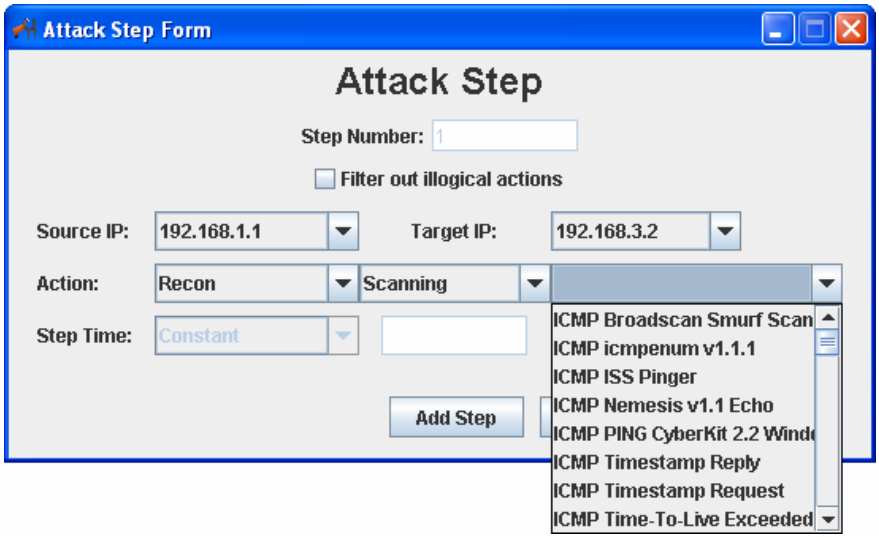

Figure 8.6: Attack Step Details

8.2.2 Stage Progression

The process for automatically generating attack steps for an attack relies heavily on the proper use of the guidance template. The guidance template limits controls what stages need to precede which other stages, and the attacks generated using the simulator need to demonstrate this stage precedence. An experiment was setup to test the implementation of the guidance template using the network model displayed in Figure 8.5.

The stage progression experiment uses an attack scenario with nine separate automatic attacks that have various efficiency values, stealth values, target machines, and goal categories. The characteristics of each attack are listed in Table 8.4. These characteristics are varied to account for the different complexities and situations that the automatic attack generation process can encounter. The attacks are generated and processed during the simulation run, and the resulting ground truth output is observed to verify that the stage progression is appropriate.

Table 8.4: Attack Details for Stage Progression Experiment

Name	Efficiency	Stealth	Skill	Target	Goal Category
Attack 1	0.2	0.9	1.0	192.168.2.59	Espionage
Attack 2	0.5	0.9	1.0	192.168.4.16	DOS
Attack 3	0.9	0.9	1.0	192.168.3.1	Corruption
Attack 4	0.2	0.5	1.0	192.168.2.24	Pilfering
Attack 5	0.5	0.5	1.0	192.168.1.1	Backdoor
Attack 6	0.9	0.5	1.0	192.168.3.2	Backdoor
Attack 7	0.2	0.2	1.0	192.168.7.16	Espionage
Attack 8	0.5	0.2	1.0	192.168.2.1	Corruption
Attack 9	0.9	0.2	1.0	192.168.4.11	DOS

The ground truth indicates that the action categories and respective stages used in each attack follow the correct guidance template progression. For example, Attack 2 consists of twelve actions that occur in a sequence that complies with the stage

precedence defined in the guidance template class, and the actions are displayed in Table 8.5. This attack begins with a footprinting action (stage 0) and then an intrusion/other action targeting the Web Server from an external location. With access gained to the web server, the attack continues with misc/virus-trojan and reconnaissance/enumeration actions that perform additional malicious activity against the Web Server and install worms. These actions account for stage 5 of the guidance template progression. Next, an intrusion/root (stage 6) action is performed against a machine in the Subnet group to gain access to a Subnet machine. From this point, a reconnaissance/enumeration (stage 5) action and escalation/service (stage 7) action are performed directed at the Admin machine and gaining access to this machine. An escalation/service action is performed again (internally) to gain additional access, and then another escalation/service action is performed against a machine in the Offices group. Access is then gained to the attack's final target machine (192.168.4.16) with another escalation/service action. Finally, the attack's final goal is accomplished using a goal/dos (stage 9) action.

Table 8.5: Attacks Steps Generated for Attack 2

Step	Action Category	Action	Path
Step 1	Recon Footprinting	RPC portmap sadmind request UDP	209.153.125.235 -> 192.168.1.1
Step 2	Intrusion Other	SHELLCODE x86 stealth NOOP	212.97.19.144 -> 192.168.1.1
Step 3	Misc VirusTrojan	MISC ramen worm	192.168.1.1 -> 192.168.1.1
Step 4	Recon Enumeration	FINGER account enumeration attempt	192.168.1.1 -> 192.168.1.1
Step 5	Misc VirusTrojan	MISC ramen worm	192.168.1.1 -> 192.168.1.1
Step 6	Intrusion Root	MS-SQL/SMB xp_showcolv possible buffer over	192.168.1.1 -> 192.168.2.22
Step 7	Recon Enumeration	WEB-PHP read_body.php access attempt	192.168.2.22 -> 192.168.3.1
Step 8	Escalation Service	WEB-MISC changepw.exe access	192.168.2.22 -> 192.168.3.1
Step 9	Escalation Service	SMTP VRFY overflow attempt	192.168.3.1 -> 192.168.3.1
Step 10	Escalation Service	WEB-COLDFUSION CFUSION_VERIFYMAIL a	192.168.3.1 -> 192.168.4.8
Step 11	Escalation Service	SMTP RCPT TO overflow	192.168.4.8 -> 192.168.4.16
Step 12	Goal Dos	DDOS TFN client command BE	192.168.4.16 -> 192.168.4.16

The stages used by the actions performed in the attack are tracked to determine the sequence that the stages occur in. This sequence is as follows: stage 0, stage 1, and

stage 5 (three times), stage 8, stage 5, stage 7 (four times), and finally stage 9. This attack sequence abides perfectly by the rules defined in the guidance template class.

8.2.3 Attack Parameters

The three automatic attack generation parameters are intended to provide a significant level of control in generating the attacks simply by varying the parameter values between 0 and 1. A separate experiment is performed to verify that each parameter provides this level of control.

The efficiency parameter affects the directness and complexity of the attacks. Highly efficient attacks are expected to consist of only a few steps that all follow a critical path to the target, while highly inefficient attacks are expected to have many steps that traverse across the network. The experiment performed to verify this outcome is established using the network model shown in Figure 8.5 and consists of twenty attacks using the same final target and goal category. For these attacks, the efficiency parameter is varied between 0.05 and 1.0, while the other parameters remain at 1.0. With the efficiency value as the only difference between the attacks, the number of steps and complexity of the attacks can be used to determine the effectiveness of the parameter. Ten replications of this scenario are used (with a different random seed for each replication) to provide a sufficient amount of step-related data for each parameter value. Figure 8.7 displays the number of steps generated for each of the ten replications across all of the efficiency values used. A trend line is displayed to indicate the correlation of the efficiency parameter with the number of steps.

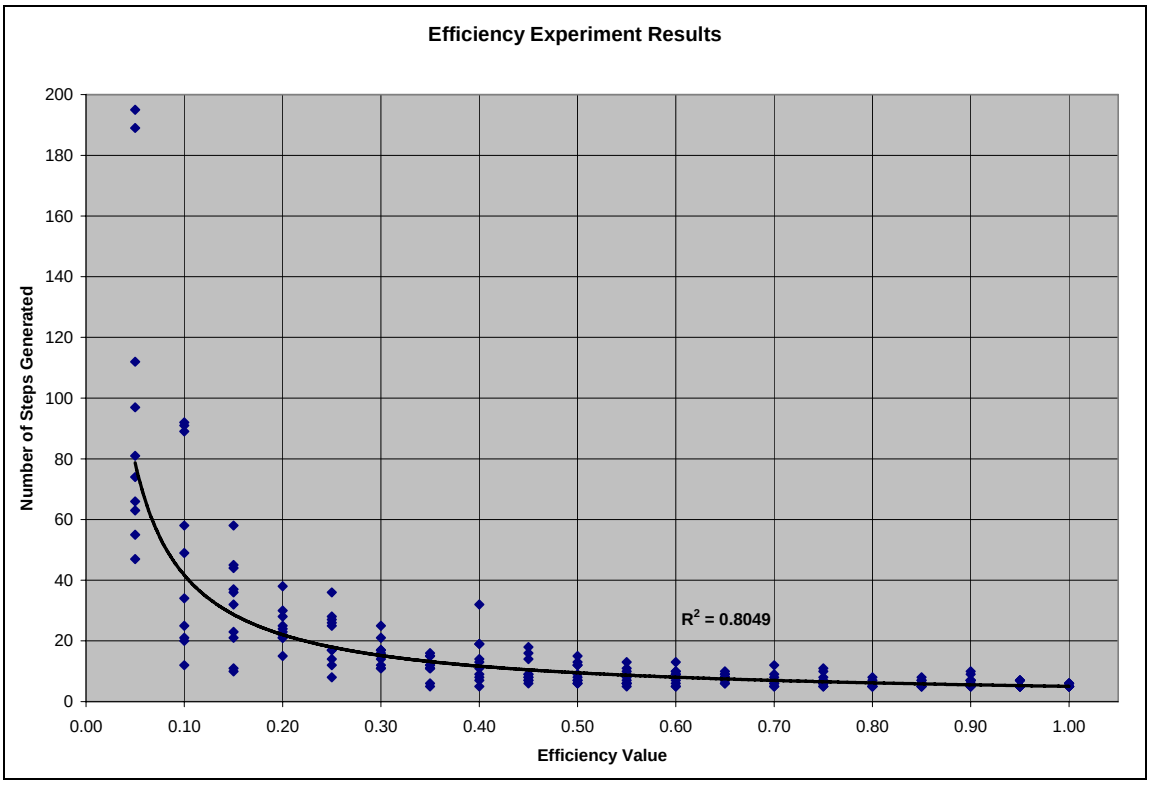

Figure 8.7: Relationship between Efficiency and Number of Steps

The lower efficiency values are determined to significantly produce more attack steps. The trend in Figure 8.7 indicates that experiment produces an 80.5% correlation between the efficiency and the number of steps generated. Observation of the individual attack progressions developed for each efficiency value also indicates that the attacks using a low efficiency value have a more complex attack path than the attacks using a high efficiency value. For example, nine out of ten replications of the attack associated with an efficiency of 0.9 progress along the critical connector path to the final target (in the Office group). However, six replications of the attack associated with an efficiency of 0.3 branch out to the Subnet group before advancing to machines connected to Router 3. The number and complexity of attack steps produced for this experiment indicate that the efficiency parameter has the appropriate effect in the automatic attack generation process.

The stealth parameter affects how well intermediate goal steps are avoided during the attack progression, and these steps are more likely to be produced when using a low stealth value than when using a high stealth value. The experiment used to verify the proper implementation of the stealth parameter also uses the network model in Figure 8.5. In a similar manner to the efficiency experiment, this experiment's scenario consists of twenty attacks where only the stealth parameter is varied (between 0.05 and 1.0). Also, ten replications are run to provide a significant amount of data. The primary metric within the attacks generated is the number of intermediate goal steps per attack. Figure 8.8 displays this metric across the different replications and stealth parameter values.

