2010

Network attacks and securing streaming content

Liang Lu
University of Wollongong

Network Attacks and Securing Streaming Content

A thesis submitted in fulfillment of the
requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Liang Lu

School of Computer Science and Software Engineering
July 2010

Dedicated to
My mother and my father

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Liang Lu
July 22, 2010

Abstract

Despite many years of effort by the industry as well as the research community, attacks on computer systems via access networks are still a severe threat. In the battle against network attacks, firewalls and Intrusion Detection Systems (IDSs) have played one of the most important roles. However, conventional firewalls and IDSs have technical limitations and as such have difficulties dealing with emerging network applications, a notable example of which being streaming content. Besides, configuring firewall rule tables for large networks with complex security requirements is a difficult and error prone task.

In this thesis, we study the behavior of streaming content applications and look into techniques for enhancing firewalls/IDSs capabilities to cater for this new network application requirement. To assist system administrators to correctly implement organisational policies, we also develop a method of representing a firewall rule table that allows comparison of two tables, and provide an algorithm that determines if two tables are equivalent.

Even enhanced with techniques we provided, conventional firewalls/IDSs themselves still have difficulties dealing with complicated network threats and challenges. A notable example is multi-stage attacks where each stage itself does not violate security policy and is not detected by firewalls/IDSs.

A new mechanism, namely attack graphs, has emerged to model and defend against multi-stage attacks. However like any other new technologies, attack graphs have technical limitations such as sizing or scaling issues. In this thesis, we present our contribution to the area of ranking attack graphs. Our contribution lies in two major areas: accurate ranking of attack graphs, and efficient ranking by an artificial intelligence approach.

Acknowledgement

My experience as a graduate student in the University of Wollongong has been wonderful. I am grateful to my principal supervisor Prof. Rei Safavi-Naini for this opportunity. Rei has been an excellent supervisor who has offered me directions and yet enough freedom for me to explore different areas. I would like to express my gratitude to my co-supervisor Prof. Willy Susilo and Dr. Jeffrey Horton for their guidance. Willy and Jeffrey have always been nice to me.

My sincere thanks to Dr. Markus Hargenbuckner and his artificial intelligence research group for all their support and advice. Their help has been invaluable for my research work on combining techniques in the network security field and artificial intelligence field. Special thanks to S. L. Yong with his valuable comments on much of my work. All the discussions are very helpful in the development of this thesis.

I am fortunate to be here with a team of interesting people. Discussions are always stimulating and rewarding. I have enjoyed discussions with Angela Piper, Noi Rungrat, Xinyi Huang, Mohammad Reza Reyhanitabar, Man Ho Au, Shams Ud Din Qazi, Siamak Fayyaz Shahandashti, etc. The list goes on and on.

I would like to acknowledge the support that I have received from all the academic and general staff in the School of CS & SE from the University of Wollongong, Australia.

Finally, my love and gratitude to my parents and families for their enduring love and support. A special thanks to my beloved girl friend Yi Gao, nicknamed colored-piggy.

Publications

The following papers have been published or presented, and contain materials based on the content of this thesis.

1. Liang Lu, Rei Safavi-Naini, Markus Hagenbuchner, Willy Susilo, Jeffrey Horton, Sweah Liang Yong, Ah Chung Tsoi. Ranking Attack Graphs with Graph Neural Networks. The 5th Information Security Practice and Experience Conference (ISPEC 2009), Lecture Notes in Computer Science 5451, pp. 345 - 359, 2009.

2. Liang Lu, Jeffrey Horton, Rei Safavi-Naini and Willy Susilo. Transport Layer Identification of Skype Traffic. The International Conference on Information Networking (ICOIN 2007), Lecture Notes in Computer Science 5200, Springer-Verlag, pp. 465 - 281, 2008.

3. Liang Lu, Jeffrey Horton, Rei Safavi-Naini and Willy Susilo. An Adversary Aware and Intrusion Detection Aware Attack Model Ranking Scheme. The 5th International Conference on Applied Cryptography and Network Security (ACNS'07), Lecture Notes in Computer Science 4521, Springer-Verlag, pp. 65-86, 2007

4. Liang Lu, Rei Safavi-Naini, Jeff Horton and Willy Susilo. Comparing and Debugging Firewall Rule Tables. IET Information Security, Vol. 1 No. 4, pp. 143 - 151, 2007.

