A Hybrid and Cross-Protocol Architecture with Semantics and Syntax Awareness to Improve Intrusion Detection Efficiency in Voice over IP Environments

Bazara I. A. Barry

This thesis is submitted in partial fulfillment of the academic requirements

for the degree of

Doctor of Philosophy in Electrical Engineering

in the Faculty of Engineering and The Built Environment

University of Cape Town

August 2008

As the candidate's supervisor, I have approved this dissertation for submission.

Name: Professor H. Anthony Chan

Signed: Signed by candidate

Date: _ August 12 2008_____

Declaration

I hereby declare that: (1) the above thesis is my own unaided work, both in conception and execution, and that apart from the normal guidance of my supervisor, I have received no assistance apart from that stated below; (2) except as stated below, neither the substance or any part of the thesis has been submitted in the past, or is being, or is to be submitted for a degree in the University or any other University.

I am now presenting the thesis for examination for the Degree of (PhD) in Electrical Engineering. I also grant the University free license to reproduce the above thesis in whole or in part, for the purpose of research.

Bazara I. A. Barry	August 15 2008
Name	Date

To my family, who firmly stood by and believed in me.

Abstract

Voice and data have been traditionally carried on different types of networks based on different technologies, namely, circuit switching and packet switching respectively. Convergence in networks enables carrying voice, video, and other data on the same packet-switched infrastructure, and provides various services related to these kinds of data in a unified way. Voice over Internet Protocol (VoIP) stands out as the standard that benefits from convergence by carrying voice calls over the packet-switched infrastructure of the Internet.

Although sharing the same physical infrastructure with data networks makes convergence attractive in terms of cost and management, it also makes VoIP environments inherit all the security weaknesses of Internet Protocol (IP). In addition, VoIP networks come with their own set of security concerns. Voice traffic on converged networks is packet-switched and vulnerable to interception with the same techniques used to sniff other traffic on a Local Area Network (LAN) or Wide Area Network (WAN). Denial of Service attacks (DoS) are among the most critical threats to VoIP due to the disruption of service and loss of revenue they cause. VoIP systems are supposed to provide the same level of security provided by traditional Public Switched Telephone Networks (PSTNs), although more functionality and intelligence are distributed to the endpoints, and more protocols are involved to provide better service. A new design taking into consideration all the above factors with better techniques in Intrusion Detection are therefore needed.

This thesis describes the design and implementation of a host-based Intrusion Detection System (IDS) that targets VoIP environments. Our intrusion detection system combines two types of modules for better detection capabilities, namely, a specification-based and a signature-based module. Our specification-based module takes the specifications of VoIP applications and protocols as the detection baseline. Any deviation from the protocol's proper behavior described by its specifications is considered anomaly. The Communicating Extended Finite State Machines model (CEFSMs) is used to trace the behavior of the protocols involved in VoIP, and to help exchange detection results among protocols in a stateful and cross-protocol manner. The signature-based module is built in part upon State Transition Analysis Techniques which are

used to model and detect computer penetrations. Both detection modules allow for protocol-syntax and protocol-semantics awareness. Our intrusion detection uses the aforementioned techniques to cover the threats propagated via low-level protocols such as IP, ICMP, UDP, and TCP.

An intrusion detection prototype is implemented using a network simulator for proof-of-concept of the proposed architecture. The network simulator is also used to launch a variety of attacks against hosts equipped with our IDS. The system shows excellent hit rate and performs well under both low- and high-load conditions causing little overhead to the network. The performance of the network is not impeded or degraded in any way that badly affects the Quality of Service (QoS). The results presented in this thesis show that the IDS has only a minor impact on the performance of the protected hosts.

Acknowledgements

First and foremost, I thank the Almighty God for the grace He bestowed upon me, without which, this work would not have been possible.

I am indebted to my advisor, Prof. H. Anthony Chan, for the outstanding motivation, guidance, support, and knowledge he has provided throughout the course of this work. He kindly took me into his group and provided me with a great amount of freedom to work on things that I liked.

The laboratory of the Communications Research Group (CRG) provided an ideal research environment for my PhD study at University of Cape Town. I am particularly grateful to Mr. Neco Ventura for his support and trust which allowed me to overcome many of the research hurdles.

It would be unfair if I mentioned some of the names of my colleagues at the CRG lab. My thanks go to all of them. Their ideas, efforts, and contributions have significantly added to my work.

I am extremely grateful to The Ministry of Higher Education – The Republic of The Sudan and University of Khartoum for their generous financial support during this study.

Last, but not the least, I thank my parents and my sisters. It has been their lasting love and support that enabled me to reach this point.

Table of Contents

List of Figures

List of Tables

Glossary

Access Control: Protection of system resources against unauthorized access; a process by which use of system resources is regulated according to a security policy and is permitted by only authorized entities (users, programs, processes, or other systems) according to that policy.

AOR: An Address Of Record (AOR) is a SIP Uniform Resource Identifier (URI) that points to a domain with a location service that can map the URI to another URI where the user might be available. Typically, the location service is populated through registrations. An AOR is frequently thought of as the "public address" of the user.

Authentication: The process of verifying an identity claimed by or for a system entity.

Authorization: An authorization is a right or a permission that is granted to a system entity to access a system resource. An authorization process is a procedure for granting such rights.

Availability: The property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system; i.e., a system is available if it provides services according to the system design whenever users request them.

CODEC (Coder/Decoder): Hardware or software that converts analogue signals (e.g., video, audio) to or from digital form for transmission or storage. It may perform compression or other optimizations such as silence suppression.

Cryptography: The study of ways to convert information from its normal, comprehensible form into an obscured guise, unreadable without special knowledge.

CSRC: Contributing Source is a source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer.

Data Integrity: The property that data have not been changed, destroyed, or lost in an unauthorized or accidental manner.

DoS: A Denial-of-Service attack is an attempt to make a computer resource unavailable to its intended users.

IDS: Intrusion Detection System is software/hardware that detects and logs inappropriate, incorrect, or anomalous activity.

IETF: The Internet Engineering Task Force is the body that defines standard Internet operating protocols such as TCP/IP. The IETF is supervised by the Internet Architecture Board (IAB). Standards are expressed in the form of Requests for Comments (RFCs).

IP Softphones: IP Softphone makes it easy to place and receive phone calls on your PC or laptop, making it ideal for working from any location with an internet connection.

ITU-T: The Telecommunication Standardization Sector of the International Telecommunication Union is the primary international body for fostering cooperative standards for telecommunications equipment and systems. It was formerly known as the CCITT.

Jitter: The variation in the time between packets arriving caused by network congestion, timing drift, or route change. Queuing delay and processing time experienced by each packet are the main contributors to jitter.

LAN: A Local Area Network (LAN) is a group of computers and associated devices that share a common communications line or wireless link.

Mixer: An intermediate system that receives RTP packets from one or more sources, possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet.

Mobile ad-hoc network: A mobile ad-hoc network is a kind of wireless ad-hoc network, and is a self-configuring network of mobile routers (and associated hosts) connected by wireless links – the union of which forms an arbitrary topology.

OLSR: The Optimized Link State Routing Protocol (OLSR) is an IP routing protocol specified in RFC 3626, and is optimized for mobile ad-hoc networks but can also be used on other

wireless ad-hoc networks.

Packetization: The process of filling a packet payload with encoded/compressed data. It transforms a native IP core transaction into packets.

PSTN: Public Switched Telephone Network is an umbrella term that represents the carriers that make up the worldwide telephone service.

RFC: Request For Comments is a memorandum published by the Internet Engineering Task Force (IETF) describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems.

Serialization: The process of clocking a voice or data frame onto the network interface. It is directly related to the clock rate on the trunk.

SLA: A Service Level Agreement is a formally negotiated agreement between two parties such as customers and their service providers. It may specify the levels of availability, serviceability, performance, operation, or other attributes of the service like billing and even penalties in the case of violation of the SLA.

SPIT: SPam over Internet Telephony is unsolicited bulk messages broadcast over VoIP to phones connected to the Internet.

URI: A Uniform Resource Identifier (URI) is a compact string of characters used to identify or name a resource on the Internet. The main purpose of this identification is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.

Virus: A computer virus is a program that copies itself into (infects) other programs. It may or may not perform other tasks. It can be passed by infected files on a disk, by files download from another computer, or by attachments sent via e-mail.

WAN: A Wide Area Network is a geographically dispersed telecommunications network. The term distinguishes a broader telecommunication structure from a local area network (LAN).

WiMAX: Also known as the IEEE 802.16 group of standards, defines a packet-based wireless technology that provides high-throughput broadband connections over long distances.

Worm: A computer worm is a program that spawns running copies of itself over a computer network and usually performs malicious actions, such as using up the computer's resources and possibly shutting the system down.

Chapter 1 Introduction

1.1 Intrusion Detection Systems: Definitions and Taxonomies

The threat of intruding into computer systems is gaining momentum in today's interconnected networks. Due to the rise in connectivity, more and more systems are subject to attacks by intruders. Anderson, who introduced the concept of intrusion detection in 1980 [1], defined an intrusion attempt or a threat to be the potential possibility of a deliberate unauthorized attempt to:

1. Access information,
2. Manipulate information, or
3. Render a system unreliable or unusable.

The role of an Intrusion Detection System (IDS) is to detect such attempts, and to inform system administrators about such threats in order to take countermeasures.

Although many attacks and intrusions can be prevented if proper authentication and cryptographic methods are used, inevitably, the best intrusion prevention system will fail. Therefore, intrusion detection systems should be used as a second line of defense with intrusion prevention systems to provide a securer environment. The significance of IDSs stems from many factors including the following:

1. Building a completely secure system is still a dream far from being realized. Software reuse, relying on Internet protocols designed without security awareness in mind, and the need to connect to outsiders make the job extremely difficult.
2. Cryptographic methods have their own problems. Passwords can be cracked, users can lose their passwords, and entire crypto-systems can be broken [2].
3. Authentic insiders can abuse their privileges.

4. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.

5. Intrusion detection enables the collection of information about the intrusion techniques that can be used to strengthen the intrusion prevention facility [3].

1.1.1 Classification of Attacks and Intrusions

Since intrusion detection systems deal with the threat of attacks and intrusions, it is in order to take a closer look at these dangerous activities. The intent is to provide a better understanding of attacks that an organization or an individual may need to counter.

Based on their effect, attacks can be categorized as passive and active attacks. Passive attacks try to obtain information that is being transmitted without altering or affecting system resources. Therefore, they violate the confidentiality rules of the system. Passive attacks are accomplished by monitoring a system performing its tasks and collecting information regarding its operation. Once this information is collected, it may be used later to attack the same system or one related to it. There are several types of passive attacks, however, we mention two of the most prevalent ones, namely, eavesdropping and traffic analysis. Eavesdropping is accomplished by monitoring traffic passing through a network, and aims at revealing the contents of the messages being transmitted. Eavesdropping attempts can be blocked if the contents of the transmitted messages are properly encrypted. Traffic analysis is another type of passive attacks where an opponent tries to determine the nature of the messages being exchanged and the identities of the communicating parties by observing the pattern of the messages. Traffic analysis could be accomplished even if encryption is applied to the contents of messages.

Active attacks on the other hand attempt to alter system resources or affect their operation violating several security services such as authentication, access control, data integrity, and availability. Some of the most common types of active attacks include masquerade, message content modification, and denial of service. A masquerade attack is a type of attack in which one system entity illegitimately poses as another entity to gain access to confidential systems. In message content modification attacks the attacker removes a message from the wire, modifies it, and then resends it. Either the content or the recipient of the message is modified. Denial of

service attacks are incidents in which legitimate users are deprived of the services of a resource they would normally expect to have. Denial of service attacks can be accomplished by disabling the network or overloading it with messages so as to degrade performance. The severity of Denial of Service attacks vary from making the victim unresponsive for a few seconds to causing serious damage such as crashes and reboots.

Another possible classification of attacks is the one based on the origin of the attack. In this context attacks can be classified as internal or external. Internal attacks are originated from the enterprise own employees, business partners, customers, or any entity that has an authorized access to some of the enterprise resources. External attacks on the other hand come from outside, usually via the Internet or a Wide Area Network (WAN) [73].

1.1.2 Classification of Intrusion Detection Principles

Intrusion detection systems can be classified based on the detection principle into three main types as shown in Figure 1.1:

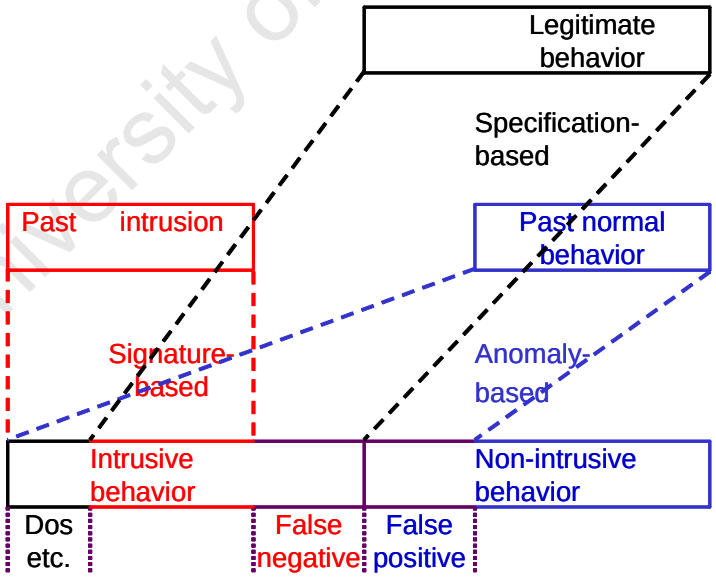


Figure 1-1. The three main approaches in intrusion detection

1.1.2.1 Anomaly-based Detection

Anomaly-based Detection approach explores issues in intrusion detection associated with

deviation from normal system or user behavior. Anomaly detection's philosophy is that, if we could establish a normal profile for a system, we could in principle report all system states varying from the normal profile as intrusion attempts. Anomaly-based detection has the advantage of detecting previously-unknown attacks but at the cost of relatively high false alarm rate, which is due to the fact that systems often exhibit legitimate but previously-unseen behavior. These false alarms are indicated as false positives in Figure 1.1. The main issues in anomaly detection are to select the threshold upon which we measure the deviation from the normal profile and to select the features to monitor.

1.1.2.2 Signature-based Detection

Instead of modeling the system's normal behavior, signature-based detection technique models intrusive behaviors in the form of patterns or signatures. Consequently, such signatures are used by the system to distinguish between normal and anomalous behavior. A signature is a pattern that we look for in traffic received or activities performed by a system. For example, if administrators want to ban a certain IP address from establishing a connection with monitored systems, that IP address could serve as a pattern we look for in incoming packets. Another more complex example could be a Denial of Service (DoS) attack on a mail server with the attack issuing the same command thousands of times. A possible signature for such attacks is to count how many times the command is issued and raise an alarm when that number of times exceeds a certain threshold. Signature detection is capable of detecting all known attack patterns, but is of little use for unknown ones, as no signatures are available for such attacks [4]. The failure to detect such unknown attacks is indicated as false negatives in Figure 1.1. The main issue in signature-based detection is to write a signature that encompasses all possible variations of a certain attack and does not match non-intrusive behavior.

1.1.2.3 Specification-based Detection

A third technique that has been overtaking both previous approaches is intrusion detection by static program analysis which was first proposed by Wagner and Dean [5]. This technique performs a static analysis of the program to create an abstract automata model of the functions and system calls. The program's behavioral specifications are used to create the model and also used as a basis to detect attacks. If the program executes a function or invokes a system

call which violates the model, the IDS assumes that an intruder has corrupted the program. This approach has become more mature with the inputs of Balepin et al [7] and has had the name specification-based intrusion detection. Specification-based technique has the capacity to detect previously-unseen attacks with the lowest false alarm rate. This advantage is due to the programmatic nature of the IDS which contains a model that represents all possible legal paths through the program or the protocol session ensuring that any detected deviation from the model is not caused by the program's code but by code inserted by a bug or an attacker. In a narrow sense, specification-based detection means looking for behavior in network traffic that is peculiar in terms of the specification for the protocol the traffic is using. In this case, detection is interested in syntax violation. In a broader sense, the term could mean applying anomaly detection on the semantics of traffic as expressed using the protocol. In this approach, traffic is not peculiar due to a particular protocol element it is using, but rather what in aggregate it is trying to achieve with the protocol. Semantics violations are the main concern here [8]. Despite their obvious advantages, specification-based systems are not very effective against certain types of attacks such as Denial of Service (DoS) and failed logins.

Other types of intrusion detection systems can be safely classified under any of the abovementioned three main approaches.

1.1.2.4 Statistical Detection

Another name for this type of detection is Threshold detection. Statistical detection is one of the most rudimentary forms of intrusion detection. The goal of this type of detection is to record each occurrence of a specific event and detect when the number of occurrences of that event surpasses a reasonable amount that one might expect to occur within a specified time period. As the monitored system continues running, the statistical monitor builds a profile of normal statistical behavior by collecting descriptive statistics on a number of parameters. Such parameters can be the number of unsuccessful logins, the number of network connections, and the number of commands with error returns. Although statistical approaches have the advantage of adaptively learning the behavior of users and systems, they can gradually be trained by intruders so that eventually, intrusive behavior is considered normal. Furthermore, false positives and false negatives are generated depending on whether the threshold is set too low or too high.

Statistical detection can be considered a branch of anomaly detection.

1.1.2.5 Expert Systems

Expert systems contain a set of rules that describe attacks. Such rules are usually formulated by a security professional. A security professional translates the events of the system audit trail into facts and rules, which are used by the IDS inference engine to draw conclusions on system activities. This method increases the abstraction level of the audit data by attaching a semantic to it [64]. The rules of an expert system are as strong as the security personnel who program them, and hence there is a real chance that expert systems can fail to flag intrusions. Expert systems are considered signature-based detection systems.

1.1.2.6 Model-based Detection

Model-based intrusion detection systems attempt to model either proper or intrusive behaviors at a higher level of abstraction than audit records. The objective is to build scenario models that represent the characteristic behavior of systems or intrusions. The philosophy of this approach is that certain scenarios are inferred by certain other observable activities. If these activities are monitored, it is possible to find intrusion attempts by looking at activities that infer a certain intrusion scenario. Model-based systems can predict the attacker's next move based on the intrusion model, which allows security officers to take preventive measures. However, patterns for intrusion scenarios must be easily recognized and must not be associated with any other normal behavior. Model-based approaches can be used to implement specification-based and signature-based intrusion detection systems, however they differ from signature-based approaches in that they do not simply pattern match audit records to expert signatures [9].

1.1.3 Classification of the Monitored Resources

Considering the resources they monitor, IDSs can also be differentiated into the following popular categories:

1.1.3.1 Network Intrusion Detection System (NIDS)

NIDSs are placed at strategic points within the network such as inside a firewall, outside it, or at the perimeter of a network. Their aim is to monitor traffic to and from all devices on the network. Ideally, NIDSs would scan all inbound and outbound traffic; however doing so might

create a bottleneck that would impair the overall performance of the network. Therefore, researchers have been devising new methods to improve memory management and packet processing capabilities in NIDSs. Figure 1.2 shows a very basic but common scheme which is potentially able to detect attacks from the external Internet.

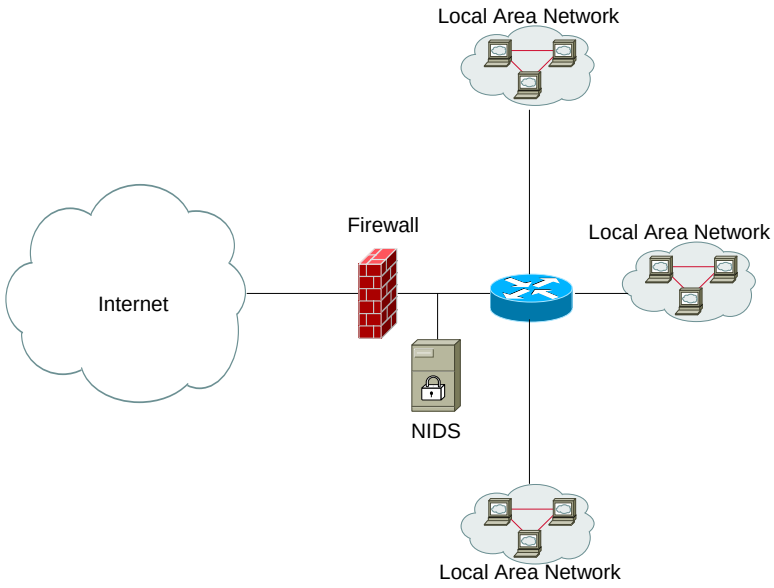


Figure 1-2. Typical deployment scheme of a network intrusion detection system

1.1.3.2 Host Intrusion Detection System (HIDS)

HIDSs are installed locally on host machines. They evaluate the activities and access to individual hosts or devices in the network. HIDSs are usually placed at business critical hosts and external servers. Four different types of host intrusion detection systems can be recognized in the literature.

1. File system monitors which check the integrity of files and directories.
2. Log file analyzers which analyze log files for patterns indicating suspicious activities.
3. Connection analyzers which monitor connection attempts to and from a host.
4. Kernel-based monitors which detect malicious activity on an operating system kernel level.

Figure 1.3 shows the abovementioned taxonomies.

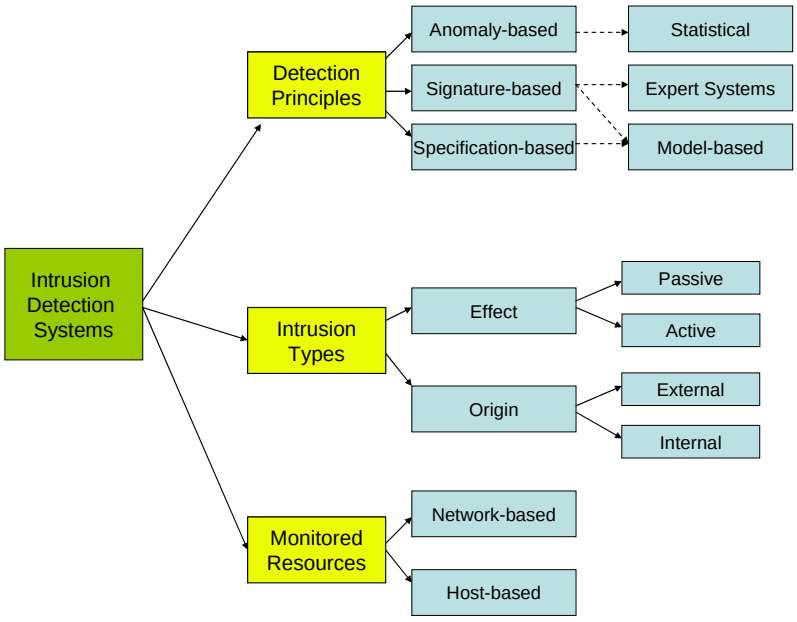


Figure 1-3. Taxonomies of Intrusions and Intrusion Detection Systems

1.1.4 Measurable Characteristics of Intrusion Detection Systems

Characteristics of IDSs can be measured quantitatively. Some of these characteristics are:

1. Coverage: Determines which types of attacks the IDS can detect under ideal conditions.

2. Probability of false alarms: Determines the rate of false alarms produced by an IDS in a given environment during a particular time frame. A false alarm is an alert caused by normal non-malicious background traffic.

3. Hit rate: Determines the rate of attacks detected correctly by an IDS in a given environment during a particular time frame.

4. Ability to handle high bandwidth traffic: Demonstrates how well an IDS will function when presented with a large volume of traffic.

5. Ability to detect novel attacks: Demonstrates how well an IDS can detect attacks that have not occurred before [10].

1.2 Convergence and VoIP: Advantages and Security Implications

Convergence in networks refers to the structures and processes that result from design and implementation of a common networking infrastructure that accommodates data, voice, and multimedia communications [11]. Network convergence is the first step towards application convergence which happens above the network layer. Convergence in applications refers to the building of applications that span over different protocols/specifications [12]. Voice over IP (VoIP) is emerging as a standard that benefits from convergence and replaces older PSTN systems.

From a management and maintenance point of view, VoIP networks and applications are less expensive than two separate telecommunications infrastructures. Although implementation can be expensive at the beginning, it is repaid in the form of lower operating costs and easier administration. For example, a single management station or cluster can be used to monitor both data and voice components and performance. However, this reliance upon the infrastructure provided by data networks makes VoIP susceptible to all the security flaws suffered by IP based applications. Furthermore, VoIP comes with its own challenges.

VoIP standards separate signaling and media on different channels. These channels run over dynamic IP address/port combinations and are controlled by different protocols each with its own security issues. In addition, VoIP distributes applications and services throughout the network. IP phones which are considered smart devices, can act as clients for a number of network protocols which means that more processing capabilities and intelligence are shifted to the edge of the network. It also dramatically increases the number of systems to be protected. This model is a reversal of the traditional security model, where critical data are centralized, bounded, and protected.

PSTN networks provided security via obscurity. The use of closed standards and proprietary solutions made it hard for attackers to penetrate such networks. In contrast, VoIP standards have a tendency towards openness and simplicity which gives attackers the opportunity to manipulate the protocols used to their advantage and use their very features to launch attacks. Availability is a key VoIP performance metric. Denial of Service attacks (DoS)

remain the most difficult VoIP threat to defend against [8].

All these unique features of VoIP applications have significant security implications which make VoIP more challenging to secure.

1.3 Thesis Approach

Our approach takes advantage of a combination of technologies to enhance the efficiency of intrusion detection in VoIP environments. It starts with developing host-based specifications for the protocols involved in SIP-based IP telephony starting from application layer protocols such as Session Initiation Protocol (SIP) and Real-time Transport Protocol (RTP), to transport and network layer protocols such as TCP, UDP, ICMP, and IP. The host-based architecture is encouraged by the current shift of interest in VoIP environments from the center to the edges of the network. The specifications are derived from the IETF Request For Comment documents (RFCs) which describe the protocols. The developed specifications form the basis of our specification-based detection modules. As can be seen from our earlier discussions, no intrusion detection approach stands alone to detect all computer penetrations. Each approach is technically suited to identify a subset of the security violations to which a computer is subject. Given the complementary nature of the strengths and weaknesses of signature-based and specification-based intrusion detection, we combine both approaches in such a hybrid way that we realize the combination of their strengths and avoid the weaknesses of either one.

Both specification-based and signature-based modules combine protocol-syntax and protocol-semantics anomaly detection techniques. Such a feature is vital in the detection process to cover all aspects of the protocols being monitored. It allows us to report any violations of the standards in the protocol packets, alongside reporting any deviation from the expected protocol behavior during a session.

Our specification-based detection modules are developed in part based on the Communicating Extended Finite State Machines (CEFSMs) model which allows us to represent each of the involved protocols as an Extended Finite State Machine (EFSM). An EFSM is a suitable structure to model the control flow of the protocol and its data, and to spot any deviation from the expected behavior. The Communicating Extended Finite State Machines model enables

a combination of stateful and cross-protocol intrusion detection. Stateful detection implies building up relevant state within a session and across sessions and using the state in matching for possible attacks. Cross-protocol detection allows the IDS to access packets from multiple protocols in a system to perform its detection. Both types of detection are important to keep the state of the session and to guard against anomalies of involved protocols.

On the other hand, our signature-based detection modules are based in part on describing an attack at the session level as a sequence of actions that the attacker performs to compromise the security of a computer system. This sequence of actions is best modeled by state transition analysis techniques. State transition analysis techniques represent an attack as well-defined states and transitions between these states. This representation allows signatures to complement the EFSM-based module in the specification-based component and to overcome some of its shortcomings. It also allows the creation of semantics-aware signatures.

As proof-of-concept of the proposed architecture, we implement an IDS prototype in a network simulator in addition to implementing a variety of attacks that target application, transport, and network layer protocols. The hit rate of the system is tested by launching these attacks against hosts equipped with our IDS prototype. The network simulator is also used to analyze the performance of the design quantitatively under various network conditions.

1.4 Thesis Scope and Limitation

Security threats in VoIP environments can be classified into three main categories:

1. **Network level threats**: Threats to VoIP at the network level are propagated via the use of low-level protocols such as IP, ICMP, UDP, and TCP. In most network architectures and corresponding communications protocol stacks, network layer protocol data units are transmitted in the clear, meaning that they are not cryptographically protected during their transmission. Consequently, it is relatively simple to do malicious things, such as inspecting the contents of the data units, forging the source or destination addresses, modifying the contents, or even replying old data units. IP as a network layer protocol is no exception to this. IP has no built-in security features such as authentication and encryption unless network administrators choose to enable IP Security (IPSec) which is disabled by default.

The lack of built-in security for IP packets makes it relatively easy to launch attacks such as malformed packets, flooding, denial of service, and buffer overflow attacks, which can result in a full or partial service loss.

2. **Application and protocol specific threats**: VoIP protocols at the application level can be classified according to their role during the session. Signaling protocols are responsible for call setup, tear down, and modification. Media transport protocols are involved in end-to-end transport of voice and multimedia data. In addition to that, support protocols are used to enable services and features required for proper network operation. A major source of vulnerabilities in signaling protocols is that they transmit packet headers and payloads in clear text by default. Media transport protocols introduce several vulnerabilities due to the absence of authentication and encryption [13]. Even support protocols transmit sensitive data such as session IDs in clear text by default. It is therefore easy for attackers to cause call termination and call flooding as well as to spoof caller ID or to exercise other attacks.

3. **Content related threats**: These threats can be well-presented by Spam over Internet Telephony (SPIT), and malicious payloads containing worms and viruses. A SPIT threat sends unsolicited calls to legitimate users. The calls contain mostly pre-recorded messages that annoy people or congest a voice mail system to overflowing. Viruses and worms cause Denial of Service (DoS) conditions due to the network traffic they generate as they replicate and seek out other hosts to infect. The lack of proper authentication at the application level and the poor network-level security control on the internal IP network make it easy respectively to spoof e-mail senders and to propagate viruses and worms.

This thesis is concerned with both network level threats and VoIP-applications and protocol specific threats. In today's VoIP networks, two major application level signaling protocols dominate: SIP and H.323. Our research is confined to threats in SIP and its accompanying media transport protocol RTP. The threats covered by this thesis include:

1. Message flow attacks which are used by attackers to exploit vulnerabilities in the flow of messages used by signaling and transport protocols.

2. Parser attacks which aim at hampering proper parsing by constructing invalid messages.

3. Flooding attacks which are used by attackers to deny legitimate users access to network resources.

Our specifications for SIP and RTP are based on RFC 3261 [14] and RFC 1889 [15] respectively.

The same techniques used to address threats at the application level are used at the network level to detect its attacks. For that purpose, we cover some of the threats posed by IP, ICMP, UDP, and TCP. Content related threats such as SPIT, worms, and viruses are beyond the scope of this research.

This thesis is not concerned with flaws in particular implementations of the protocols. Rather, generic problems of the protocols themselves are discussed, and various requirements for intrusion detection systems are addressed. For the most, there is no discussion of vendor-specific protocols or protocol features.

VoIP environments can include nodes from different heterogeneous access networks that include WLAN, WiMAX, and cellular networks among others. In addition, mobility can have its own security implications in converged environments. The implemented prototype and the simulation scenarios are however tested and conducted in LAN. Our implementations do not take into account mobility requirements, and so no performance evaluation of the intrusion detection system's effectiveness is done in a mobile environment. However, theoretical analysis is presented based on some mobile related attacks.

The proposed architecture is compared analytically to several similar existing systems. However, a comparison by implementing these existing systems is beyond the scope of this study.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 consists of two main parts. The first part provides the necessary background with regard to VoIP systems. It discusses various issues related to VoIP architecture and

protocols. The second part of the chapter compares our design against several systems that tend to adopt similar approaches in intrusion detection. The comparison revolves around certain key features that are suitable for intrusion detection in VoIP environments. This second part clearly shows the advantages of our IDS compared to others.

Chapter 3 discusses the formal model upon which the IDS is designed. It starts with discussing the features and advantages of State Transition Analysis Techniques which form the basis of our signature-based detection module. Next, the chapter looks at Finite State Machines, their mathematical model, implementation approaches, and unique characteristics that qualify them to model networking protocols. That discussion is meant to prepare the reader for the more advanced Extended Finite State Machines and the Communicating Extended Finite State Machines model which form the basis of our specification-based detection module. Chapter 3 concludes with some of the limitations of the abovementioned approaches as a prelude to the discussion on our system design.

Chapter 4 presents the architecture of the proposed IDS from an application layer point of view. It follows a systematic approach by first discussing the involved protocols, namely, SIP and RTP and their threat model. Next, the chapter sheds some light on the specification development and the design of the state machines and verifiers for each of the protocols involved. The chapter then demonstrates the components of the proposed architecture and the role played by each component in the detection process. A discussion on the advantages of the proposed architecture follows. Finally, the chapter presents several implemented attacks, and shows how they can be detected by the various components of the system.