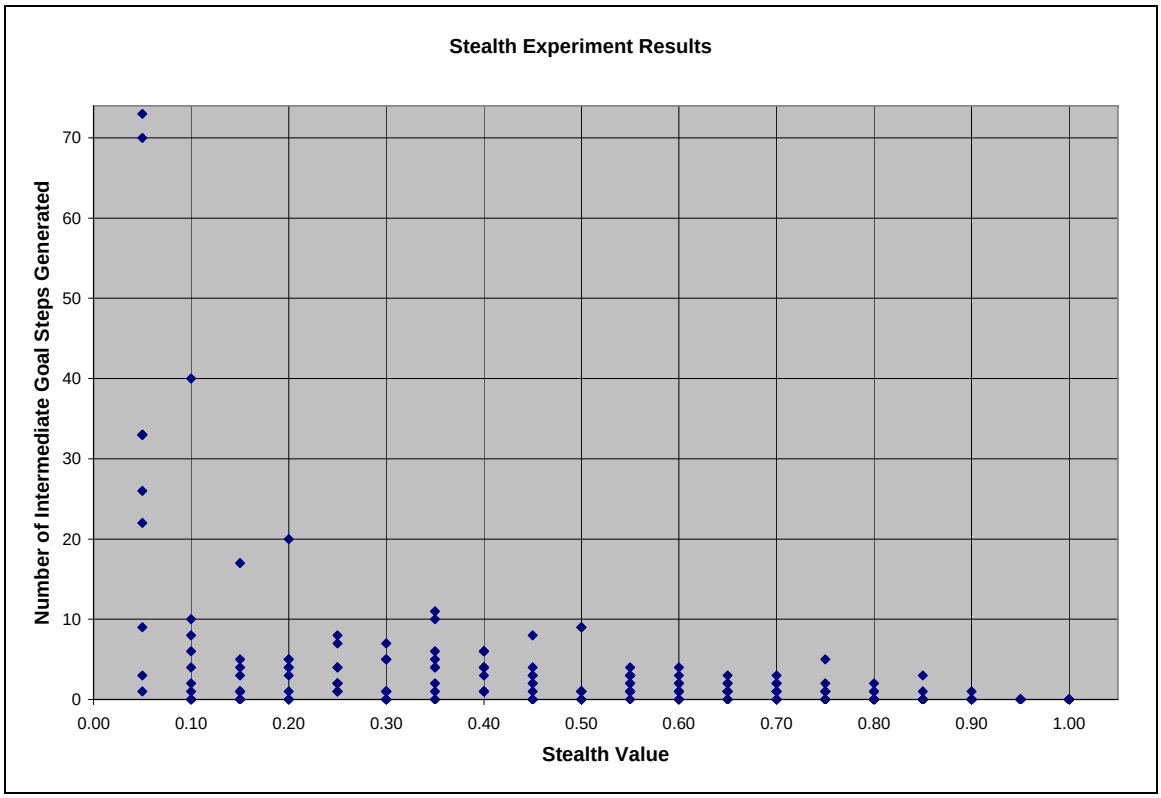

Figure 8.8: Relationship between Stealth and Number of Intermediate Goal Steps

Figure 8.8 illustrates that the lower stealth values overall produce a larger amount of intermediate goal steps. However, the relationship between the stealth parameter and

this metric does not fit well to any particular distribution. Still, though, the significance of the parameter can be verified by comparing the results when using lower parameter values with the results when using higher parameter values. For this comparison, a paired t-test is performed comparing the number of goal steps produced using moderately low stealth values (ranging from 0.1 through 0.3) with the number of goal steps produced using moderately high stealth values (ranging from 0.7 through 0.9). The test indicates with 99.9% confidence that results are significantly different.

The skill parameter affects how likely each attack is in succeeding. Highly skilled attacks will typically be successful while unskilled attacks are likely to fail at some particular point (step). This likeliness is verified with an experiment using the network model in Figure 8.5 and an attack scenario of twenty attacks with different efficiencies and targets. The experiment involves running ten replications of the attack scenario for each of nine possible skill parameter values ranging from 0.1 through 0.9. The primary metric of this experiment is the number of successful attacks (out of the twenty possible attacks) in the attack scenario. Figure 8.9 displays a plot of this metric indicating the number of completed attacks for each of the ninety simulation runs. A fitted curve illustrates a fairly positive linear relationship between the skill parameter and the attack success. The parameter, though, is not intended to strictly represent the probability of an attack's success. The skill value merely guides approximately how successful attacks are relative to other skill values (more successful or less successful).

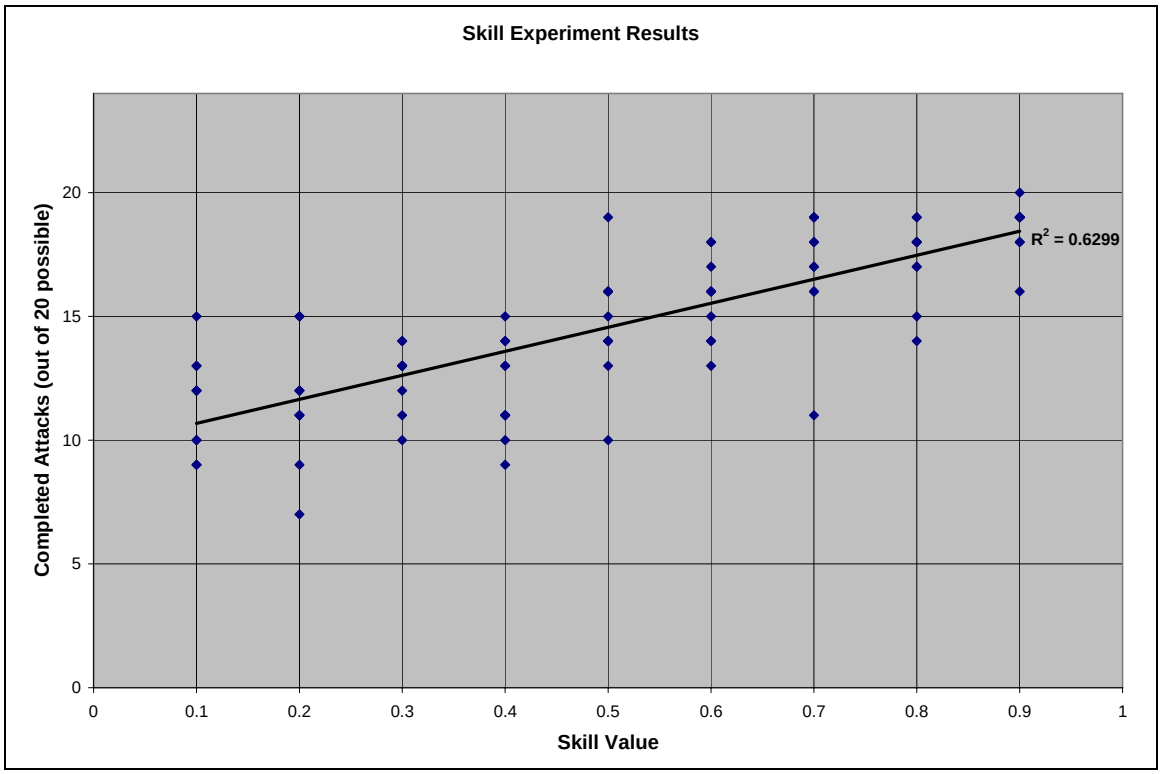

Figure 8.9: Relationship between Skill and Completed Attacks

For skill parameter, as well as the other parameters, will give varying results depending on the network used. In the examples discussed in this section, the same exact network setup is used for each experiment. However, different modeled networks may cause the parameters to have differing relationships. For instance, a skill value of 0.6 in a simple and fairly insecure network may result in the majority of attacks succeeding while a skill value of 0.6 in a complex and secure network may lead to only a small percentage of attacks succeeding. Similarly, an efficiency of 0.3 in a large and complex network model will typically produce many more attack steps than the same efficiency used in a small network model. The automatic attack parameters are intended to simply guide the generation process rather than provide specific results, and differing values for these parameters are effectively shown to establish different attacks when considering a common network.

8.2.4 Entity Routing

The routing of traffic entities between nodes within the modeled network is the primary simulation-based feature of the simulator. This task represents the execution of both attack steps and common false-positive noise activity. The task is also necessary to determine the success of each action performed and trigger functions of the sensor objects used in the model. An experiment is set up to demonstrate the proper implementation of traffic routing, and the experiment is associated with tracking the routing of an entire attack progression. Again, the experiment utilizes the network illustrated in Figure 8.5.

The routing experiment involves an attack scenario with one automatically generated attack that runs successfully to completion. For this experiment, both the entity and node classes are temporarily modified to output information pertaining to the routing process. Each time an entity enters or leaves a specific node, a string is displayed indicating the action associated with the entity and the name of the node. This information is used to verify that all of the attack steps of an attack are routed as expected through the network. The source and destination of the steps generated for the attack are displayed in Table 8.6.

Table 8.6: Routing Experiment Attack Steps

Step	Source	Destination
1	84.31.173.66	192.168.1.1
2	112.124.121.1	192.168.1.1
3	143.9.55.104	192.168.1.1
4	192.168.1.1	192.168.1.1
5	192.168.1.1	192.168.1.1
6	192.168.1.1	192.168.3.1
7	192.168.3.1	192.168.3.1
8	192.168.3.1	192.168.4.9
9	192.168.4.9	192.168.4.9

Simulating this attack in the network results in the routing paths depicted in Table 8.7. These routing paths are compared to the network display in Figure 8.5 to verify that the correct node (connector/machine) links are used. For each attack step, the associated traffic entity follows the appropriate path through the network topology. This experiment therefore verifies that the traffic entities used in the simulation can logically move from the source machine to the destination machine through the proper network nodes.

Table 8.7: Attack Scenario Routing Paths

Step (Entity #)	Routing Path
1	External → Router 1 → 192.168.1.1
2	External → Router 1 → 192.168.1.1
3	External → Router 1 → 192.168.1.1
4	192.168.1.1 → 192.168.1.1
5	192.168.1.1 → 192.168.1.1
6	192.168.1.1 → Router 1 → Router 3 → 192.168.3.1
7	192.168.3.1 → 192.168.3.1
8	192.168.3.1 → Router 3 → Router 4 → 192.168.4.9
9	192.168.4.9 → 192.168.4.9

8.2.5 Noise Generation

The generation of noise activity is an important part of providing representative network traffic that can trigger IDS alerts, and an experiment is performed to illustrate that the noise activity is properly generated. The experiment is designed to illustrate both the proper generation of noise traffic and the independence of the attack generation process and the noise generation process. Essentially, the noise generation process should not affect the set of attacks produced and executed during the simulation run. To test the independence of these processes, two replications of an attack scenario are executed. The first scenario replication does not include any noise generation, while the second scenario replication generates approximately 500 noise activities per hour. 15% of this activity is

associated with intrusion actions, while the remaining 85% is associated with reconnaissance actions. Also, 95% of the noise activity is on the external portion of the network. The attack scenario consists of three automatically generated attacks, and the inputs for the three attacks are displayed in Table 8.8.

Table 8.8: Attack Parameters for First Experiment

Attack	Efficiency	Stealth	Skill	Target	Goal	Step Time	Start Time
1	0.7	0.9	1.0	192.168.2.23	Dos	4	0
2	0.5	0.8	0.8	192.168.3.2	Corruption	3	21
3	0.8	1.0	1.0	192.168.4.16	Espionage	3	33

Running the scenario without any noise results in six steps generated for the first attack, four steps generated for the second attack, and six steps generated for the third attack. All three attacks run successfully to completion, with a total scenario time of 0:50:49. When including the 500 noise activities per hour, the same exact steps are generated for each of the attacks and the attacks are again successful. This process generates the same results when repeated again using the common random number option. These results demonstrate that the automatic attack generation and noise generation are processed independently. Furthermore, 452 noise activities are properly generated during the simulation run.

To verify that the parameters for setting up the noise work as intended, the same scenario is replicated over ten runs (including the initial run) using different random numbers. This process is used to demonstrate that the number of noise activities, percentage of external noise, and action categories of the noise are appropriately represented. Table 8.9 indicates the replication results with respect to these three metrics.

Table 8.9: Noise Metrics over Ten Replications

Replication	# of Noise Activities	% External Noise	% Reconnaissance Actions
1	452	93.1%	84.1%
2	505	94.7%	85.0%
3	393	96.2%	85.2%
4	371	95.4%	86.5%
5	424	94.8%	85.1%
6	424	92.2%	83.0%
7	605	95.0%	83.8%
8	334	92.8%	84.4%
9	384	95.8%	86.2%
10	374	94.7%	86.6%
Avg.	**427**	**94.5%**	**85.0%**

The specified noise rate of 500 activities per hour suggests that approximately 425 noise activities should be generated during the 51-minute simulation run. Table 8.9 indicates that the ten replications produced an average of 427 noise activities, which is very representative of the expected amount. Also, an average of 94.5% of the noise is external-based and an average of exactly 85% of the noise is associated with reconnaissance actions. These results verify that the noise-based parameters effectively control the amount and type of noise produced.

8.2.6 Alert Generation

IDS alerts are generated during the simulation run as traffic entities move through network nodes that have associated IDS sensors. The set of alerts generated for each sensor become the primary outputs at the end of the simulation run. In order for accurate alerts to be generated, the model must be able to identify the action signature associated with the traffic and match the signature with the sensor's alert database to determine whether an alert is to be generated. An experiment is used to verify that these tasks are performed as expected.

The experiment for this verification procedure involves the same attack scenario as is used for the routing experiment. Of the network devices involved in the attack scenario, Router 1, Router 3, Router 4, and the Web Server all include IDS sensors. For each attack step, the path is checked to determine if any of these four devices are along the traffic path. For each device along the path, the IDS sensor output (after running the simulation) is checked to verify that an alert is generated and that the alert corresponds with the attack step's action. Table 8.10 displays the action associated with each attack step and the devices that produce alerts for this action.