5. Liang Lu, Rei Safavi-Naini, Jeffrey Horton and Willy Susilo. On Securing RTP-Based Streaming Content With Firewalls. The 4th International Conference on Cryptology and Network Security (CANS2005), Lecture Notes in Computer Science 3810, Springer-Verlag, pp. 304 - 319, 2005.

6. Liang Lu, Rei Safavi-Naini and Willy Susilo. Design of Policy Tables For Implementation of Hybrid Distributed Firewalls. Australian Telecommunication Networks and Applications Conference (ATNAC) 2004, pp. 68 - 73, 2004.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Network Attack and Firewalls

Network attacks can take various forms from eavesdropping at physical layer, to session hijacking at transport layer, and to attacks at application level such as buffer overflow exploits. As computer systems are becoming larger and more complicated every day, the complexity of network attacks also increases. It is rarely the case that an attacker can launch a single attack and achieve their goal. Typically, the attacker needs to combine several techniques and execute several simple attacks to bypass a number of tools deployed to protect the system against unauthorised access [16].

In the battle against network attacks, firewalls have played one of the most important roles. A firewall is a security system consisting of a combination of hardware and/or software that limits the exposure of a protected network to attacks from outside [117]. It protects networks against malicious attacks by filtering out unwanted network traffic from entering the network. In the primitive form of firewalls, filtering decisions are made using a table of rules (or policies) that defines the forms of traffic allowed into the network [51]. In more advanced firewalls, filtering decision can also be based on application layer data or session metadata [117]. Figure 1.1 illustrates a typical deployment of a firewall that separates the protected Local Area Network (LAN) and the outside world, Wide Area Network (WAN), where any computing element can be potentially a threat to the LAN.

In addition to blocking malicious traffic to a protected network, firewalls can also be used to enforce organisational network access policy. In this case, it is used to filter out traffic that is considered of no value to the organisation. Examples include but are not limited to undesired contents such as pornographic web sites, instant messaging software, and online games.

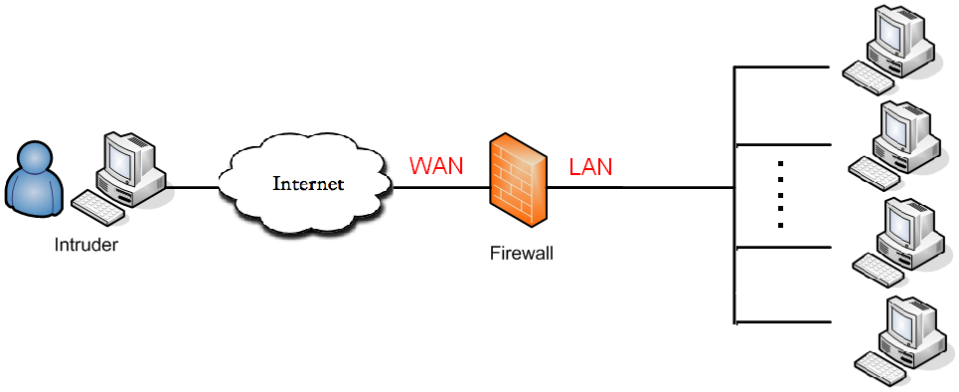

Figure 1.1: Illustration of that how a firewall works

1.2 The Challenge

Recently computer systems have become more complicated and hence difficult to model. This presents challenges to conventional network security tools such as firewalls that rely on a thorough understanding of network topology and architecture to define a strict security policy. As computer systems grow larger, there also exist potentially more security vulnerabilities that can be exploited by attackers to compromise the system or to gain unauthorised access. This further challenges conventional network security tools as well as people using these tools. A notable example is that conventional firewalls are unable to deal with newly emerged Internet applications such as streaming media. The rest of this section elaborates threats that conventional security tools are facing amid computer technology advancement.

1.2.1 Conventional Firewalls

In the face of new network attacks and threats, firewalls have gone through generations of evolution from primitive packet filters that only examine packets based on user-defined rules, to stateful firewalls that make decision based on packet series as well as individual packets, and to application layer firewalls that have specific and detailed knowledge of a set of Internet application and are able to perform an in-depth, complete scan for traffic generated by these applications.

Although a set of new technologies have been created to stop new forms of network attacks, firewall technology has not been able to catch up with advancement

of new Internet technologies. Situations have arisen where conventional firewalls are incapable of securing a network that employs new Internet technologies [53, 55].