Chapter 5 follows the footsteps of its predecessor discussing the extensions added to the proposed architecture to accommodate lower layer protocols. The same systematic approach is followed by first discussing the lower layer protocols involved, namely, IP, ICMP, UDP, and TCP and their threat model. Next, the chapter looks at the specification development and the design of the state machines and verifiers for each of the protocols involved. The chapter then demonstrates the extended architecture alongside its added benefits to the overall design. Finally, the chapter presents several implemented attacks and shows how they can be detected by various components of the extended architecture.

Chapter 6 is dedicated to implementation and simulation issues. It starts with mentioning some of the intrusion detection implementation approaches alongside the hurdles that face the implementation and testing of intrusion detection system. Then the chapter discusses the simulation tool used to implement our architecture, its support for the networking protocols at different layers, and how it overcomes implementation and testing hurdles. Next, the chapter sheds some light on the simulator features that are used to implement the attacks and the detection methodologies. A description of the simulated network topology and its configuration is presented, followed by the different traffic generation scenarios used to test the system.

Chapter 7 presents and analyzes the produced results. The chapter consists of two main parts. The first part discusses the issues relating to detection accuracy. Several axes are covered in this part such as the IDS coverage, hit rate, and probability of false alarms. The effect of the implemented IDS on the performance of hosts is demonstrated in the second part of the chapter. The performance of the system is demonstrated in terms of end-to-end delay, call setup delay, processing delay, packet loss, and memory usage. The chapter concludes by drawing attention to some of the optimization techniques used to improve the performance.

Chapter 8 presents a set of conclusions derived from the results and theoretical analysis done in the previous chapters. A summary of the concluding remarks on the various issues encountered in the previous chapters are illustrated. Further, this chapter gives some recommendations to improve the system and the evaluation methodology. Areas of research relating to this discipline that will be of interest in the near future are outlined.

Chapter 2 Literature Review

2.1 Introduction

In the previous chapter we shed some light on the different types of intrusion detection systems with regard to detection techniques and monitored resources. A major part of the previous chapter was dedicated to discussing the special security needs of converged networks and applications. The aim of that discussion was to highlight the importance of adopting new methods and combining existing ones in intrusion detection for more efficiency and accuracy.

We demonstrated our approach in the first chapter by bringing to light the techniques adopted for better intrusion detection in VoIP environments. These techniques include a hybrid detection mechanism that combines specification-based and signature-based approaches. Our architecture also provides a cross-protocol detection mechanism in our specification-based and signature-based modules to exchange detection information across protocols to help the IDS make more accurate decisions. Although specification-based detection is very powerful by itself, it is limited to examining a single request or response [16]. Many attacks involve multiple requests and responses and can not be detected by looking at individual requests and responses. Therefore, stateful detection forms an important technique in our approach. Another technique adopted by our approach is the ability to create session-level and semantics-based signatures to cope with the rising complexity of VoIP attacks.

Since this research focuses on intrusion detection in VoIP environments, we start this chapter by introducing some basic concepts about VoIP systems and architecture. Several similar IDSs have been proposed and implemented. The chapter will also provide a discussion on these systems and how they relate to our proposed architecture. The discussion will revolve around the techniques we mentioned in the previous paragraph. The chapter will also shed some light on IDSs designed for VoIP and how they are compared to our design.

2.2 VoIP Architecture

Circuit-switched networks have traditionally been suitable to carry voice. Prior to

sending data, a point-to-point static channel is allocated in the circuit-switched network. Such a channel is dedicated entirely to the voice call and not shared with others which assures a certain level of quality. IP protocol forms the decisive difference between circuit-switched networks and VoIP networks. It is being used to carry voice alongside data. IP networks which are packet-switched break voice and data into packets that are routed to a certain destination. Upon arrival at the destination, the packets are reassembled into their original format. Contrary to circuit-switched networks, packets in packet-switched networks can travel across multiple independent paths to the final destination. This feature can benefit the network in terms of self-recovery with failed link paths because paths can be allocated dynamically. These differences between traditional circuit-switched and VoIP networks entail changes in the infrastructure and protocols used.

2.2.1 VoIP Infrastructure

Components in VoIP infrastructure can be generally classified into servers, endpoints, and routing nodes. VoIP servers are the components responsible for various duties aiming at maintaining the service and enhancing it. For example, Address resolution servers are responsible for providing address translation and access control to VoIP terminals and gateways. Registration servers provide better mobility support by registering users and routing their traffic regardless of their geographical location. Furthermore, redirect servers can inform callers if the information of the callee they are trying to reach has changed. The abovementioned examples of servers are not meant to be comprehensive, but rather to give an overview on the various functions performed by servers in VoIP environments.

Endpoints are the devices capable of initiating and terminating a call. They can take the form of IP softphones which can look and feel like traditional phones but have more functionality and intelligence. Other forms of endpoints are instant messaging clients and video clients.

Routing nodes in VoIP environments have the capacity to connect IP networks to either other IP networks or circuit-switched networks. When connecting two or more IP networks, IP routers and switches inspect packets' addresses to determine their final destination and forward

them to their recipients. When circuit-switched networks are involved, routing nodes act as gateways to translate between incompatible interfaces and protocols.

Figure 2-1 shows an example of VoIP infrastructure.

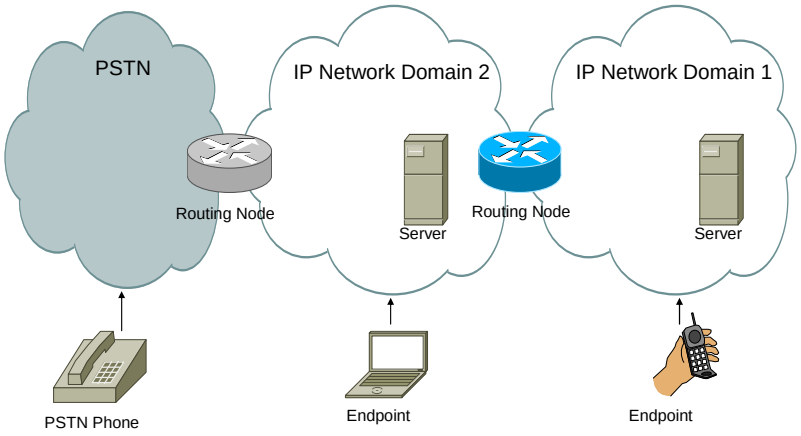


Figure 2-1. VoIP infrastructure

2.2.2 VoIP Protocols

In this subsection, we shed some light on the most dominant multimedia suites, namely, H.323 and SIP. Both protocols are used for signaling and with them come other protocols that cater for functions other than signaling in VoIP environments.

H.323 was originally designed based on ITU-T Signaling System Number 7 [17] (also known as SS7) which defines how devices in the PSTN network exchange data. This feature gives H.323 advantage when it comes to internetworking efficiency with PSTN networks. H.323 specifications [18] define many sub-protocols that address different aspects of a session. For instance, H.225 is a sub-protocol that is responsible for call setup and tear-down. H.245 is another sub-protocol that is responsible for call control. H.323 specifies other protocols for data transport and quality of service such as Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP).

On the other hand, SIP is much simpler than H.323, but with less internetworking capabilities with PSTN systems. It was developed by the Internet Engineering Task Force (IETF) in RFC 2543 [19] which was later updated by RFC 3261 [14]. SIP was designed to address some important issues in setting up and tearing down sessions such as user location, user availability, and session management. The versatility of SIP makes it the choice of instant messaging, video conferencing, and multiplayer game applications among others.

2.3 Cross-Layer and Cross-Protocol Intrusion Detection

Cross-layer and cross-protocol intrusion detection are very relevant to VoIP environments. Services at the application layer in a converged environment could easily span across multiple protocols and specifications. A typical VoIP session involves the Session Initiation Protocol (SIP) for call management and the Real-time Transport Protocol (RTP) for data delivery along with the rest of protocols at lower layers. Each of the protocols involved has its own set of vulnerabilities, and several attacks are based on sequences that cross protocol boundaries.

The concept of integrating multiple layers of the protocol stack for efficient intrusion detection was introduced by Zhang and Lee [20]. They developed a cross-layer based IDS architecture to study the abnormalities in the network using anomaly detection. Another remarkable attempt was made by G. Thamilarasu et al [21] in the CIDS intrusion detection architecture. However, both architectures are based on anomaly-based detection techniques in wireless networks rather than Specification-based techniques which are more reliable as mentioned earlier. In addition, CIDS system is confined to a certain type of Denial of Service (DoS) attacks.

The WebSTAT system [22] provides intrusion detection for web servers. It works by correlating both network-level and operating system-level events with entries contained in server logs. However, no correlation is performed within concurrently executing protocols at the same layer.

Our architecture provides cross-protocol detection at the application layer to cover the threats posed by VoIP application layer protocols. Other lower layers can benefit as well from

the cross-protocol capabilities. The architecture also provides cross-layer detection techniques to allow different layers to exchange detection information efficiently.

2.4 Stateful Intrusion Detection

Stateful detection involves performing analysis for an entire connection or session, capturing and storing certain pieces of relevant data seen in the session, and using these data to identify attacks that involve multiple requests and responses [8]. State transition analysis and state machines are some of the techniques used by IDSs to perform stateful detection.

The state transition analysis technique was conceived as a method to describe computer penetrations as a sequence of actions that an attacker performs to compromise the security of a computer system. Attacks are graphically described by using *state transition diagrams*. States represent snapshots of system's volatile, semi-permanent, and permanent memory locations. A description of an attack has a "safe" starting state and at least one "compromised" ending state. States are characterized by means of *assertions* which are predicates on some aspects of the security state of the system. *Transitions* between states are indicated by signature actions that represent the actions that if omitted from the execution of an attack scenario, would prevent the attack from completing successfully [23].

State transition analysis technique was adopted by STAT system [24] which targeted hosts. NetSTAT [23] used the same techniques targeting networks. NetSTAT takes advantage of peculiar characteristics of intrusion detection based on analysis of network traffic. Networks provide detailed information about computer system activity and can provide this information regardless of the installed operating systems or the auditing modules available on the host. However, both STAT and NetSTAT are considered signature-based systems. Being only signature-based limited the ability of these systems to detect new attacks. On the other hand, J. M. Orset, B. Alcalde, and A. Cavalli [25] proposed Extended Finite State Machine-based IDS that used specifications of routing protocol OLSR to detect anomalies in Ad Hoc networks. However, their solution was not complemented by a signature-based component, which made it difficult to detect attacks such as DoS attacks. C. C. Michael and A. Ghosh [89] proposed variants of state-based intrusion detection algorithms. One is a simply constructed finite-state

machine and the other monitor statistical deviations from normal program behavior. The absence of a solid specification based approach in their algorithms leaves a chance to a relatively high false alarm rate.

Our design uses an advanced form of state transition analysis and state machines in its signature-based and specification-based detection modules respectively to provide stateful detection.

2.5 Signature-based Intrusion Detection

One of the most popular signature-based intrusion detection systems is Snort [26]. Snort targets networks and performs real-time analysis of packets and triggers alarms when certain conditions are met. The Snort database design defines the lowest level of detail as an event, which is the combination of a collection of packet header and data and an active Snort rule called a signature. However, Snort's approach falls short of providing a basis for semantics-aware signatures at the session level. Sommer and Paxon [27] proposed adding connection-level context to signatures to reduce false positives in signature-based detection. However, their aim was to complement the most common form of signature matching, which is low-level string matching, with context. In [86] and [87], M. Zulkernine, M. Graves, U. Khan, and M. Raihan proposed integrating two languages for software specification and security specification to reduce duplication of efforts and conflicting requirements. In [87] specifically, M. Zulkernine, M. Graves, and U. Khan used this technique to automatically translate attack scenarios written in these two languages into Snort rules. However, all these efforts were confined to adding context information to Snort signatures.

Our signature database combines both types of signatures, byte-level and semantics-aware. Our semantics-awareness is based on the state transition analysis technique that is used to describe the semantics of attacks and penetrations. The state transition analysis technique can model attacks at the session level rather than lower and semantics less levels. The lowest level of detail in our semantics-based module is the *state* instead of the traditional *event*. This feature enables our database to store a higher-level abstraction of attacks than previous works and support more general signatures.

2.6 Hybrid Intrusion Detection

Hybrid IDSs can trace their origins back to systems such as IDES [28], NADIR [29], and W&S [30]. IDES provides an anomaly-based component that identifies expected behavior at the user, group, remote host, and target system level. The IDES anomaly component in combination with the IDES penetration identifier rule-base provides a way of learning and representing normal behavior, as well as a way of representing improper behavior. NADIR's implementation is similar to IDES in that it employs statistical anomaly component in conjunction with a small rule-base of expert penetration rules. NADIR's expert rule-base consists of penetration rules that are developed by interviewing and working with security personnel. In W&S, the penetration detection component is combined into the same rule-base to represent site-specific policies, expert penetration rules, and other administrative data. However, it was difficult in all these systems to establish proper behavior patterns, resulting in a relatively large number of false alarms. Using system specifications as the detection baseline in our architecture reduces the false alarm rate significantly.

Sekar et al [6] proposed a hybrid intrusion detection system that combined specification-based and anomaly-based detection. Their reliance on protocol specifications helped to simplify the process of feature selection which plays a major role in anomaly detection approaches. However, their reliance on anomaly-based detection hindered the system ability to detect attacks that were not based on repetition. Such attacks can be detected by our architecture. In [88], P. Uppuluri and R. Sekar proposed a detection system that combines specification-based and signature-based approaches. Although their proposed system showed a high detection rate, its signature-based component only stored simple and less sophisticated attacks.

2.7 Intrusion Detection Systems for VoIP Environments

Our discussion in the previous sections aimed at highlighting the features that should be adopted by intrusion detection systems in VoIP environments. We discussed the techniques used by some related systems to enable such features and we mentioned the advantages of our design over these systems. We believe it is in order to bring to light some of the intrusion detection systems specially designed for VoIP environments. We will show in this section how these

intrusion detection systems either succeed or fail to adopt the aforementioned features, and how our architecture is compared to them.

The special needs of converged networks and applications, and the prevalence of VoIP telephony resulted in the introduction of SCIDIVE [13] intrusion detection system. SCIDIVE is a stateful and cross-protocol intrusion detection system for VoIP environments. SCIDIVE can be considered a signature-based detection system rather than an anomaly-based system. It works by accessing packets from multiple protocols in a system and comparing them against well-created cross-protocol rules. As mentioned previously, signature-based systems lack the ability to detect new and novel attacks, and the rule database needs to be updated on a regular basis following new attacks. These drawbacks are addressed by vIDS [31]. Instead of relying entirely on a rule database, vIDS is based on interacting protocol state machines. However, all the attacks used to test the efficiency of vIDS were known attacks and had to be encoded in the system as attack patterns. The capabilities of vIDS in detecting attacks based on normal behavior specifications were not shown. Moreover, the design of vIDS covers the issues relating to protocol-semantics anomaly detection, while not addressing protocol-syntax anomaly detection. We mentioned in Section 1.1.1.3 the importance of supporting both protocol-semantics and protocol-syntax anomaly detection. Both SCIDIVE and vIDS are dedicated only for VoIP application layer protocols and not addressing threats at lower layers using the same techniques.

Another attempt to address the special needs of VoIP environments has been made by vFDS [32]. vFDS is an online statistical detection mechanism. Its detection goes beyond detecting attacks at the application layer, to detect transport layer attacks as well. However, the reliance of vFDS on pure statistical anomaly approaches affects its sensitivity negatively. In addition, vFDS is limited to detecting flooding attacks.

Our design provides a combination of specification-based and signature-based detection techniques to bring the false alarm rate to its lowest level. It also addresses protocol-syntax and protocol-semantics-related issues to cover a wider range of attacks, in addition to addressing lower layers threats alongside application layer ones.

Table 2-1. Comparison against other IDSs for VoIP environments

	Support for syntax and semantics-based detection	**Stateful detection**	**Cross-protocol detection**	**Statistical anomaly detection**	**Specification-based detection**	**Signature-based detection**
vIDS	Support for syntax-based detection is unclear	Yes	Yes	No	Yes	Yes
vFDS	No	Yes	Yes	Yes	No	No
SCIDIVE	Syntax-based detection is supported. Semantics-based detection is partially supported through signatures	Yes	Yes	No	No	Yes
Our Proposed Architecture	Yes	Yes	Yes	No	Yes	Yes

Table 2-1 summarizes the differences between our proposed architecture and other intrusion detection systems that are designed for VoIP environments. Every column in the table represents a desirable feature that should be supported by the detection system. Rows represent different proposed systems including ours.

2.8 Chapter Discussion

This chapter aimed at highlighting the unique features possessed by VoIP environments and the special needs of intrusion detection systems in these environments. Our discussion on

VoIP environments revolved around their architecture, infrastructure, and protocols. Based on that discussion the chapter suggested some detection features that should be present in intrusion detection systems for VoIP environments. Those desired features were: cross-protocol awareness, stateful detection, and a combination of signature-based and specification-based techniques with semantics-awareness. Some of the systems that adopt the aforementioned features were mentioned, and how our proposed design differed was discussed. The chapter then dedicated the last section to present whether some VoIP intrusion detection systems were in line with these features or not and how our design excelled over them. The next chapter will discuss in detail the State Transition Analysis and State Machines with their use in intrusion detection.

Chapter 3 State Transition Analysis and State Machines: Applications in Intrusion Detection

3.1 Introduction

The previous chapter introduced basic concepts on VoIP architecture, infrastructure, and protocols. Furthermore, features that are deemed necessary for intrusion detection in VoIP environments were discussed. That discussion followed a systematic approach by showing how comparable systems succeeded or failed to adopt those features, and how our approach applied the same features in its detection techniques. The previous chapter then concluded by comparing intrusion detection systems designed for VoIP environments to our proposed architecture. That comparison demonstrated the advantages of our architecture over other architectures.

This chapter lays the theoretical foundation of our proposed architecture by discussing State Transition Analysis and State Machines which are used in our signature-based and specification-based modules. The use and application of these methods in intrusion detection will be discussed as well.

3.2 State Transition Analysis

State Transition Analysis was developed by the Reliable Software Group at University of California, Santa Barbara to represent computer penetrations. State transition analysis forms a method for representing the sequence of actions that the attacker performs to achieve a security violation. In State Transition Analysis, a penetration is viewed as a sequence of actions performed by an attacker which leads from some initial state on a system to a target compromised state, where a state is a snapshot of the system representing the values of all volatile, semi-permanent and permanent memory locations on the system. The initial state corresponds to the state of the system just prior to the execution of the penetration, and the compromised state corresponds to the state of the system resulting from the completion of the penetration. Between the initial and compromised states are one or more intermediate state transitions that an attacker performs to achieve the compromise.

A state transition diagram is the graphical representation that the state transition analysis uses to represent a penetration. This representation is useful for describing attacks in that it provides an interesting level of abstraction to the analyst: just above the system call and below English description [24]. This level of abstraction allows for higher-level, semantics-aware representation of the attack scenario. A state diagram can take the form of a directed graph which consists of: labeled circles to represent system states, input symbols to represent the input at each state, output symbols to represent the output from each state, edges to represent transitions between states, a start state, and a final state.

It was noticed by the development team that if the state changes of a computer system could be monitored, then state transition analysis could be developed from a method for representing computer penetrations to a method for detecting penetrations as well. A mechanism to record system state changes already existed in the form of audit facilities. Audit facilities record the state changes made by users on monitored system attributes. The information kept in the system's audit trail can be analyzed by appropriate tools and compared to state transition diagrams of known penetrations.

A major problem with signature-based detection systems other than state transition analysis-based ones is their entire dependence on audit record fields. Such systems pattern match their signatures to audit records, which lets some penetrations go unnoticed if they manage to make a slight change in their scenario. One solution to this problem is to use a higher-level representation for signatures which is what state transition analysis does. The attack representation used by state transition analysis is audit record independent and can accommodate slight variations by the same penetration. In addition, other signature-based systems have no ability to foresee an impending compromise and preempt or limit the damage before it occurs. State transition analysis can anticipate an impending compromise at the state prior to the compromised one and forewarn administrators accordingly. Figure 3-1 sheds more light on this important feature that is missed in other signature-based intrusion detection techniques. The figure shows a state transition diagram of a certain attack. To successfully execute the attack, the attacker needs to reach the system state $State_0$, have Assertion 1 and Assertion 2 hold, and perform a sequence of actions represented by $State_1$ and $State_2$. If the system reaches $State_0$ and finds the specified assertions hold, the signature-based detection mechanism can raise an alarm

warning about the potential impending *Compromised State*. The intermediate states $State_1$ and $State_2$ can be used to represent variants of the penetration.

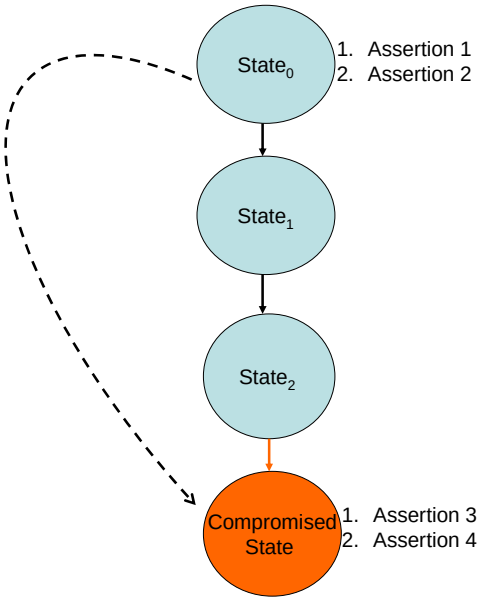


Figure 3-1. State Transition Diagram of an Attack

State transition diagrams form the common factor between State Transition Analysis and State Machines which will be discussed in the next section.

3.3 Finite State Machines

A Finite State Machine (FSM) is a model of behavior that consists of a set of input events (actions), a set of output events (actions), a set of states, and a set of transitions [33].

3.3.1 Finite State Machine Representation

Finite State Machines can be represented using several types of *state transition tables*. One of the most common is shown in table 3-1.

A combination of *Current State* and *Input* produces *Output* and makes the machine move to *Next State*. *Input* and *Output* are regarded as two finite and disjoint sets of signals, namely, input signals and output signals. When the value of input signals meets a certain condition, a

transition rule is executed by the machine and a change occurs to the value of the output signals. A state transition table can be easily transformed into a state transition diagram as shown in figure 3-2.

Table 3-1. State Transition Table

Current State	Input	Output	Next State
$State_0$	$Input_0$	$Output_0$	$State_1$
$State_1$	$Input_1$	$Output_1$	$State_2$
$State_2$	$Input_2$	$Output_2$	$State_3$

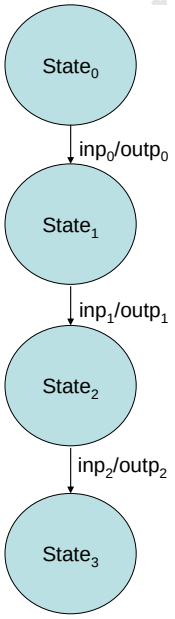


Figure 3-2. State Transition Diagram for Table 3-1

Finite State Machines are used widely to model systems in many different areas such as circuit design, linguistics, and logic. Communication protocols are good candidates to be represented by finite state machines. A protocol is often most easily understood as a state machine. Design criteria can also easily be expressed in terms of desirable or undesirable

protocol states and state transitions. In a way, the protocol state symbolizes the assumptions that each process in the system makes about the others. It defines what actions a process is allowed to take, which events it expects to happen, and how it will respond to those events [34].

3.3.2 Mathematical Model and Definitions

Finite State Machines (FSMs) can usually be modeled by *Mealy* machines that produce outputs on their state transitions after receiving inputs.

Definition 3.1. A finite state machine (FSM) M is a quintuple

$$M = (I, O, S, \delta, \lambda)$$

where I, O, and S are finite and nonempty sets of input symbols, output symbols, and states, respectively.

δ: $S \times I \rightarrow S$ is the state transition function;

λ: $S \times I \rightarrow O$ is the output function.

When the machine is in a current state s in S and receives an input a from I, it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$.

A *Mealy* machine has a finite number of states and produces outputs on state transitions after receiving inputs. Another popular type of finite state machines is *Moore* machines. The theory is very similar for the two types. Mealy machines, which are mainly considered here, model finite state systems more properly and are more general than Moore machines.

The finite state machine in the aforementioned definition is fully specified in a sense that at a state and upon an input there is a specified next state by the state transition function and a specified output by the output function. Otherwise, the machine is partially specified; at certain states with some inputs, the next states or outputs are not specified. Also the machine defined is *deterministic*; at a state and upon an input, the machine follows a unique transition to a next state. Otherwise, the machine is *non-deterministic*; the machine may follow more than one transition and produce different outputs accordingly [35]. Our proposed architecture uses

deterministic state machines in its implementation.

3.3.3 Implementation of Finite State Machines

The way finite state machines are implemented for a system affects the performance of that system greatly. Like all software, finite state machines are subject to change and maintenance. Implementers of FSMs should plan foremost whether it would be easy with their implementation methodology to add, remove, or change a state, event, or transition during maintenance. More than one instance of a finite state machine is often needed to be kept in the memory of a system. Implementers of finite state machines should bear in mind how the scheme they are using is adding to the cost of instance creation and if they can prevent unnecessary duplication of objects.

Mainly, two methods are widely used to implement finite state machines, namely, the procedural approach and the object oriented approach. In procedural languages, FSMs can be implemented efficiently using a double *case* statement. The outer case statement selects on the current State of the FSM. Then the inner case statement selects the appropriate behavior for the current State given the type of event that was sent to the FSM. A major disadvantage of using this approach is when the structure of the FSM needs some change due to a change in the specifications. A lot of code has to be modified even for very simple changes in the specifications. However, the procedural approach is very efficient regarding instance creation and memory management.

By using object orientation, the use of case statements can be avoided through the use of dynamic binding. Each case becomes a State class and the correct case is selected by looking at the current state-object. This approach makes understanding of the implementation much easier than the procedural approach. However, code for a transition can be scattered vertically in the class hierarchy. Furthermore, this approach could be very inefficient with regard to the cost of objects creation and memory management. Other approaches have been introduced to overcome the problems of the aforementioned ones [36].

3.4 Extended Finite State Machines

In principle, finite state machines model appropriately the control portions of communication protocols. However, in practice the usual specifications of protocols include variables and operations based on variable values; ordinary FSMs are not powerful enough to model in a succinct way the physical systems any more. To accommodate these needs of protocols and other complicated systems, Extended Finite State Machines (EFSMs) were introduced.

The model of a Mealy Finite State Machine (FSM) extended with input and output parameters, context variables, operations, and predicates defined over context variables and input parameters is what is understood by an Extended FSM (EFSM). The FSM underlying an EFSM is often said to model the control flow of a system, while parameters, variables, predicates, and operations reflect its data flow (or context).

3.4.1 Mathematical Model and Definitions

To define a general type of EFSMs, we use finite disjoint sets for signal parameters and context variables, denoted, respectively, *R* and *V*, as follows: Input or output signals of FSM are associated with a subset of parameters, so that the signal and a valuation of parameters from the set *R* associated with the signal constitute a parameterized signal, input or output. A state and a valuation of the context variables in the set *V* constitute a so-called configuration of the EFSM. Input and output parameters do not contribute to the configuration space. Thus, if one flattens an EFSM into a normal FSM (assuming finite domains for all parameters and variables), parameterized signals of the EFSM become inputs and outputs of the FSM, while configurations of the EFSM constitute the states. Signal parameters and context variables of an EFSM are all parameters of various objects in the EFSM. The difference is that all the context variables parameterize states, while only subsets of parameters are used to define input or output (this is why we call them differently). Signal parameters and context variables could share common types (i.e., have the same value) and sometimes they are all just called variables associated with EFSM [37] [38].

In the following definitions, let X and Y be finite sets of inputs and outputs, R and V be finite disjoint sets of parameter and variable names. For $x \in X$, we note $R_x \subseteq R$ the set of input parameters and D_{Rx} the set of valuations of parameters in the set R_x. For $y \in Y$, we define similarly R_y and D_{Ry} . Finally, D_V is a set of context variable valuations v.

Definition 3.2. An extended finite state machine (EFSM) M over X, Y , R, V , and the associated valuation domains is a pair (S, T) of a finite set of states S and a finite set of transitions T between states in S, such that each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where

- $s, s' \in S$ are the initial and final states of the transition, respectively;
- $x \in X$ is the input of the transition;
- $y \in Y$ is the output of the transition;
- P, op, and up are functions, defined over input parameters and context variables V , namely,
    - P: $D_{Rx} \times D_V \rightarrow$ {True, False} is the predicate of the transition.
    - op: $D_{Rx} \times D_V \rightarrow D_{Ry}$ is the output parameter function of the transition.
    - up: $D_{Rx} \times D_V \rightarrow D_V$ is the context update function of the transition.

Definition 3.3. Given input x and a (possibly empty) set of input parameter valuations D_{Rx}, a *parameterized* input is a tuple **(x, p_x)**, where $p_x \in D_{Rx}$. A sequence of parameterized inputs is called a parameterized input sequence.

Similarly, we define parameterized outputs and their sequences.

Definition 3.4. A context variable valuation $v \in D_V$ is called a context of M. A configuration of M is a tuple (s, v) of state s and context v.

In case of an empty set of context variables, we do not distinguish between configuration and state.

Definition 3.5. A transition is said to be enabled for a configuration and parameterized input if the transition predicate evaluates to True.

The EFSM operates as follows: The machine receives input along with input parameters (if any) and computes the predicates that are satisfied for the current configuration. The predicates identify enabled transitions. A single transition among those enabled fires. Executing the chosen transition, the machine produces output along with output parameters, which, if they exist, are computed from the current context and input parameters by the use of the output parameter function. The machine updates the current context according to the context update function and moves from the initial to the final state of the transition. Transitions are atomic and cannot be interrupted. The machine usually starts from a designated configuration, called the initial configuration. A pair of an EFSM *M* and an initial configuration is called a strongly initialized EFSM [38].

We can build elaborate systems of interacting machines by connecting the output signals of one machine to the input signals of another to form Communicating Extended Finite State Machines (CEFSMs) [34].

3.4.2 EFSMs in Specification-based Intrusion Detection

Internet protocols can be easily modeled as EFSMs. A protocol can be viewed as a sequence of processes (states) chained by a set of events (transitions). A running protocol EFSM receives packets (input signals) through one of the available ports. Packets usually contain header fields with values (input parameters). Upon receiving a packet, a check is performed to identify the packet type (predicate) and to determine the appropriate event (transition). Some transitions represent unexpected packets, which usually occur due to network failures or an attack. Similarly, absence of expected packets, and the consequent transition on a timeout event, suggests a failure or an attack. Another source of input to a protocol state machine could be a signal sent by another protocol state machine (synchronization signals).

The execution of the chosen event (transition) could result in producing and sending a packet with its header values (parameterized output signal) by a dedicated function (output parameter function). The protocol then updates its state information (context variables) by a pre-

defined set of instructions (context update function).

Figure 3-3 shows a state transition diagram for a protocol that has three states and two transitions based on its specifications. When the state machine is in state 1, and upon receiving input signal (inp_1), a predicate is computed to choose the appropriate transition which leads to state 2. The dotted transition, which leads to the attack state, represents an unexpected input received at state 1. The unexpected input results in the predicate failing to enable a legitimate transition and the machine raising a protocol violation flag [8].

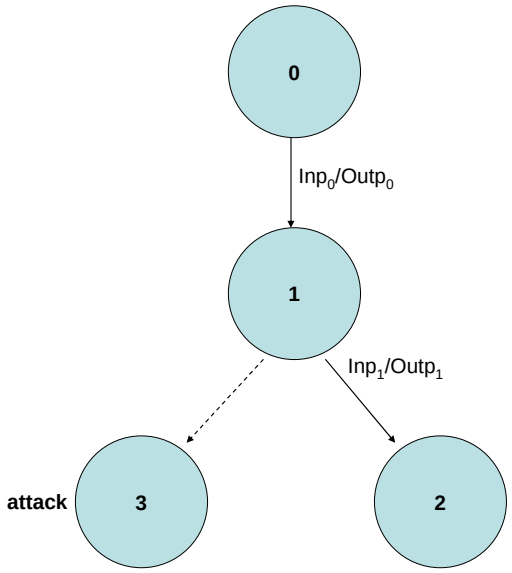


Figure 3-3. A state transition diagram that shows normal and potential abnormal protocol behavior.