Table 8.10: Attack Step Alerts

Attack Step	Action	Sensor Alerts
1	SCAN cybercop os PA12 attempt	Router 1 Sensor, Web Server Sensor
2	WEB-IIS .cnf access	Router 1 Sensor, Web Server Sensor
3	WEB-CLIENT readme.eml download attempt	Router 1 Sensor, Web Server Sensor
4	BACKDOOR Infector 1.6 Server to Client	Web Server Sensor
5	INFO Connection Closed MSG from Port 80	Web Server Sensor
6	WEB-MISC changepw.exe access	Web Server Sensor, Router 1 Sensor, Router 3 Sensor
7	WEB-PHP shoutbox.php directory traversal attempt	No alerts
8	DNS EXPLOIT named overflow attempt	Router 3 Sensor, Router 4 Sensor
9	POLICY SMTP relaying denied	No alerts

Attack steps seven and nine do not produce any associated alerts since there is no sensor along the path of these steps. For all other steps, the appropriate sensors generate alerts indicating that the correct action is being performed. Also, the difference between the NIDSs and the HIDSs is verified by running the scenario again using a lower skill value. With this modification, one of the attack steps targeting the Web Server fails to be executed. The sensor associated with Router 1 still generates an alert indicating that the

action was attempted, but the sensor associated with the Web Server does not generate an alert since the action was not successfully executed.

8.3 Validation of Attacks and Alerts

In order to provide accurate alert output using the simulation model, the generation of simulated attacks and sensor alerts must validly represent real network attacks and sensor alerts, respectively. Ideally, the validation process involves comparing the results of the simulator with the information available in a real network setup to check for consistencies between the simulated results and the real network results. However, such a real network setup needs to provide the *a priori* knowledge of the attacks that occur along with the entire set of IDS alerts. Very few existing networks provide this capability, and the networks that do provide this capability generally keep the results proprietary. For these reasons, a different approach must be taken that involves logically comparing the generated attacks and alerts to what is expected in a real network. The remainder of this section will discuss how specific aspects of the simulation model are validated as representing the corresponding real-network counterparts.

Providing a simulated form of communication between devices in the network is a necessary component of validly representing attacks. The simulator models this communication using traffic entities that represent a set of network packets associated with one particular action (since actions can be spread across multiple packets). The same representation is used for both attack-based actions and noise-based actions. During a simulation, the traffic entities are routed through the appropriate nodes in the network model in the same manner as the actual traffic would be routed through devices in a physical network. Furthermore, the actions executed by the traffic entities are limited by

the attributes of the target machine (such as the services and operating system) and the permissions of the connectors involved. Limiting the traffic in this manner reflects how real computer networks are not simply susceptible to just any action, but rather to actions that correspond with the vulnerabilities in the computers and routing devices on the network. Therefore, the network model is validated as providing representative traffic entities and appropriate network details that effectively restrict the actions performed with the traffic entities.

With respect to generating attacks, the auto-attack parameters and the Guidance Template input allow for the attack generation process to produce the desired types of attacks. Since the Guidance Template integration is verified as functioning as desired, this input can be adjusted or updated to reflect either a more accurate or more detailed attack stage progression. Using this approach, a subject matter expert can clearly define the types of attacks possible with the simulator in order to ensure that only valid attacks are generated. The current Guidance Template input uses stage progressions that are in accordance with a graph-based stage precedence defined by subject matter experts. Therefore, the model is valid with respect to the classification and precedence of attack stages by subject matter experts, and the flexibility of the simulator's Guidance Template input allows for updated or more detailed attack stages to be specified when needed. The maintenance of this Guidance Template input, though, is a necessary part of providing both valid and up-to-date attacks using the simulator.

In a real network, the IDS sensors specifically are unable to discern between the activities associated with an attack and the activities that are just common, non-malicious network noise. This same feature of IDS sensors is mimicked within the simulation

model. When either the attack steps or the noise activities are routed between nodes in the network, the associated entities are cast into traffic entities. The traffic class used in this model is the super-class of both the attack step and noise class; thus, a traffic object still contains all of the necessary information to route the entities. Each IDS sensor triggers an alert when a traffic entity entering the node containing the sensor is associated with an alert signature in the respective sensor's alert definitions database. For this whole process, the IDS sensors are not provided with any indication of whether the entity is associated with an attack or with noise. Therefore, the IDS sensor objects validly react to the modeled network traffic in the same manner that a real IDS reacts to real network traffic.

The alert output of the simulation model is intended to represent the real alerts that would be generated when performing an attack scenario in an actual network. For this reason, the alerts generated by modeled sensors must provide the same information that is included in real sensor alerts. For snort alerts specifically, this information consists of a timestamp, alert message, protocol, classification, priority ranking, and network path. The alert definitions input used by the modeled snort sensors contains alert data compiled using information from both the Snort website (www.snort.org) and the Skaion test network alert data that contains real snort sensor alerts. Therefore, all of the same alert information provided in real snort alerts is available for use by the simulated alerts. Furthermore, the simulated snort alerts use the same message and output format used by real alerts, allowing the simulated alerts to be handled in the same manner that real alerts are handled. Additionally, the simulator provides an add-on (the sensor management package) that can be used to limit the alerts output during a simulation run. This type of

functionality can potentially be used in a real network where bandwidth and other constraints force the network's sensors to only output a portion of all of the alerts produced. All of these alert generation considerations ensure that valid sensor alerts are produced with the simulator.

8.4 Model Capabilities

The simulation model has many capabilities that address both the initial intentions noted in the problem statement and other issues that have become of importance. The primary capabilities of the model include modeling detailed networks and simulating attack scenarios and intrusion detection systems within such networks. A competent user is able to effectively make use of the modeling features, simulate desired attack scenarios, and utilize the intrusion detection results. This section will discuss specific capabilities that the model provides.

The simulation model allows for both large and complex networks to be easily (and graphically) defined. Such networks can be many levels in depth and contain subnets with hundreds of host machines. Figure 8.10 illustrates one such large network model consisting of five network levels and over 1500 machines. The network modeling functionality allows for each machine, connector, and subnet to have a unique set of attributes along with IDS sensors that can easily be toggled on and off. Furthermore, the connectors involved can define specific permissions for each unique connection path between network devices. The network depicted in Figure 8.10 saves and loads quickly, and the network details are also readily available when setting up an attack scenario.

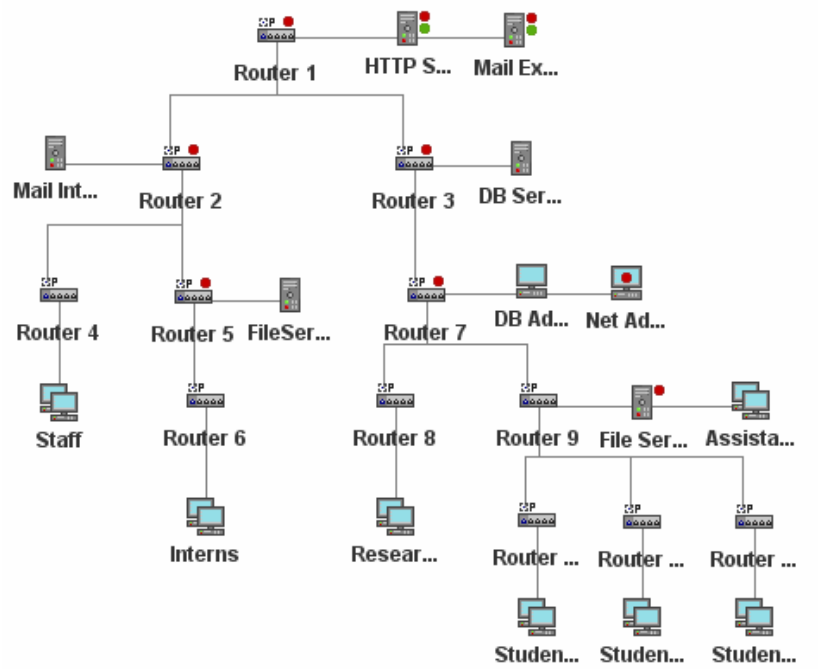

Figure 8.10: Large Network Modeled

The ability to import a network model from a virtual terrain document or export a modeled network to a virtual terrain document is another useful capability of the simulation model. This interaction with virtual terrain documents essentially allows modeled networks to be used by external applications for a variety of means. Similarly, networks intended to be used by the simulator can be defined in an external application and imported. Regardless of external applications, the use of virtual terrain imports and exports provides an alternative way to store modeled networks. Also, networks stored in this manner can be loaded even when application-related changes are made to the simulator. In such cases, the typical object files used to store network models may be incompatible.

The guidance template input represents part of the flexibility provided by the simulation model. While the guidance template is responsible for the attack generation capabilities of the simulator, the guidance template can also be redefined in a variety of

ways in order to change the requirements for attack progressions. Therefore, the guidance template can be updated to reflect more detailed attack progressions and still be compatible with the simulation model.

The attack modeling options are a major capability of the simulation model. The user is able to specify individual attacks at various levels of detail. Very detailed attacks can be created through indicating each action performed through the progression of the attack. Broader attacks can be created by providing several attack parameters to reflect the directness, stealth, and skill level of the hacker performing the attack. An entire attack scenario can even be generated based on a set of parameters that include the attack rate (in attacks per hour), average efficiency, standard deviation of the efficiency, average attack length, and entire scenario length (minutes to run the scenario). The manual and automated attacks have no specified size limits, so the number of steps that make up an attack is essentially only limited by the processing capability of the machine running the application or the patience of the user in setting up the attacks. For example, in setting up an automatically generated attack in the network displayed in Figure 8.10 using a very low efficiency value (0.005), an attack is generated with 2009 attack steps in a matter of seconds. This attack is also simulated within the network in only a few seconds time. The total size of the attack scenario (with respect to both number of attacks and steps) is also unbounded within the simulation model. Using the automated attack scenario option, an attack scenario is generated with 197 attacks ranging from 5 to 91 steps per attack. All of these attacks are also generated and simulated within a matter of seconds. After either manually specifying the attacks in a scenario or invoking the attack generation process on

attacks in a scenario, the entire attack scenario can be stored as an XML file for later use and observation.

In addition to modeling specific attacks and attack scenarios, the simulation model is also capable of modeling noise traffic that occurs within the network. This noise represents network activity that is not malicious in any way but can still cause false positives (false alerts) when checked by IDS sensors. Since the majority of alerts produced by a real IDS represent noise, the simulation model must be able to generate and process a sufficient amount of noise traffic. An experiment is established to test the limits of the noise generation process. This experiment consists of running the attack scenario from the automated scenario generation example (with 197 attacks) along with 10,000 noise activities generated per hour with the category breakdown displayed in Table 8.11. The total simulated time for the run is 84.5 minutes, and this scenario processed to completion within about four actual minutes. Running the scenario with noise takes significantly longer time due to the large number of alerts generated; especially considering that the majority of noise activity occurs on the external portion of the network where all three devices (Router 1, HTTP Server, and Mail External Server) contain IDS sensors and generate alerts for each activity. For instance, the HTTP Server alone generated 8055 alerts over the 84.5 simulated minutes. The alerts, though, are still generated within a reasonable amount of time when considering the length and size of the scenario and the volume of noise activity.

Table 8.11: Noise Activity Category Breakdown

Category	Recon.	Intrusion	Escalation	Goal	Misc.
%	70%	10%	10%	5%	5%

The primary outputting capabilities of the simulation model include the IDS sensor alerts and the scenario ground truth. The sensor alerts indicate what activity, both malicious and noise-based, is detected during the simulation run. The simulation produces a separate file of alerts for each active sensor placed in the modeled network. These alert files are produced using a commonly accepted text-format for each alert, and the model places no specific limits on the size of the alert files. This setup allows for all of the alerts generated in a scenario to be easily organized, examined, and used as input to other applications. The ground truth output created represents all of the actions performed in the attack scenario that are associated with malicious activity. Providing this output allows for comparisons to be made with the sensor alert output to determine which activities failed to be detected by the IDS. This ground truth output can also be used in other applications to compare with the sensor alerts for the same or alternate reasons.

8.5 Model Limitations

In spite of the many capabilities of the model, there are several limitations that affect the use and applicability of the model. Many of the limitations do not directly impact the validity of the model, but addressing these limitations can allow for more detailed networks, attacks, and alerts to be modeled using the application presented. This section will discuss aspects of the simulation model that are limited with respect to the desired functionality, which include the connector modeling features, integrated sensor types, auto-attack specification, and exploit and vulnerability database.

Although the network modeling features provide the user with a variety of options, the representation of connectors is still fairly generic. The network model represents a connector as a device that allows other connected devices to communicate.

In reality, though, a variety of connector types exist (routers, hubs, switches, etc.), each with a specific functionality. Therefore, the model is limited when constructing a network model that represents a network consisting of a series of specific connector types and functionalities. Also, one of the basic assumptions of the model is that specific malicious activities do not move more than one connector level through the network at a time. However, through the use of routers and other more functional connector types that can determine the appropriate path through the network, such activity is easily capable in a real network.