A challenge to conventional firewalls is to handle the traffic generated by new types of Internet applications, such as applications delivering streaming content. These applications do not follow conventional Internet application models, and therefore present other challenges to conventional network security tools. For example, applications delivering streaming content tend to communicate on dynamically negotiated ports at the time of connection establishment. As a result, conventional firewalls need to open a wide range of ports to let through traffic generated by such applications [37, 67, 40]. Opening unused ports increases the chance that an attacker can penetrate through the firewall and this is unacceptable. A better solution is required, instead of simply opening the entire range of ports that streaming applications may randomly choose one from.

The behavior of randomly selecting a port to carry traffic also increases difficulties for conventional firewalls to enforce a network access policy. Conventional firewalls can no longer block the use of a streaming application, such as Skype, by blocking all ports that it may use, without interfering with other applications. Coupled with the fact that Skype, like many other streaming applications, has traffic encrypted, not even an application level proxy is capable of analysing packet payloads. This is not even to mention the fact that many streaming applications are built on proprietary protocols such that building an application level proxy to analyse its payload is not legal nor practical.

Conventional firewalls also have the problem of the correct implementation of organisation security policies. Translating a high-level security policy written in a natural language into firewall rule tables, a much lower-level description of the policy, is an error prone task. Moreover, correct translation of a high-level security policy into a firewall rule table is not unique. The "middle-level" policy languages [5, 120, 104, 39] designed to simplify the task of designing firewall rules also have similar deficiencies. In this thesis, we develop an algorithm to compare and analyse firewall rules at a low level to assist firewall users in design and analysis of their implementation of organisation security policies. The algorithm is based on a technique that we propose to uniquely represent any firewall rule table.

1.2.2 Modeling Multi-Stage Attacks

Having devoted much effort into enhancing conventional network security tools and techniques to deal with new threats and challenges, the research community has also placed a strong focus on new techniques to fight against new threats such as multi-stage attacks [41, 87, 98]. An example of such techniques is the use of a model checking mechanism [73] to discover all potential multi-stage attacks as well as their execution sequences within a modeled network [75], and also the use of attack graphs to intuitively present discovered attack and the execution sequences[95].

However, like any other new technologies, the use of attack graphs is limited by a number of factors such as sizing or scaling problems. In other words, generation and analysis of attack graphs is difficult because of the state explosion problem. In recent years, a fair amount of research has been conducted to resolve this issue with the main focus on two aspects: scalable attack graph generation [74, 99, 105], and improving attack graph visualisation by highlighting important portions of attack graphs [111, 85, 42, 59].

In this thesis, attention is given to the research area of improving the visualisation of attack graphs. In particular, the thesis looks into a scheme that highlights important portions of attack graphs by computing a rank for each node in an attack graph. Ranks are computed based on "importance" of a node. By ranking all the nodes, the focus of an attack graph is given to the nodes that have higher ranks.

The current ranking scheme of attack graphs is based on Google PageRank [89]. In Page Rank, the rank of a web page is defined as the probability that a web surfer hits the page eventually when starting from a random page. Similarly, the existing attack graph ranking scheme [111] computes the rank of an attack graph node based on the probability that an intruder reaches the node after exploiting a sequence of individual vulnerabilities. This scheme has not given much consideration to the dissimilarity between web surfing scenarios and computer system intrusion scenarios. This is one of the areas that this thesis contributes to by presenting a scheme that better models computer system intrusion scenarios.

PageRank is not an efficient algorithm [62]. It is mainly suitable to be used on a relatively stable structure such as the Web graph that will have little change over time. On the other hand, a network model may vary into many different forms. These can be a real improvement or change of design on the network, or can be a hypothetical alteration in order to observe the effect by any proposed change.

Because of PageRank inefficiency, it may be difficult to rank many attack graphs each resulting from one of many possible changes in the modeled network. This thesis presents an alternative scheme to estimate ranks of attack graphs based on the Graph Neural Network (GNN) [29], a new neural network model capable of processing graph structures. By learning from a certain number of training samples, GNN can learn the the attack graph ranking functions and quickly produces new results for any new input.