3.4.3 Limitations of EFSM model in Specification-based Detection

Despite the remarkable convenience State Transition Diagrams (STDs) provide for the EFSM model in specification-based IDSs, there are also problems with state machines approach in this type of detection:

- It can model abnormal behavior as a simple and straightforward sequence of events rather than more complex forms. This limitation will become clear when we discuss some of the complex cross-protocol attacks in the implemented attacks sections of this thesis (4.6 and 5.6).

- Some attacks, which are launched by abusing legitimate features of the system, can pass EFSM-based anomaly detection without being detected. This limitation is the reason why STD-based EFSMs cannot directly detect attacks such as Denial of Service and failed logins in anomaly mode [39].

Clearly, a signature-based approach that is based on a flexible and more advanced state transition analysis is needed to assist EFSM-based specification detection approaches to detect such attacks. The suggestion of an accompanying signature detection mechanism based on state transition analysis is a natural product of the remarkable similarities between EFSM-based and state transition analysis-based approaches.

3.5 Chapter Discussion

This chapter presented the theoretical foundation of our detection methodologies. It started by discussing state transition analysis techniques and how they were used to model and detect computer penetrations. The chapter highlighted the advantages of state transition analysis over other approaches and why it was preferred for signature-based intrusion detection. Furthermore, finite state machines, their representation, mathematical model, and implementation issues were discussed in detail. The chapter mentioned the shortcomings of finite state machines in protocol modeling as a prelude to discussing extended finite state machines. The mathematical model of extended finite state machines and their use in specification-based intrusion detection were detailed. The chapter concluded the discussion on extended finite state machines by shedding some light on their limitations. That conclusion suggested the importance of combining state transition analysis as a signature-based detection mechanism with extended finite state machines as specification-based one to form a hybrid detection methodology. The next chapter will show how we adopt the principles demonstrated in this chapter and translate them into the proposed design.

Chapter 4 System Design: An Application Layer Perspective

4.1 Introduction

The previous chapter detailed the theoretical background of the proposed architecture. State Transition Analysis techniques which form a basis in our signature-based detection modules were discussed. Furthermore, Extended Finite State Machines which constitute a foundation in our specification-based modules were thrashed out. The conclusion of the previous chapter was to draw the attention to the convenience of combining both methods in intrusion detection based on the remarkable similarities between them.

This chapter kicks off the discussion on the design of the proposed architecture. The chapter will focus on the application layer protocols and their threat model. It will then show how the proposed architecture adopts the methods mentioned in the previous chapter to build a concrete IDS. The components of the architecture and their functionality will be demonstrated. The attacks implemented to test the credibility of the design and the related issues will also be discussed.

4.2 Related Protocols and Threat Model

As mentioned in the first chapter, we consider SIP suite for our discussion on the related protocols. Session Initiation Protocol (SIP) is a standard signaling protocol for VoIP, and is appropriately coined as the "SS7 of future telephony." It was developed by the Internet Engineering Task Force (IETF) in RFC 2543 which was updated by RFC 3261. SIP was designed to address some important issues in setting up and tearing down sessions such as user location, user availability, and session management. The simplicity and versatility of SIP make it the choice of instant messaging, video conferencing, and multiplayer game applications among others. SIP uses other protocols to perform various functions during a session such as Session Description Protocol (SDP) to describe the characteristics of end devices, Resource Reservation Setup Protocol (RSVP) for voice quality, and Real-time Transport Protocol (RTP) for real-time

transmission.

In this section we concentrate on SIP and RTP for the vital role they play in the establishing, tearing down, and carrying the data of the session.

4.2.1 SIP Message Format

The SIP message is made up of three parts: the start line, message headers, and body. The start line contents vary depending on whether the SIP message is a request or a response. For requests it is referred to as a request line and for responses it is referred to as a status line. Figure 4-1 shows SIP message format.

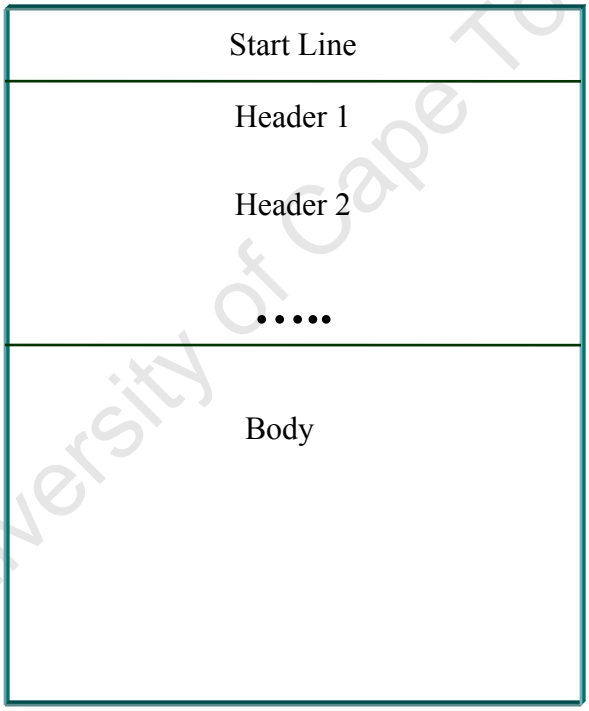


Figure 4-1. SIP Message Format.

The base SIP specifications define six types of request: the INVITE request, CANCEL request, ACK request and BYE request are used for session creation, modification, establishment, and termination; the REGISTER request is used to register a certain user's contact information; and the OPTIONS request is used as a poll for querying servers and their capabilities.

Response types or codes are also classified into six classes. *1xx* for provisional/informational responses, *2xx* for success responses, *3xx* for redirection responses, *4xx* for client error responses, *5xx* for server error responses, and *6xx* for global failure responses. The "*xx*" are two digits that indicate the exact nature of the response: for example, a "180" provisional response indicates ringing by the remote end, while a "181" provisional response indicates that a call is being forwarded.

Header fields contain information related to the request like the initiator of the request, the recipient, and call identification. Some headers are mandatory in every SIP request and response. These are: To (carries the recipient of the request), From (carries the initiator of the request), Call ID (carries the unique identifier of the call), CSeq (used to identify the order of transactions), Via (contains the transport protocol and the address where the response is to be sent), Max-Forwards (used to limit the number of hops a request traverses and to avoid loops), and Contact (contains the address of the host where the request originated).

Message bodies can carry any text-based information whose interpretation is determined by request and response codes.

4.2.2 SIP Architecture

Elements in SIP can be classified into user agents (UAs) and intermediaries (servers). In an ideal world, communications between two endpoints (or UAs) happen without the need for servers. However, this is not always the case as network administrators and service providers would like to keep track of traffic in their network.

A SIP UA or terminal is the endpoint of dialogs: it sends and receives SIP requests and responses, it is the endpoint of multimedia streams, and it is usually the user equipment (UE) which is an application in a terminal or a dedicated hardware appliance.

SIP servers are logical entities where SIP messages pass through on their way to their final destination. These servers are used to route and redirect requests. These servers include:

- Proxy server—receives and forwards SIP requests.

- Redirect server—maps the address of requests into new addresses.
- Location server—keeps track of the location of users.
- Registrar—a server that accepts REGISTER requests.
- Application server—an Application Server (AS) is an entity in the network that provides end users with a service.

4.2.3 SIP Session

Figure 4-2 shows the establishment of a SIP session between two users in the same domain.

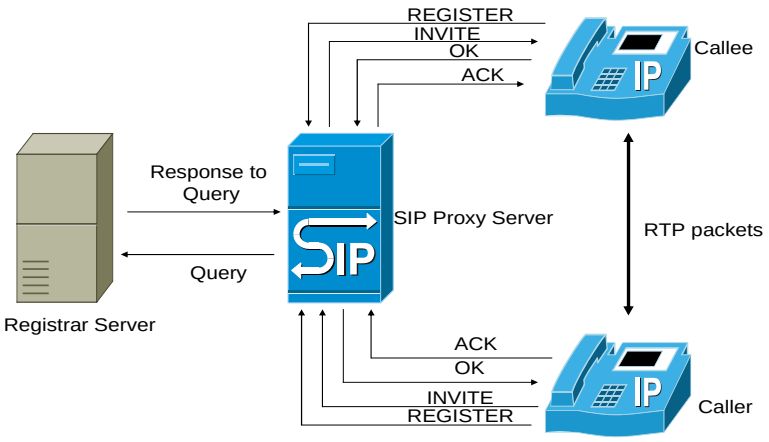


Figure 4-2. Establishment of a typical SIP Session

When turning on their devices, both users register their availability and their IP addresses with the SIP proxy server using REGISTER request. The proxy server then sends this information to the relevant Registrar server. The caller tells the proxy server that he/she wants to contact a certain callee using INVITE request. The SIP proxy server relays the caller's invitation to the callee. The callee informs the proxy server that the caller's invitation is acceptable with OK response. The SIP proxy server communicates this response to the caller who sends ACK response establishing a session. The users then create a point-to-point RTP connection enabling

them to interact. Any of the parties involved in a session can end it by sending a BYE request.

4.2.4 RTP Message Format

Real-time Transport Protocol (RTP) is an application layer protocol that provides end-to-end delivery services for real-time audio and video. It was developed by the Internet Engineering Task Force (IETF) in RFC 1889 which was updated by RFC 3550. Figure 4-3 shows RTP message format.

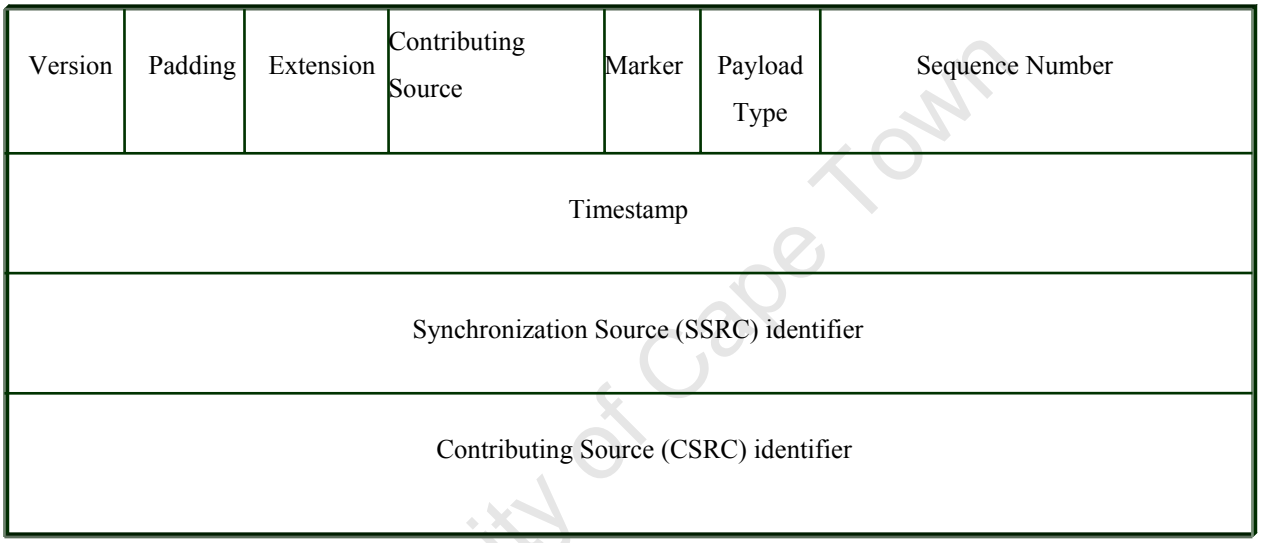


Figure 4-3. RTP Message Format.

The message fields are: Version (contains the version of RTP), Padding (indicates whether the message contains padding octets or not, and may be needed by some encryption algorithms), Extension (indicates if there is an RTP header extension), Contributing Source Count (contains the number of contributing source (CSRC) IDs that follow the fixed header), Marker (interpreted by an application profile), Payload Type (identifies the payload format), Sequence Number (increments by one with each packet and is used by the receiver to reorder the packets), Timestamp (indicates the time when the first octet in the payload was sampled), Synchronization Source Identifier (identifies the source of RTP packets), and Contributing Source Identifier (if a mixer has been used, this field carries a list of sources that have contributed to the mixed media stream).

4.2.5 SIP Threat Model

In this subsection and the following one (4.2.6), we shed some light on the threats that surround SIP and RTP. The aim is to prepare the reader for the more detailed discussion on attacks in Section 4.6. SIP is susceptible to the following threats and attacks:

- Denial of service: The consequence of a DoS attack is that the entity attacked becomes unavailable. DoS attacks include scenarios like targeting a certain UA or proxy and flooding them with requests.

- Eavesdropping: If messages are sent in clear text, any malicious user can eavesdrop and get session information, making it easy for attackers to launch a variety of hijacking-style attacks.

- Tearing down sessions: An attacker can insert messages like a CANCEL request to stop a caller from communicating with someone else. It can also send a BYE request to terminate the session.

- Session hijacking: An attacker can send an INVITE request within dialog requests to modify requests en route to change session descriptions and direct media elsewhere.

- Man in the middle: This attack is where attackers tamper with a message on its way to a recipient [40].

4.2.6 RTP Threat Model

Attackers can inject artificial packets with higher sequence numbers that will cause the injected packets to be played in place of the real ones [13]. Flooding with RTP packets not only deteriorates the perceived quality of service (QoS) but also may cause phones dysfunctional and reboot operations [31].

4.3 Specification Development

Our specifications for SIP and RTP EFSMs and packet verifiers are based on RFCs 3261

and 1889 respectively. Request for Comment (RFC) documents provide designers and programmers with rich information regarding the operation and message flow of a certain protocol. However, RFC documents usually contain very detailed descriptions that could be time-consuming if implemented precisely. Furthermore, precise implementation of RFCs may be undesirable due to the inevitable discrepancies among different implementations of a protocol EFSM. Such discrepancies could make the same traffic be classified differently by different EFSMs of the same protocol. Therefore, we implement the essential details that describe a protocol in a more abstract way.

4.3.1 Session Initiation Protocol

For SIP, our state machine implementation is based on the base types of requests defined in RFC 3261.

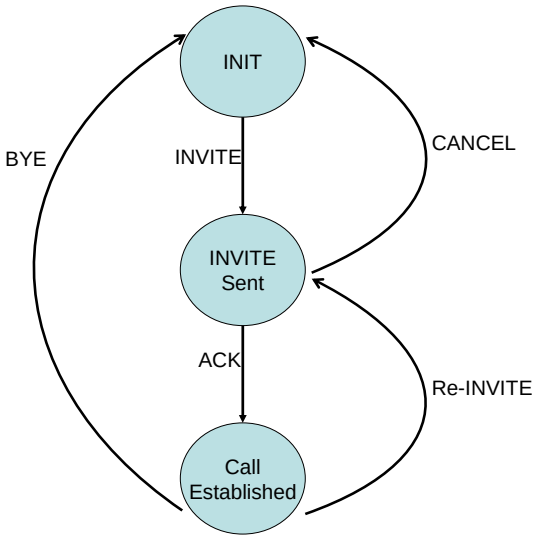


Figure 4-4. Simplified SIP State Machine.

A certain client starts at the initial state *INIT* where no connection is established. An INVITE request is sent by the client if it wishes to start a call. If the client does not want to proceed with the call attempt, it can send a CANCEL request setting the state machine back to the initial state. Otherwise, an ACK message is sent following the callee's acceptance to start a

call and change the state to *Call Established*. After call establishment, a client can send a Re-INVITE request if it wishes to move the call to another device without tearing down the session. A client can terminate a call by sending a BYE request. Figure 4-4 shows the above described state machine of a SIP client. For the sake of simplicity, we have not included the remaining two requests, namely, REGISTER and OPTIONS nor have we included the various types of SIP responses and message codes in the figure.

SIP packet verifier is designed to accept messages that are conformant to SIP specifications. A SIP message consists of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. The start-line, each message-header line, and the empty line must be terminated by a carriage-return line-feed sequence (CRLF). The empty line must be present even if the message-body is not.

For SIP requests, the start line, which is referred to as the request line in this context, contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character. Method names are discussed in detail in Section 4.2.1. Request-URI, which indicates the user or service to which this request is being addressed, must not contain non-escaped spaces or control characters and must not be enclosed in "<>". The SIP-Version string is case-insensitive, and includes the version of SIP in use.

For SIP responses, the start line, which is referred to as the status line in this context, consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by a single SP character. The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request and is discussed in detail in Section 4.2.1.

Each header field consists of a field name followed by a colon (":") and the field value. Table 4-1 shows the mandatory headers in every SIP request and response and their format. It is important to note that the brackets around parameters indicate that they are optional and are not part of the header syntax. Whenever (;parameters) appears it indicates that multiple parameters can appear in a header and that semicolons separate the parameters. For the sake of simplicity, we do not mention the different requirements for messages inside or outside a dialog although

they have been implemented.

Table 4-1. Format of mandatory SIP Headers

Header Name	Header Format	Examples and Comments
To	To: SIP-URI(;parameters)	To: Carol <sip:carol@chicago.com>. The display name *Carol* is optional
From	From: SIP-URI(;parameters)	From: Alice <sip:alice@atlanta.com>;tag=1928301774. The tag parameter contains a random string that is used for identification purposes.
Call-ID	Call-ID: unique-id	Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com. Call-IDs are case-sensitive and are simply compared byte-by-byte.
CSeq	CSeq: digit method	CSeq: 4711 INVITE. The method must match that of the request. The sequence number value must be expressible as a 32-bit unsigned-integer and must be less than 2^{31}.
Via	Via: SIP/2.0/[transport-protocol] sent-by(;parameters)	Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds. The protocol name and protocol version in the header field must be SIP and 2.0, respectively. The Via header field value must contain a branch parameter that is used to identify the transaction created by that request and is used by both the client and the server.

Max-Forwards	Max-Forwards: digit	The value of this header field should always be 70.
Contact	Contact: SIP-URI(;parameters)	Contact: <sip:alice@pc33.atlanta.com>. This header field is mandatory for requests that create dialog.

4.3.2 Run-time Transport Protocol

For RTP, the implementation of the state machine is simpler due to the lesser number of states in the protocol state machine.

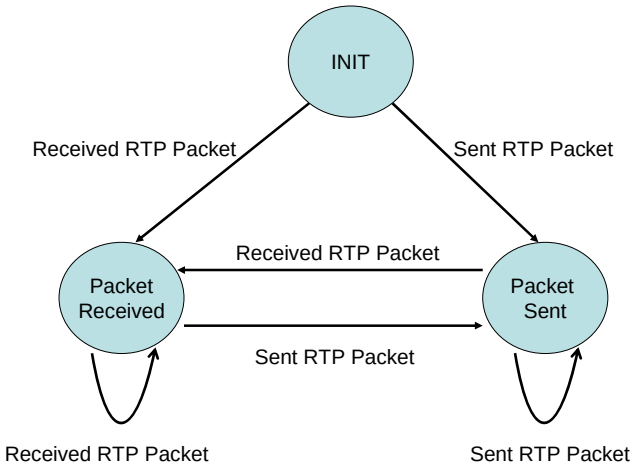


Figure 4-5. Simplified RTP State Machine.

A client starts at the initial state *INIT* where it can either receive or send packets. Upon receiving a packet, the state changes to *Packet Received*. Whilst at that state, a machine can either send a packet changing the state to *Packet Sent*, or remain at the same state receiving more packets. Similarly, at the *Packet Sent* state a machine can either receive packets changing the state to *Packet Received*, or send more packets staying at *Packet Sent*. Figure 4-5 shows the simplified RTP state machine.

RTP packet verifier follows the protocol specifications when examining packets. Table 4-2 shows the fixed header fields of RTP packets and some constraints on their lengths and values.

Table 4-2. RTP Header Fields Sizes and Requirements

Header Name	Header Format
Version	2 bits. The version identified by RFC 1889 is 2
Padding	1 bit. If set, the packet contains one or more additional padding octets at the end, which are not part of the payload.
Extension	1 bit. If set, the RTP fixed header is followed by exactly one header extension
CSRC count (CC)	4 bits
Marker	1 bit
Payload type	7 bits
Sequence number	16 bits
Timestamp	32 bits
SSRC	32 bits
CSRC	0 to 15 items, 32 bits each. The number of items is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified.

4.4 Proposed Architecture

4.4.1 Architecture Components

The proposed architecture of our host-based intrusion detection system is shown in figure 4-6. The following is a detailed description of the architecture components.

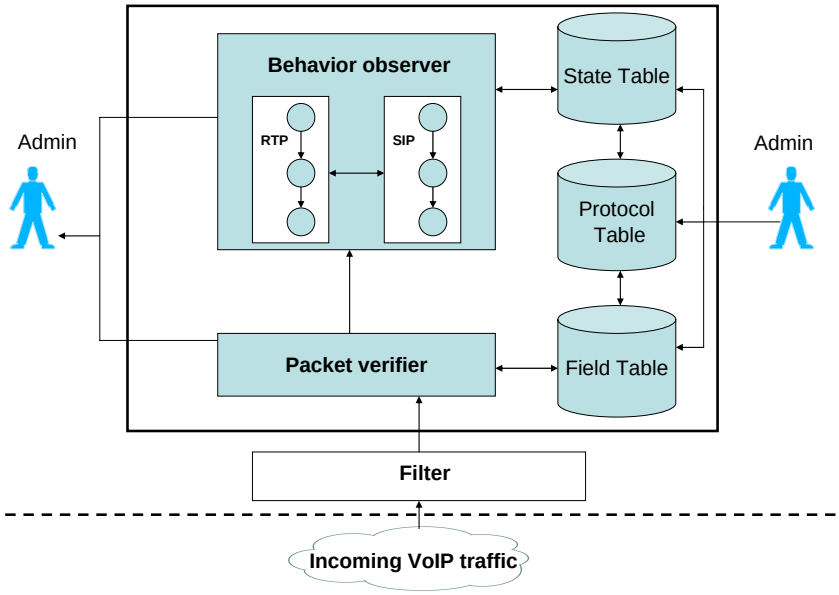


Figure 4-6. System Architecture.

1. *The Filter*: It classifies the incoming VoIP traffic into signaling and media packets. Currently, the filter supports SIP for signaling and RTP for media delivery.

2. *The Packet Verifier*: Its purpose is to validate compliance with protocol *syntax* according to standards. It checks the length of the fields, validates in terms of mandatory fields, and examines the structure of the message. This way, many unknown attacks can be detected such as attacks aiming at exploiting a vulnerability in the endpoint implementation by sending invalid protocol fields which can lead to inadvertent leakage of sensitive network topology information, call hijacking, or Denial of Service (DoS) attacks.

3. *The Behavior Observer*: The main duty of the behavior observer is to guard against *semantics* anomalies. It performs stateful detection by keeping the Extended Finite State Machines of the protocols involved in a call. Protocol EFSMs are designed based on protocol specifications, so they can detect any deviation from normal protocol behavior. This way, the behavior observer can detect unknown attacks. Each protocol EFSM is provided with *getter* functions, so that other protocol EFSMs can get values of header fields and protocol state, which benefits the system in terms of detection accuracy [72].

4. *The Protocol Table*: This table is responsible for defining protocols at a high-level of abstraction. Each record in this table defines a specific protocol supported by the intrusion detection system, and each field defines a high-level attribute of the protocol. This table is meant for organizational purposes and to add some normalization to the design of the signature database. The following is a list of the major fields in the *Protocol Table* and their functionality:

Table 4-3. Example of Protocol Table Content

Field Name	Field Content 1	Field Content 2
Protocol ID	53	54
Protocol Name	SIP	RTP
Layer	Application Layer	Application Layer
Description	A protocol used for session initiation	A protocol used for real-time transmission

- **Protocol ID**: A unique identifier that identifies the protocol supported by the system.
- **Protocol Name**: A name given to the protocol.
- **Layer**: The layer on which the protocol operates such as transport or application layer.
- **Description**: As its name suggests, this field describes the protocol and its role in the message exchange.

Table 4-3 shows an example of the content of two records from the protocol table. The table shows how our database defines SIP and RTP at a high level. The Protocol ID field is also used to join various tables as will be shown shortly.

5. *The Field Table*: Each record in this table represents a certain field in the protocol's header and a suspicious pattern associated with it. Multiple records in this table can be used to form a signature that spans across many fields and protocols. Below, is a list of the main fields and their descriptions.

- **Protocol ID**: The same as in the Protocol Table and the joiner of the two relations.
- **Field ID**: A unique identifier that identifies the field of the protocol header.
- **Field Name**: A name given to the field.
- **Description**: A description that shows the function of the field.
- **Type**: The data type of the field.
- **Pattern**: This field usually contains suspicious patterns the administrator is interested in detecting.
- **Stand-Alone Pattern**: A Boolean field to identify whether the above-described pattern forms an attack on its own, or as part of other fields. This feature enables the database to hold signatures, which span across multiple fields and multiple protocols.
- **Next Protocol ID**: If the Stand-Alone Pattern field contains False, this field points to the protocol ID of the next field in the multi-field signature.
- **Next Field ID**: If the Stand-Alone Pattern field contains False, this field points to the field ID of the next field in the multi-field signature.
- **Impact**: The effect of the attack on the system.

Table 4-4. Example of Field Table Content

Field Name	Field Content 1	Field Content 2
Protocol ID	53	53
Field ID	1	5
Field Name	Start Line	From
Description	To distinguish requests from responses	The sender of the message
Type	String	String
Pattern	INVITE	sip:alice@domain.com
Stand-alone	False	False
Next Protocol ID	53	Null
Next Field ID	5	Null
Impact	INVITE requests from *Alice* should not be received for administrative reasons	INVITE requests from *Alice* should not be received for administrative reasons

Table 4-4 depicts an example of the content of the field table. It shows two records representing a signature that includes two SIP's header fields, namely, the start line and from fields. The signature indicates that the system should raise an alert whenever an INVITE request is received from *sip:alice@domain.com*. A False in *Stand-alone Pattern* field instructs the retrieval system to retrieve the record with the *Next Protocol ID* and *Next Field ID* to form the full signature with the current field. Null values in *Next Protocol ID* and *Next Field ID* denote the end of the retrieval process [84].

6. *The State Table*: Each record in this table represents a state in the protocol's EFSM.

When a session reaches a certain protocol state, the IDS retrieves all the records associated with that state from the State Table. A record could contain various values suitable for threshold detection such as the upper limit for the number of requests allowed within a specific amount of time at that state to avoid Denial of Service saturation attacks. Furthermore, a record could contain a stored procedure to be executed upon arriving at the certain state. Such a procedure is meant to predict an impending compromise at the current system's safe state, and to limit the damage before it occurs. This strategy stems from the fact that for multi-step attacks, there are benign steps that precede the attack sequence. The administrator can provide the state table with the necessary procedures to be taken at the safe state that precedes the attack. By providing this feature, the State Table reflects the design philosophy adopted by State Transition Analysis. It should be obvious from the aforementioned description that this table deals with input that has the perfect syntax, but is trying to achieve something that violates the semantics of the protocol. Hence, it is the semantics-based component of the database. The following is a list of the main fields in the table.

- **Protocol ID**: The same as in the Protocol Table, and the joiner of the two tables.
- **State ID**: A unique identifier that identifies a state in the protocol EFSM.
- **State Name**: A name given to the state.
- **Description**: A description of the state and the system upon reaching it.
- **Threshold**: Identifies the upper limit for the number of requests that can be received at this state.
- **Time Unit**: Denotes the period of time during which the threshold is measured.
- **Timer**: Denotes a value for a timer that can be used at the state.
- **Recommended Action**: The procedure that should be executed by the system upon reaching the state to detect potential attacks.

- **Impact**: The effect of the attack on the system.

Table 4-5. Example of State Table Content

Field Name	Field Content
Protocol ID	53
State ID	20
State Name	REGISTER Received
Description	The system state after receiving REGISTER
Threshold	100 MSG
Time Unit	1 SEC
Timer	Null
Recommended Action	REGISTER_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the Registrar server

Table 4-5 shows an example of a signature from the state table. The signature shown in Table 4-5 indicates that the IDS should raise an alert whenever the number of REGISTER messages exceeds 100 within 1 second at the REGISTER Received state.

4.4.2 How Architecture Components Interact with Each Other

The *filter* is the first component to receive the incoming VoIP traffic. It helps classify the traffic into signaling and media packets and forward packets to the right verifier. The *packet verifier* receives packets from the filter and parses them. The parsing process examines the packet in terms of its size and structure. Too big and malformed packets are rejected by the packet verifier in order not to deplete the processing power of the endpoint. After examining the

general structure of the packet the verifier starts checking the header fields individually. It checks whether mandatory fields are present and if their values are within the limits defined by the protocol specifications. After checking compliance with specifications for a certain field, the system retrieves all the records of the field from the *field table* to perform signature detection. If approved, packets are sent to the *behavior observer*.

The *behavior observer* keeps track of the session and whether it progresses according to specifications. This session awareness is achieved by keeping EFSMs for the protocols involved to guard against any unacceptable behavior that violates proper protocol semantics. When reaching a certain state in the EFSM, the system retrieves all the records of that state from the *state table* to perform further checks on semantics violations. Clearly, detecting and reporting attacks take place in real-time [72].

4.4.3 How Architecture Components Reflect the Formal Model

As mentioned in Chapter 3, the theoretical foundation of our design is based on Communicating Extended Finite State Machines and State Transition Analysis Techniques. These two formal models are adopted by two components in our design, namely, the *Behavior Observer* and the *State Table*. In this subsection we show how these two components reflect the essence of the formal models they are built upon.

The *Behavior Observer* starts by using *enumerated values* to define EFSM *states*. In addition, it provides *state variables* associated with each EFSM to act as *context variables*. Such variables can serve various purposes including holding values carried by *input* or *output signals*. The *Behavior Observer* has dedicated *methods* working as *predicates* to enable *transitions* based on the current *state* and *input signal*. Upon enabling a transition, a special *method* is triggered changing the *state* of the EFSM. Among the methods dedicated by the *Behavior Observer* are those which update the value of the *state variables* and produce output. The *Behavior Observer* introduces *error states* to model the case when invalid input is received at a certain state. An EFSM in the *Behavior Observer* does not work as a single independent system, but rather as part of a large system of Communicating EFSMs. Therefore, the *Behavior Observer* provides *getter* functions associated with every EFSM to help other EFSMs to access the values of *context*

variables easily.

Figure 4-7 delineates the relationship between the *State Table*, and State Transition Analysis Techniques. In the figure, a *State Table* lies on the right side, whereas attacks represented by State Transition Analysis lie on the left. Each record in the *State Table* represents a *safe state* in State Transition Analysis. As mentioned earlier, a *safe state* usually precedes a *compromised* one. A *compromised state* can be predicted and dealt with using the *stored procedure* that forms an important part in every record in the *State Table*. *Stored procedures* use *Threshold* and *Timer* fields associated with records to provide more flexible detection parameters. Furthermore, *Thresholds* and the associated *Time Units* in the *State Table* can be used to extend the capabilities of State Transition Analysis to detect Denial of Service attacks and the likes. Therefore, the *State Table* adopts the design approach of State Transition Analysis bringing semantics awareness to attack modeling and improves the same approach to model attacks that were difficult to model previously.

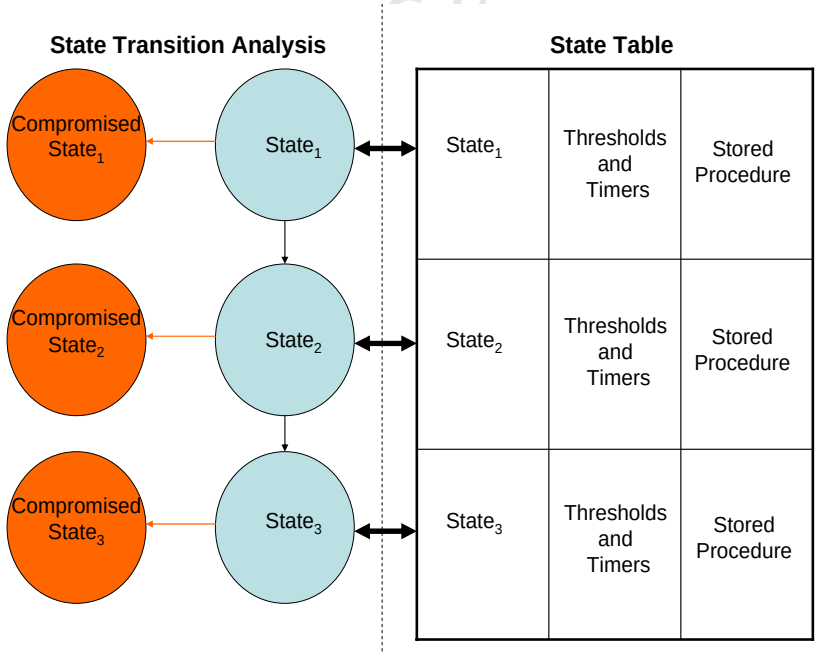


Figure 4-7. The Relationship Between The State Table and State Transition Analysis.