The current sensor input used in the model is also fairly limited. Snort IDS is the only instantiated sensor type that can be added to a network device. Also, the functionality of the Snort sensor is more specific to NIDSs than to HIDSs. Ideally, though, the model should represent NIDSs and HIDSs as they really function. Currently, the only functionality difference between the sensor classes is that HIDSs will not detect failed actions while NIDSs will. However, genuine HIDSs also provide more specific information about the action performed and the affect on the host that are not reflected in the simulation model.

The simplicity of the automatic attack parameters provided illustrates another manner in which the simulation model is limited. The generation of attacks using the parameters only (exhaustively) accounts for a subset of the potential types of attacks that are possible within the network. Also, some of the parameters result in an uncontrollable degree of variability in the attacks generated. Having variability in the attack generation process is necessary to reduce user bias and account for the alternative attack

progressions, but the lack of control exhibited in this variability effectively reduces the specific influence of each parameter.

Lastly, the database of exploits and vulnerability mappings used by the model is hardly comprehensive when compared to the databases used by other cyber security applications. Also, the addition of new exploits and vulnerability mappings is primarily a manual process, which further illustrates the limitations of the model in keeping up-to-date databases. Consequently, modeling newly identified attack types and new service vulnerabilities is a challenging task with the current exploit and vulnerability mapping infrastructure.

8.6 Applications

Although the cyber attack simulator is developed as a stand-alone simulation model, the simulator has effectively been able to use input from or provide input for several external applications. For instance, the virtual terrain document allows for external applications to provide a network structure that can be created using the simulator. Also, the alerts and ground truth data produced by the simulator can be used by applications focused on processing sensor alerts in some manner. This section discusses specific applications that can and have been used with the simulator, which include the Nessus network scanning tool, the INFERD fusion tool and the VTAC assessment tool.

Nessus (Tenable Network Security, 2007) is a network scanning tool that allows for many of the details of a computer network to be obtained and stored. Recent research efforts have included generating a virtual terrain XML file to store the network information detected by Nessus. The network modeling capabilities of the simulator

allow for these virtual terrain files to be easily visualized and modified. Thus, the simulator can effectively depict a network that is scanned using Nessus. For example, the Skaion test network (Skaion Corporation, 2007) is scanned using Nessus scanner and the details are stored in a virtual terrain XML file. The XML file is imported into the simulator, and the simulator graphically displays the Skaion test network with all of the same features available as with a typical network generated with the simulator. Figure 8.11 displays the network model of the imported Skaion test network.

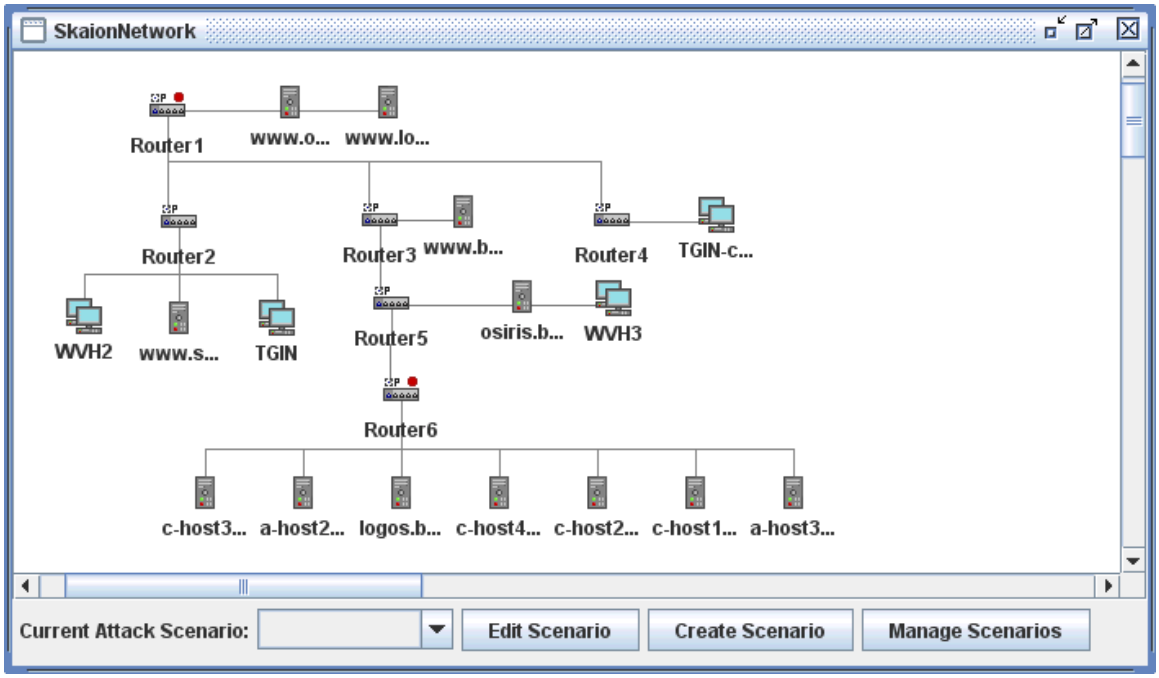

Figure 8.11: Skaion Test Network Imported into Simulator

INFERD is an information fusion tool that takes a set of sensor alert messages as input and correlates the alert messages to form attack tracks that correspond with potential attacks (Holender, Stotz, and Sudit, 2005). Although the intended application of INFERD is correlating alert messages from real IDS sensors on a network, the development of INFERD requires a generous amount of sample alert messages with which to test the correlation features. Relying on a physical network for the sensor alert

messages restricts INFERD to only demonstrating successful correlations with respect to a small subset of potential network and attack types. The use of the simulator to provide sensor alert data offers a wider range of possibilities with respect to the network structure and attack types.

VTAC is the simplified name given to the Virtual Terrain Assisted Impact Assessment for Cyber Attacks application. This application combines a network's virtual terrain file and a set of sensor alerts generated in the network to determine the associated impact to the network. In this situation, the virtual terrain file indicates some crucial machine attributes, such as the services running, importance, and operation criticality. The sensor alert input for VTAC is then used to determine what actions can potentially compromise a machine in the network, and the impact score is generated by comparing this information with the virtual terrain information (Argauer, 2007). With respect to this process, the Cyber Attack Simulator is used to generate sensor alerts for the network specified in the virtual terrain. The simulator reads in the virtual terrain to build the appropriate network model, and attack scenarios are specified with the desired targets and goal types. For this application, the networks typically represented in the virtual terrain and assessed by VTAC are not physical networks, so the simulator provides a quick way to generate alert data which would otherwise require manually defining the exploits and determining the alert correlations.

9 Conclusions and Future Work

9.1 Conclusions

Accurate and detailed IDS sensor alert data is strongly desired in the cyber security realm for the purposes of testing both awareness and response tools. The cyber-attack simulator application developed over the course of this thesis provides a means for quickly and efficiently generating representative alert data. The application also provides the functionality to manipulate the virtual network representations, the cyber attacks simulated, the sensor alerts output, and the a priori inputs. Overall, the application can be viewed as a tool that effectively manages the entire process of simulating IDS sensor alerts. Although this tool is not exhaustive in generating alert data, the tool provides a sufficient underlying structure that can be easily built upon and improved to account for the wide variety of potential alert data.

The application provides network modeling capabilities that allow for well-defined network representations to be created. These virtual networks include the connectors and machines/subnets that actual networks consist of, and appropriate attributes are given to these connectors and machines. For example, connectors include firewall permissions that dictate what type of network communication can occur between a specific source and destination within the network. Also, machines include attributes such as the operating system, services running, and the machine type that are used to determine the specific vulnerabilities for the machine. IDS sensors can be placed on both the connectors and machines to properly monitor the network traffic flowing through these network devices. The devices can be connected to form specific, organized levels within the network that indicate how deep a machine is within the network topology (in

other words, how many connector links separate the machine from the external network portion). This level structure, in combination with the network device attributes, offers a large variety of potential network topologies that can be produced with the application.

The application also provides attack modeling capabilities that are used to generate detailed attack scenarios and simulate the individual attack actions within an established network model. Additionally, the detection of such actions by sensors produces alerts that can be manipulated and output as the primary simulation results. The functionality for providing such capabilities includes modeling the network traffic associated with attack actions, storing a set of attack actions for each attack and a set of attacks for each attack scenario, and allowing for attacks to be generated through either a manual process or an automated and parameterized process. This automated attack generation process uses a series of parameters that give a user significant control over the type of attack generated and the manner in which an attack progresses through the network. This process also relies on the details of the network structure to ensure that feasible attack progressions are produced.

The functionality of the network model and the attack simulation are verified and validated through some different approaches. Several of the network modeling features, including the connector level structure, subnet implementation, machine service representation, and connector firewall permissions, are verified through the use of conditional statements and temporary debugging output. Other network features are visually (or graphically) verified through observation of the resulting network model. Such features include the connection links, IP addresses, saving process, and virtual terrain exportation. The attack simulation features are verified using a unique approach

for each feature. The proper exploit filtration is verified by observing the details of individual exploits listed. The stage progressions produced in the automated attack generation process are verified by observing scenario results and comparing the results to the guidance template. The level of control provided by attack parameters is verified by observing attack trends and other effects upon varying the parameters. The entity (traffic) routing is verified by tracing the appropriate path through the network model. The noise generation is verified by comparing noise results with the noise parameter values. Lastly, the alert generation is verified by checking the sensor output against the specific attack actions simulated. The validation of attack and alert generation process is performed by evaluating the modeling capabilities with respect to the manner in which a real network functions. The use of a common network traffic entity to represent both attack and noise actions is a valid means of representing network communications and imitates the real issue of not being able to clearly distinguish attack-related actions with typical noise actions. The use of automated attack generation parameters allow for knowledgeable users to easily sculpt a set of valid attacks to suite specific needs without the requirement of defining each specific action. Also, the alert output produced through simulating an attack scenario is validated as providing the same information as real alerts of the corresponding sensor type.

The network model and attack simulation in combination give the overall application a wide array of capabilities. Some of the key capabilities include the variety of network configurations that can be modeled, the interactions with virtual terrain, the flexibility of the guidance template input, the methodology provided for generating attacks, the modeling of noise activity, and the generation of both IDS sensor alerts and

attack scenario ground truth. However, the application also contains some limitations that encourage the continued development and improvement of the application features. These limitations consist of the generic representation of connecting devices, the lacking set of default IDS sensor types, the simplicity and observed variability of the automated attack parameters, and the primitive databases provided for exploit and vulnerability information.

The use of the cyber attack simulator in collaboration with other applications and related areas of research further demonstrates the capabilities and utility of this application. The virtual terrain interactions and the simulated sensor alert and ground truth output allow the application to work effectively with the Nessus network scanning tool, the INFERD fusion tool, and the VTAC assessment tool. The network modeling features allow for a virtual terrain generated with data from Nessus to be imported as a new network model and treated just as a network model generated within the application. The sensor alerts produced by the simulator are used by INFERD correlate attack actions, and these results are compared with the simulator's ground truth output to assess the performance of INFERD. Also, both the virtual terrain and the sensor alert output are used by VTAC to determine what impact particular actions have on a network.

In response to the initial problem statement of this thesis, the application developed successfully portrays both the progression of attacks through a modeled network and the generation of accurate IDS sensor alerts as a result of such attacks. Network models and attack scenarios can be created and controlled with great detail. Additionally, the attack scenarios can be easily run and modified. The sensor alerts produced during a simulation run represent valid IDS sensor alerts that associated with

the particular attack actions simulated. Also, the cyber attack simulator successfully integrates with other applications and related research to fully utilize the network modeling, attack simulation, and sensor alert generation features.

9.2 Future Work

The Cyber Attack Simulator developed in this thesis comprises several of the accomplishments made toward modeling computer networks, cyber attack scenarios, and IDS alerts. Overall, the simulator is a step forward in providing representative intrusion detection data; however, the simulator alone does not complete this task. The application itself has significant opportunity for both improvements to existing features and addition of new features. Also, the inputs used by the simulator need to be either restructured or properly maintained in order to provide the required and up-to-date information that the simulator relies on. Section 9.2.1 discusses some of the recommended tasks in improving the functionality of the simulator. Section 9.2.2 indicates the changes and updates that can be made with respect to the various inputs.

9.2.1 Recommended Functionality Improvements

With respect to the network modeling functionality, a recommended improvement is the use of more specific connector types. The existing representation of a connector (through the connector class) is very generic. More specific subclasses of connecting devices, such as hubs, routers, and switches, can be used in the model by simply extending the connector class and inheriting the attributes and methods provided for connector objects. In this situation, each specific connector class can contain additional methods and attributes that reflect the particular functionality of the connector type.

The sensors modeled within the network structure can also be improved by giving the different sensor types more unique characteristics. For instance, actual network-based and host-based IDS sensors operate and produce output in a different manner. Only a few of the differences between the two types are depicted in the simulator, and the majority of implemented sensor functionality represents network-based IDS sensors. By acquiring both the type of information output by a host-based sensor and the alert definitions associated with a specific host-based sensor brand, the network can be enhanced to more accurately portray these sensor types.

With respect to the attack modeling features, the primary recommended feature to add is the ability to model an internal attack. Although this thesis focuses on external-based attacks, internal attacks are another growing threat to network security. Modeling this type of attack is currently only possible in the manual attack specification process. To integrate this type of attack into the automatic attack generation process, some additional auto-attack attributes and machine selection processes need to be created.

The auto attack generation process can further be improved by the addition of new attack parameters. While the efficiency, stealth, and skill parameters allow for a significant level of control by the user, the actual attacks generated can still be fairly unpredictable. Additional parameters can be made by either breaking down the current three parameters into more specific sub-parameters, providing an additional standard deviation parameter associated with one or more of the existing parameters, or by identifying another feature of the attack progression that should be controlled.

9.2.2 Adjustments to Inputs

Of the inputs used by the simulator, the most critical input is the array of exploits needed in creating attacks. This database of exploits must be extensive and kept up-to-date to generate attacks that are representative of modern hacker attacks. The model currently uses about 2000 exploits of various types, but this array of exploits is far from exhaustive and can be tedious to update. One possibility for improving this setup is to synchronize the simulator's exploit input with a well-maintained exploit database. Some available databases include ArachNIDS, CVE, and BugTraq. ArachNIDS (www.whitehats.com) is a database of signatures and network-based intrusion detection information, CVE (www.cve.mitre.org/cve) is short for Common Vulnerabilities and Exposures and associates exploits with service vulnerabilities, and BugTraq (www.online.security.focus.com/bid) is a database of exploits that provides a unique ID for each exploit (Crothers, 2003).

The set of available sensor types (or brands) to be used in the model is also an important input. The modeling input currently only contains alert definitions for snort sensors. Many other sensor types are available and frequently used in networks, though. Such sensors include Dragon, IIS, and Apache IDS sensors. These additional sensor types each tend to include specific alert signatures and detection capabilities not available in other sensors. Providing the appropriate input for these sensor types and adjusting the simulator's output to match the sensors' alert formats can improve the model's representation of a full-fledged intrusion detection system.

The guidance template input is another area for improvement. Although the input is designed and integrated to allow for a variety of unique guidance templates to be

defined, only one specific guidance template is provided and used in the simulator. The current guidance template is also fairly lenient with respect to the types of stage progressions that are possible. Additional research can therefore be performed to either define multiple guidance templates with varying levels of specificity or maintain the guidance template to appropriately represent attacks based on the views and findings of subject matter experts.

The correlation of machine services with exploit vulnerabilities is an important input that the model requires in filtering out the potential set of exploits. This thesis, though, presents only the first implementation of this service to vulnerability mapping. Therefore, the correlation input has significant room for improvement with respect to both the types and the details (such as the vulnerability IDs and ports) of services defined. The exploit databases mentioned previously, along with other resources as well, can help identify other services and service-specific information that should be provided as an input to the simulator.

References

Argauer, B. J. (2007). Virtual Terrain Assisted Impact Assessment for Cyber Attacks. Unpublished Masters of Science Thesis for Rochester Institute of Technology, Rochester.

Allen, J., Christie, A., & McHugh, J. (2000). Defending Yourself: The Role of Intrusion Detection Systems. *IEEE Software, September,* 42-51.

Baezner, D., and Lomow, G. Object Oriented Simulation. Retrieved May 10, 2007 from Compendex Online Database.

Bass, T. (2000). Intrusion Detection Systems and Multisensor Data Fusion. *Communications of the ACM, 43*(4), 99-105.

Bischack, D. P., and Roberts, S. D. (1991). Object-Oriented Simulation. *Proceedings of the 1991 Winter Simulation Conference*, 194-203.

Brunner, D. T., and Schriber, T. J. (2005). Inside Discrete-Event Simulation Software: How it Works and Why it Matters. *Proceedings of the 2005 Winter Simulation Conference*, 167-177.

Cheung, S., Fong, M.W., & Lindqvist, U. (2003). Modeling Multistep Cyber Attacks for Scenario Recognition. *Proceedings of the Third DARPA Information Survivability Conference and Exposition, 1,* 284-292.

Crothers, T. (2003). *Implementing Intrusion Detection Systems: A Hands-On Guide for Securing the Network.* Indianapolis, IN, Wiley Publishing, Inc. New York, NY.

Darmont, J. (2000). DESP-C++: A Discrete-event Simulation Package for C++. *Software – Practice and Experience, 30,* 37-60.

DeLooze, L.L., Graig, C., McKean, P., & Mostow, J.R. (2004). Incorporating Simulation into the Computer Security Classroom. *IEEE, 34.*

Dougherty, E.T., & Gonslaves, P.G. (2006). Adaptive Cyber Attack Modeling System. *Proceedings of SPIE, 6201.*

Fuchsberger, A. (2005). Intrusion Detection Systems and Intrusion Prevention Systems. *Information Security Technical Report, 10,* 134-139.

Garg, A., Kwiat, K., & Upadhyaya, S. (2006). Attack Simulation Management for Measuring Detection Model Effectiveness. *Department of Computer Science and Engineering at Stony Brook University*

Hall, D.L., & Llinas, J. (1998). An Introduction to Multi-Sensor Data Fusion. *IEEE,* 537-540.

Holender, M., Stotz, A., & Sudit, M. (2006). INFERD and Entropy for Situational Awareness.

Holender, M., Stotz, A., & Sudit, M. (2005). Situational awareness of coordinated cyber attacks. *Proceedings of The International Society for Optical Engineering,* Orlando, April 2005.

Joines, J. A., and Roberts, S. D. (1998). *Object-Oriented Simulation.* Handbook of Simulation. Banks, J. John Wiley & Sons. New York, NY.

Kemmer, R.A., & Vigna, G. (2002). Intrusion Detection: A Brief History and Overview. *Reliable Software Group, Computer Science Department at the University of CA.*

Kistner, J. (2006). *Cyber Attack Simulation and Information Fusion Process Refinement Optimization Models for Cyber Security.* Unpublished Masters of Science Thesis for Rochester Institute of Technology, Rochester.

Kuhl, M., & Kistner, J. (2005). *Generation of synthetic cyber attack data using simulation*. Final report for AFRL/IF VFRP Program, Rome, NY.

Kurtz, G., McClure, S, & Scambray, J. (2005). *Hacking Exposed*. Emeryville, California, McGraw-Hill.

L'Ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. (2002). An Objected-Oriented Random-Number Package with Many Long Streams and Substreams. *Operations Research, 50*(6), 1073-1075.

Lin, J. T., Sheu, L. & Yeh, K. (1996). A Context-Based Object-Oriented Application Framework for Discrete Event Simulation. *Computers Ind. Engng, 30,* 579-597.

McConky, K. T. (2007). Design and Analysis of Information Fusion, Dynamic Sensor Management Rules for Cyber Security Systems Using Simulation. Unpublished Masters of Science Thesis for Rochester Institute of Technology, Rochester.

Sarjoughian, H. S., and Singh, R. (2003). Software Architecture for Object-Oriented Simulation Modeling and Simulation Environments: Case Study and Approach. Department of Computer Science and Engineering, Arizona State University. Retrieved April 30, 2007 from Compendex Online Database.

Skaion Corporation. (2007). www.skaion.com

Snort. (2007). Snort - The De Facto Standard for Intrusion Detection/Prevention. www.snort.org

Tenable Network Security. (2007). The Nessus Vulnerability Network Scanner. www.nessus.org/nessus.

Appendix A: User Guide

This appendix provides an overview of how to use the Cyber Attack Simulator application. An example network is created in which an attack scenario is setup and executed. Section A.1 covers the basic principles of running and managing the application. Section A.2 demonstrates how to construct a network model. Section A.3 explains how to setup attack scenarios within a modeled network. Lastly, section A.4 demonstrates the process of running a simulation and reviewing the results.

A.1 Running the Simulator

Prior to running the Cyber Attack Simulator, several needed input and preference files must be readily accessible by the application. These files include ConnectorSettings.txt, ListOfActions.txt, NoiseSettings.txt, SensorSettings.txt, ServiceSettings.txt, SnortAlertDefs.txt, VulnerabilityMapping.txt, XMLSettings.txt, and GuidanceTemplateXML.xml, and the files should be located in the “config” folder within the application’s directory.

Figure A.1 displays the common directory layout for the Cyber Attack Simulator. The application itself is stored in an executable .jar file. The application, though, also requires the jdom.jar file to provide XML input and output functions. Therefore, the application is best opened by using the Run.bat file located within the application’s directory. This file basically executes a command prompt that includes jdom.jar when running CyberAttackSimulator.jar.

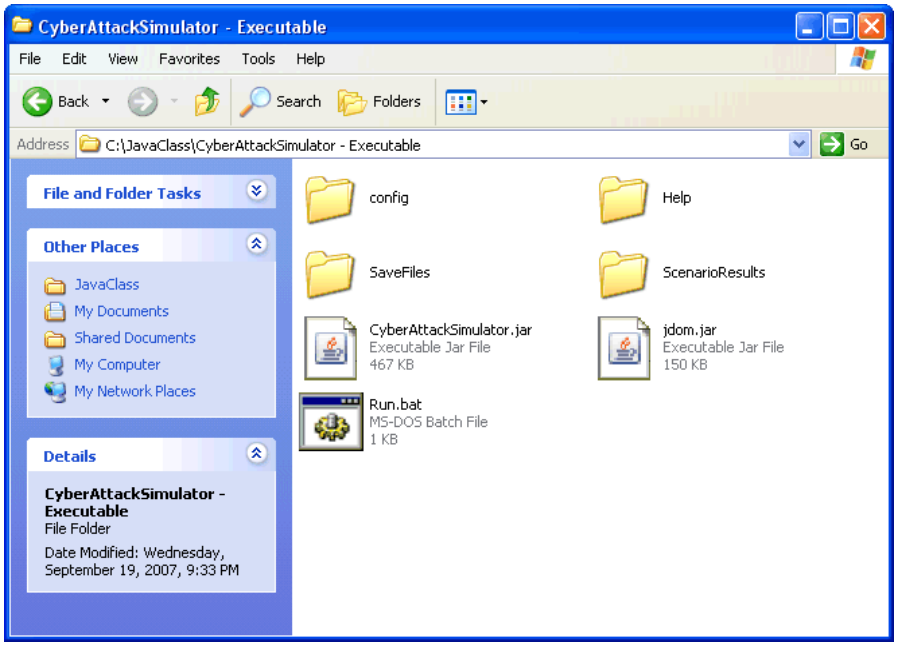

Figure A.1: Running the Application

Upon opening the application, the user is presented with the window displayed in Figure A.2. Only two menus are initially available, the File menu and Help menu. The File menu includes options to create, open, or import a network, modify the application preferences, or exit the application.

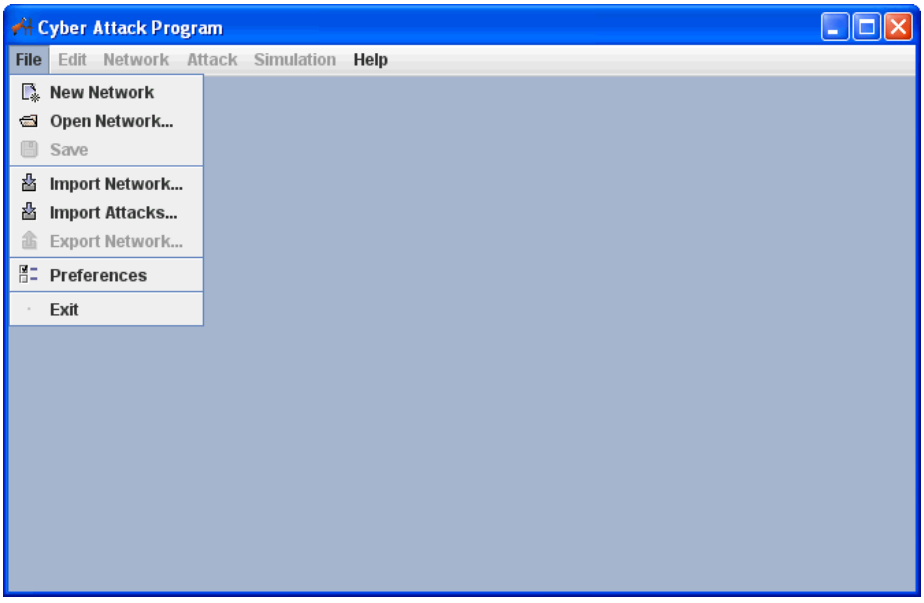

Figure A.2: Application Window

When selecting the “Preferences” option, a tabbed menu of the various default settings and inputs for the application is displayed. This menu is displayed in Figure A.3. The menu allows for the database of available actions to be viewed, the available sensor types to be setup and viewed, the default machine services to be specified, the default connector configurations to be specified, the default noise parameters to be specified, and the XML input and output options to be configured.