4.5 Advantages of Architecture

1. Our architecture provides a stateful and cross-protocol detection in specification-based and

signature-based modules. The *behavior observer* performs stateful detection by keeping the EFSMs of all the involved protocols and assembling state from multiple packets. It also performs cross-protocol detection by providing external interfaces between protocol EFSMs in the form of callable functions which return values of important protocol state variables. On the other hand, signature-based modules have a sense of statefulness and cross-protocol awareness. The *field table* has the ability to store signatures that cross protocol boundaries. Furthermore, the *state table* follows the progress of protocol sessions carefully providing stateful detection. The functions stored in the *Recommended Action* field have the ability to perform cross-protocol detection.

2. The design of the database tables is simple and clean. This advantage is achieved by separating the anomalies in protocol traffic from specific attacks. The *packet verifier* and *behavior observer* eliminate anomalies according to protocol specifications. They also remove ambiguities in the incoming traffic which lets the *field table* and the *state table* focus on the modeling of specific attacks rather than all anomalous behaviors. For example, if the *packet verifier* were not present in the design, we would have – somehow – to store signatures for all malformed SIP messages. Bearing in mind that there are different types of SIP messages, and for each message there are multiple headers with different combinations, we would end up storing a huge number of signatures. Since signatures are stored in the external memory which has a considerably longer access time than the random access memory, simplifying the design and limiting the number of signatures should be a goal sought by IDS designers.

3. Our design maintains a reasonable balance between database normalization and performance. A normalized database has many one-to-one relationships, and many tables to reflect these relationships. Such a design suggests small tables with a relatively few attributes for each. Although high normalization provides a clearer overview on the database for designers, it usually comes at the expense of retrieval time. For example, the primary type of information in Snort system [26] which is called the event, is represented in no less than six tables. Accessing the information of a single event may require joining six or more tables which could affect performance negatively [42]. The number of tables to be accessed grows directly with the number of protocols involved in a signature. On the

other hand, and as shown in figure 4-8, we provide a less normalized database (two levels of hierarchy) with more attributes per table. A signature in our database is entirely stored in a single table (either *field* or *state table*). For reporting purposes, another table (the *protocol table*) is accessed, which puts the cost of accessing information in our database at no more than two tables. Furthermore, unlike other comparable signature databases, a cross-protocol signature can be stored in a single table (either *field* or *state table*), which benefits the performance positively.

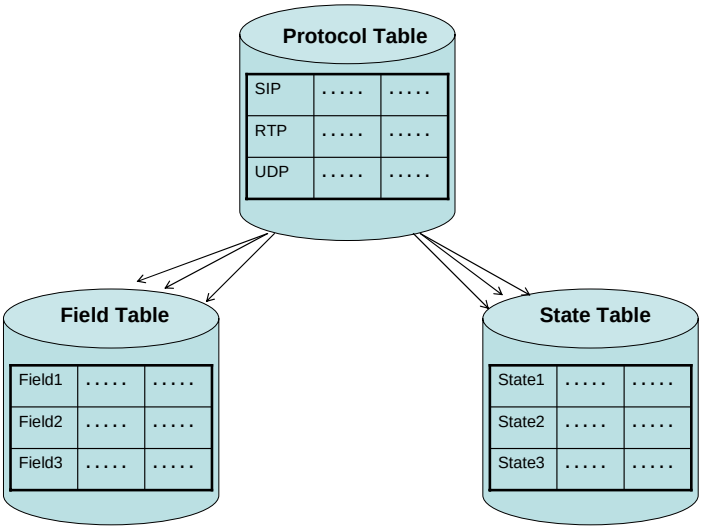


Figure 4-8. A High-level Hierarchy of the database.

4. Our system can thwart obfuscation attempts made by attackers to evade detection. Obfuscation is a technique used by attackers to introduce slight modifications in a certain penetration hoping to do the same damage, without being detected by the signature database that stores a signature for the unmodified penetration. Thus, even if the attack scenario is represented in the signature database, a minor variation of the attack can go unnoticed. Our system tackles this problem by representing attacks in the *state table* using a higher-level representation. Upon reaching the *safe* state that precedes a *compromised* one in an attack, the system executes the procedure in the *Recommended Action* field. Such a procedure can deal with different types of *intermediate* states that represent variants of

the attack. Furthermore, more than one procedure can be stored in the *Recommended Action* field to add more flexibility and deal with more variants of the attack [84].

5. Extending State Transition Analysis Techniques and combining them with specification-based ones allow our IDS to detect attacks that are difficult to detect using either of the approaches. Denial of Service attacks and variations from normal usage are clear examples of attacks that cannot be represented easily by State Transition Diagrams [39] which form the base of our EFSM model and State Transition Analysis techniques. However, by improving State Transition Analysis and incorporating it into specification-based module that uses EFSMs, we are able to detect the abovementioned attacks efficiently.

4.6 Implemented Attacks and Detection Methodologies

We implement six attacks to demonstrate the functionality of the intrusion detection system at the application layer. The attacks are launched exploiting various vulnerabilities in SIP as a signaling protocol and RTP as media transport protocol. The implemented attacks can be classified either as flooding attacks, message flow attacks, or parser attacks. Some of these attacks can be found in classifications such as the one released by VoIPSA for threats that IP telephony is vulnerable to [41]. Such attacks are common in VoIP environments since current SIP specifications do not mandate authentication for all types of requests used by the protocol. Furthermore, existing security mechanisms that guarantee message integrity, confidentiality, and origin authentication can only protect against outsiders and not against insiders who abuse their privileges. The rest of this section discusses the attacks and the detection methodology for each

4.6.1 The BYE Attack

As mentioned earlier, a BYE request can be sent by either the caller or the callee to terminate the session. An attacker can abuse this feature by sending this message to either the caller or the callee to fool them into tearing down the session prematurely. The User Agent that receives the faked BYE message will immediately stop sending RTP packets, whereas the other User Agent will continue sending its RTP packets. BYE attack is common in VoIP environments and can be accomplished either by sniffing the network or performing a man-in-the-middle

attack to insert a BYE request into the session. Wherever there is no authentication mechanism in place, and considering the attacker's ability to discover the current session parameters, this attack can be launched successfully. BYE attack is considered a Denial of Service (DoS) attack

Table 4-6. BYE Attack Signature

Field Name	Field Content
Protocol ID	53
State ID	30
State Name	BYE Received
Description	The system state after receiving BYE
Threshold	Null
Time Unit	Null
Timer	20 MSEC
Recommended Action	BYE_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the endpoint

Although BYE attack occurs within the signaling protocol (SIP), checking the status of RTP flow in the endpoint is vital in the detection process. A genuine BYE sender will stop sending RTP packets immediately after sending a BYE message. Receiving RTP packets from the original sender on the original port after seeing the BYE message is an indicator of a BYE attack. To detect such an attack, we store a signature in the *state table* of our database. The stored signature represents the state of a SIP session upon receiving a BYE message. We set a value to the *timer* field in the signature. The recommended action includes a cross-protocol detection procedure that checks RTP status after receiving the BYE message. If the system receives any RTP packets before the timer expires, it is an indication a BYE attack is taking

place. Table 4-6 shows the signature. The pseudo-code of BYE_Procedure( ) which is the recommended action appears in figure 4-9.

```
Procedure BYE_Procedure ( )

  while (Timer > 0)

   { if (RTP packets are received from original address)

        Raise_Alarm (BYE_attack)

     else

        Timer = Timer -1

   }
```

Figure 4-9. The Pseudo-code for BYE Attack Detection.

In this example we choose the *Timer* value to be 20 milliseconds, which is the time each voice packet represents on average.

A point worth noting is that network conditions could scupper the aforementioned strategy. If RTP packets are delayed beyond the average time after receiving a legitimate BYE request due to network congestion, our database will generate a false positive.

4.6.2 The Re-INVITE Attack

Another name for this attack is Call Hijacking. SIP clients use Re-INVITE message if they want to move the phone call from one device to another without tearing down the session. This feature is called call migrating. An attacker can abuse this feature by sending a Re-INVITE message to one of the parties involved in a session to fool it into believing that the other party is going to change its IP address to a new address. The new address is controlled by the attacker. This attack can be seen as a DoS attack. Furthermore, it breaches the privacy of the call since the attacker will be able to receive voice that is not meant for it. Lack of authentication enables attackers to launch this attack against endpoints.

To detect Re-INVITE attacks we use an approach similar to the one used to detect BYE attacks. Clearly, continuing to receive RTP packets from the original address on the original port after receiving a Re-INVITE denotes a call hijacking attempt. We create a signature in the *state table* denoting the system state upon receiving a Re-INVITE. Similar to the approach used in BYE attack, we set a value to the *timer* field in the signature.

Similar to BYE attack, if a benign Re-INVITE arrives before RTP packets due to taking a different path or any other network conditions, the system will raise a false flag. Packets between two endpoints in an IP-based network are not confined to a certain route. Such a scenario is rare although it is possible.

4.6.3 The CANCEL Attack

CANCEL message is sent if the caller decides not to proceed with the call attempt. It asks the callee to cease processing the previous request and generate an error response designating that request. It is sent usually after receiving a provisional response from the callee. Provisional responses indicate that the request has been received, and is being processed by the callee. Without proper authentication, the receiving user agent cannot differentiate a faked CANCEL message from a genuine one, which leads to a denial of communication between user agents.

Our system detects this attempt by carefully monitoring the behavior of the signaling protocol in the *behavior observer*. Sending a CANCEL after receiving OK response or not receiving a provisional response would be incorrect protocol behavior. Deploying our IDS prototype on all components of the network guarantees that CANCEL is sent only if a provisional response is received and any OK response is not received. This way the attack is detected early on the attacker side. This detection methodology shows the statefulness and compliance to specifications of our system. It is important to realize that CANCEL attack can be detected by our system without having to encode its pattern or store its signature. The *behavior observer* which follows correct protocol behavior is able to detect the attack without priori knowledge of its signature.

4.6.4 The REGISTER Flooding Attack

Overwhelming victim resources by flooding it with malicious traffic is the most basic and probably the most difficult to defend against DoS attack. A number of SIP clients can launch a REGISTER flooding attack to swamp a single registrar server within a short duration of time. REGISTER requests are accepted by registrar servers to store a binding between a user's SIP address and the address of the host where the user is currently residing or wishing to receive requests. REGISTER flooding attack can be viewed as a DoS attack. Even proper authentication would not stop such an attack if the attackers are insiders with bad intentions.

To detect this attack, we create a signature in the *state table* denoting the system state upon receiving a REGISTER request. Two values are set to the *Threshold* and *Time Unit* fields respectively. Whenever the number of REGISTER requests exceeds the threshold within the specified time unit, the system raises a REGISTER flooding attack flag.

4.6.5 Voice Injection Attack

This attack targets RTP which is used to carry call data such as voice and video. Lack of integrity checking could allow an attacker to inject an alternative RTP stream to one of the parties involved in a session. An attacker can send artificial RTP packets with higher sequence numbers than the original ones, which causes the receiver to play the artificial ones instead.

To detect such an attack we can store a signature in the *state table* to denote the system state upon receiving an RTP packet. A special procedure in the *Recommended Action* field should compare the sequence number of the packet to that of the previous one. Whenever there is an increase that exceeds the number in the *Threshold* field, an alarm is raised. Table 4-7 shows the signature.

Table 4-7. Voice Injection Signature

Field Name	Field Content
Protocol ID	54
State ID	7
State Name	RTP Received
Description	The system state after receiving an RTP packet
Threshold	50
Time Unit	Null
Timer	Null
Recommended Action	Voice_inj_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the endpoint

4.6.6 Malformed Messages Attack

Attackers can create extra-long messages with fields of increased length or huge message body. They can also omit some of the mandatory fields in the messages being sent. Such an attack targets the protocol parser at the endpoint and aims at depleting its processing power and increasing its network utilization. Different implementations of the protocol could respond to such messages in different ways. It is likely that attackers try various malformed message combinations to discover a flaw in the end system. In addition, such malformed messages could lead some endpoints to crash which is considered a DoS situation. Malformed messages attack targets both signaling and transport protocols.

To detect such attacks the *packet verifier* provides input validation for the incoming packets before they are passed to the parser. It checks every incoming packet for adherence to

the protocol specifications in terms of field presence, length, and other criteria. Therefore, the system does not have to possess priori knowledge of the attack signature to detect it.

4.7 Attack Summary

Table 4-8 recapitulates the attacks mentioned in the previous section. For each attack, we provide a brief description and the violation type.

Table 4-8. Application Layer Attack Summary

Attack Name	Brief Description	Violation Type
BYE Attack	A faked request sent by attackers to fool the parties involved in a session into tearing it prematurely.	Protocol-Semantics Violation
Re-INVITE Attack	A faked request sent by attackers to one of the parties involved in a session to fool it into redirecting the call to the attacker.	Protocol-Semantics Violation
CANCEL Attack	A faked request sent by attackers to cancel a call attempt made by legitimate users.	Protocol-Semantics Violation
Malformed Messages	Malformed protocol messages created by attackers to hamper victim processing	Protocol-Syntax Violation
REGISTER Flooding	Overwhelming registrar servers with too many requests within a short time.	Protocol-Semantics Violation
Voice Injection	Injecting an alternative voice stream to one of the parties involved in a session.	Protocol-Semantics Violation

4.8 Chapter Discussion

This chapter started the discussion on the design of our system from an application layer

point of view. It started by mentioning the two major protocols involved in SIP-based IP telephony, namely, Session Initiation Protocol (SIP) and Real-time Transport Protocol (RTP). The roles played by these protocols in a call, their message formats, and threat models were mentioned preparing the reader to apprehend the magnitude of the threats. The specifications of the involved protocols were developed based on abstract models that focus on the essential details of each protocol, and the benefits of that decision were mentioned.

The chapter then thrashed out the design of the proposed architecture focusing on its components and their interaction with each other. The faithfulness of our design to the formal models of Extended Finite State Machines and State Transition Analysis was clearly shown. The chapter discussed thoroughly the advantages of the proposed design. These advantages included cross-protocol and stateful detection in specification and signature-based modules, efficiency in storing signatures and retrieving them, and resisting obfuscation. Most importantly, we showed how our design improved the formal model to detect attacks that were hard to detect previously. A variety of attacks were shown alongside their detection methodologies to highlight the system's ability and versatility. Next chapter will continue with the design addressing lower layers protocols, threats, and attacks.

Chapter 5 Extending the Design to Lower Layers

5.1 Introduction

In the previous chapter we started a thorough discussion on the design of our intrusion detection system. As an introduction we shed some light on the involved protocols with their threat model and how their specifications were developed. Then we dived into the details of our proposed host-based intrusion detection architecture dissecting its components and highlighting its advantages. Six implemented attacks with their detection methodologies were discussed showing the capabilities of our system.

In this chapter we follow the same path that was started in the previous chapter. However, we shift our focus from application layer protocols to protocols at lower layers. Our discussion will follow the same systematic approach to show how our design addresses lower layer issues using the same principles utilized for application layer.

5.2 Related Protocols and Threat Model

TCP/IP networking is implemented in a layered fashion that has four levels [43]. The previous chapter dealt with the application layer level that is seen by the average user. The Transport layer and Network layer levels are the focus of this chapter. The lowest level which is the Network Interface layer has the task of interfacing with the network hardware. Special protocols are implemented at the Network Interface layer level for data transmission across different kinds of networks. Attacks that target protocols at this level are beyond the scope of this thesis, so they will not be covered in any detail.

The Network layer handles communication from one machine to another. It accepts a request to send a packet from the Transport layer along with an identification of the machine to which the packet should be sent. This layer defines the basic unit of transfer across the network and includes the concepts of destination addressing and routing [44].

The Transport layer is where reliability of delivery is implemented. This layer also

provides communication from one application program to another. Such communication is often called end-to-end [44].

5.2.1 Internet Protocol (IP)

The most fundamental TCP/IP internet service consists of a packet delivery system. The service is defined as an unreliable, best-effort, connectionless packet delivery system. It is unreliable because packets may be lost, duplicated, delayed, or delivered out of order without the service detecting such conditions or informing the sender or receiver. The service is connectionless because each packet is treated independently from all others and may travel over a different path from the rest. The service is best-effort because it does its best to deliver packets. However, exhaustion of resources or failure of underlying network could scupper that best effort.

The Internet Protocol (IP) is the protocol that defines the delivery system described in the previous paragraph. It defines the basic unit of data transfer used throughout the TCP/IP internet. Furthermore, the IP software performs the routing function, choosing a path over which packets will be sent. Therefore, IP is the underlying language that all machines on the Internet must understand in order to communicate. Figure 5-1 shows the IP packet format.

The Version field is always set to 4 indicating IPv4. The length field gives the packet header length in 32-bit words. The service type field specifies how the packet should be handled and routed. The Total Length field gives the length of the packet in bytes. Each packet has an identifier contained in the identification field. The Flags and Fragment Offset fields are used when the packet is too large to traverse a given network and must be broken into smaller packets. The Time To Live field specifies how long the packet is allowed to remain in the internet system. Each gateway that handles a packet decrements the Time To Live field, and the packet gets dropped when the field reaches zero. The Protocol field tells which protocol is used by the Transport layer. The Header Checksum ensures integrity of header values. Fields Source IP Address and Destination IP Address contain the 32-bit IP addresses of the packet's sender and intended recipient. The Options field allows the selection of a number of possible routing and recording choices.

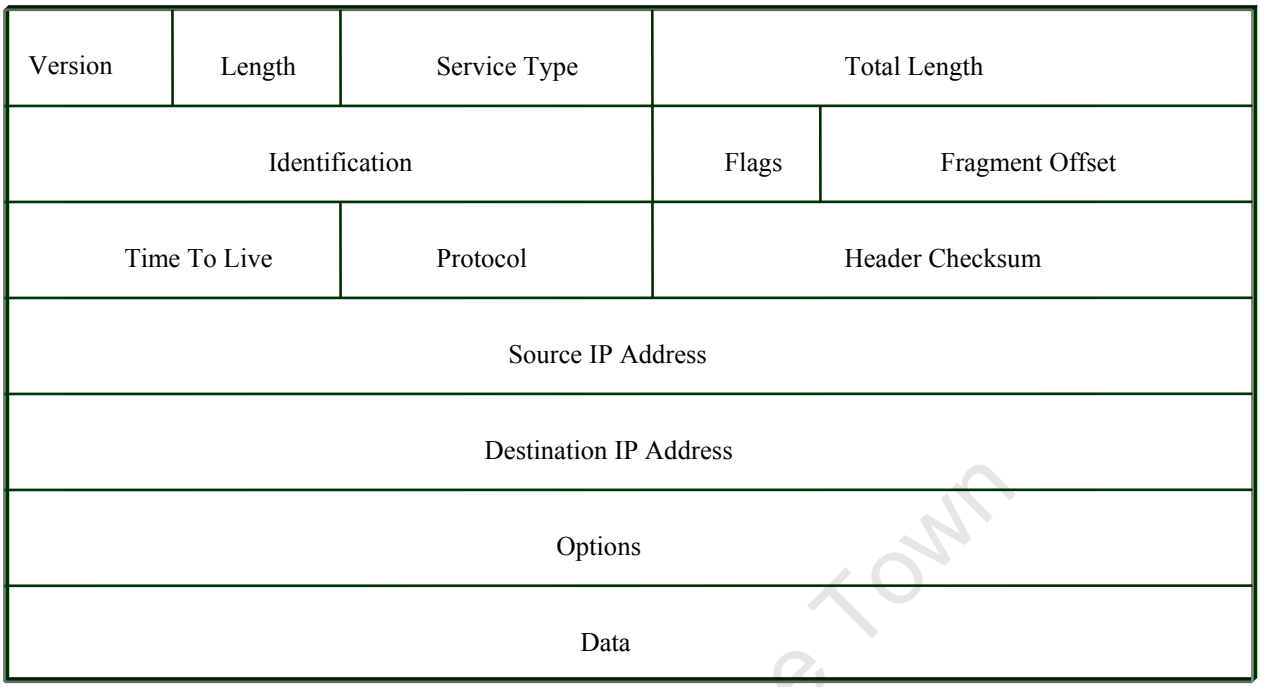


Figure 5-1. IP Packet Format.

5.2.2 Internet Control Message Protocol (ICMP)

If IP software on a gateway cannot route or deliver a packet due to the unavailability of the destination machine, or if the gateway detects an unusual condition, like network congestion, that affects its ability to forward the packet, the original source of the packet needs to be instructed to take actions to avoid or correct the problem. To allow gateways to report errors or provide information about such unexpected circumstances, the Internet Control Message Protocol (ICMP) is implemented as part of IP implementation. ICMP is used to send error messages or other information pertinent to the functioning of the network. It provides communication between the IP software on one machine and the IP software on another.

Although each ICMP message has its own format, they all begin with the same three fields: the Type field that identifies the message and defines its meaning and format, the Code field that provides further information about the message type, and the Checksum field which insures integrity. The Data field of ICMP messages can contain extra header fields for specific types and codes. Figure 5-2 shows the ICMP general header, whereas Table 5-1 contains a description of the message types.

Table 5-1. ICMP Message Types

Type	Description	Purpose
0	Echo Reply	Query
3	Destination Unreachable	Error
4	Source Quench	Error
5	Redirect	Error
8	Echo Request (Ping)	Query
9	Router Advertisement	Query
10	Router Solicitation	Query
11	Time Exceeded	Error
12	Parameter Problem	Error
13	Timestamp Request	Query
14	Timestamp Reply	Query
15	Information Request	Query
16	Information Reply	Query
17	Address Mask Request	Query
18	Address Mask Reply	Query

Type	Code	Checksum
Data		

Figure 5-2. ICMP Header.

5.2.3 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) provides a mode of communication between applications. UDP implements the concept of *protocol ports* used to communicate with different applications. Protocol ports are logical constructs rather than physical ones. Each protocol port is identified by a positive 16-bit integer. The local operating system on a machine provides an interface mechanism that applications use to specify a port or access it. To communicate with a foreign port, a sender needs to know both the IP address of the destination machine and the protocol port number of the destination application within that machine. UDP provides protocol ports to distinguish among multiple applications executing on a single machine. It uses the underlying IP and provides unreliable connectionless packet delivery.

Source Port	Destination Port
Length	UDP Checksum
Data	

Figure 5-3. UDP Header.

The UDP header appears in figure 5-3. The Source Port and Destination Port fields contain the 16-bit UDP protocol port number used to direct packets to applications waiting to receive them. When the Source Port is used, it specifies the port to which replies should be sent. The Length field contains the length of the UDP packet in bytes, including the UDP header and

the user data. The Checksum field is used to insure integrity.

5.2.4 Transmission Control Protocol (TCP)

As mentioned in Section 5.2.1, the lowest level of service provided by a TCP/IP network is the unreliable packet delivery system. In such a system packets can be lost due to transmission errors or hardware failures. Furthermore, packets can be delivered out of order or after a substantial delay. Such an unreliable connectionless delivery system can hinder the operation of application programs that need to send large volumes of data from one machine to another. Application programmers in this system are required to build error detection and recovery into their applications which entails a special technical background possessed by few application programmers. Therefore, a reliable transport service on top of the unreliable connectionless one was proposed to provide reliable stream delivery system to be used by all application programs.

The Transmission Control Protocol (TCP) is the protocol that implements reliable communication on the Internet. It implements the concept of a connection which can be thought of as a communication channel, where both sides have agreed on the communication, and mechanisms are put in place to ensure that all packets arrive unchanged at their destination. TCP has several key features that provide reliability. First, TCP acknowledges the receipt of each packet. It maintains a timer, and if the acknowledgement is not received within the predefined time limit, it resends the packet. Second, it includes a unique identifier for each packet to ensure that the receiving application can reconstruct the correct order and also detect when packets are lost. Finally, TCP ensures that the sending machine never sends too much data to be stored in the receiver's buffer.

Figure 5-4 depicts the TCP header. The Source Port and Destination Port fields play the same role played by the Source Port and Destination Port fields in the UDP header, indicating which application is the sender and recipient of the packet. The 32-bit Sequence and Acknowledgment numbers are used to ensure that the packet ordering is maintained and that no packets are lost. The Length field includes the length of the header in 32-bit multiples. The Reserved field is an area reserved for future extensions to TCP. The Flags are bit values within a 6-bit field, used to implement and control the connection. The URG flag indicates that the

Urgent pointer (which will be discussed shortly) is valid. ACK flag is used to acknowledge the receipt of a packet. PSH flag indicates that the data should be pushed up to the application as soon as possible. The RST flag resets the connection. SYN flag is used to initiate the connection and synchronize it. The FIN flag is used to finish the connection and to indicate that the sender is done sending data. The Window Size is the number of bytes that the receiver is willing to accept. The Checksum is used to ensure the integrity of the packet. The Urgent pointer is a way for an application to send emergency data to the receiver. When the Urgent flag is set, the Urgent pointer carries the offset to be added to the current sequence number to indicate the last byte of the urgent data. The Options provide various choices for the connection.

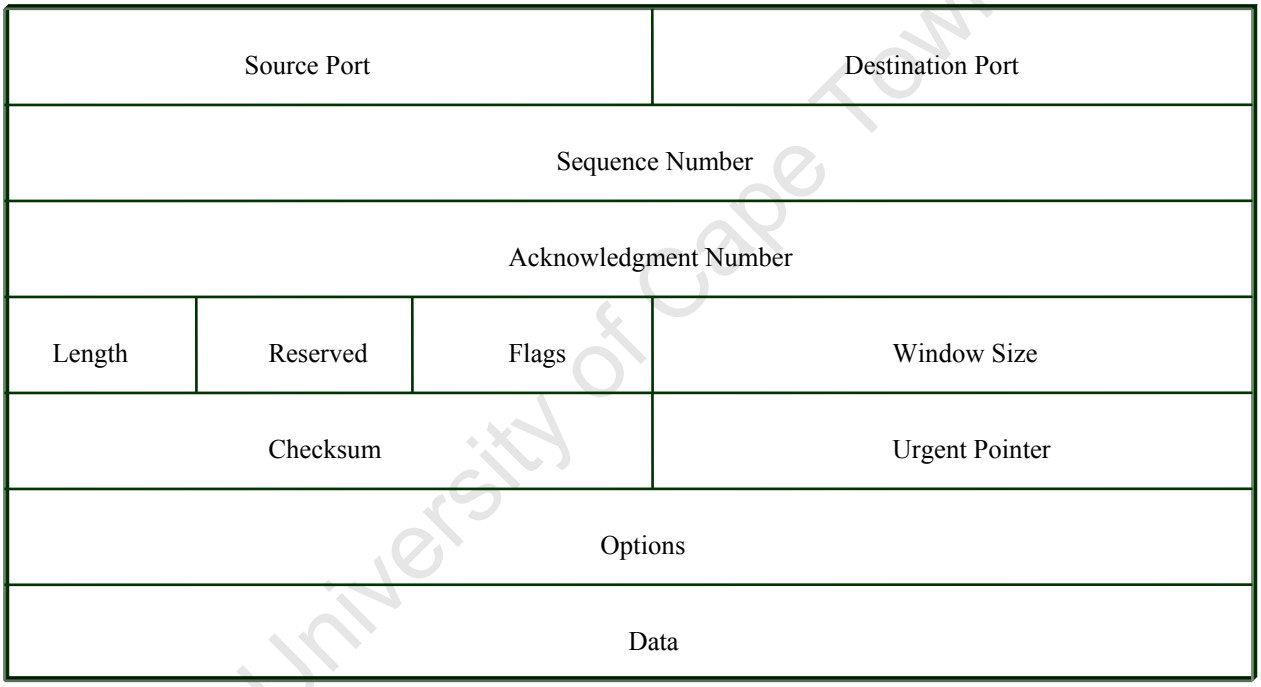


Figure 5-4. TCP Header.

5.2.5 Threat Model

Ever since Bellovin's paper on the security problems in the TCP/IP protocol suite [45], dozens of papers have been published on the same issue. The way TCP accepts new connections allows attackers to launch denial-of-service attacks to prevent anyone from using a particular host. Generally, attackers can take advantage of the stateful nature of TCP to cripple the protocol whilst in a certain state. The lack of built-in authentication with IP packets makes it easy for

attackers to spoof packet addresses and get unauthorized privileges. Some could argue that lower layers attacks are probably not effective in today's environments for they have been around for a long time. However, due to software reuse and potential coding errors in future systems, there is always the possibility that these attacks, like any other, may become effective once again against some future systems [46]. Some recent reports on the performance of a newly released version of Microsoft Windows Vista proved the effectiveness of some of these attacks against the protocol stack of the operating system [47]. That was despite the fact that Microsoft removed a large body of tried and tested code and replaced it with freshly written code.

5.3 Specification Development

Our specifications for IP, ICMP, UDP and TCP EFSMs and packet verifiers are based on RFCs 791 [48], 792 [49], 768 [50] and 793 [51] respectively. In developing the state machine specifications, we have followed the same approach mentioned in Section 4.3 by implementing the essential details of each protocol. Using more abstract specifications provides a twofold benefit. First, specification development becomes less tedious and time-consuming. Second, the IDS gets to avoid undue false alarms that result from differences in traffic interpretation. For the packet verifiers, we have benefited from some previous works in the area of traffic normalization [83] to address the issues related to ambiguities in the traffic stream that are exploitable by attackers to either evade detection or launch various probing and unknown attacks.

5.3.1 Internet Protocol

Figure 5-5 shows a simplified IP state machine for a client. The state machine starts at the initial state *INIT* where it can make a transition to either *Packet Sent* or *Packet Received* state. Upon receiving an IP packet, the state machine makes a transition to the *Packet Received* state. Whilst in this state, the state machine can receive more IP packets staying at the same state or send IP packets making a transition to *Packet Sent* state. Whilst in the *Packet Sent* state, the state machine can continue sending more IP packets staying at the same state or receiving IP packets moving to *Packet Received* state.

Table 5-2. Some IP Header Constraints

Header Name	Checks Performed
Version	For IPv4, this field must have the value 4.
Header Length	If the value of this header is less than 20 bytes or more than the packet length, the packet is dropped.
Total Length	If this field exceeds the actual length of the packet, the packet is dropped. Otherwise, the packet is trimmed to the length indicated by the field.
Protocol	In our case, we enforce the use of either TCP or UDP in this field.
Header Checksum	Any packet with incorrect header checksum is dropped.
IP Source and Destination Addresses	Any packet with invalid IP addresses is dropped. Examples of invalid IP addresses are *localhost* and *broadcast* addresses.
IP Options	In our case, we remove any IP options from the packet. Not all options are implemented by all machines. For the most part, they are not used in modern networks, and some of them can have security threats.

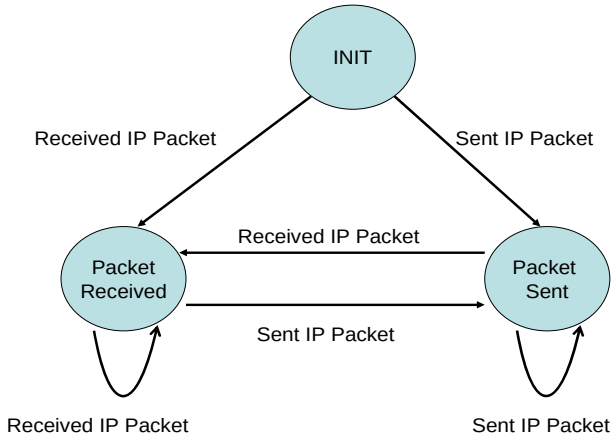


Figure 5-5. Simplified IP State Machine.

Table 5-2 shows how our IP packet verifier addresses some of the issues related to packet headers and their associated values.

Since we are implementing a host-based intrusion detection system, we have not mentioned the header fields that are more related to the operation of networks and network-based intrusion detection systems.

5.3.2 Internet Control Message Protocol

An ICMP state machine at the initial state *INIT* can either send an error or a request message.

Table 5-3. Some ICMP Constraints

ICMP Type	Checks Performed
Echo request (ping)	If the destination of the message is either a multicast or broadcast address, the message is dropped. Such action prevents the host from participating in a Smurf attack (will be discussed shortly). If the checksum is incorrect, the packet is dropped. The Code field is set to Zero.
Echo reply	If the checksum is incorrect, the packet is dropped. If the reply does not match a request, the packet is dropped. The Code field is set to Zero.
Source Quench	Source quench requests are dropped to prevent possible Denial of Service attacks. Such a measure is justifiable since routers in IP are not allowed to originate a source quench and are not obligated to act on a received source quench [85].