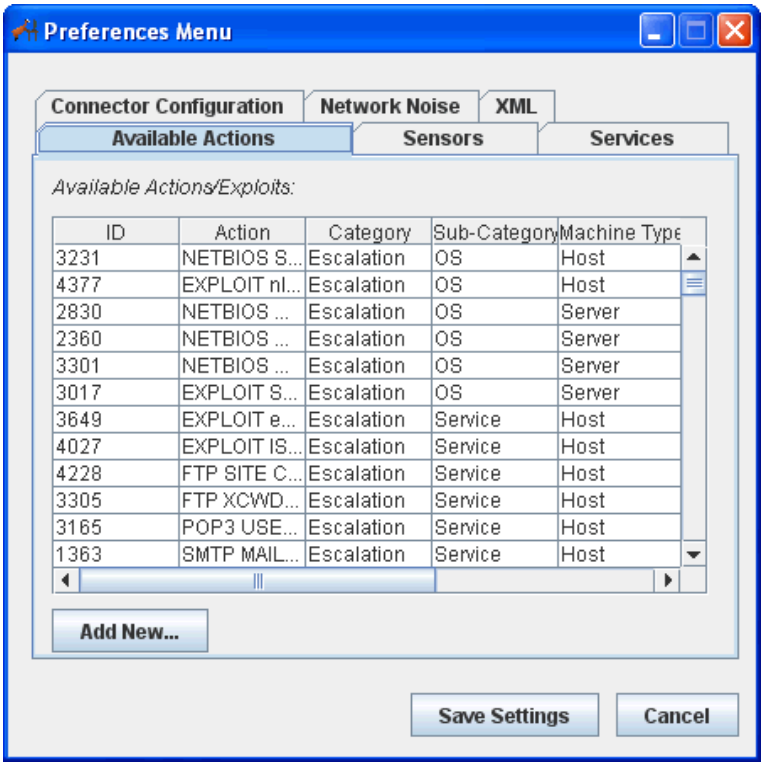

Figure A.3: Preferences Menu

A.2 Building a Network

This section walks through the process of constructing an example network. Individuals reading this user guide should follow along with the process outlined in this and successive sections.

The first step in creating a network is to choose the “Create Network” option from the File menu. This option invokes a window to appear, as displayed in Figure A.4, in which the basic network details are setup. Enter “My Network” as the network name and press the create button. The application window now displays a network window, a toolbar of network-building buttons, and several other menus. The network window is where the graphical representation of the created network is displayed. The toolbar buttons invoke other menus to appear, and this toolbar can be dragged around to form a separate window.

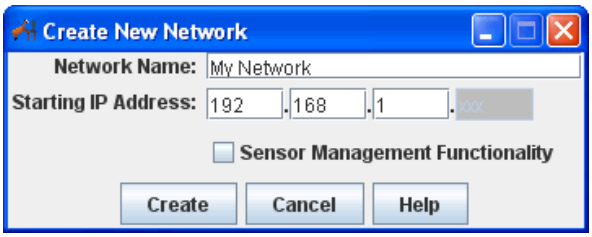

Figure A.4: Create Network Form

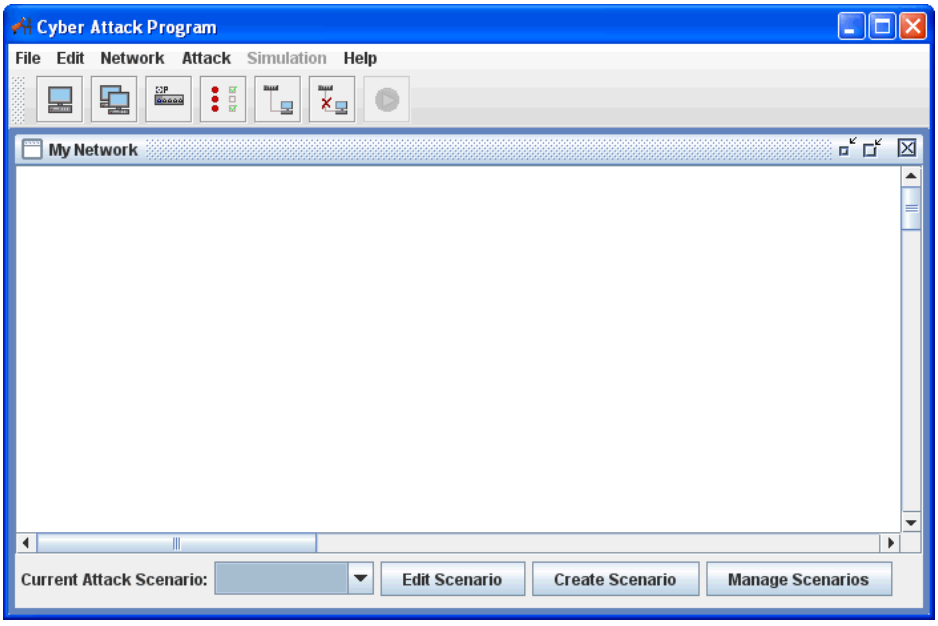

Figure A.5: Network Window

A connector is added to this network by clicking the “Create Connector” toolbar button highlighted in Figure A.6. This button invokes the “Create Connector Form” displayed in Figure A.7. In this form, enter “Router A” as the connector name, select the “Snort” option from the IDS combo-box, and click “Create”. When the form closes, a new connector block is added to the network window that graphically represents the connector specified.

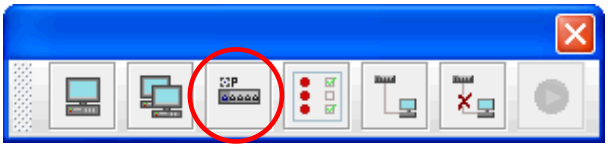

Figure A.6: Create Connector Button

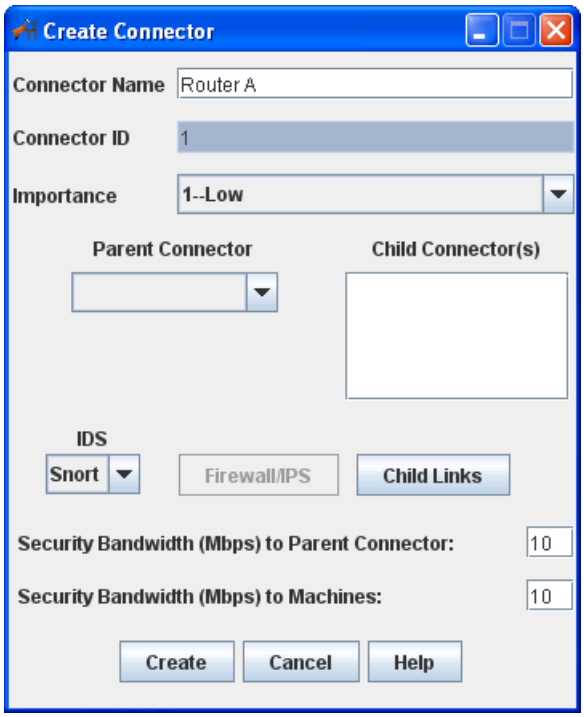

Figure A.7: Create Connector Form

Similarly, a machine is added by selecting the “Create Machine” toolbar button indicated in Figure A.8. This button invokes the “Create Machine Form” displayed in Figure A.9. In this form, enter “HTTP Server” as the machine name, select “Router A” as

the connector, indicate that external access is true, select “Server” as the machine type, select “Snort” as the IDS sensor, select “Windows XP” as the operating system, and select the “Services” button to open the service options menu. In the services menu, select the “Web”, “SSH”, “ICMP”, and “Apache” options and click the “Save” button. This option returns the view to the create machine form, at which point the “Save” button can be pressed to create the specified machine.

Figure A.8: Create Machine Button

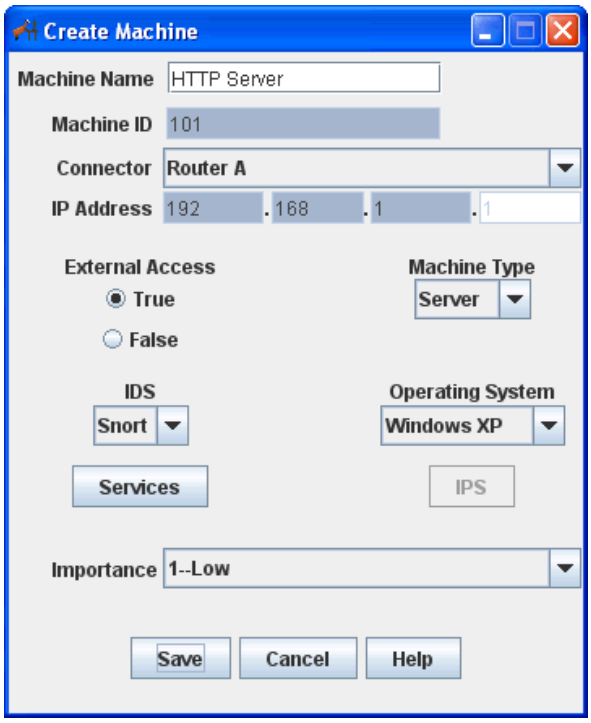

Figure A.9: Create Machine Form

In addition to the connector block, a machine block is created in the network window that is connected directly with “Router A” as displayed in Figure A.10. These blocks can be dragged around as desired while still maintaining the connection.

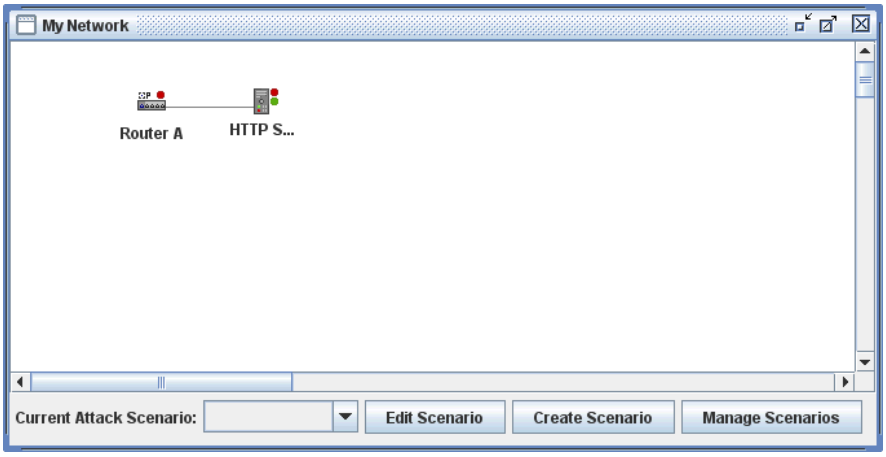

Figure A.10: Network View

To illustrate the manual connection process, create another connector named "Router B" with the same attributes as "Router A". This connector will appear in the network without any existing connections, as displayed in Figure A.11. Select the "Create Connection" toolbar button that is indicated in Figure A.12 to enable the connection drawing mode.

Figure A.11: Additional Connector

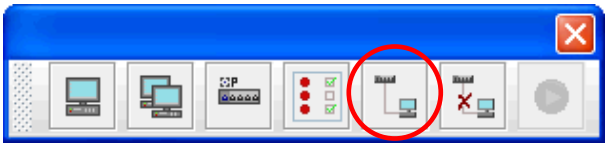

Figure A.12: Create Connection Button

In this mode, clicking on the desired child connector and then the parent connector establishes the appropriate child-parent relationship between the two connectors. Click on "Router B" and then on "Router A". This action results in a

connection drawn as displayed in Figure A.13 and indicates that “Router B” is a child connector of “Router A”.

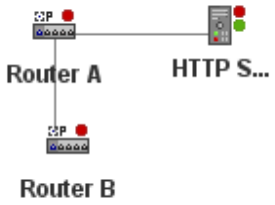

Figure A.13: Making a Connection

A whole subnet of machines is added to a network model by selecting the “Add Subnet” toolbar button indicated in Figure A.14. This action presents the window displayed in Figure A.15 to input the size of the subnet. Indicate “12” as the value and select the “Create” button.