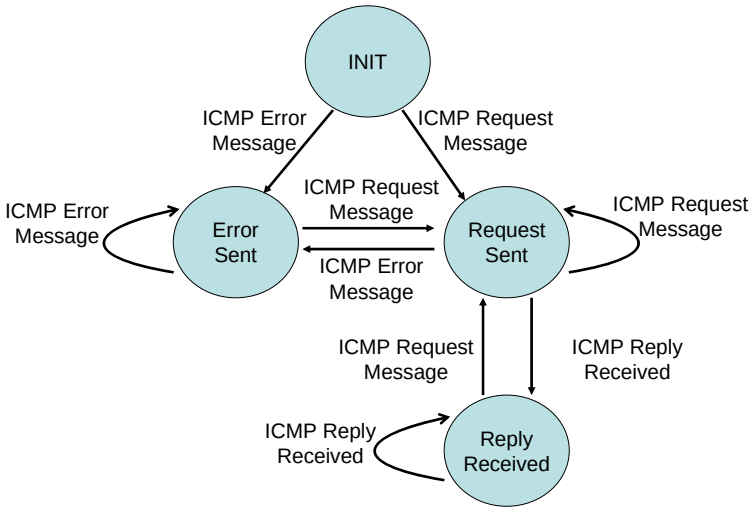


Figure 5-6. Simplified ICMP State Machine.

By sending an error message, the machine makes a transition to the *Error Sent* state. Sending more error messages keeps the state machine at this state. By sending a request message, the machine makes a transition from the *INIT* state to the *Request Sent* state. At the *Request Sent* state a machine can either send more requests staying at the same state or receive replies moving to the *Reply Received* state. Figure 5-6 shows a simplified ICMP state machine.

Table 5-3 shows some of the measures taken by ICMP packet verifier to deal with requests and replies.

5.3.3 User Datagram Protocol

A UDP state machine starts at the initial state *INIT* where it can make a transition either to *Packet Received* or *Packet Sent* state. Upon receiving a UDP packet, the machine moves to the *Packet Received* state. Whilst at the *Packet Received* state, the machine can either keep receiving more packets to stay at the same state or send packets to move to the *Packet Sent* state. Similarly, at the *Packet Sent* state the machine can either keep sending more packets to stay at the same state or receive packets to move to the *Packet Received* state. Figure 5-7 shows a simplified UDP state machine.

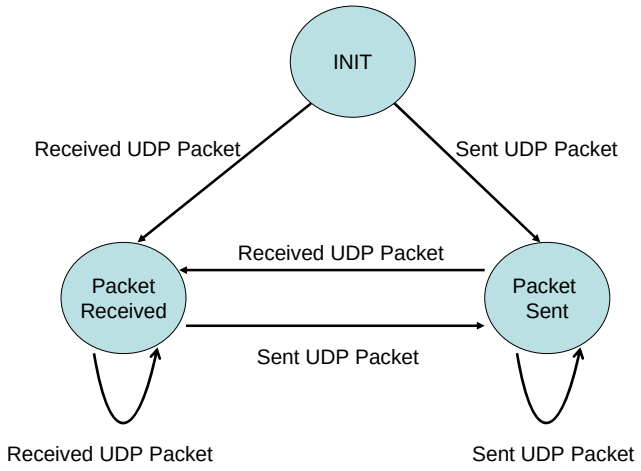


Figure 5-7. Simplified UDP State Machine.

Table 5-4 shows some of the measures taken by UDP packet verifier to deal with some header fields.

Table 5-4. Some UDP Constraints

UDP Field	Checks Performed
Length	If the value of this field does not match the length as indicated by IP total length, the packet is dropped.
Checksum	If the checksum is incorrect, the packet is dropped.

5.3.4 The Transmission Control Protocol

A TCP state machine starts at the initial *INIT* state. For a TCP server application at this state, *a passive open* command is issued to wait for a connection from another machine. The passive open command results in a transition made by the state machine to the *Listen* state, where the server application is willing to accept connections. When a TCP client application wants to establish a connection with a server, it sends a TCP packet with the SYN flag set. Upon receiving the packet, the TCP server application sends a packet with SYN and ACK flags set and moves to the *SYN Received* state. A received RST packet at this state changes the state to *Listen*. Upon receiving the acknowledgement from the client, the server makes a transition to the *Established* state and data transfer can take place accordingly. When a TCP client application wants to tear down the connection, it sends a packet with the FIN flag set to the server. Upon receiving the FIN packet the server sends a packet to the client with the FIN and ACK flags set, which closes communication from the client to the sever and makes a transition to *Close Wait* state. At this stage, the server can continue sending packets to the client until it sends its closing FIN packet, which is acknowledged by a FIN/ACK.

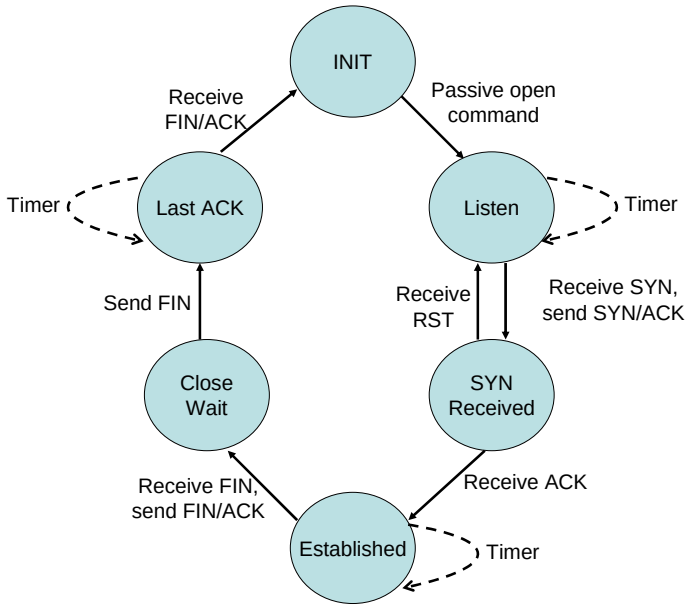


Figure 5-8. Simplified TCP State Machine.

It should be obvious that the abovementioned description of a TCP state machine applies to a server application. A TCP state machine that describes a client application would have slight differences in the form of some disparate states and transitions. For instance, a client application at the *INIT* state sends a SYN packet to a server to initiate a connection moving to the *SYN Sent* state. When a client application sends a FIN packet to end the connection, it moves to the *FIN Wait 1 state*, where it waits for the server's acknowledgment. Upon receiving the acknowledgment, the client application makes a transition to the *FIN Wait 2* state, receives a final FIN packet from the server and acknowledges it, and closes the connection after a predefined amount of time. To keep things simple, we only show the states and transitions of a TCP state machine describing a server application in figure 5-8.

Table 5-5 shows some specific measures taken by TCP packet verifier to deal with some header fields.

Table 5-5. Some TCP Constraints

TCP Field	Checks Performed
SYN	If SYN=1 and RST=1, the packet is dropped. If SYN=1 and FIN=1, the FIN flag is cleared. If SYN=0 and ACK=0 and RST=0, the packet is dropped. If SYN=1, the packet data are removed.
RST	If RST=1, the packet data are removed.
FIN	If FIN=1 and ACK=0, the packet is dropped.
PUSH	If PUSH=1 and ACK=0, the packet is dropped.
URG	If URG=1 and ACK=0, the packet is dropped.
Header Len	If the value of this field is less than 5 or beyond the end of packet, the packet is dropped.
Reserved	In our case, we remove any values in this field.
Checksum	If the checksum is incorrect, the packet is dropped.

It is clear from table 5-5 that all the checks are performed to insure adherence to the protocol specifications. For example, a TCP packet with SYN flag set is meant to initiate a connection and hence it should not carry data.

5.4 Extended Architecture

To accommodate the protocols discussed in section 5.2, we extend the system architecture that was detailed in section 4.4 to cover the threats posed by lower layer protocols. The extended architecture is shown in figure 5-9.

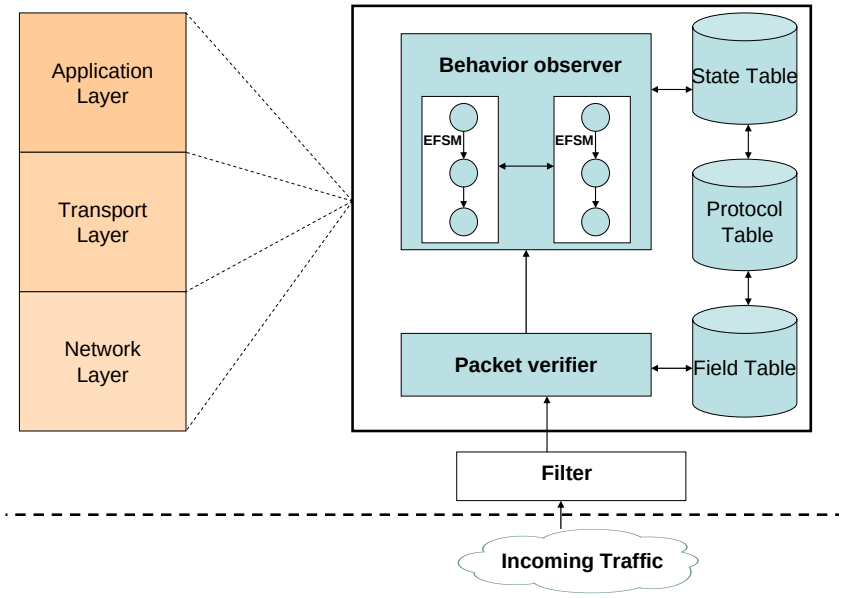


Figure 5-9. Extended System Architecture.

The system architecture that was discussed in the previous chapter is cloned for the transport and network layers. The cloning aims at addressing the threats and vulnerabilities of lower layers using the same principles utilized at the application layer. For instance, the *Filter* classifies the traffic based on the active protocols at the receiving layer and forwards packets to the right verifier. It classifies the packets coming from the network layer to the transport layer into TCP and UDP packets. Furthermore, various signature database tables in the extended architecture are used to store information on lower layer protocols in exactly the same way used with application layer protocols.

5.5 Advantages of Extended Architecture

In addition to all the advantages mentioned in Section 4.5, our architecture provides cross-layer detection in specification-based and signature-based modules. Protocols at a lower layer append information to the messages crossing to upper layers. The *behavior observer* performs cross-layer detection by using such information to make more informed decisions on potential attacks. The *field table* has the ability to store signatures that cross layer boundaries. Furthermore, the functions stored in the *Recommended Action* field of the *state table* have the ability to perform cross-layer detection.

Protocol layering has conceptual and structural advantages. It provides a structural way to discuss system components. Modularity makes it easier to update system components. However, some researchers and networking engineers are vehemently opposed to layering [52]. One potential drawback of layering is that one layer may duplicate lower-layer functionality. For example, many protocol stacks provide error recovery on both a link basis and an end-to-end basis. A second potential drawback is that functionality at one layer may need information (for example, a timestamp value) that is present only in another layer, which violates the goal of separation of layers [53].

5.6 Implemented Attacks and Detection Methodologies

We implement nine attacks to demonstrate the functionality of the intrusion detection system at network and transport layer. The implemented attacks have been widely used to test intrusion detection systems. These attacks target protocols such as TCP, UDP, ICMP, and IP and can be found in data sets such as 1999 DARPA Intrusion Detection Evaluation [54]. The rest of this section discusses the attacks and the detection methodology for each.

5.6.1 UDP Storm

This attack causes two machines to attack each other. The idea is that there are a number of ports that will respond with another packet if a packet is sent. For example, Echo (port 7) will echo a packet back, while Chargen (port 19) will generate a stream of characters. Consider a UDP packet with source port 7 and destination port 19. The packet generates some characters from the destination machine, headed for the echo port of the source machine. The source machine echoes these packets back, generating even more packets, and so on. Eventually, both machines are spending all their time sending packets back and forth which could result in a Denial of Service.

Table 5-6. UDP Storm Signature

Field Name	Field Content 1	Field Content 2
Protocol ID	25	25
Field ID	1	2
Field Name	Source Port	Destination Port
Description	The source port of the sender of the packet	The destination port of the receiver of the packet
Type	Number	Number
Pattern	7	19
Stand-alone	False	False
Next Protocol ID	25	Null
Next Field ID	2	Null
Impact	Such action causes Denial of Service (DoS) at the endpoint	Such action causes Denial of Service (DoS) at the endpoint

To detect such an attack we create a signature in the *field table* for the source and destination port fields. Whenever an incoming UDP packet has the number 7 in its Source port and the number 19 in its Destination port, a UDP storm flag is raised by the system. Table 5-6 shows the signature. The number 25 is used as the protocol ID for UDP.

5.6.2 Land Attack

In this attack, a TCP SYN packet is constructed with the source and destination IP addresses the same and both set to the target machine. A direct impact of this attack is causing the target to reply to itself which could result in the machine being locked up or unresponsive. Attackers launching this attack take advantage of the lack of built-in authentication in IP which makes it easy to spoof packet addresses. Land attack can be considered a DoS attack.

Table 5-7. Land Attack Signature

Field Name	Field Content
Protocol ID	26
State ID	5
State Name	SYN Packet Received
Description	The system state after receiving SYN packet
Threshold	Null
Time Unit	Null
Timer	Null
Recommended Action	SYN_Rec_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the endpoint

To detect this attack a signature is created in the *State table* representing TCP state upon receiving a SYN packet. The *recommended action* field includes a cross-protocol procedure that checks the IP addresses of the packet, and raises an alarm if they are equal. State variables in the

behavior observer can be used to store and compare IP addresses. Table 5-7 shows the signature. The number 26 is used as a protocol ID for TCP.

5.6.3 Blat Attack

This attack is an advanced version of the land attack. In addition to spoofing IP addresses, the attacker sends a TCP SYN packet to the target with an urgent pointer that points beyond the end of the packet. In some stack implementations this could lead to crashes and unresponsiveness.

Blat attack can be detected by our *packet verifier* which ensures that all fields of the packet header have appropriate values that are conformant to specifications. Clearly, a value that is greater than the end of the packet in the urgent pointer violates protocol specifications. Therefore, a priori knowledge of the signature of this attack is not required to perform detection.

5.6.4 Smurf Attack

Smurf attack is a way of generating a lot of traffic targeting a victim host. The attack floods the target system via spoofed broadcast echo request (ping) messages. The attacker constructs echo requests (ping) with the target as the source IP and broadcast them to an intermediary network to maximize the number of machines responding. The machines at the intermediary network all respond to the echo request with packets destined for the target machine. The target machine cannot process the large number of packets received and goes down under the load. Two main factors contribute to the success of Smurf attack, namely, the absence of cryptographic authentication at the network layer which allows attackers to spoof addresses, and poor network configuration by administrators which allows attackers to use networks as intermediaries.

A signature can be stored in the *State Table* representing the state of ICMP upon receiving an echo request. Two values are set to the *Threshold* and *Time Unit* respectively. Whenever the number of arriving echo requests exceeds the *Threshold* within the *Time unit*, a Smurf attack flag is raised. Table 5-8 shows the signature. The pseudo-code of the recommended action is shown in figure 5-10. The number 21 is used as a protocol ID for ICMP.

Table 5-8. Smurf Attack Signature

Field Name	Field Content
Protocol ID	21
State ID	2
State Name	Echo Request Received
Description	The system state after receiving an echo request
Threshold	100
Time Unit	500 MSEC
Timer	Null
Recommended Action	Smurf_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the endpoint


```
Procedure Smurf_Procedure ( )
    while (Time_Unit > 0)
     {
       if (echo_request is received)
          number_of_requests = number_of_requests + 1
          if (number_of_requests >= Threshold)
             Raise_Alarm (Smurf_attack)
```

```
    else
        Time_Unit = Time_Unit - 1
  else
    Time_Unit = Time_Unit - 1  }
```

Figure 5-10. The Pseudo-code for Smurf Attack Detection.

5.6.5 Neptune

Neptune, or "SYN flood", exploits the way a connection is established in TCP. In this attack, an attacker sends a SYN packet to a server using a spoofed IP address initiating a connection. The spoofed packet source address used by the attacker is unreachable. Therefore, the attacker never responds to the SYN/ACK packet from the server, which leads to a situation known as "half-open" TCP connection on the server. For each half-open TCP connection made to a server, the TCP software creates a record in a data structure to hold the information about the connection. If the connection is not completed within a certain amount of time, the connection times out and the record is freed. If attackers can initialize enough connections before the timeout occurs, the data structure can overflow, causing a segmentation fault and locking up the computer.

A signature can be stored in the *State Table* representing the state of TCP upon receiving a SYN packet. Two values are set to the *Threshold* and *Time Unit* respectively. Whenever the number of arriving SYN packets exceeds the *Threshold* within the *Time unit*, the procedure in the *Recommended Action* field can raise a Neptune attack flag.

5.6.6 Stealthy Probing Attacks

Another name for this attack is fingerprinting. Probing refers to methods for determining specific information about individual machines. Probing usually preludes more serious attacks and aims at determining the operating system or other unique identifiers of a system. Obtaining such knowledge allows attackers to launch specific attacks targeting specific operating systems. One basic technique in operating system fingerprinting is to send a packet with a strange flag

combination or when unexpected. For example, an unexpected TCP FIN packet will cause some systems to respond even though the correct action is to ignore the packet. Similarly, different operating systems will respond to such packets differently. Thus, by looking at the response to such packets, one obtains information that can help to identify the operating system of the machine of interest [55].

Such attacks can be detected by the *behavior observer* component, which contains the EFSM of the targeted protocol. As we have mentioned earlier in the chapter on the formal model, a well-designed protocol EFSM treats unexpected packets at a certain state as an implementation error or an attack. In both cases, the system should not respond in such a way that helps attackers to determine the unique identifiers of the machine. Packets with strange flag combinations can be detected by the *packet verifier* component which guarantees adherence to protocol specifications. Both components do not require priori knowledge of patterns of such attacks.

5.6.7 Ping of Death

The Ping of Death is an ICMP echo request (ping) packet with an illegally long payload. The packet is specially crafted by an attacker to be of a size greater than 65535 bytes, which is the maximum IP packet size. Such packets can make the receiving host lock up or reboot when the buffer into which the incoming packet is stored overflows.

Ping of Death can be detected by our *packet verifier* which ensures that all packets have the appropriate sizes. Clearly, an IP packet size that exceeds 65535 bytes violates protocol specifications. Therefore, detecting this attack does not entail priori knowledge of its signature.

5.6.8 Teardrop

This attack takes advantage of the fact that some vulnerable systems do not properly handle overlapping fragments. An attacker sends a series of IP packets carefully crafted to look like a normal packet that has been fragmented but such that the fragments overlap instead of being disjoint. Packet fragments are deliberately fabricated with overlapping offset fields, so that the IP software fails to reassemble the packet properly. Such an attack can result in

a stack corruption or failure of the IP module on the host.

Teardrop can be detected by our *packet verifier* which ensures that the second fragment of a packet specifies a fragment offset that resides within the data portion of the first fragment, and has a length such that the end of the data carried by the second fragment is short enough to fit within the length specified by the first fragment. Following protocol specification in the *packet verifier* component renders priori knowledge of this attack unnecessary.

5.6.9 TCP Hijacking

TCP hijacking takes advantage of the way a connection is established and a trust relationship between two machines A, and B. The attacker SYN floods machine B to make sure it does not respond to any packets from machine A. Then the attacker initiates a connection with machine A using a SYN packet spoofed to appear to be from trusted machine B. Machine A acknowledges the connection request from the presumed trusted machine accordingly. The attacker then sends an ACK packet with the correct sequence number to machine A using his/her real address, which results in a TCP connection established between machine A and the attacker. This sequence of events assumes that the attacker has previously determined the sequence number algorithm that A uses, and has determined the next sequence number that A will use. The issuance of CERT advisory CA-01-09 [56] shed some light on the issue of generating Initial Sequence Numbers (ISNs). The CERT advisory points out that even if the ISN gets incremented with pseudo-random numbers, it is still possible to guess the increment, given enough attempts. The advisory points out that, given a series of ISNs, they will tend to fall around an "average" value over time, and given enough attempts, the attack could still guess an ISN.

TCP hijacking can be foiled if the change in IP addresses is detected. If the TCP layer specifically looks back at the source address of the ACK packet and sees whether it is different from the source address of the SYN packet, the attack can be detected [67]. Therefore, we create a signature in the *state table* representing the system state upon receiving the ACK packet which aims at establishing a connection. The procedure in the *recommended action* field performs cross-layer detection by using address information provided by the network layer, and determines if a change in IP addresses occurs. Table 5-9 shows the signature, whereas Figure 5-

11 shows the pseudo-code of the recommended action.

Table 5-9. TCP Hijacking Attack Signature

Field Name	Field Content
Protocol ID	26
State ID	8
State Name	ACK Received
Description	The system state after receiving an ACK packet
Threshold	Null
Time Unit	Null
Timer	Null
Recommended Action	TCP-Hijacking_Procedure( )
Impact	Such action causes TCP Session Hijacking


```
Procedure TCP-Hijacking_Procedure ( )
    SYN_Address = The source IP address of the SYN packet
    ACK_Address = The source IP address of the ACK packet
    if SYN_Address == ACK_Address
        Enter ESTABLISHED state    // connection established
    else
```

Raise a TCP hijacking flag

Figure 5-11. The Pseudo-code for TCP Hijacking Attack Detection.

5.7 Attack Summary

Table 5-10 recapitulates the attacks mentioned in the previous section. For each attack, we provide a brief description and the violation type.

Table 5-10. Lower Layers Attack Summary

Attack Name	Brief Description	Violation Type
UDP Storm	A specially crafted UDP packet that causes two machines to attack each other.	Protocol-Syntax Violation
LAND Attack	A specially constructed TCP packet with the source and destination addresses set to the victim.	Protocol-Semantics Violation
Blat	An advanced version of LAND with illegal Urgent pointer.	Protocol-Syntax Violation
Smurf	Flooding a victim with ping requests.	Protocol-Semantics Violation
Stealthy Probing	Sending packets with strange flag combination or when unexpected.	Protocol-Semantics Violation
Ping of Death	Sending a ping request with illegally long payload.	Protocol-Syntax Violation
Neptune	Flooding a victim with "half-open" TCP connections.	Protocol-Semantics Violation
Teardrop	Sending IP packets with overlapped fragments.	Protocol-Syntax Violation
TCP Hijacking	Spoofing packet addresses to make a victim establish a TCP connection with the attacker instead of a trusted machine.	Protocol-Semantics Violation

5.8 Chapter Discussion

In this chapter we followed the same systematic approach of the previous chapter to show how our design tackled lower layer threats. We started by mentioning the related protocols at the network and transport layer with their threat model. A discussion on how specifications for the related protocols were developed followed. The chapter then showed how the architecture originally designed for application layer protocols could be extended to accommodate lower layer protocols using the same principles. Nine different attacks targeting lower layer protocols were detailed to show the detection capabilities of the extended architecture. The next chapter will discuss the implementation and simulation issues surrounding the design.

Chapter 6 Implementation and Simulation

6.1 Introduction

The previous chapter concluded our discussion on the system design. The last two chapters detailed the design of our architecture discussing the system's components and showing their role in the detection process. The attacks used to test the system were discussed along with the system detection capabilities.

This chapter discusses the issues involved in the implementation and simulation of the proposed architecture. We start by mentioning some of the IDS implementation approaches followed by the hurdles that face the implementation and testing of intrusion detection systems in general. Then we introduce the main simulation concepts adopted by OMNeT++ simulator and how they can be used to overcome the hurdles. We also show how the simulator features are harnessed to implement the proposed architecture and launch the various attacks. We detail the simulated network in terms of topology and parameter settings and demonstrate the various traffic generation scenarios used to test the system

6.2 Intrusion Detection Systems Implementation Approaches

The way in which an IDS is implemented influences its operation and effect on the monitored resource greatly. Whether it is host-based or network-based, an IDS should not hamper the performance of the host or the network in a way that renders the users unhappy or unsatisfied. Some approaches to implementing IDSs are based on artificial intelligence such as neural networks and expert systems. Others are computationally based such as special purpose languages and Bayesian. Moreover, some are based on biological concepts such as immune systems and genetics. The rest of this section shows some of the most prevalent approaches to implementing IDSs and some examples of systems that have applied these approaches.

6.2.1 Neural-based Intrusion Detection Systems

An artificial neural network is a type of technique that allows the identification and

classification of network activities based on incomplete and limited data sources. The system consists of a collection of processing elements that are highly interconnected. These highly interconnected processing elements transform a set of inputs to a set of desired outputs. The result of the transformation is determined by the characteristics of the elements and the weights associated with the interconnections among them. By modifying the connections between the nodes, the network is able to adapt to the desired outputs. In general, the neural network gains the experience initially by training the system to correctly identify pre-selected examples of the problem. The response of the neural network is reviewed and the configuration of the system is refined until the neural network's analysis of the training data reaches a satisfactory level. In addition to the initial training period, the neural network also gains experience over time as it conducts analyses on data related to the problem.

Neural network based IDSs are able to process data from a number of sources, predict events, and accept nonlinear signals as input. They are also fast and can perform supervised learning by mapping input signals to desired response. Furthermore, they can adapt weights to the environment, and be retrained easily. Moreover, they are fault tolerant and have a graceful degradation of performance if damaged. However, training of the neural network is required and enabling an industrial application requires complex hardware and software. If process conditions change from those used when training the neural network, data must once again be collected, analyzed, and used for retraining the system.

Ryan, Lin, and Mikkulainen [78] developed a keyword count based misuse detection system with neural networks. They presented the attack-specific keyword counts in network traffic to the neural network. Ghosh, Schwartzbard, and Shatz [79] employed neural networks to analyze program behavior profiles instead of user behavior profiles. In this method the normal system behavior of certain programs is identified and compared to the current system behavior.

6.2.2 Bayesian-Based Intrusion Detection Systems

Bayesian logic is a branch of logic that is applied to decision making and inferential statistics that deals with probability inference. Bayesian inference uses the knowledge of prior events to predict future events. With a Bayesian based IDS, an optimal statistical model to fit

experimental data can be established. It can also be used in data analysis when there is a need for extracting complex patterns from sizable amounts of information that contains a significant level of noise. It is considered very consistent and robust in the sense that small alterations in the model do not affect the performance of the system dramatically. It also allows combining expert knowledge with statistical data in a very practical way and the Bayesian networks can be constructed directly by using domain expert knowledge, without a time-consuming learning process. However, uncertainty may arise especially when the input parameters to an IDS are independent from one another and the number of parameters needed for defining the models is too high.

Scott [80] described a model-based approach to designing network intrusion detection systems using Bayesian methods. His approach considers general methods applicable to many different types of networks, using specific algorithms as examples. The central theme is that latent variable hierarchical models constructed using Bayesian methods lead to coherent systems that can handle the complex distributions involved with network traffic. Bayes' rule provides a means of combining competing intrusion detection methods such as anomaly detection and pattern recognition. Bayesian methods present evidence of intrusion as probabilities, which are easy for human fraud investigators to interpret. Hierarchical models allow transactions to communicate information about possible intrusions across time and accounts. These hierarchical models contain a transaction level model describing how well individual network transactions fit user and intruder profiles, an account level model parameterizing bursts associated with network intrusion, and a network-level model that adjusts account level model parameters when an intrusion on one or more accounts is suspected.

6.2.3 Special-Purpose Languages for Intrusion Detection

Such systems are designed based on program behavior, which reduces false positive and false negative rates. Language rules are usually defined in terms of system resources and not attacks, which requires few updates while the program is running, and detection is usually done in real time. However, since the rules are based on a specific program version and since different versions of the application may access different resources, every version will require modified set of rules. They can also be computationally expensive.

Sekar et al. [81] identify network attacks by collecting and aggregating information across many network packets, and act on the basis of this information. Their implementation consists of a compiler and a runtime system. The compiler is responsible for translating the intrusion specifications into C++ code and performs type-checking for packet data types and the compilation of pattern-matching. Their pattern-matching is based on compiling the patterns into a kind of automaton in a manner analogous to compiling regular expressions into finite-state automata. The runtime system provides support for capturing network packets either from a network interface or from a file.

Sekar et al. [6] employ state-machine specifications of network protocols, and augments these state machines with information about statistics that need to be maintained to detect anomalies. They were able to show that protocol specifications simplify manual feature selection process that often plays a major role in other anomaly detection approaches. The specification language developed for modeling state machines made it easy to apply their approach to other layers such as HTTP and ARP protocols.

6.2.4 Rule-Based Intrusion Detection Systems

Rule-based IDSs including expert system-based IDSs provide consistent answers for repetitive decisions, processes and tasks. They hold and maintain significant levels of information, reduce employee training costs, centralize the decision making process, and reduce time needed to solve problems. They also reduce the amount of human errors and can review transactions that human experts may overlook. However, since they are rule-based, they need frequent updates to remain current. Furthermore, the acquisition of these rules is a tedious and error-prone process. They also lack human common sense, human creativity, and ability to adapt to changing environments.

Ilgun, Kemmerer and Porras [24] presented an approach to detect intrusions in real time based on state transition analysis. Their approach has been discussed earlier in this thesis as a model that is represented as a series of state changes that lead from an initial secure state to a target compromised state. The state transition graphs identify the requirements and represent them as critical events that must occur for a successful completion of the attack. The state

transition analysis tool (STAT) is a rule-based expert system that is fed with the graphs. The authors developed USTAT which is a UNIX specific prototype of this expert system. In general, STAT extracts and compares the state transition information recorded within the target system audit trails to a rule-based representation of a known penetration that is specific to the system.

6.2.5 Immune-Based Intrusion Detection Systems

Immune based IDSs mimic the ability of the innate immune system to detect intrusions and stop them. They mimic the ability of the adaptive immune system to detect new types of intrusions that have not been seen before, and do not require a human expert to indicate that the intrusion is actually true. They have a faster response to previously seen intrusions and are distributed requiring no local coordination, which means that there is no single point of failure. Such system provides multi layer security as different mechanisms are combined to provide high overall security. The system is robust and goes through a dynamically changing coverage where cells die and others are reproduced to provide a random sample that can cover a larger space. However, the base models are simple since the actual human immune system is still under study and it lacks a theoretical foundation.

Pagnoni and Visconti [82] implemented a Native Artificial Immune System (NAIS), an artificial immune system for the protection of computer networks. Their system was able to distinguish between normal and abnormal processes. NAIS was also able to detect and protect web and FTP servers against new and unknown attacks and was able to deny access of foreign processes to the server.

6.3 Hurdles Facing the Implementation and Testing of Intrusion Detection Systems

There are several challenges that face the implementation and testing of intrusion detection systems. Some of these challenges are:

1. **Collection of attack scripts:** An important aspect of the testing of any IDS is testing its ability to detect a wide range of attacks. Collecting a wide range of attack scripts and codes is a difficult task. Although many of these scripts and codes are available on the Internet, it

entails a considerable time and effort to adapt them to the particular testing environment. Once the script of an attack is identified, it must be reviewed, automated, and smoothly integrated into the testing environment. Such tasks could be very challenging due to the fact that these scripts are developed by different people with different technical backgrounds to work in different environments.

2. **Use of different tools to launch and detect attacks:** Testing of intrusion detection systems usually involves two main phases. The first phase is to develop the intrusion detection algorithms and architecture using a specific tool. The second phase is to develop the attacks and scenarios necessary to test the system using a different tool. This separation of tools creates complications when it comes to integrating these tools to work together and into the specific testing environment.

3. **Generation of background traffic:** Most IDS testing approaches can be classified in one of four categories with regard to their generation of background traffic [10]. Each of these categories has its advantages and disadvantages. In the following we summarize the four approaches and the challenges they pose:

   - **Testing using no background traffic:** In such a scheme, an IDS is set up on a host or network on which there is no activity. Then, computer attacks are launched on this host or network to determine whether or not the IDS can detect the attacks. This approach is useful for verifying that an IDS has signatures for a set of attacks and that the IDS can properly label each attack. Furthermore, testing schemes using this approach are often much less costly to implement than the other approaches. However, such a scheme can neither say anything about false alarms, nor about the IDS ability to detect attacks at high levels of background activity [68].

   - **Testing using real background traffic:** This approach is very effective for determining the hit rate of an IDS given a particular level of background activity. Hit rate tests using this technique may be well received because the background activity is real and it contains all of the anomalies and subtleties of background activity. However, this approach could be ineffective in determining false alarm rates. It is

virtually impossible to guarantee the identification of all of the attacks that naturally occurred in the background activity which hinders false alarm rate testing. It is also difficult to publicly distribute the test since there are privacy concerns related to the use of real background activity [69].