Figure A.14: Create Subnet Button

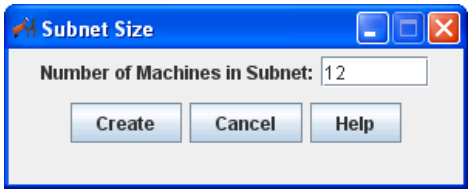

Figure A.15: Subnet Size Form

The next window displayed is the “Create Subnet Form”, which is displayed in Figure A.16. In this form, indicate “Staff” as the machine name. This name will be used for all machines within the subnet. Next, select “Router B” as the connector, indicate that the external access is false, select “Host” as the machine type, select “none” as the IDS sensor, and select “Windows XP” as the operating system. The services can be left at the

default values. Select the “Create” button to close the form and display a new subnet block in the network model. Figure A.17 shows that the “Staff” subnet includes a connection drawn to “Router B”, which represents a connection from each machine in the subnet with “Router B”.

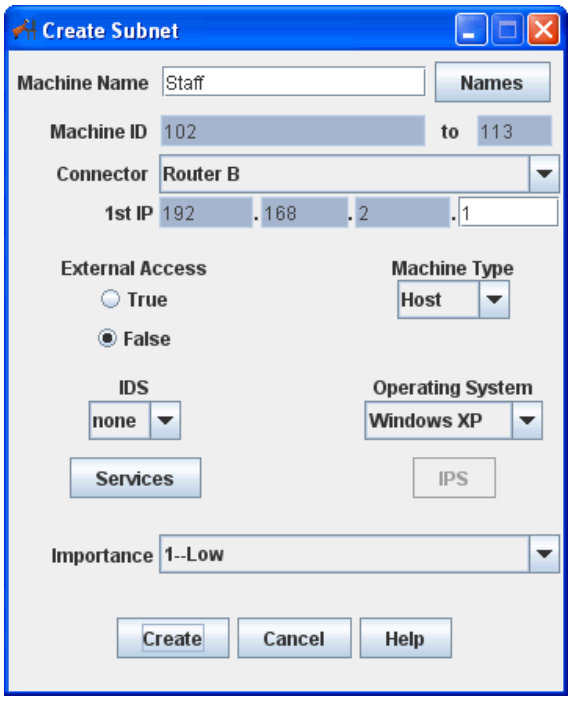

Figure A.16: Create Subnet Form

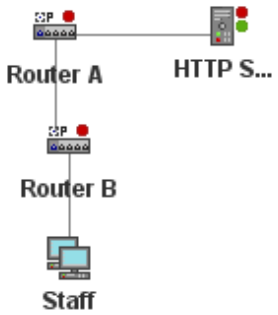

Figure A.17: Network with Subnet

Now that the basic principles of setting up a network structure have been covered, try adding additional network devices to create the network topology depicted in Figure A.18. The specific details of the devices are not important at this point. If a mistake is made in making a connection, use the “Delete Connection” toolbar button highlighted in Figure A.19 after selecting the child connector (or machine) of the connection.

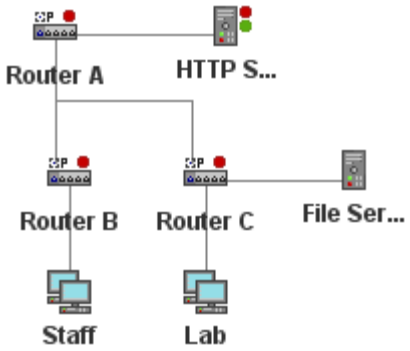

Figure A.18: Example Network Model

Figure A.19: Delete Connection Button

The created network structure can be saved by selecting either the “Save” or “Export Network…” option from the File Menu. The “Save” option will save the network as an object file (.obj) that can only be properly viewed with the Cyber Attack Simulator. The “Export Network…” option will save the network as a virtual terrain XML document that can be used by other applications and also imported back into this application with most of the initially defined details intact. The graphical placement of the devices will most likely be different, though.

A.3 Creating Attack Scenarios

Attack scenarios are created and managed in a modeled network using the attack scenario panel located at the bottom of the network window and indicated in Figure A.20.

Figure A.20: Attack Scenario Panel

A new attack scenario is created by selecting the “Create Scenario” button in this panel. This action brings up an option window, displayed in Figure A.21, that allows for the type of scenario to be specified. Select “Manual” to indicate that the scenario’s attacks will be specified separately.

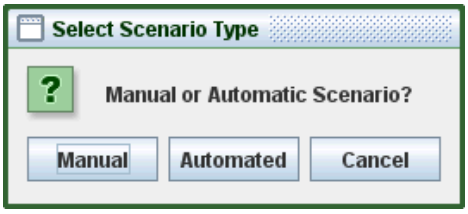

Figure A.21: Scenario Options

The manual option brings up the “Create Attack Scenario” form displayed in Figure A.22. This form includes a section to specify noise parameters and a section to specify attacks. Enter “My Scenario” as the scenario name and also enter “15” for the intrusion noise percentage. The reconnaissance percentage adjusts to 85. Additionally, enter “200” as the number of alerts per hour and “2” as the number of minutes to run the scenario after the attacks complete. Next, select the “Add Attack” button to add a new attack to this scenario.

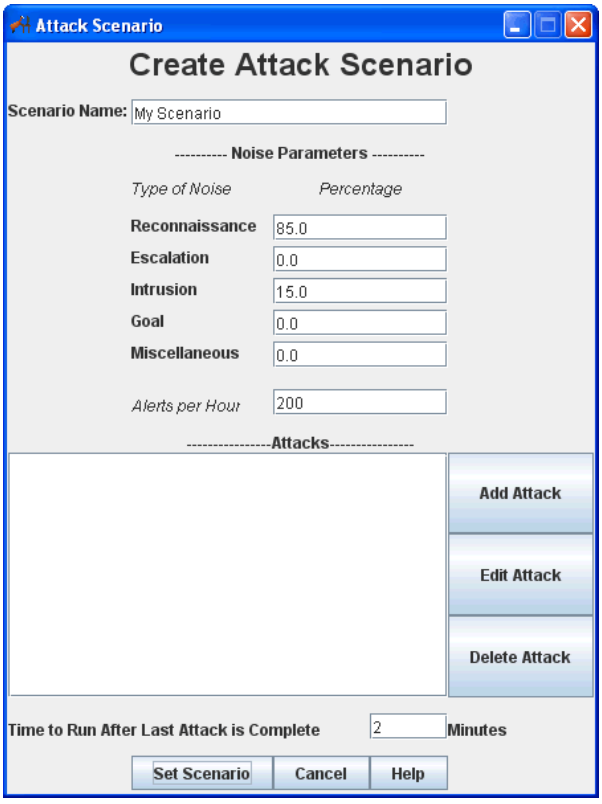

Figure A.22: Create Scenario Form

Adding a new attack will first invoke an option window, displayed in Figure A.23, where the type of attack is selected. Select the “Manual” button to proceed with a manual attack where the individual steps are specified.

Figure A.23: Attack Options

The resulting “Setup Manual Attack” form is displayed in Figure A.24. Enter “Attack A” as the attack name, select the option to use individual attack times, and indicate “3” as the time (in minutes) to delay the attack. Individual attack steps are added to the form’s step table by clicking the “Add Step” button.

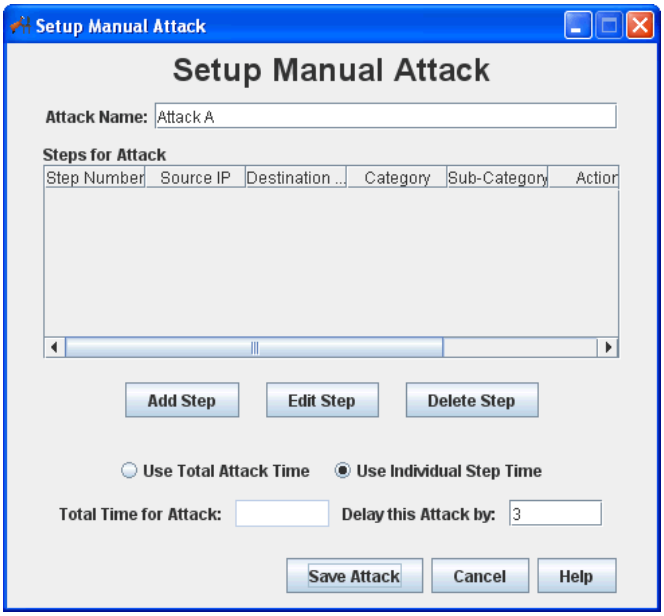

Figure A.24: Manual Attack Form

The “Add Step” button opens the “Attack Step” form where the detailed step information is specified. Ensure that the “Filter out illogical actions” option is selected and choose “External” as the source IP and “192.168.1.2” as the target IP. This target IP represents the “HTTP Server” within the “My Network” network model. Next, select “Recon” and “Scanning” as the primary and secondary action category to filter out the list of available actions. Select “ICMP PING Windows” as the action performed in this attack step. Lastly, indicate a constant step time of “1” and click the “Add Step” button.

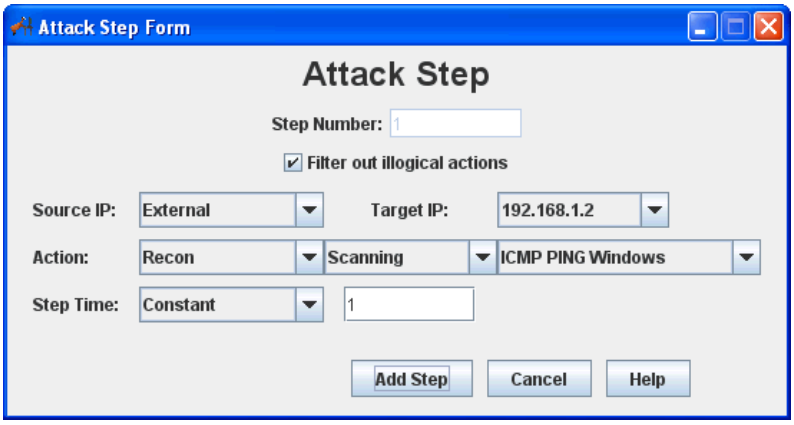

Figure A.25: Attack Step Form

This specified attack step is added to the list of attacks in the “Setup Manual Attack” form. Add a few additional attack steps similar to the list of steps displayed in Figure A.26. Select “Save Attack” on the “Setup Manual Attack” form to add the attack to the scenario and return to the “Create Attack Scenario” form.

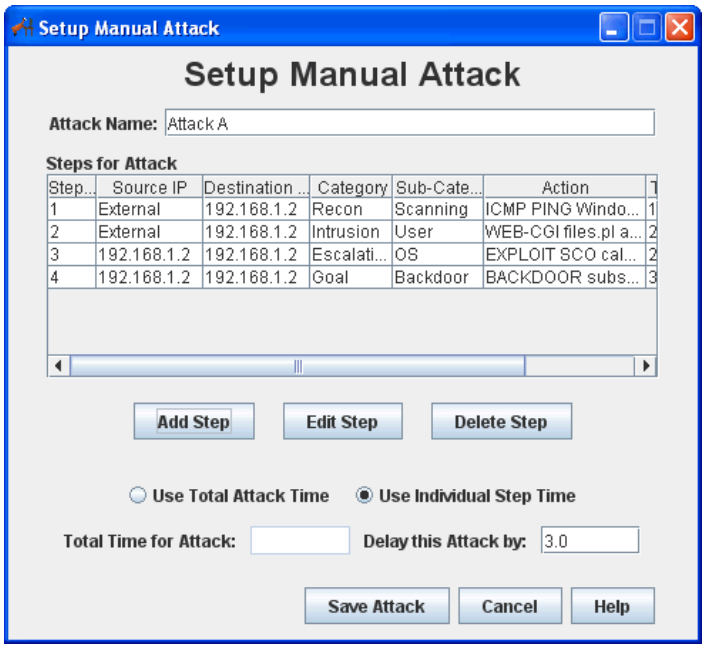

Figure A.26: Manual Attack Form with Attack Steps

From the “Create Attack Scenario” form, select the “Add Attack” button again. This time, though, choose to create an “Automated” attack. This option opens the “Setup Automatic Attack” form displayed in Figure A.27. In this form, enter “Attack B” as the attack name, “0.6” as the efficiency, “0.8” as the stealth, and “1.0” as the skill. Select “192.168.2.8” as the target machine. This represents the eighth machine in the “Staff” subnet. Also, select “Dos” (Denial of Service) as the goal type. Enter a total attack time of “15” and a start time of “10”. Click the “Save Attack” button to add this attack to the scenario and return to the “Create Attack Scenario” form. This form now displays both “Attack A” and “Attack B”, as illustrated in Figure A.28.