- **Testing using sanitized background traffic:** In this approach, real background activity is prerecorded and then sanitized to remove any sensitive data. This sanitization is performed to overcome the political and privacy problems of using, analyzing, and distributing real background activity. Then, attack data are injected within the sanitized data stream. Attack injection can be accomplished either by replaying the sanitized data and running attacks concurrently or by separately creating attack data and then inserting these data into the sanitized data. The advantage of this approach is that the test data can be freely distributed and the test is repeatable. However, sanitization attempts may end up either removing much of the content of the background activity thus creating a very unrealistic environment, or removing information needed to detect attacks [70].

- **Testing by generating background traffic:** In this scheme, a test bed or simulated network is created with hosts and network infrastructure that can be successfully attacked. The simulated network includes victims of interest with background traffic generated by complex traffic generators that model the actual network traffic statistics. An advantage of this approach is that the data can be distributed freely since they do not contain any private or sensitive information. Another advantage is that we can guarantee that the background activity does not contain any unknown attacks since we created the background activity using the simulator. Therefore, false alarm rates using this technique are well-received. Lastly, IDS tests using simulated traffic are usually repeatable since one can either replay previously generated background activity or have the simulator regenerate the same background activity that was used in a previous test [71].

6.4 Implementation and Simulation Environment

We use OMNeT++ [57] simulator as the platform for our design. OMNeT++ is an object-oriented discrete event simulation tool that uses a modular structure. It may be used for traffic modeling of telecommunication networks, protocol modeling, and evaluating performance aspects of complex software systems among other things.

An OMNeT++ model consists of hierarchically nested modules which communicate through message passing. OMNeT++ models are often referred to as *networks*. The top level module is the system module. The system module contains sub-modules which can also contain sub-modules themselves. The structure of models is shown in figure 6-1. The depth of module nesting is not limited which allows the user to reflect the logical structure of the actual system in the model structure. Modules that contain sub-modules are termed *compound modules*, whereas *simple modules* lie at the lowest level of the module hierarchy. Simple modules contain the algorithms in the model.

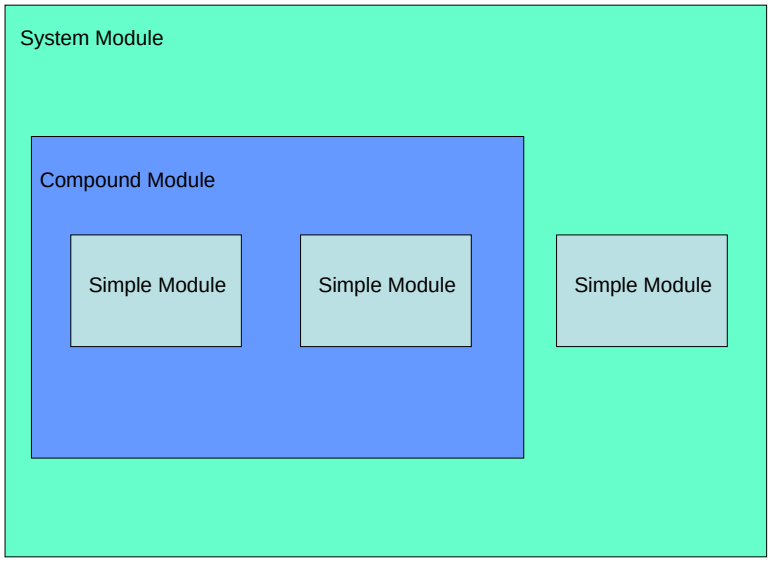


Figure 6-1. OMNeT++ Model Structure.

As previously mentioned, modules communicate by exchanging messages. In an actual simulation, messages can represent frames or packets in a computer network and can contain

arbitrarily complex data structures. Messages can arrive from another module or from the same module. When a message arrives from the same module it is called a *self-message*, and is usually used to implement timers. Simple modules can send messages either directly to their destination or along a predefined path through *gates* and *connections*.

Gates are classified into output and input gates. Output gates are the interfaces through which messages are sent out, whereas input gates are the interfaces through which messages arrive. Connections are the links used to connect gates. Connections can be assigned three parameters which facilitate the modeling of communication networks. These three parameters are *propagation delay* (which is the amount of time the arrival of the message is delayed by when it travels through the channel), *bit error rate* (which specifies the probability that a bit is incorrectly transmitted and allows for simple noisy channel modeling), and *data rate* (which is specified in bits/second, and used for calculating transmission time of a packet).

OMNeT++ uses two programming languages, namely NED (Network Description) Language and C++. NED language is used to describe the model structure and the topology of a network and its modules. A network description may consist of a number of component descriptions that can be reused in another network description, which facilitates the modular description of a network. On the other hand, C++ is used for the actual implementation of the simple modules such as messages and queues. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library. The simulation programmer can freely use object-oriented concepts (inheritance, polymorphism, etc) and design patterns to extend the functionality of the simulator. Therefore, the design of a topology and the implementation of the modules that exist in the topology are separated. In addition, OMNeT++ provides a high degree of parameterization through the use of NED and initialization files and a solid support for Finite State Machines in the form of ready-to-use classes and functions.

6.4.1 TCP/IP Modeling in OMNeT++

OMNeT++ support for TCP/IP protocols such as IP, ICMP, UDP, and TCP started with the Internet Protocol Suite (IPSuite), and has culminated in the more recent INET Framework. Both IPSuite and INET framework have been faithful to the TCP/IP protocol suite with its

layered approach. The modularity that distinguishes OMNeT++ is reflected on the modeling of TCP/IP protocols, where all components of protocols are divided into a number of different modules, and each module can have several parameters.

6.4.2 VoIP Protocols Modeling in OMNeT++

Several research groups at the University of Karlsruhe developed MMSim [58] which is a model to simulate multimedia protocols using OMNeT++. The MMSim model provides support for SIP, RTP, and Real-Time Streaming Protocol (RTSP).

The actual details of each protocol are implemented in C++ programming language, where every major operation of the protocol is implemented as *a member function* in the *class* files that represent the protocol. All implementations of protocols follow the specifications detailed in relevant Request for Comment (RFC) documents.

6.4.3 How OMNeT++ Can Be Used to Overcome the Implementation and Testing Hurdles

C++ programming language can be exploited efficiently to implement attacks, which alleviates the burden of integrating attack scripts and codes written in different programming languages and styles into the testing environment. The full flexibility and power of the programming language, supported by the OMNeT++ class library can be used to implement protocol-related attacks. The same powerful features can also be used to implement both attacks and detection algorithms without the need to switch tools or products. Furthermore, OMNeT++ can generate background traffic that is guaranteed to be free of unwanted attacks, which gives credibility to hit rate and false alarm tests, and the test scenario is usually repeatable.

6.5 Attack Implementation and Detection using the Simulator

As mentioned earlier in this thesis, attacks that target networked environments take advantage of vulnerabilities in networking protocols. Such attacks can be classified into (1) message flow attacks which are used by attackers to exploit vulnerabilities in the flow of messages used by protocols, (2) parser attacks which aim at hampering proper parsing by

constructing invalid messages, and (3) flooding attacks which are used by attackers to deny legitimate users access to network resources.

In light of the above, we classify the implemented attacks based on the targeted protocols for implementation reasons. As mentioned earlier, protocols in OMNeT++ are implemented in an object-oriented manner as *classes* using C++ programming language. The main operations of each protocol are implemented as *member functions* in the class files. Therefore, we follow the same concept and implement protocol-related attacks as member functions in the class files that represent the protocol.

Detection algorithms are implemented in some of the member functions that perform tasks related to the protocol operation. For instance, *handleMessage()*, which is a member function responsible for handling messages coming to the protocol, could be a good choice for the implementation of the detection algorithms responsible for checking the validity of the incoming messages and packets.

All attacks are given identification numbers, which are stored in a system text file. The code that launches attacks (calls the member function that represents the attack) chooses a number randomly from the range of the identification numbers and launches the associated attack accordingly. Furthermore, the attack launching code itself is activated in the endpoints based on a randomly selected number that should exceed a certain threshold. This technique guarantees that the majority of the simulated network background traffic remains benign. Such techniques are made possible by the random number generation features provided by OMNeT++. OMNeT++ enjoys the support of several Random Number Generators that can be configured in the initialization files.

Events in the simulator environment can be controlled to occur at a specific time. Message/event related functions can be used to send messages to other modules, schedule an event, or delete a scheduled event. This feature facilitates the detection and launching of attacks that require accurate timing such as flooding attacks, and message flow attacks.

Message manipulation functions provided by protocol modules allow for creating malformed packets and launching parser attacks easily. The simulator library contains various

functions to set the value of different fields and the length of the entire message. Similar functions can be used to get the value of message fields to perform detection.

MMSim module provides interaction between SIP and RTP which makes cross-protocol detection at the application layer possible. RTP attributes can be captured by SIP through a specialized function that can be called from SIP module. In addition, messages in OMNeT++ have a field called *control info* that carries auxiliary information to facilitate communication between protocol layers which makes cross-layer detection possible.

On the other hand, C++ streams which are associated with files are used to emulate our signature database. Functions that perform the recommended actions are given names that reflect the associated protocol and state. This way, functions are linked to records in the *state table*. When an administrator defines a function to perform a certain recommended action, he/she should store the function name in a system text file. When the detection process reaches a certain state of a protocol during a session, the IDS searches the system text file for the function name using a combination of the protocol and state ID. If the function name is found, it means that there is a function defined for that state of the protocol. Therefore, the IDS calls the function accordingly to perform the recommended action. More than one function can be defined for a single state of a protocol by adding different suffixes to the end of the function name.

System files can also be used to aid the IDS in terms of performing stateful detection. Values of header fields of incoming packets are stored in temporary system files associated with sessions. Such files are named in a way that reflects the ID of the affiliated session, and the files contain records for the packets belonging to the active protocols of the session. This feature allows modules such as the *Field Table* to store signatures that span across multiple packets that could belong to different protocols. Since the relevant information is kept in these system files, the IDS can perform detection for the entire session or connection. Such system files get deleted automatically once the session is terminated.

6.6 Network Topology and Configuration

Figure 6-2 shows the simulated network topology. Our network comprises two domains each with a Proxy and Registrar Server. Each domain also contains a set of User Agents

(endpoints) which are connected to the servers by a 10Base-T Ethernet. Proxy servers and endpoints in each domain also run TCP-based server and client applications. TCP-based server and client applications simulate web users issuing HTTP requests. This setting of client and server applications is meant to emulate the operation and functionality of converged services. A converged service is an application that spans communications over multiple protocols to provide higher level function and better user experience.

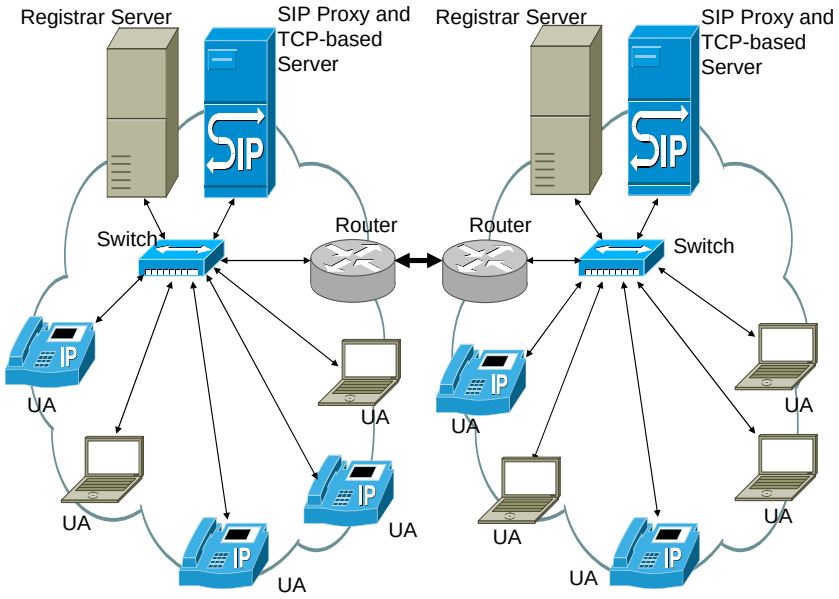


Figure 6-2. Simulated Network Topology.

For the VoIP application, we use the Audio/Video profile with minimal control (RTP/AVP), with UDP as the underlying protocol. An application profile describes how audio and video data may be carried within RTP. Our payload type is static with the identification number 10, and has the encoding L16. The payload type defines how a particular payload is carried in RTP. The clock rate, which is used to generate RTP timestamps, is 44100 Hz and the number of transmission channels is 2. Endpoints in a domain make calls to other endpoints in the other domain randomly and without predefined durations. The abovementioned parameter setting is recommended as one of the standard operating parameter settings for audio encoding and payload type [74].

On the other hand, the TCP-based server and client applications use TCP Reno

algorithm. We use a Maximum Segment Size (MSS) of 536 bytes per segment, and an advertised window of 7504 (MSS * 14) bytes. Similar to voice-based applications, TCP-based applications establish sessions with servers randomly, and sessions last for random durations. TCP Reno algorithm forms the base of the most modern TCP implementations and introduces major improvements over previous algorithms. Our choice of the Maximum Segment Size conforms to the default setting for TCP. TCP is designed to restrict the size of the segments it sends to a certain maximum limit to cut down on the likelihood that segments will need to be fragmented for transmission at the IP level. The Minimum Maximum Transmission Unit (MTU) for IP networks is 576 bytes. All networks are required to be able to handle an IP datagram of this size without fragmenting. From this number, we subtract 20 bytes for the TCP header and 20 for the IP header, leaving 536 bytes which is the standard MSS for TCP [75].

Our IDS is installed on all endpoints and servers in both domains. The Internet connection between the two domains is assumed to have a delay of 40 ms and a packet loss of 0.2%. Such values for delay and packet loss are acceptable by most network Service Level Agreements (SLAs) for backbone providers [76].

6.7 Generated Traffic and Performance Tests

The thesis aims include proving that the intrusion detection system can operate under different network conditions, adds little overhead to the network, and is robust. To do this, performance tests are conducted on the simulated network. The different conditions that are tested include a low-load scenario and a high-load scenario. The experiment runs under these two types of load with five different runs for each load. Each run lasts for 120 minutes which gives as an overall simulation time of 20 hours. The results which will be shown in the next chapter are averaged across the different runs and taken with and without the operation of the IDS to observe the difference.

For the VoIP application, the tests that are performed under low-load conditions use background traffic sent at a frequency of 1 call per 15 seconds. The TCP-based applications use a background traffic sent at a frequency of 645 packets per minute under low-load conditions. Figures 6-3 and 6-4 show the calls captured at the proxy server and the packets captured at the

TCP-based server in one of the domains respectively.

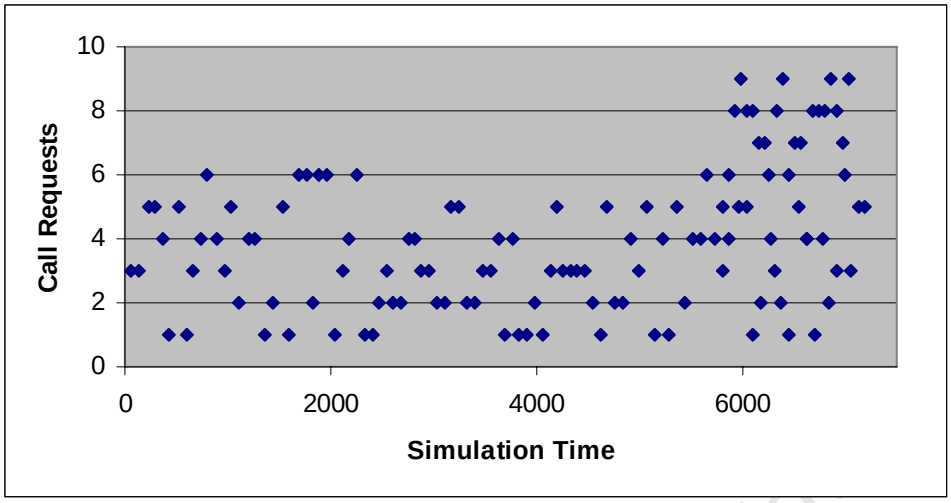


Figure 6-3. Call Requests at a Proxy Server under Low-Load.

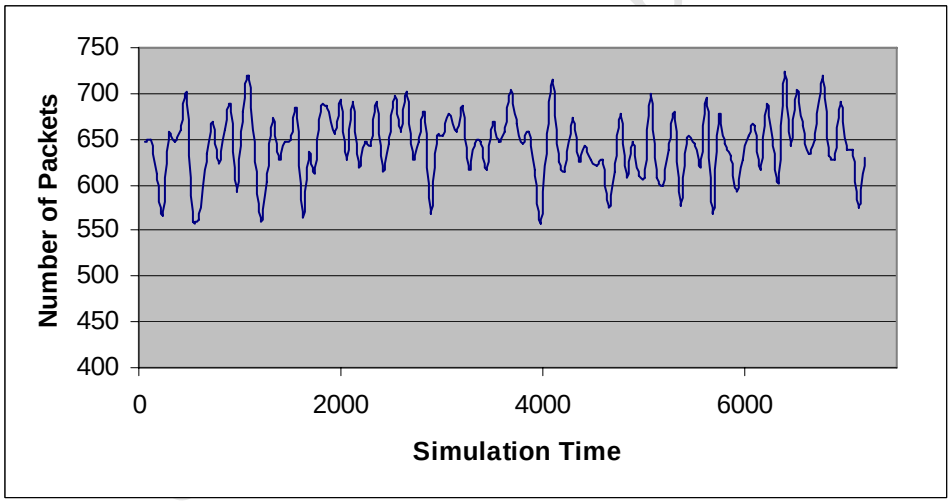


Figure 6-4. Amount of TCP-Based Traffic under Low-Load.

For the VoIP application, the tests that are performed under high-load conditions use background traffic sent at a frequency of 1.5 calls per 1 second. The TCP-based applications use a background traffic sent at a frequency of 3200 packets per minute under high-load conditions. Figures 6-5 and 6-6 show the calls captured at the proxy server and the packets captured at the TCP-based server in one of the domains respectively.

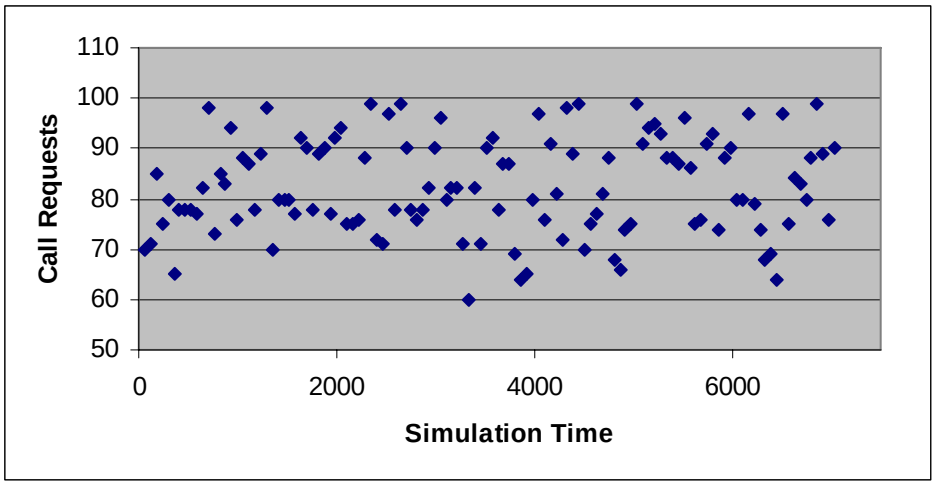


Figure 6-5. Call Requests at a Proxy Server under High-Load.

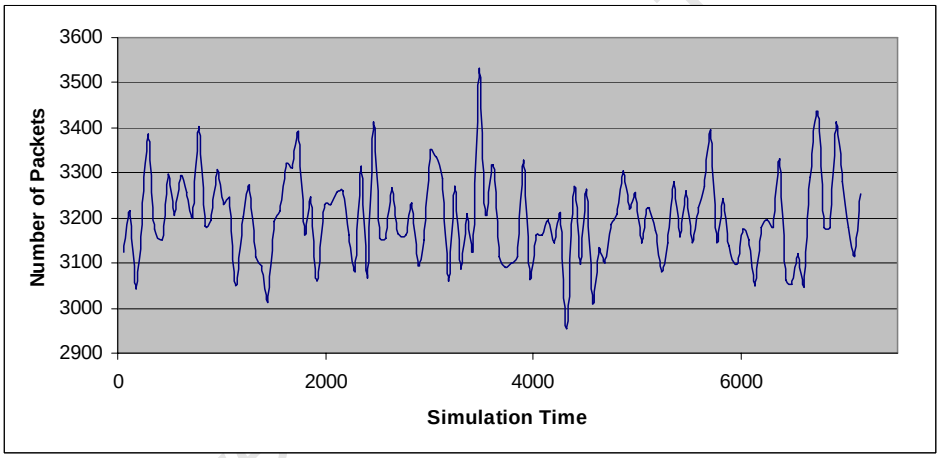


Figure 6-6. Amount of TCP-Based Traffic under High-Load.

In his critique of the 1998 and 1999 DARPA off-line intrusion detection as performed by Lincoln laboratories, McHugh [90] criticizes the test traffic - including the background traffic and attack data – for (1) the lack of discussion on the data rate and its variation with time and (2) the presence of attacks in the training data. As mentioned clearly in this chapter, we address these two issues by providing the traffic rate under low- and high-load scenarios, and generating background traffic that is free of unwanted attacks.

6.8 Chapter Discussion

In this chapter, we introduced the reader to the simulation environment and how it was harnessed to implement the system. The advantages of our testing approach were highlighted. We also presented our simulated network with its parameters and settings. The two scenarios used to test the performance of the system were also shown. The next chapter will demonstrate and analyze the detection and performance evaluation results.

Chapter 7 Results and Analysis

7.1 Introduction

The previous chapter shed some light on the simulation environment used to implement the proposed intrusion detection system. The chapter started by showing the different intrusion detection system implementation approaches alongside the hurdles that faced the implementation and testing of IDSs in real environments. It then showed the simulator features and how they were used to overcome the implementation and testing hurdles and implement both attacks and detection algorithms. The chapter then concluded by presenting the simulated network topology with its parameters, and the scenarios used to test the performance.

This chapter will discuss the results produced by our system and analyze them. Our discussion on simulation results will revolve around two axes, namely, detection accuracy and performance evaluation. We will show how the IDS detects all the attacks presented, and provide an analysis of the causes of false alarms. We will also demonstrate quantitatively how the operation of the IDS has a minor impact on the performance of hosts and servers.

7.2 Detection Accuracy

7.2.1 IDS Coverage

Assessing the coverage of intrusion detection systems is a challenging task with many ramifications. The coverage of any intrusion detection system depends on the attacks that the IDS can detect under ideal conditions. The number of dimensions that form each attack makes the assessment difficult. Each attack has a particular goal and works against particular software. Attacks may also target a certain version of a protocol or a particular mode of operation. Different sites may consider some attacks more important than others, which affects the assessment greatly. For instance, E-commerce sites may be very interested in detecting distributed denial of service attacks, whereas military sites may pay a great deal of attention to surveillance attacks.

Table 7-1. Implemented Attacks with targeted Protocols and Effect

Attack Name	Protocols Involved	Effect
BYE Attack	SIP, RTP	Session Tear down
Re-INVITE	SIP,RTP	Session Hijacking
CANCEL	SIP	Denial of Service
Malformed Messages	All Protocols	Denial of Service
REGISTER Flooding	SIP	Denial of Service
Voice Injection	RTP	Playing Artificial Stream
UDP Storm	UDP	Denial of Service
LAND	IP, TCP	Denial of Service
Blat	IP, TCP	Denial of Service
Smurf	ICMP	Denial of Service
Stealthy Probing	TCP	Identifying OS
Ping of Death	ICMP	Denial of Service
Neptune	TCP	Denial of Service
Teardrop	IP	Denial of Service
TCP Hijacking	IP, TCP	Session Hijacking

As stated in the first chapter of this thesis, we are neither interested in flaws in particular implementations of the protocols nor in vendor-specific protocols or protocol features. Instead, we are interested in generic protocol problems exploited by attackers to cause damage. Therefore, we list in table 7-1 all the attacks implemented to test the system along with the

protocols they target and the effect they have on the attacked system.

There are several dimensions that can be taken from table 7-1. It is important to realize the diversity of the attacks used to test the system in terms of the protocols involved and the effect they have. The protocols involved range from application layer protocols such as SIP and RTP to network layer ones such as IP and ICMP. The wide range of supported protocols reflects the wide coverage of the intrusion detection system. Some attacks are cross-protocol which forms another dimension. As shown in the table, the effect of the attacks varies widely. The attacks violate many of the security services that should be provided by systems such as availability, confidentiality, authentication, and data integrity. Therefore, we can safely say that the abovementioned dimensions constitute the coverage of the IDS.

7.2.2 IDS Hit Rate

Generally, measuring the hit rate of an IDS is largely dependant on the set of attacks used during the test. Furthermore, intrusion detection systems can be configured in a way that could either favor detection of new attacks or minimizing false alarms. Table 7-2 shows all the attacks used to test the system during the experiment. With each attack, the table shows the number of instances launched during the experiment, the number of instances detected, and the module of the IDS responsible for detecting the attack. It is important to state that we have used the same configuration during testing for hit rates and false alarms.

Table 7-2 shows how various components in the proposed architecture contributed to detecting *all* attacks launched during the experiment. Some of the attacks such as CANCEL, malformed packets, stealthy probing, ping of death, and teardrop were unknown to the IDS prior to the experiment. In other words, we did not encode any special signatures, and hence all detections for such attacks were based on normal behavior specifications. Our IDS has also identified all the detected attacks successfully by labeling them with names.

It is important to realize that our IDS ability to detect attacks is not confined to the attacks used during the experiment. The implemented attacks are meant to represent a wide range of security service violations and attack categories. The proposed intrusion detection components are capable of detecting other attacks that violate the syntax or semantics of protocols.

Table 7-2. Hit Rate Results

Attack Name	Instances Launched	Instances Detected	Detecting Module
BYE Attack	6	6	State Table
Re-INVITE	5	5	State Table
CANCEL	3	3	Behavior Observer
Malformed Messages	6	6	Packet Verifier
REGISTER Flooding	3	3	State Table
Voice Injection	4	4	State Table
UDP Storm	2	2	Field Table
LAND	7	7	State Table
Blat	6	6	State Table
Smurf	4	4	State Table
Stealthy Probing	8	8	Behavior Observer and Packet Verifier
Ping of Death	3	3	Packet Verifier
Neptune	3	3	State Table
Teardrop	5	5	Packet Verifier
TCP Hijacking	6	6	State Table

7.2.3 Probability of False Alarms

Some of the difficulties surrounding the measurement of the false alarm rate can be attributed to the fact that intrusion detection systems may have different false alarm rates depending on the conditions of the underlying network environment. Furthermore, for intrusion detection systems that can be configured and tuned flexibly to control the rate of false alarms, it is always difficult to determine the ultimate configuration that should be used for a certain false positive test. In this part of the thesis, we mention some of the potential causes of false alarms in our design and show how the system overcomes them.

1. Whenever the detection of an attack relies upon a certain order of packet arrivals within a predefined amount of time, there is a possibility of a false alarm. This phenomenon is due to network conditions and non-guaranteed delivery within a specific time window. During the experiment, we simulated false BYE and Re-INVITE attacks by delaying RTP packets in both after receiving a BYE message, and a Re-INVITE request respectively. Our IDS raised false flags on both occasions. We believe abnormal network conditions are to blame for these false positives, and not our detection mechanism. Delay as a result of propagation, handling, or queuing is a major issue in packet-based VoIP environments. However, our parameterized *State Table* can be used to overcome such situations. The choice of the values for timers is left to the discretion of the system administrator. Hence, system administrators can set these values in a way that reflects the conditions of the underlying network to avoid unwanted false alarms. It is relevant to mention that the same abnormal network conditions can result in false negatives. For example, in BYE attack, if the system does not receive any RTP packets within the life time of the timer due to network delays, it will miss the attack producing a false negative

2. A major source of false alarms in intrusion detection is threshold detection. The goal of threshold detection is to record each occurrence of a specific event and detect when the number of occurrences of that event surpasses a reasonable amount that one might expect to occur within a specified time period. An unnaturally high number of occurrences within a short period of time may indicate an attack. Once the threshold number of occurrences is surpassed, the threshold detector can notify the administrator. The relatively high rate of

false alarms stems from the difficulty to identify the threshold number and the time frame for the specific event. These two parameters are highly dependent upon the security relevance of the event and the historical number of occurrences. Therefore, the choice of these values is often left to administrators [9], which is the policy adopted by our system to reduce false alarms originating from threshold detection. The assumption here is that in a well-designed system, any alarm contains information. For example, one may see a few packets that look like a probe for vulnerable systems. The administrator may want to know about this, even though it is not yet a problem and even though in reality it may not be a prelude to an attack at all. In this scheme the system reports only alarms for events that are meaningful to administrators, and hence reduces the amount of false alarms significantly. The origins of this school of thought are discussed in detail in [59].

3. We believe strict sticking to protocol specifications and standards might lead to some false alarms. In his paper about Bro system [60], Vern Paxon mentioned what he called "The Problem of Crud". Based on monitoring a large volume of network traffic, he realized that legitimate traffic exhibits abnormal behavior. He stated that the diversity of legitimate network traffic, including the implementation errors sometimes reflected within it, leads to a very real problem for intrusion detection, namely, discerning in some circumstances between a true attack versus an innocuous implementation error. He concluded by mentioning the difficulty of relying on "clearly" broken protocol behavior as definitely indicating an attack because it very well may simply reflect the operation of an incorrect implementation of that protocol. As mentioned earlier in the section on specification development, our system uses more abstract specifications with the state machines concentrating on the essential details of each protocol and omitting the details that differ between different implementations of a protocol. We believe such an approach provides a satisfactory solution to the problem.

4. The detection of some attacks depends on checking the source IP address of the incoming packets. We have already mentioned the example of TCP hijacking in a previous chapter. Another good example could be *fake instant messaging*. By faking the header of an instant message appropriately, the attacker can forge a message to *A* and mislead it into believing the message is from *B*. A direct way of detecting such an attack is to look at the IP addresses

of the incoming messages and raise an alarm once the IP address of one of the messages changes. This direct methodology however, does not take the issues of mobility and user motion into account, which leads to false alarms. SIP fully supports the concept of user mobility. A user can make himself/herself available for communication by explicitly binding his/her Address Of Record (AOR) with a certain host address. This feature allows for user mobility since the user can register from any device that supports SIP including personal computers, wireless devices and cellular phones. We believe the flexibility provided by the threshold values and procedures in the recommended action fields in the *state table* can allow for mobility-aware detection. Our detection can take the rate of user mobility into account and allows for changes in the IP address according to a tuned rate of user motion.

7.3 Performance Evaluation

It is vital that any security measure to be implemented in a VoIP network does not impede the performance of the network. Quality of service (QoS) is very important to the operation of VoIP networks. The implementation of various security measures in a VoIP network can introduce some complications that can degrade QoS. These complications range from delaying call setups to delaying delivery of data packets.

In this section we evaluate the performance of the proposed architecture, and show its effect on the hosts. Our discussion will focus on end-to-end delay, call setup delay, processing delay, packet loss, and memory usage at endpoints and servers. Since our IDS inspects individual packets and monitors the progress of sessions, it is expected to affect these performance parameters.

7.3.1 End-to-end Delay

End-to-end delay in VoIP refers to the time it takes for a voice transmission to go from its source to its destination. The ITU-T G.114 standard describes that a 150 milliseconds one-way delay is acceptable for high voice quality [61]. Every element along the voice path adds to this delay. This includes switches, routers, and public Internet connections. Some delays such as those added by the coder, packetization, and serialization are fixed, which places a genuine

constraint on the amount of security that can be added. A typical delay budget such as the one provided by Goode [62], puts the cost of fixed delays at 121 milliseconds, leaving only 29 milliseconds for various security implementations to compete for.

Figure 7-1 shows the end-to-end delay experienced by an endpoint in the network with and without our IDS installed. The figure shows measured end-to-end delay for individual RTP voice packets (y-axis) versus simulation time (x-axis). The figure shows that during simulation, end-to-end delay, without the operation of the IDS, varies from 115 milliseconds to 134 milliseconds. It is clear from the figure that the average end-to-end delay without the IDS is around 120.0 milliseconds. On the other hand, end-to-end delay with the operation of the IDS varies from 117 milliseconds to 136 milliseconds. From the figure, the average end-to-end delay with the IDS is around 122.7 milliseconds. Therefore, our IDS adds about 2.7 milliseconds on average to the voice transmission delay.