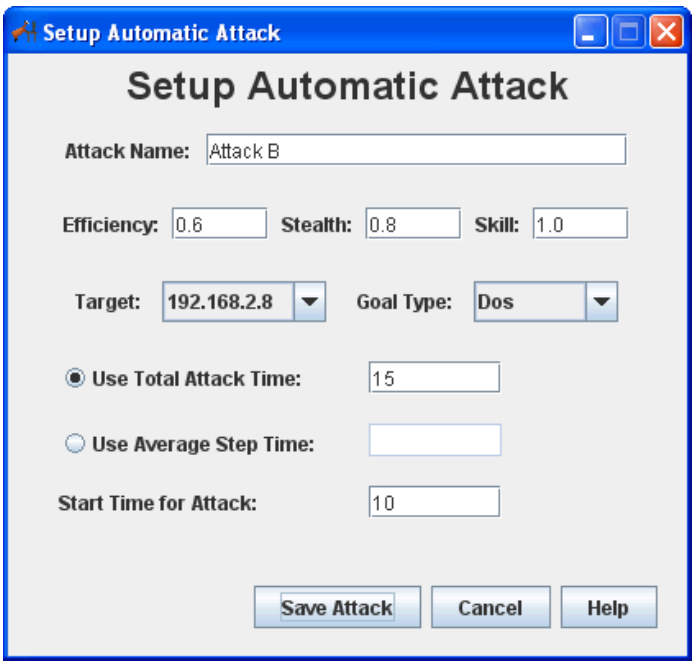

Figure A.27: Automatic Attack Form

Figure A.28: Attack Scenario Form with Attacks

Press the “Set Scenario” button to make this attack scenario the active scenario for the network. This network can also be edited later if needed by selecting the “Edit Scenario” button in the network window. The “Manage Scenarios” button in the network window allows for individual scenarios to be copied or deleted.

A.4 Running a Simulation

After an attack scenario has been specified and set as the active scenario for the network, the scenario can be run (simulated) by selecting the “Run Simulation” toolbar button highlighted in Figure A.29.

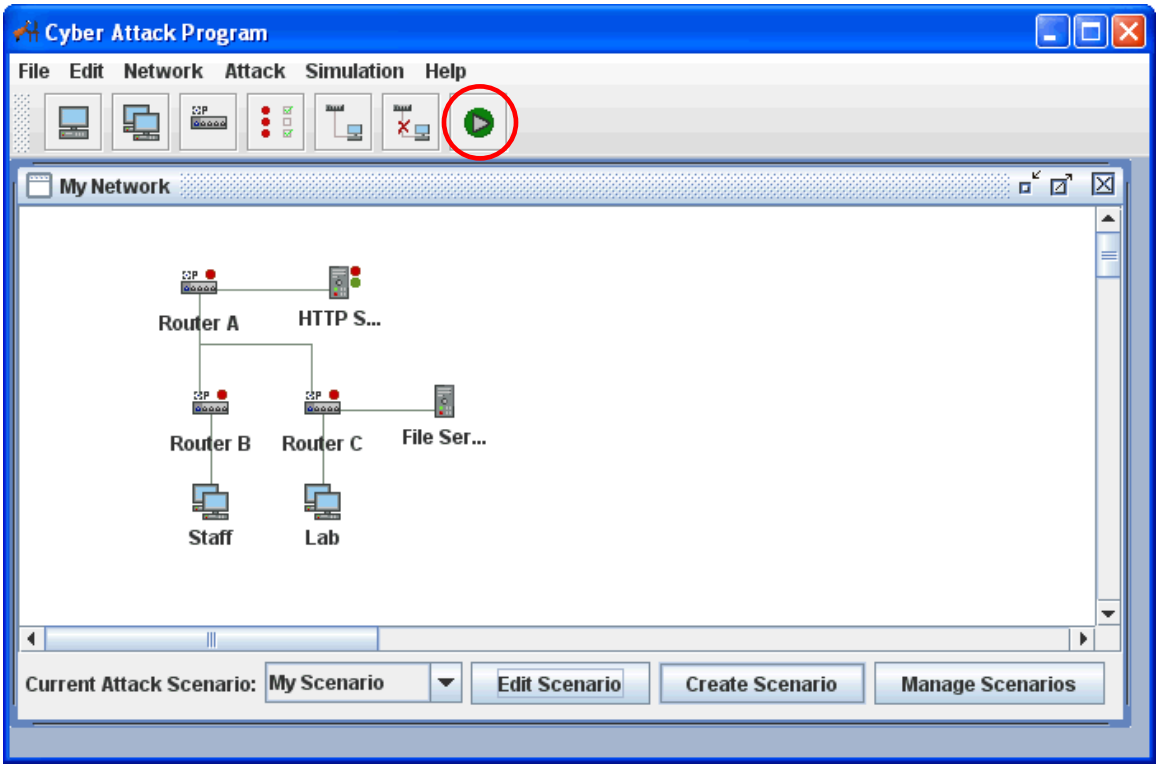

Figure A.29: Network with Attack Scenario

In running an attack scenario, an option window (Figure A.30) is first displayed that allows the scenario to be run interactively or to simply proceed to the results.

Running directly to the results uses default simulation options. Select “Interactively” from this window.

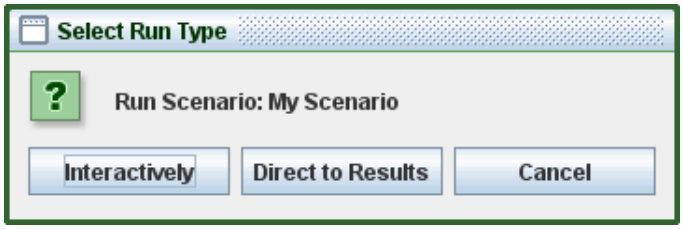

Figure A.30: Attack Generation Display

An interface is provided that consists of three sections (tabs). The first section is displayed in Figure A.31 and represents the process of initializing attacks and generating the automated attacks. Press the “Initialize Attacks” button to perform this process. Click the “Next >” button when the messages box indicates that the initialization is complete.

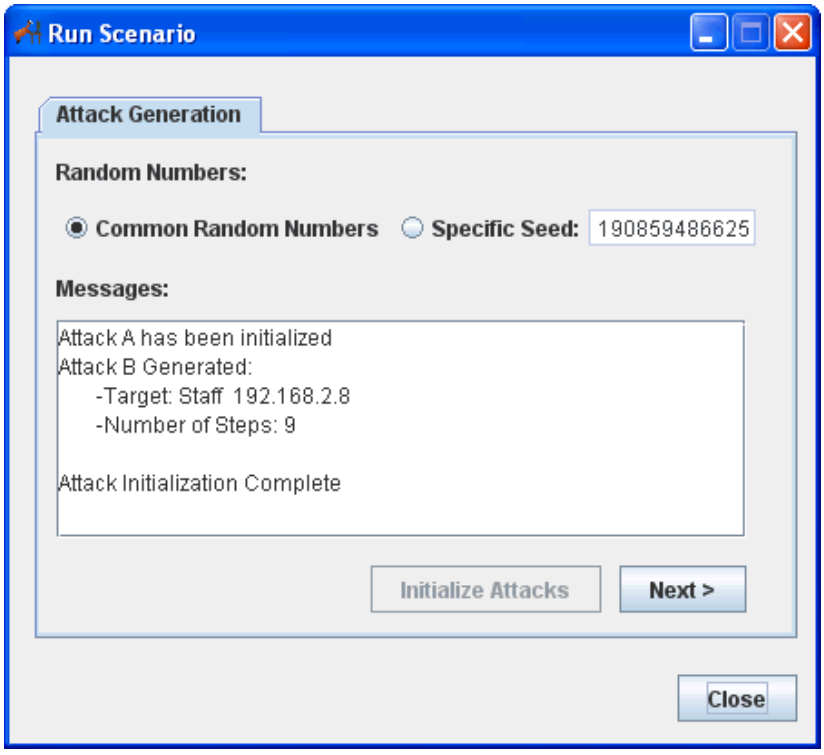

Figure A.31: Attack Generation Display

The next section represents the actual scenario implementation where the attack and noise actions are simulated through the network model. This section is displayed in Figure A.32. Press the “Run Scenario” button to begin the simulation. Attack-based actions are displayed in the messages box. Upon completion of the scenario, click the “Next >” button to proceed to the final section.

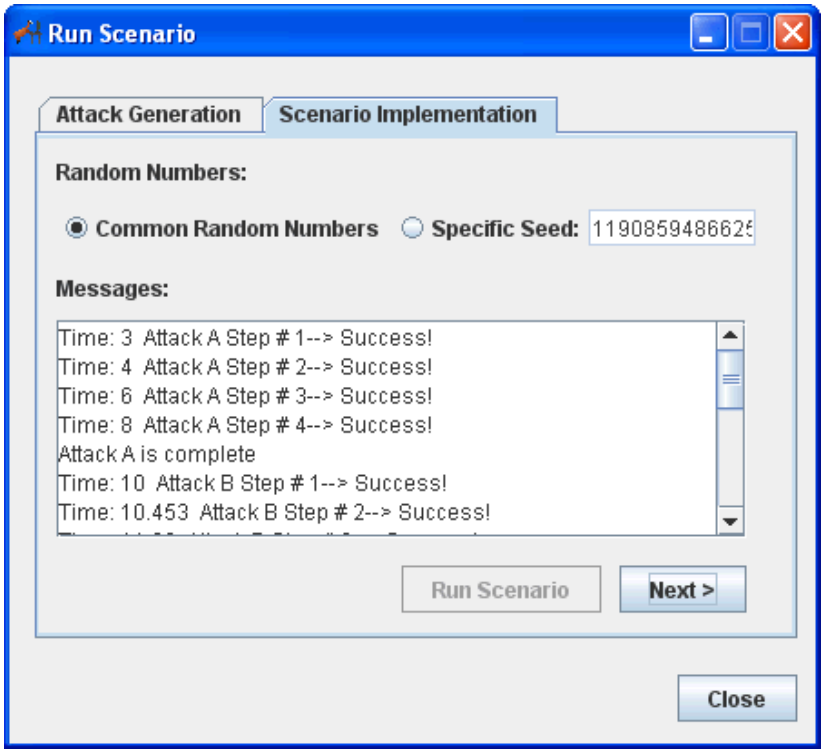

Figure A.32: Scenario Implementation Display

The final section in the simulation interface includes the results of the simulation run. This section is displayed in Figure A.33 and includes a listing of the ground truth file and sensor alert files output by the application. Each file can be individually viewed (as a text file in Notepad) by selecting the file and clicking the “View Result” button.

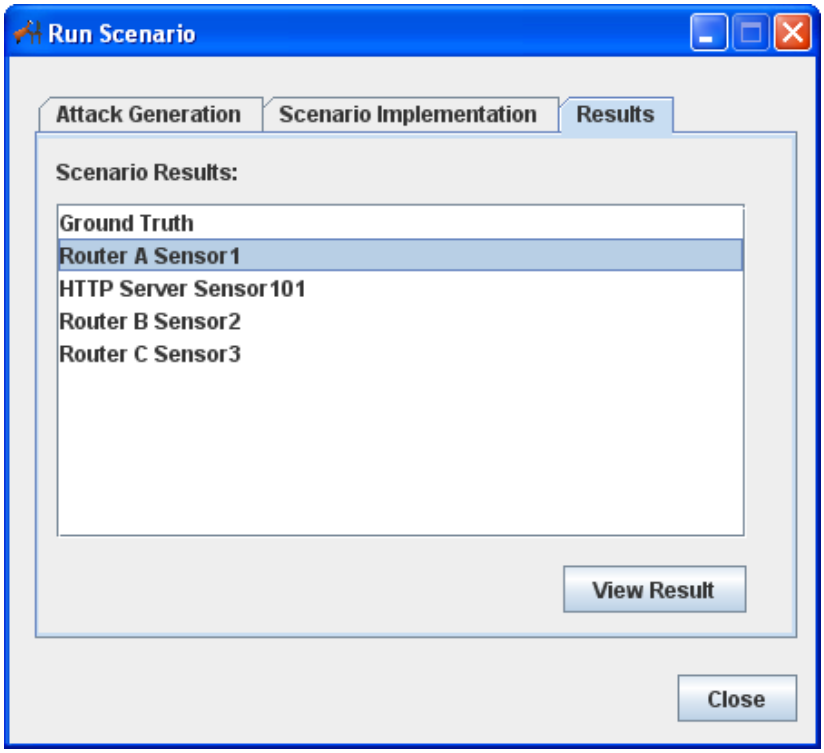

Figure A.33: Scenario Results Display

Appendix B: Software CD

This thesis includes a CD containing the software developed over the course of this research and any dependencies this software has. The CD contains these primary folders:

- **CAS Source –** The complete source code for the Cyber Attack Simulator. This includes all classes for the six packages and all images used in the user interface.
- **CAS Executable –** An executable version of the Cyber Attack Simulator that can be run on any machine supporting java. A Windows-based java runtime environment is also included to allow the simulator to run with the same java virtual machine as it was tested with. Also, the JDOM library is included with the executable file as it is a dependency for working with XML. An example network is provided that can be opened with the Cyber Attack Simulator.