Figure 7-2 shows a similar measurement of end-to-end delay versus time with emphasis on the operation of the signature database. From the figure, end-to-end delay is averaged around 122.672 milliseconds when the signature database is disabled, whereas it is averaged around 122.727 milliseconds when the signature database is enabled. Therefore, the operation of the signature database adds about 0.055 milliseconds on average to the end-to-end delay.

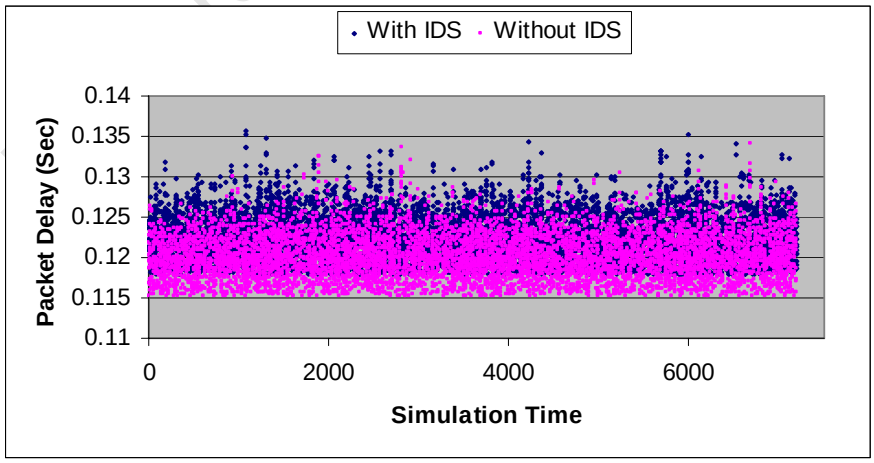


Figure 7-1. End-to-end Delay.

The end-to-end delay increases by a maximum of 0.6% under high-load. As shown in the

two figures, the overall delay remains considerably less than the upper bound of 150 milliseconds despite the 40 milliseconds delay on the Internet connection. The delay variation (jitter) remains around 2 milliseconds with a slight addition of $1.6 * 10^{-5}$ seconds by our IDS. Therefore, our hybrid IDS has a trifling impact on end-to-end delay.

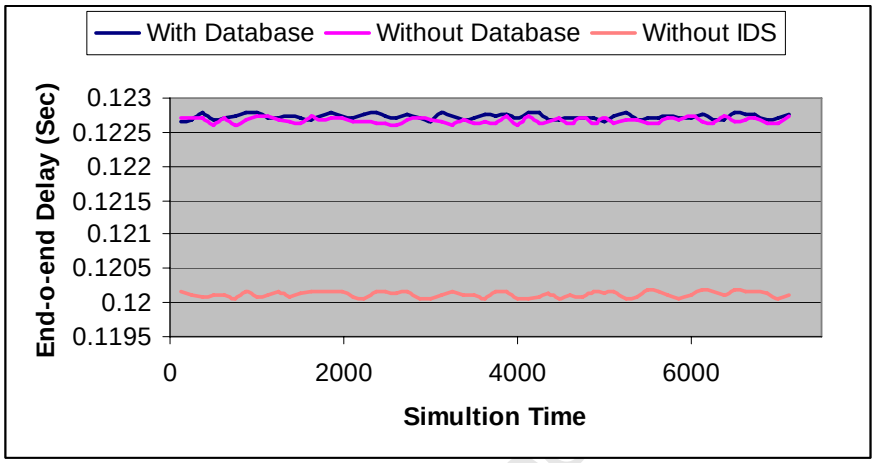


Figure 7-2. End-to-end Delay with the Effect of the Signature Database.

7.3.2 Call Setup Delay

Call setup delay in VoIP environments is the period that starts when a caller dials the last digit of the called number and ends when the caller receives the last bit of the response. VoIP systems are expected to give a performance comparable of that of PSTNs. Users may be annoyed with a setup process that requires more than a few seconds.

Figure 7-3 shows the call setup delay introduced by our IDS at a certain endpoint during the simulation. The figure shows measured call setup delay for 60 calls initiated by the endpoint (y-axis) versus simulation time (x-axis) with and without our IDS installed. The figure shows that during simulation, call setup delay, without the operation of the IDS, varies from 242 milliseconds to 253 milliseconds. From the figure, the average call setup delay without the IDS is around 248.5 milliseconds. On the other hand, the call setup delay with the operation of the IDS varies from 310 milliseconds to 320 milliseconds. From the figure, the average end-to-end delay with the IDS is around 315.6 milliseconds. Therefore, the hybrid IDS adds about 67.1 milliseconds to the call setup process.

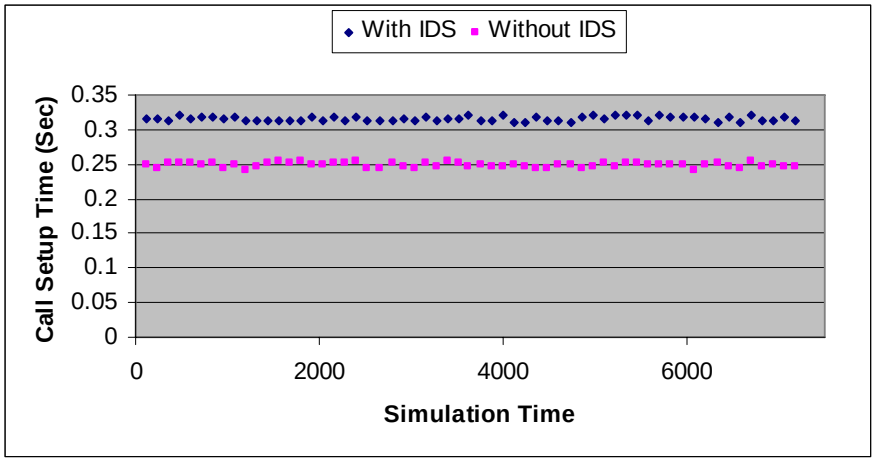


Figure 7-3. Call Setup Delay.

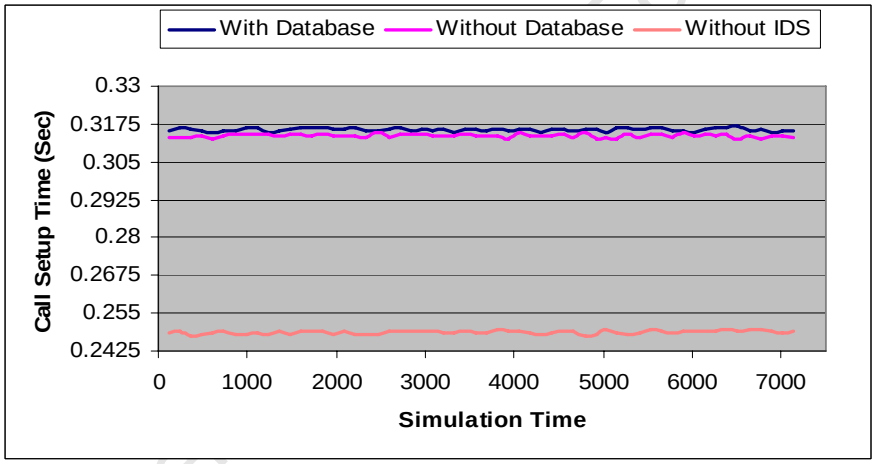


Figure 7-4. Call Setup Delay with the Effect of the Signature Database.

Figure 7-4 shows a similar measurement of call setup delay versus time with emphasis on the operation of the signature database. From the figure, call setup delay is averaged around 313.597 milliseconds when the signature database is disabled, whereas it is averaged around 315.642 milliseconds when the signature database is enabled. Therefore, the operation of the signature database adds about 2.04 milliseconds on average to the call setup delay.

The call setup delay increases by 1.3% on average under high-load. Such an increase in the call setup time is tolerable by VoIP users. Furthermore, the overall call setup delay remains within the limit of one or two seconds mentioned in [63].

7.3.3 Processing Delay

End-to-end and call setup delays can be affected by another important factor which is processing delay. Processing delay is the time required by an endpoint or a server to process a message.

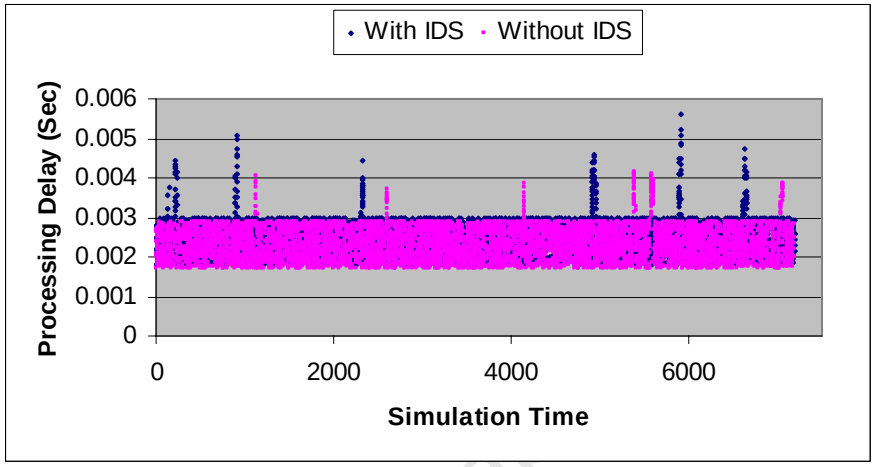


Figure 7-5. Processing Delay.

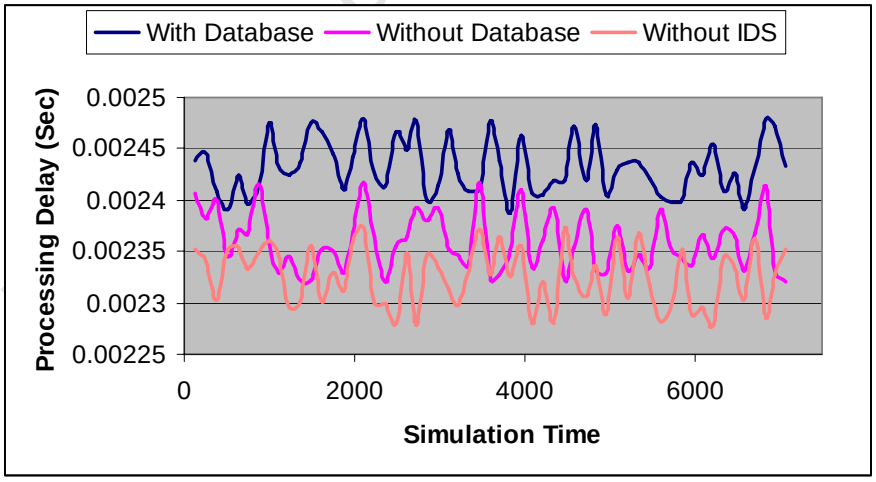


Figure 7-6. Processing Delay with the Effect of the Signature Database.

We show in figure 7-5 the processing delay at an endpoint with and without our IDS. The figure shows measured processing delay for individual packets (y-axis) versus simulation time (x-axis). The figure shows that during simulation, processing delay, with the operation of the

IDS, varies from 1800 microseconds to 5588 microseconds. From the figure, the average call setup delay with the IDS is around 2429 microseconds. On the other hand, the processing delay without the operation of the IDS varies from 1700 microseconds to 4147 microseconds. From the figure, the average processing delay without the IDS is around 2325 microseconds. Therefore, our IDS adds about 4.4% increase to the processing delay on average.

Figure 7-6 shows a similar measurement of processing delay versus time with emphasis on the operation of the signature database. From the figure, processing delay is averaged around 2359 milliseconds when the signature database is disabled, whereas it is averaged around 2429 milliseconds when the signature database is enabled. Therefore, the operation of the signature database adds about 70 microseconds on average to the processing delay.

The processing delay increases by 0.8% under high-load. An important observation that can be taken from the figures is the processing spikes indicating extra processing time needed for certain packets. When a spike exceeds the incoming packet rate, there is the danger of packet drops.

7.3.4 Packet Loss

High sending rates can lead to packet drops especially in UDP-based transmissions. Unlike TCP, UDP lacks built-in transmission control mechanism that makes senders adapt themselves to the buffer capacity of receivers. The absence of such a mechanism in UDP could lead to a situation where the receiver is unable to keep up with the high sending rate of the sender, which results in some packet drops from the receiver's buffer. Another contributing factor to packet loss is processing spikes. Processing spikes mean that the CPU is spending too much time on some packets which has the consequence of missing subsequent ones.

It is important to mention that packet loss does not only happen due to transmission errors. Based on practical experiences, it has been found that buffer overflow is a major cause of packet loss. Such buffers may be in network hardware (e.g. switches and routers) or in operating systems. Consequently, packet loss does not only happen in networks. Buffers of hosts in a networked environment can also experience packet losses due to congestion [77].

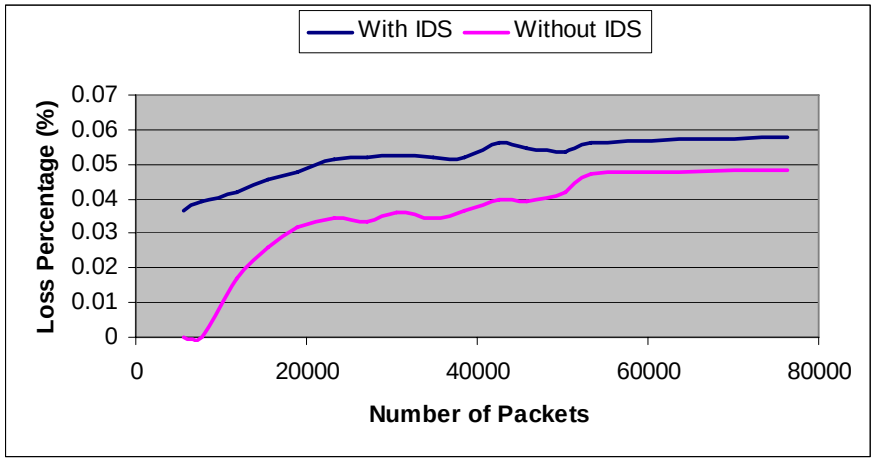


Figure 7-7. Packet Loss.

Ideally, there should be no packet loss for VoIP. Losses of 3 and 4 percent could place the quality in VoIP networks encoded by certain codecs at a level below the quality of service level of PSTNs. Figure 7-7 shows the packet loss rate at servers and endpoints buffers with and without our IDS. The figure shows the percentage of packet loss (y-axis) versus the various amounts of traffic (x-axis). Without the operation of the IDS, the system starts without packet drops till the number of packets exceeds 10000. Then, the loss percentage grows linearly to 0.02 as the number of received packets exceeds 15000. Thereafter, the packet loss percentage stabilizes around 0.035% till the number of packets reaches 45000. The loss percentage climbs to and stabilizes around 0.05% for packet rates greater than 75000.

On the other hand, when the IDS operates on the host, the loss percentage starts at about 0.035% and increases linearly to around 0.05% as the number of received packets reaches 22000. Thereafter, the loss percentage stabilizes around 0.055% as the number of received packets surpasses 75000.

From figure 7-7 and the abovementioned discussion, we conclude that the packet loss rate with our IDS is only 0.02 % higher than the rate without it on average. The overall packet loss remains at 0.05% on average, which is considerably less than the 1 percent level specified by many codecs as the upper limit. The figure does not show a separate curve for the operation of the signature database because its effect could not be distinguished from the effect of the hybrid IDS during the experiment.

7.3.5 Memory Usage

Measuring the memory consumption of an intrusion detection system is vital in gauging its effect on the host. Some IDSs could exhaust all the available memory after a relatively short runtime, leaving the host with the possibility of crashing. We find that two factors dominate overall memory consumption, namely, the total amount of state kept by the system and the traffic volume. In order to understand the memory usage of a host intrusion detection system, we need to track where it stores the state. Therefore, we have identified the IDS's main data structures and added methods to track their size during simulation.

Figure 7-8 shows the memory usage at a server under high-load of traffic. The figure shows the amount of memory consumed by the IDS (y-axis) versus simulation time (x-axis). The figure exhibits the gradual increase in memory consumption as call and session establishment requests arrive.

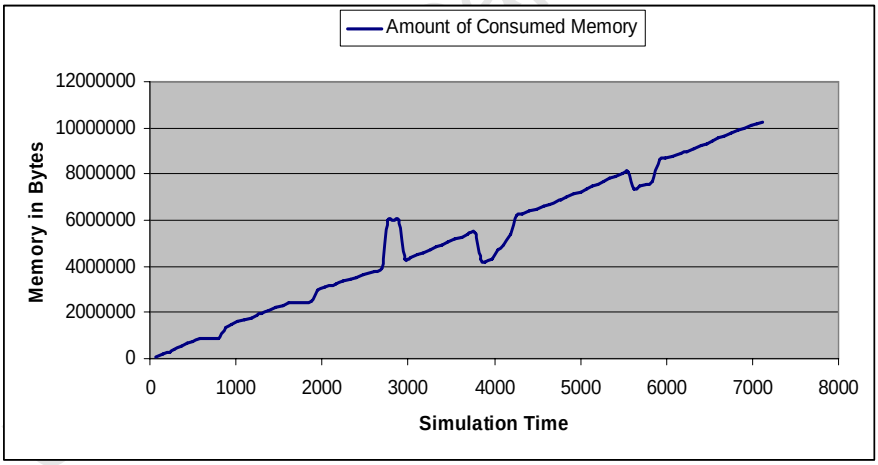


Figure 7-8. Memory Consumption.

Memory consumption at the server starts at 96.2 KB and grows linearly till it reaches 3.8 MB as the simulation time passes the 40 minutes mark. The figure shows a surge in consumption that brings the amount of consumed memory to 6 MB. The surge can be attributed to a sudden increase in the number of connection and session establishments. Thereafter, the amount of consumed memory is decreased to remain around 4 MB as 1 hour of simulation time elapses. Afterwards, the figure shows a linear increase in memory consumption followed by a decrease

before the overall consumption stabilizes around 10 MB. Such a figure is acceptable considering the plenty amounts of memory enjoyed by servers these days. Section 7.3.6 of this thesis describes how the IDS can efficiently decrease and curb the amount of consumed memory.

7.3.6 Performance Enhancers

The good performance figures shown by our architecture can be attributed to three main factors:

1. We use a flexible state management policy to curb the system's memory consumption. For a stateful IDS, it is vital to limit the overall memory requirements for state management to a tractable amount. The amount of memory required for connection state is determined by two factors: (i) the size of each state entry, and (ii) the maximum number of concurrent, still active connections. The flexible options provided by the *State Table* allow administrators to limit the number of new connections arriving at a host and consequently the amount of memory required by these connections. Allocating memory dynamically as required by new connections can lead to crashes due to memory exhaustion. Furthermore, the same flexible options in the *State Table* can be used to implement timeouts to improve state expiration. The procedures in the *Recommended Action* Field can be used to flush a connection's state if for some time no new activity is observed. Such procedures can even override the default values of timers provided by the protocol specifications, which helps the system to restore memory occupied by practically inactive connections.

2. The *Behavior Observer* implements finite state machines in switch-like (case) statements, which makes memory management efficient. Considering the cost of creating objects, there is no need in this scheme to create a new object for each transition or state in the finite state machine. Information that identifies calls and sessions uniquely can be stored at the cost of a few hundred bytes per entry. This low cost allows servers to accommodate hundreds of calls and sessions simultaneously without degrading the performance of the system.

3. Retrieving from the database requires going through only one level of hierarchy. Specification-based modules directly retrieve from *Field* and *State* tables which contain the actual signatures. In addition, Administrators can store more than one procedure in the

Recommended Action Field of the database for a single record. Therefore, the database can store in one record multiple signatures with slight variations.

7.4 Chapter Discussion

In this chapter, we showed and analyzed the results produced by our intrusion detection system in terms of detection accuracy and performance evaluation. Our discussion on detection accuracy addressed the IDS coverage, hit rate, and how our design went about solving the problem of false alarms. We presented the wide coverage of the IDS by highlighting the various protocols supported by the system and the different violations of security services our IDS was able to detect. Consequently, the IDS was able to detect all the attacks launched during the experiment. We demonstrated how our design empowered system administrators to reduce false alarms with the flexible options available in both specification-based and signature-based modules.

The chapter also presented quantitatively the effect of the IDS on the operation of servers and endpoints. Some of the vital parameters in VoIP networks and endpoints, such as end-to-end delay, call setup delay, processing delay, packet loss, and memory usage, were chosen for the discussion, and the minor impact of the IDS on these parameters was clearly shown. The next chapter concludes the thesis.

Chapter 8 Conclusions and Recommendations

8.1 Conclusions

This thesis started with clarifying the threats surrounding VoIP environments and showing the various security challenges facing service provisioning in such environments. That start constituted the prelude towards proposing a hybrid host-based intrusion detection system that is suitable for VoIP environments. The hybridism of our proposed intrusion detection system is the result of combining specification-based and signature-based approaches. The two detection approaches join forces to detect attacks targeting various network layers such as application, transport, and network layer.

The specification-based modules of our IDS are based in part on the Communicating Extended Finite State Machines model (CEFSMs), where each protocol is represented as an Extended Finite State Machine (EFSM). On the other hand, our signature-based modules are based in part on State Transition Analysis Techniques, where attacks are modeled as a sequence of actions leading from a system safe state to a compromised one.

Both specification-based and signature-based detection modules provide stateful and cross-protocol detection to enhance the system's efficiency. EFSMs representing monitored protocols maintain the state of a session and allow state to be collected from multiple requests and responses that constitute an entire session or connection. Furthermore, EFSMs exchange detection information in real-time to help the IDS to deal with cross-protocol attacks and make more accurate decisions. The use of State Transition Analysis Techniques in our signature-based detection module allows for the modeling of state-aware signatures. In addition, signatures that cross protocol and layer boundaries can be stored efficiently in our signature database.

This thesis has proposed vital improvements to the traditional State Transition Analysis Techniques. These improvements enable our signature database to model attacks that are difficult to model using similar State Transition Diagram-based approaches, such as Denial of Service (DoS) attacks. The improvements come in the form of configurable detection settings to enable administrators to customize the IDS based on the local environment and added detection

procedures to tackle more complex attacks.

Our system's adoption of the Communicating Extended Finite State Machines model allows specification-based detection modules to detect violations of protocol semantics. Furthermore, the improved State Transition Analysis Techniques enable our signature database to store semantics-aware attack signatures. Syntax violations are also taken care of by a certain specification-based module, and a dedicated table in the signature database.

We have implemented and tested the proposed architecture using the network simulator OMNeT++. OMNeT++ simulator is used to implement the detection modules and the attacks used to test the hit rate. It is also used to collect various results with regard to the system performance under various loads of traffic. The features possessed by OMNeT++ simulator allow us to flexibly implement the detection methodologies discussed earlier and launch various attacks against certain targets in the simulated network. We have presented in this thesis the advantages of our approach to using a simulated environment and how it can be used to overcome the hurdles of implementing and testing IDSs.

We have demonstrated in this thesis how the hybrid intrusion detection system has detected all the implemented attacks using the proposed detection methodologies. The implemented attacks are meant to be diverse in terms of the protocols involved and the security services violated. Some of the attacks target higher application layer protocols, whereas others target lower layers such as transport and network layer. Some of the attacks violate data availability and confidentiality, whereas others violate authentication and data integrity. We have also shown how the proposed IDS can flexibly overcome the causes of false alarms.

We have proven in this thesis that the operation of the proposed hybrid intrusion detection system has a trifling impact on the quality of service provided by the network. We have numerically assessed the effect of the architecture on vital system parameters such as end-to-end delay, call setup delay, processing delay, packet loss, and memory usage. The outcome of that assessment attests to the quality of our design and its minor effect on the operation of hosts.

8.2 Contributions

This thesis has made the following contributions to intrusion detection research in VoIP environments:

- We have proposed a novel and efficient combination of specification-based and signature-based detection approaches. Both detection approaches are well-known for their accurate detection and low false alarm rate. Therefore, their combination is expected to achieve the goal of improving the intrusion detection experience in general and bringing false alarms to their lowest level. Based on the experimental results which show accurate detection and identification of all the launched attacks along with a low false alarm rate, we can safely say that we have succeeded in achieving the abovementioned goal. The use of specification-based approaches simplifies the process of feature selection which has proven to be a hard problem in anomaly-based approaches. Furthermore, the use of signature-based approaches allows the system to detect attacks that are not based on repetition, which have proven to be challenging in anomaly-based approaches.

- This thesis has introduced vital improvements to State Transition Analysis Techniques which are used to model intrusions and penetrations. As discussed earlier in this thesis, State Transition Analysis uses State Transition Diagrams (STDs) to represent attacks. Despite the remarkable convenience STDs provide for attack modeling, they fall short of representing complex and Denial of Service (DoS) attacks. Our proposed improvements have addressed these shortcomings. Firstly, the detection procedures attached to attack signatures have the capacity to predict and detect quite complex attacks such as cross-protocol ones. They can also take obfuscation resistance capabilities of State Transition Analysis one step further by tackling different types of intermediate states that represent variants of the attack. Secondly, the flexible detection parameter setting provided with attack signatures allows administrators to customize the IDS based on the security relevance of events and underlying network conditions. These two improvements enable our signatures to be flexible enough to detect variants of the same attack and tight enough to distinguish attacks reliably.

- Our signature database overtakes other signature databases by introducing the *state* model. This feature enables our database to store a higher-level abstraction of attacks than previous works and support more general signatures. It also brings semantics-awareness to attack modeling by using state transition diagrams which allow us to represent attacks at the session level rather than lower and semantics-less levels. Furthermore, the *state* model allows for a tight integration with specification-based approaches that are based on Extended Finite State Machines.

- This thesis presented an evaluation framework based on a network simulator. The framework can be used to generate attacks and implement detection methodologies. It can also be used to collect various results aiming at assessing the performance aspects of intrusion detection systems. As discussed earlier in this thesis, the framework can be reliably used to test both the hit rate and false alarm rate of intrusion detection systems. Considering the fact that there are not many instances of VoIP attacks available in public databases and the difficulty of customizing available attacks, our framework can be used for future research in this area.

8.3 Future work and Recommendations

In the following, we present some issues for consideration for future work in the field of intrusion detection systems in VoIP environments:

- The simulated network in this research included two domains. Future research involving the investigation of the performance of the intrusion detection system for several domains can demonstrate the behavior of the system for large networks. Furthermore, more applications with more protocols at various layers can be added to take our tests a step further.

- In the previous chapter, we made a mention of "The Problem of Crud" which faces intrusion detection systems. Considering the diversity of legitimate network traffic, including the implementation errors sometimes reflected within it, the IDS may find it difficult to discern in some circumstances between a true attack and an innocuous

implementation error. What worsens the problem is the reliance of IDSs on strict specifications that do not consider implementation differences. Although we have addressed this issue in this thesis by relying on more abstract specifications in our specification-based modules, we believe there is still some room for improvement. This improvement can come in the form of an intelligent mechanism that adopts fuzzy logic to help specification-based modules to be more flexible. A good reason fuzzy logic is introduced for intrusion detection is that security itself includes fuzziness [65]. Classical approaches in intrusion detection define a range value or an interval to denote a normal value. Then, any values falling outside the range are considered anomalies regardless of their distance to the interval. Unfortunately, this causes an abrupt separation between normality and anomaly. Fuzzy logic helps smooth this abrupt separation and produces more general rules which will increase the flexibility of the IDSs. The runtime impact of introducing fuzzy logic rules in our specification-based modules should also be evaluated thoroughly.

- We showed previously in this thesis that some of the attacks could be detected at the attacker side (CANCEL attack). We can envisage a situation where the attacker is an insider with bad intentions. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordinating and cooperating among intrusion detection systems across the network. Coordinating and cooperating among IDSs will help limit the damage of attacks such as the abovementioned. When it comes to coordination and cooperation among IDSs, a major issue is to define data format and exchange procedures for sharing information of interest. A possible start might be some of the work of the IETF Intrusion Detection Working Group [66], which focuses on standards to support interoperability. These guidelines could be the seeds of a future work to convert our system into a distributed intrusion detection system.

References

[1] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co., Fort Washington, PA, April 1980.

[2] W. Hu, W. Kou, and C. Lee, *Advances in Security and Payment Methods for Mobile Commerce*, Hershey, PA, USA: IGI Publishing, 2005, p. 38.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Third ed., Delhi, India: Pearson Education, 2003, p. 570.

[4] R. Oppliger, *Internet and Intranet Security*, 2nd ed., Norwood, MA: Artech House, 2002, p. 374.

[5] D. Wagner and R. Dean, "Intrusion Detection via Static Analysis," *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.

[6] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection : A new approach for detecting network intrusions," *In ACM Computer and Communication Security Conference (CCS)*, Washington DC, November 2002.

[7] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using Specification-based Intrusion Detection for Automated Response," *In Proceedings of the Sixth International Symposium, Recent Advances in Intrusion Detection (RAID'03)*, Pittsburg, PA, 2003.

[8] B. I. A. Barry and H. A. Chan, "A Hybrid, Stateful, and Cross-protocol Intrusion Detection System for Converged Applications," *Springer LNCS*, vol. 4804, OTM 2007, Part II, pp. 1616-1633, November 2007.

[9] P. A. Porras, "STAT – A State Transition Analysis Tool for Intrusion Detection," Masters Thesis, Computer Science Department, University of California, Santa Barbara, June 1992.

[10] P. Mell, V. Hu, R. Lipmann, J. Haines, and M. Zissman, "An Overview of Issues in Testing Intrusion Detection Systems," Technical Report NIST IR 7007, National Institute of Standard and Technology, August 2003. Available http://csrc.nist.gov

[11] T. Porter, *Practical VoIP Security*. Rockland, MA: Syngress, 2006, Chapter 1.

[12] N. Khan, "The SIP Servlet Programming Model," Technology White Paper (Oct. 2007). Available http://dev2dev.bea.com

[13] Y. Wu, S. Bagchi, S. Garg, N. Singh, and T. Tsai, "SCIDIVE: A Stateful and Cross Protocol Intrusion Detection Architecture for Voice-over-IP Environments," *In Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, Florence, Italy, 2004.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M.

Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, IETF Network Working Group, June 2002.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport Protocol for Real-Time Applications," RFC 1889, IETF Network Working Group, January 1996.

[16] B. I. A. Barry and H. Anthony Chan, "Towards Intelligent Cross-Protocol Intrusion Detection in the Next Generation Networks Based on Protocol Anomaly Detection," *In Proceedings of the 9th International Conference on Advanced Communication Technology (ICACT2007)*, Phoenix Park, Gangwon-Do, Korea, February 2007.

[17] International Telecommunication Union –Recommendation Q.700 (1993, March). *Introduction to CCITT Signaling System No. 7.* Available: http://www.itu.int

[18] International Telecommunication Union –Recommendation H.323 (2006, June). *Packet-based Multimedia Communications Systems.* Available: http://www.itu.int

[19] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," RFC 2543, IETF Network Working Group, March 1999.

[20] Y. Zhang and W. Lee, "Intrusion detection in Wireless Ad hoc Networks," *In Proceedings of Sixth International Conference on Mobile Computing and Networking*, Boston, Massachusetts, pp. 275–283, August 2000.

[21] G. Thamilarasu, A. Balasubramanian, S. Mishra, and R. Sridhar, "A Cross-layer based Intrusion Detection Approach for Wireless Ad hoc Networks," *In Proceedings of IEEE International Conference on Mobile Ad hoc and Sensor Systems Conference*, Washington DC, November 2005.

[22] G. Vigna, W. Robertson, V. Kher, R.A. Kemmerer, "A Stateful Intrusion Detection System for World-Wide Web Servers," *In Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*, Las Vegas, NV, December 2003.

[23] G. Vigna and R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection Approach," *In Proceedings of the 14th Annual Computer Security Application Conference (ACSAC)*, Scottsdale, Arizona, 1998.

[24] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," *IEEE Transactions on Software Engineering*, vol. 21, issue. 3, pp. 181-199, Mar. 1995.

[25] J. M. Orset, B. Alcalde, and A. Cavalli, "An EFSM-Based Intrusion Detection System for Ad Hoc Networks," *Proceedings of The Third International Symposium, Automated Technology for Verification and Analysis (ATVA)*, Taipei, Taiwan, October 2005.

[26] Snort – The de facto Standard for Intrusion Detection/Prevention. Available: http://www.snort.org

[27] R. Sommer and V. Paxon, "Enhancing Byte-level Network Intrusion Detection Signatures with Context," *Proceedings of Tenth ACM Conference on Computer and Communication*

Security, Washington DC, 2003.

[28] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey, "A Real-time Intrusion Detection Expert System (IDES), Final Technical Report," Computer Science Laboratory, SRI International, Menlo Park, CA, February 1992.

[29] K. A. Jackson, D. H. DuBios, and C. A. Stalling, "An Expert System Application For Network Intrusion Detection," *Proceedings of the 14th National Computer Security Conference,* Baltimore, MD, pp. 215-225, October 1991.

[30] H. S. Vaccaro and G. E. Liepins, "Detection of Anomalous Computer Session Activity," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, pp. 280-289, May 1989.

[31] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "VoIP Intrusion Detection Through Interacting Protocol State Machines," *In Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, Philadelphia, USA, 2006.

[32] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "Fast Detection of Denial-of-Service Attacks on IP Telephony," *Proceedings of IEEE Fourteenth International Workshop on Quality of Service*, New Haven, CT, 2006.

[33] B. Lee and E. A. Lee, "Interaction of Finite State Machines and Concurrency Models," *Proceedings of Thirty Second Annual Asilomar Conference on Signals, Systems, and Computers*, Pacific Groove, California, November 1998.

[34] J. Gerald Holzmann, *Design and Validation of Computer Protocols.* New Jersey: Prentice Hall, 1991, Chapter 8.

[35] D. Lee, and M. Yannakakis, "Principles and Methods of Testing Finite State Machines," *Proceedings of the IEEE*, vol. 84, issue. 8, pp. 1090-1123, Aug. 1996.

[36] Jilles van Gurp and Jan Bosch, "On The Implementation of Finite State Machines," *Proceedings of 3rd Annual IASTED International Conference on Software Engineering and Applications*, Scottsdale, Arizona, USA, October 1999.

[37] A.S. Krishnakumar, "Reachability and Recurrence in Extended Finite State Machines: Modular Vector Addition Systems," *Proceedings of Fifth International Conference on Computer Aided Verification*, pp. 110-122, 1993.

[38] A. Petrenko, S. Boroday, and R. Groz, "Confirming Configurations in EFSM Testing," *IEEE Transactions on Software Engineering*, vol. 30, issue. 1, pp. 29-42, January 2004.

[39] A. Sundaram, "An Introduction to Intrusion Detection," *ACM Crossroads Magazine, Special Issue on Computer Security*, vol. 2, issue. 4, pp. 3-7, April 1996.

[40] M. Poikselka, G. Mayer, H. Khartabil, and A.Niemi, *The IMS: IP Multimedia Concepts and Services in the Mobile Domain*. Sussex: Wiley, 2004, Chapter 8.

[41] VOIPSA, "VoIP Security and Privacy Threat Taxonomy," October 2005. Available http://www.voipsa.org

[42] R. A. Wasniowski, "Network Intrusion Detection System with Data Mart," *Proceedings of The 2006 International Conference on Security and Management*, Las Vegas, USA, 2006.

[43] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, MA: Addison-Wesley 1994.

[44] D. E. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture*. Englewood Cliffs, New Jersey: Prentice-Hall 1991, Chapter 10.

[45] S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *ACM SIGCOMM Computer Communication Review*, vol. 19, issue. 2, pp. 32-48, April 1989.

[46] D. J. Marchette, *Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint*. York, PA: Springer-Verlag, 2001, Chapter 4.

[47] T. Newsham and J. Hoagland, "Windows Vista Network Attack Surface Analysis: A Broad Overview," Technical Report, SYMANTEC Advanced Threat Research, Published July 2006. Available: http://www.symantec.com

[48] J. Postel, "Internet Protocol," RFC 791, DARPA Internet Program Protocol Specification, September 1981.

[49] J. Postel, "Internet Control Message Protocol," RFC 792, IETF Network Working Group, September 1981.

[50] J. Postel, "User Datagram Protocol," RFC 768, IETF Network Working Group, August 1980.

[51] J. Postel, "Transmission Control Protocol," RFC 793, DARPA Internet Program Protocol Specification, September 1981.

[52] J. Crowcroft, I. Wakeman, Z. Wang, and D. Sirovica, "Is Layering Harmful?," *IEEE Network Magazine*, vol. 6, issue. 1, pp. 20-24, January 1992.

[53] James Kurose, and Keith Ross, "Computer Networking: Atop-Down Approach Featuring the Internet," Pearson Education, Inc, New York, 2005, p. 47.

[54] R. Lippmann, J.W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line Intrusion Detection Evaluation," *In Proceedings of Recent Advances in Intrusion Detection (RAID)*, Toulouse, France, October 2000.

[55] Fyodor, "Remote OS Detection via TCP/IP Stack Fingerprinting," June 2002. Available http://insecure.org

[56] CERT® Advisory CA-2001-09 Statistical Weaknesses in TCP/IP Initial Sequence Numbers, February 2005. Available http://www.cert.org/advisories/CA-2001-09.html

[57] OMNeT++ Simulator. Available: http://www.omnetpp.org

[58] MMSim – Simulation of Multimedia Protocols using OMNeT++. Available: http://www.ibr.cs.tu-bs.de/projects/mmsim

[59] P. E. Proctor, *The Practical Intrusion Detection Handbook*. Englewood Cliffs, New Jersey: Prentice-Hall 2001, pp. 108-111.

[60] V. Paxon, "Bro: A System for Detecting Network Intruders in Real-Time," *In Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[61] International Telecommunication Union – Telecommunication Standardization Section Recommendation G.114 (2003, May). *One-way Transmission Time.* Available: http://www.itu.int

[62] B. Goode, "Voice over Internet Protocol (VoIP)," *Proceedings of The IEEE*, vol. 90, issue. 9, September 2002.

[63] Internet Engineering Task Force – Internet Draft (1999. June). *VoIP Signaling Performance Requirements and Expectations.* Available: http://tools.ietf.org

[64] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion Detection Systems," *Computer Networks – The International Journal of Computer and Telecommunications Networking*, vol. 31, issue 8, pp. 805-822, 1999.

[65] J. Luo, "Integrating Fuzzy Logic With Data Mining Methods for Intrusion Detection," Msc Thesis, Mississippi State University, 1999.

[66] M. Wood, and M. Erlinger, "Intrusion Detection Message Exchange Requirements," RFC 4766, IETF Network Working Group, March 2007.

[67] Bazara I. A. Barry and H. Anthony Chan, "A Cross-protocol approach to detect TCP Hijacking attacks," *In Proceedings of 2007 IEEE International Conference on Signal Processing and Communications (ICSPC07)* , Dubai, United Arab Emirates (UAE), 24-27 November 2007.

[68] The NSS Group, *Intrusion Detection System Group Test (Edition 4)* 2003. Available: http://www.nss.co.uk

[69] P. Mueller and G. Shipley, "Dragon claws its way to the top," *Network Computing*, pp. 45-67, August 2001.

[70] National Laboratory for Applied Network Research, NLAR Network Traffic Packet Header Traces, 2003. Available: http://pma.nlanr.net

[71] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo, "Testing and Evaluating Computer Intrusion Detection Systems," *Communications of ACM*, vol. 42, issue 7, pp. 53-61, July 1999.

[72] B. Barry and H. A. Chan, "On the Performance of A Hybrid Intrusion Detection

Architecture for Voice over IP Systems," *In Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm'08)*, Istanbul, Turkey, September 2008.

[73] B. Barry and H. Anthony Chan, "Intrusion Detection Systems: Classifications, Implementation Approaches, Testing Methods, and Evaluation Techniques," book chapter in Handbook on Communications and Information Security, edited by Peter Stavroulakis, to be published by Springer in 2009.

[74] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control," RFC 1890, IETF Network Working Group, January 1996.

[75] C. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. San Francisco, CA: No Starch Press 2005.

[76] Voip-Info.org, *QoS*, 2004. Available: http://www.voip-info.org

[77] 29West Inc., Topics in High-Performance Messaging, 2004. Available: http://www.29west.com

[78] J. Ryan, M. Lin, and R. Mikkulainen, *Intrusion Detection with Neural Networks*, *Advances in Neural Information Processing Systems*. Vol. 10, Cambridge MA: MIT Press. 1998.

[79] A. Ghosh, A. Schwartzbard, and M. Shatz, "Learning Program Behavior Profiles for Intrusion Detection," *In Proceedings of First USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, April 1999.

[80] S. L. Scott, "A Bayesian paradigm for designing intrusion detection systems," *Computational Statistics Data Analysis*. Vol. 45, Issue 1, pp. 69-83, February 2004.

[81] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag, "A high-performance network intrusion detection system," *In Proceedings of 6th ACM Conference on Computer and Communication Security*, Singapore, November 1999.

[82] A. Pagnoni, and A. Visconti, "An innate immune system for the protection of computer networks," *In Proceedings of the 4th international symposium on Information and communication technologies*, Cape Town, South Africa, January 2005.

[83] M. Handley, V. Paxon, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," *In USENIX Security Symposium*, Washington, DC, August 2001.

[84] Bazara Barry and H. Anthony Chan, "A Signature Database for Intrusion Detection Systems Targeting Voice over Internet Protocol," *Accepted to Appear In Proceedings of the 2008 IEEE Military Communications Conference* (MILCOM'08), San Diego, CA, November 2008.

[85] F. Baker, "Requirements for IP Version 4 Routers," RFC 1812, IETF Network Working Group, June 1995.

[86] M. Raihan, and M. Zulkernine, "Detecting Intrusions Specified in a Software Specification Language," *In Proceedings of International Computer Software and Applications Conference (COMPSAC)*, 2005.

[87] M. Zulkernine, M. Graves, and U. Khan "Integrating Software Specifications into Intrusion Detection," *International Journal of Information Security (IJIS)*. pp. 345-357, Springer Verlag, September 2007.

[88] P. Uppuluri, and R. Sekar, "Experiences with Specification-based Intrusion Detection," *In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2001.

[89] C. C. Michael, and A. Ghosh, "Simple, State-based Approaches to Program-based Anomaly Detection," *ACM Transactions on Information and System Security*. Vol. 5, Issue 3, pp. 203-237, August 2002.

[90] John McHugh, "Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Off-line Intrusion Detection as Performed by Lincoln Laboratories," *ACM Transactions on Information and System Security*. Vol. 3, Issue 4, pp. 262-294, November 2000.

Appendix A: Published Work

Some research outputs during this thesis research have also been published or are being published:

1. Bazara Barry and H. Anthony Chan, "Intrusion Detection Systems: Classifications, Implementation Approaches, Testing Methods, and Evaluation Techniques," Book chapter in *Handbook on Communications and Information Security*, edited by Peter Stavroulakis, to be published by Springer in 2009.

2. Bazara Barry and H. Anthony Chan, "A Signature Database for Intrusion Detection Systems Targeting Voice over Internet Protocol," *Accepted to Appear In Proceedings of the 2008 IEEE Military Communications Conference* (MILCOM'08), San Diego, CA, November 2008.

3. Bazara Barry and H. Anthony Chan, "On the Performance of A Hybrid Intrusion Detection Architecture for Voice over IP Systems," *In Proceedings of the 4th International Conference on Security and Privacy in Communication Networks* (SecureComm'08), Istanbul, Turkey, September 2008.

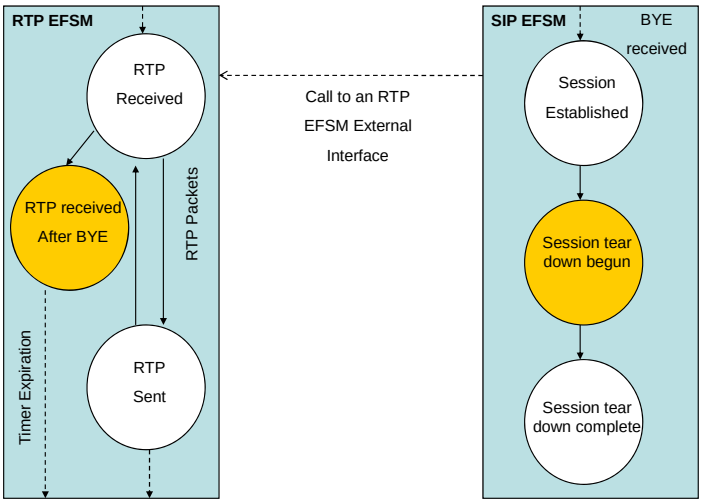
4. Bazara Barry and H. Anthony Chan, "A Hybrid, Stateful, and Cross-protocol Intrusion Detection System for Converged Applications," *Springer LNCS*, vol. 4804, OTM 2007, Part II, pp. 1616-1633, November 2007.

5. Bazara Barry and H. Anthony Chan, "A Cross-protocol approach to detect TCP Hijacking attacks," *In Proceedings of 2007 IEEE International Conference on Signal Processing and Communications* (ICSPC07) , Dubai, United Arab Emirates (UAE), 24-27 November 2007.

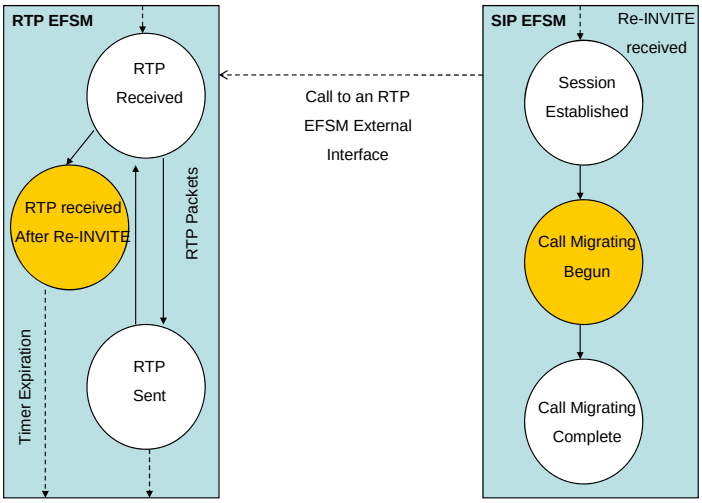
6. Bazara Barry and H. Anthony Chan, "Towards Intelligent Cross-Protocol Intrusion Detection in the Next Generation Networks Based on Protocol Anomaly Detection," *In Proceedings of the 9th International Conference on Advanced Communication Technology* (ICACT2007), Phoenix Park, Gangwon-Do, Korea, February 2007.

Appendix B: Diagrammatic Representation for Some of the Implemented Attacks and Detection Methodologies

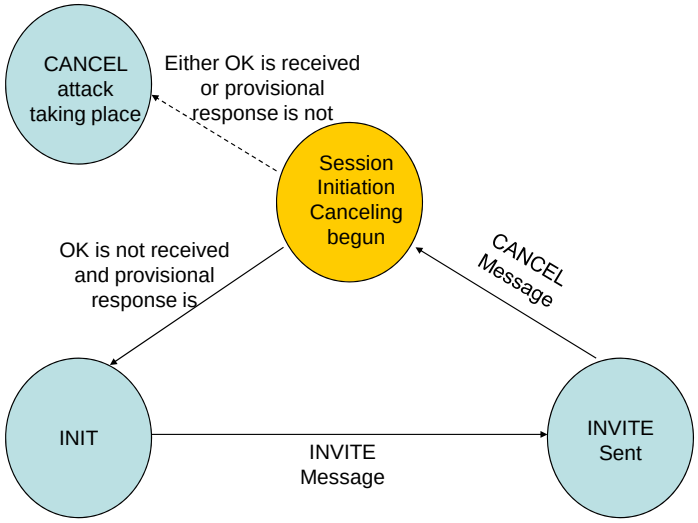
1\ BYE Attack:



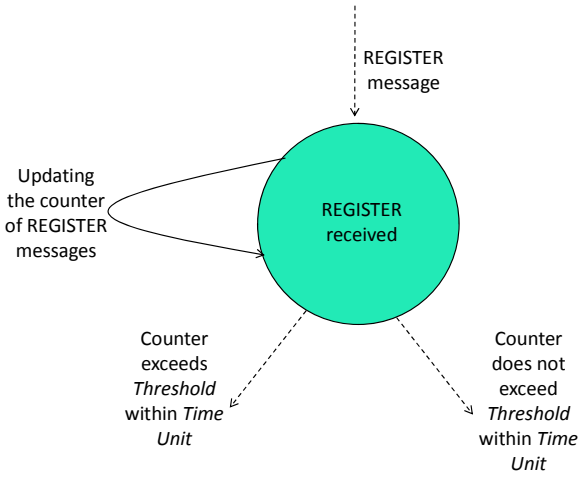
2\ Re-INVITE Attack:



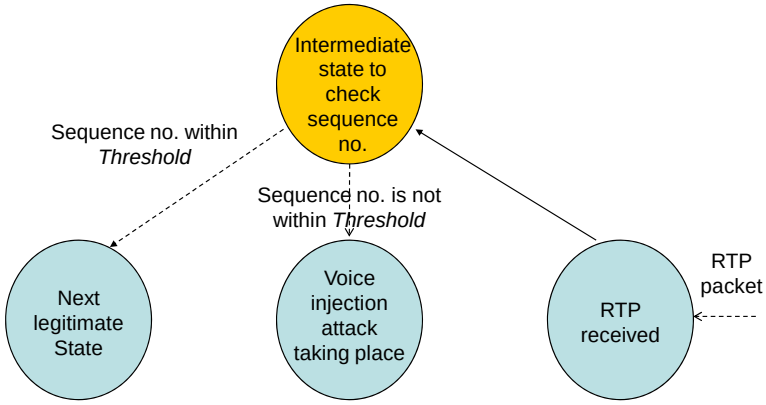
3\ CANCEL Attack:



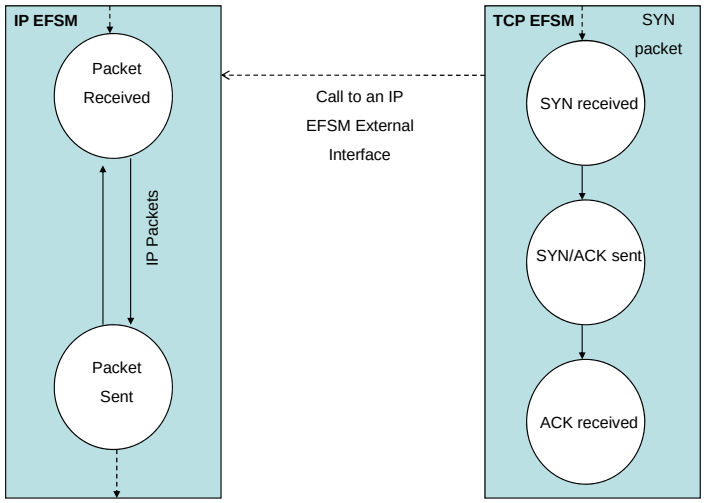
4\ REGISTER Flooding Attack:



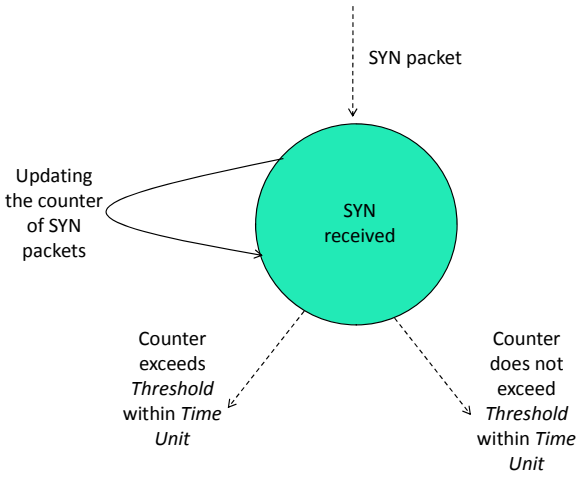
5\ Voice Injection Attack:



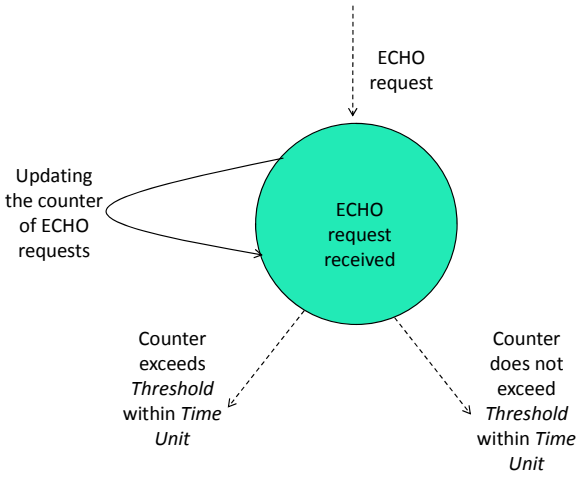
6\ Land Attack:



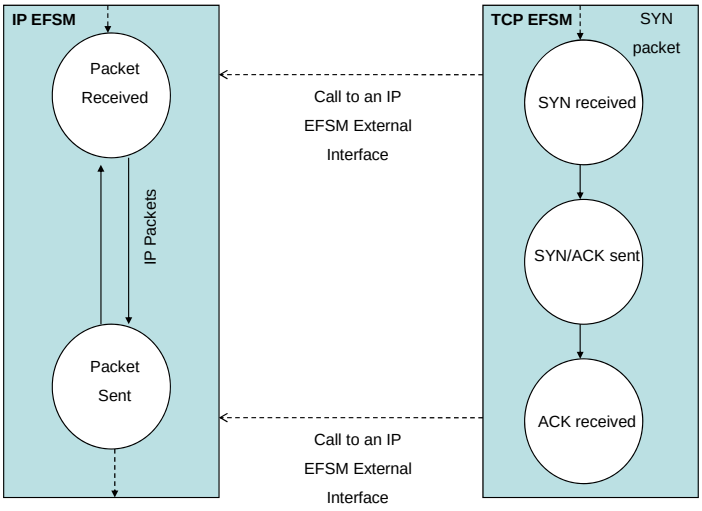
7\ Neptune Attack:



8\ Smurf Attack:



9\ TCP Hijacking Attack:



Appendix C: NED File for Simulated Network

```
import
    "Router",
    "EtherSwitch",
    "sipHost",
    "sipReg",
    "sipProxy";

channel Internet
    delay 40ms;
    datarate 512*1000000;
endchannel

channel ethernetline
    delay 15ms;
    error 1e-8;
    datarate 10*1000000;
endchannel

module simpleNetModule

    parameters:
        debug : bool;

    submodules:
        client1: sipHost;
            parameters:
                debug = debug,
                nodeName = "UA1",
            localUser = "User1",
                aktAsServer = false,
                portNumber = 5060,
                numOfPorts = 2,
                routingFile = "client1.irt";

            display: "i=image-uac;p=67,80;b=32,32";


        client2: sipHost;
            parameters:
                debug = debug,
                nodeName = "UA2",
            localUser = "User2",
                aktAsServer = false,
                portNumber = 5060,
                numOfPorts = 2,
                routingFile = "client2.irt";

            display: "p=64,141;b=32,32;i=image-uac";


        client3: sipHost;
            parameters:
                debug = debug,
                nodeName = "UA3",
            localUser = "User3",
                aktAsServer = false,
```

```
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client3.irt";

    display: "p=64,213;b=32,32;i=image-uac";

client4: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA4",
    localUser = "User4",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client4.irt";

    display: "p=64,250;b=32,32;i=image-uac";

client5: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA5",
    localUser = "User5",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client5.irt";

    display: "p=64,260;b=32,32;i=image-uac";

client6: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA6",
    localUser = "User6",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client6.irt";

    display: "p=120,80;b=32,32;i=image-uac";

client7: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA7",
    localUser = "User7",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client7.irt";

    display: "p=120,141;b=32,32;i=image-uac";

client8: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA8",
    localUser = "User8",
        aktAsServer = false,
```

```
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client8.irt";

    display: "p=120,213;b=32,32;i=image-uac";

client9: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA9",
    localUser = "User9",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client9.irt";

    display: "p=120,250;b=32,32;i=image-uac";


client10: sipHost;
    parameters:
        debug = debug,
        nodeName = "UA10",
    localUser = "User10",
        aktAsServer = false,
        portNumber = 5060,
        numOfPorts = 2,
        routingFile = "client10.irt";

    display: "p=64,260;b=32,32;i=image-uac";


proxy1: sipProxy; //
    parameters:
        debug = debug,
        nodeName = "1st Proxy",
        routingFile = "proxy1.irt",
        numOfPorts = 2;
    gatesizes:
        in[1],
        out[1]; //
                //
    display: "i=device/server_l;p=161,176;b=40,24";

registrar1: sipReg; //
    parameters:
        debug = debug,
        nodeName = "1st Registrar",
        routingFile = "reg1.irt",
        numOfPorts = 1;
    gatesizes:
        in[1],
        out[1]; //
                //
    display: "i=device/server_l;p=161,186;b=40,24";


proxy2: sipProxy; //
    parameters:
        debug = debug,
```

```
            nodeName = "2nd Proxy",
            routingFile = "proxy2.irt",
            numOfPorts = 2;
        gatesizes:
            in[1],
            out[1]; //
                    //
        display: "i=router;p=249,176;b=32,32";

    registrar2: sipReg; //
        parameters:
            debug = debug,
            nodeName = "2nd Registrar",
            routingFile = "reg2.irt",
            numOfPorts = 1;
        gatesizes:
            in[1],
            out[1]; //
                    //
        display: "i=device/server_l;p=249,186;b=40,24";

    switch1: EtherSwitch;
        gatesizes:
            in[8],
            out[8];
        display: "p=100,236;i=switch2";

    switch2: EtherSwitch;
        gatesizes:
            in[8],
            out[8];
        display: "p=200,236;i=switch2";


    router1: Router;
        display: "p=161,59;b=32,15;i=ipc";


    router2: Router;
        display: "p=249,59;b=32,15;i=ipc";


connections:
    router1.out++ --> Internet --> router2.in++;
    router1.in++ <-- Internet <-- router2.out++;

    switch1.out[0] --> ethernetline --> client1.in;
    switch1.in[0] <-- ethernetline <-- client1.out;
    switch1.out[1] --> ethernetline --> client2.in;
    switch1.in[1] <-- ethernetline <-- client2.out;
    switch1.out[2] --> ethernetline --> client3.in;
    switch1.in[2] <-- ethernetline <-- client3.out;
    switch1.out[3] --> ethernetline --> client4.in;
    switch1.in[3] <-- ethernetline <-- client4.out;
    switch1.out[4] --> ethernetline --> client5.in;
    switch1.in[4] <-- ethernetline <-- client5.out;
    switch1.out[5] --> ethernetline --> proxy1.in;
    switch1.in[5] <-- ethernetline <-- proxy1.out;
    switch1.out[6] --> ethernetline --> registrar1.in;
    switch1.in[6] <-- ethernetline <-- registrar1.out;
```

```
        switch1.out[7] --> ethernetline --> router1.in++;
        switch1.in[7] <-- ethernetline <-- router1.out++;

        switch2.out[0] --> ethernetline --> client6.in;
        switch2.in[0] <-- ethernetline <-- client6.out;
        switch2.out[1] --> ethernetline --> client7.in;
        switch2.in[1] <-- ethernetline <-- client7.out;
        switch2.out[2] --> ethernetline --> client8.in;
        switch2.in[2] <-- ethernetline <-- client8.out;
        switch2.out[3] --> ethernetline --> client9.in;
        switch2.in[3] <-- ethernetline <-- client9.out;
        switch2.out[4] --> ethernetline --> client10.in;
        switch2.in[4] <-- ethernetline <-- client10.out;
        switch2.out[5] --> ethernetline --> proxy2.in;
        switch2.in[5] <-- ethernetline <-- proxy2.out;
        switch2.out[6] --> ethernetline --> registrar2.in;
        switch2.in[6] <-- ethernetline <-- registrar2.out;
        switch2.out[7] --> ethernetline --> router2.in++;
        switch2.in[7] <-- ethernetline <-- router2.out++;

endmodule

network simpleNet : simpleNetModule
    parameters:
        debug = true;
endnetwork
```

Appendix D: Initialization File for Simulated Network

```
[General]
network = simpleNet
ini-warnings = false
preload-ned-files = *.ned @../../../nedfiles.lst
sim-time-limit = 2h
seed-0-mt=532569
num-rngs=5

[Tkenv]
default-run=1
module-messages = no
Verbose-simulation = no
plugin-path=../../../Etc/plugins

[Parameters]
*.debug = true
simpleNet.proxy1.localIp = "10.0.0.1"
simpleNet.proxy1.cmdFileName = "proxy01.cmd"
simpleNet.proxy1.localUser = ""
simpleNet.registrar1.localIp = "10.0.0.2"

simpleNet.proxy2.localIp = "10.0.10.1"
simpleNet.proxy2.cmdFileName = "proxy02.cmd"
simpleNet.proxy2.localUser = ""
simpleNet.registrar2.localIp = "10.0.10.2"

simpleNet.client1.localIp = "10.0.0.10"
simpleNet.client1.useProxy = "10.0.0.1"
simpleNet.client1.cmdFileName = "client01.cmd"

simpleNet.client2.localIp = "10.0.0.15"
simpleNet.client2.useProxy = "10.0.0.1"
simpleNet.client2.cmdFileName = "client02.cmd"

simpleNet.client3.localIp = "10.0.0.20"
simpleNet.client3.useProxy = "10.0.0.1"
simpleNet.client3.cmdFileName = "client03.cmd"

simpleNet.client4.localIp = "10.0.0.25"
simpleNet.client4.useProxy = "10.0.0.1"
simpleNet.client4.cmdFileName = "client04.cmd"

simpleNet.client5.localIp = "10.0.0.30"
simpleNet.client5.useProxy = "10.0.0.1"
simpleNet.client5.cmdFileName = "client05.cmd"

simpleNet.client6.localIp = "10.0.10.10"
simpleNet.client6.useProxy = "10.0.10.1"
simpleNet.client6.cmdFileName = "client06.cmd"

simpleNet.client7.localIp = "10.0.10.15"
simpleNet.client7.useProxy = "10.0.10.1"
simpleNet.client7.cmdFileName = "client07.cmd"

simpleNet.client8.localIp = "10.0.10.20"
simpleNet.client8.useProxy = "10.0.10.1"
```

```
simpleNet.client8.cmdFileName = "client08.cmd"

simpleNet.client9.localIp = "10.0.10.25"
simpleNet.client9.useProxy = "10.0.10.1"
simpleNet.client9.cmdFileName = "client09.cmd"

simpleNet.client10.localIp = "10.0.10.30"
simpleNet.client10.useProxy = "10.0.10.1"
simpleNet.client10.cmdFileName = "client010.cmd"


# RTP Parameter
*.portNumberRTP    = 2
*.profileNameRTP = "RTPAVProfile"
*.bandwidthRTP     = 44100
*.payloadTypeRTP = 32
*.fileNameRTP      = "../Data/moving.mpg.gdf"
*.autoOutputFileNames = true

# udp app (off)
**.numUdpApps=0
**.udpAppType="UDPBasicApp"

# tcp apps
**.client*.numTcpApps=1
**.client*.tcpAppType="TCPBasicClientApp"
**.client*.tcpApp[0].address=""
**.client*.tcpApp[0].port=-1
**.client*.tcpApp[0].connectPort=80

**.client*.tcpApp[0].startTime=exponential(5)
**.client*.tcpApp[0].numRequestsPerSession = 3500
**.client*.tcpApp[0].requestLength = truncnormal(350,20)
**.client*.tcpApp[0].replyLength = exponential(2000)
**.client*.tcpApp[0].thinkTime=truncnormal(2,3)
**.client*.tcpApp[0].idleInterval=truncnormal(3600,1200)
**.client*.tcpApp[0].reconnectInterval=30

**.proxy*.numTcpApps=1
**.proxy*.tcpAppType="TCPGenericSrvApp"
**.srv.tcpApp[0].address=""
**.srv.tcpApp[0].port=80

# ping app
**.pingApp.destAddr=""
**.pingApp.srcAddr=""
**.pingApp.interval=1
**.pingApp.hopLimit=32
**.pingApp.count=0
**.pingApp.startTime=1
**.pingApp.stopTime=0
**.pingApp.printPing=true

# tcp settings
**.tcp.mss = 536
**.tcp.advertisedWindow = 7504  # 14*mss
**.tcp.sendQueueClass="TCPMsgBasedSendQueue"
**.tcp.receiveQueueClass="TCPMsgBasedRcvQueue"
**.tcp.tcpAlgorithmClass="TCPReno"
**.tcp.recordStats=true
```

```
# ip settings
**.routingFile=""
**.ip.procDelay=10us
**.ip.stateEntry=1300

# ARP configuration
**.arp.retryTimeout = 1
**.arp.retryCount = 3
**.arp.cacheTimeout = 100

# NIC configuration
**.ppp[*].queueType = "DropTailQueue" # in routers
**.ppp[*].queue.frameCapacity = 100  # in routers

# hook names
**.qosBehaviorClass = "EnqueueWithoutQoS"

# nam trace
**.nam.logfile = "trace.nam"
**.nam.prolog = ""
**.namid = -1  # auto
```