1.3 Our Contribution and Thesis Organisation

As mentioned earlier in the Chapter, new Internet applications have brought new threats to conventional firewalls. In this thesis, we examine the mechanism for firewalls to handle traffic generated by new Internet applications, and in particular, streaming applications.Our contributions to the area of defending against network attacks include

- A mechanism provided for conventional firewalls to handle malicious streaming content into a protected network is presented.
- A technique for a conventional firewall to filter out traffic generated by application whose existence is considered a violation of network usage policy.
- A formal approach to compare and analysis firewall tables.
- A ranking scheme of attack graphs to improve accuracy in ranking attack graphs.
- An artificial intelligence approach to ranking attack graphs to improve the efficiency of ranking attack graphs.

The rest of this thesis is organised as follows. Chapter 3 presents a vulnerability in streaming protocol that allows an attacker to inject malicious traffic into a data stream, and then presents a scheme that enables conventional firewalls to detect and filter out such malicious traffic. Chapter 4 looks at the other aspect of firewalls - dropping off traffic that is considered of no value to an organisation. A technique is provided for conventional firewall to detect undesired streaming content for an organisation. Chapter 5 provides a technique that lessens the difficulties system

administrators face to correctly implement an organisation policy. This is a rule table representing method that allows comparison of two rule tables. Chapter 6 looks into a technique, namely attack graphs, for modeling and fighting against more complicated, multi-stage network attacks. It presents an accurate ranking scheme for attack graphs as well as an efficient ranking scheme based on an artificial intelligence approach. Chapter 7 concludes the thesis.

Chapter 2

Preliminaries

Delivering streaming content has gained a strong focus in the research community as well as in industry in recent years. The majority of the work is focused on delivery of streaming content other than the security aspect of streaming content delivery. Historically the Internet has been used primarily for reliable data transmission with minimal or no delay constraints. The TCP transport protocol was designed to meet this purpose and has worked well in this context. However, streaming content usually requires timely delivery other than reliable transmission. This differs significantly than the requirement driving TCP and hence requires the use of a different protocol to provide the required transmission characteristics. TCP has been proven to be not suitable for delivery of streaming content due to a number of constraints. First of all, waiting for TCP retransmission can cause noticeable and undesirable delay in playback of streaming content, no matter whether it is audio, video, or something else. Moreover, the "slow start" TCP congestion-control mechanism can interfere with the audio and video "natural" playback rate. Because there is no fixed path for packets to flow across the Internet, no mechanism exists to ensure that the bandwidth needed for streaming content delivery is available between the sender and receiver, and as such quality of service cannot be guaranteed. Finally, TCP does not provide timing information. This is highly desired in streaming content playback [102, 7].

Streaming content applications usually do not need the complexity of TCP and, instead, require a relatively simple transport protocol. Many playback algorithms can tolerate missing data better than disruptions caused by retransmission. They do not require guaranteed in-sequence delivery either. The research community and the industry have developed a number of streaming protocols to support timely delivery of streaming content like audio, video and other types of media. On the application layer, there are Real-Time Streaming Protocol (RTSP), H.323 and Session

Initialisation Protocol (SIP). On the transport layer, the most widely used protocol for streaming content transmission is Real-time Transport protocol (RTP). These real-time oriented protocols are designed to be used in both multicast and unicast network services to provide timely delivery of datagrams. Many streaming applications provide better performance when using IP multicast. Therefore, the ability to use IP multicast has been taken into account when designing these protocols. Examples include multicast routing, scalability and adaptation to large numbers of receivers and heterogeneous receivers [102, 7].

2.1 Streaming Protocols

2.1.1 Real-Time Streaming Protocol

Real-Time Streaming Protocol (RTSP) [35] is an application level protocol which aims at offering a robust mechanism to deliver streaming content over multicast and unicast networks in "one to many" applications. It takes advantage of streaming content by breaking a large chunk of data into many packets sized according to the bandwidth available between client and server. When the client has "buffered" enough packets, it can start to play the buffered data while continuing to download the rest. The client is thus able to play the media content almost in real time without having to download the entire media file first. The source for the media content can either be live data or a stored media file.

RTSP does not typically deliver streaming content itself, although interleaving of a continuous media stream with a control stream is possible. In other words, RTSP acts as a "network remote control" for streaming content providers. RTSP provides an extensible framework, more than a streaming protocol, to enable controlled, on-demand delivery of streaming content. It is designed to provide control over multiple data delivery sessions. RTSP does not mandate the underlying transport protocol to use. It can be set to use UDP, multicast UDP, TCP, or RTP as the underlying data delivery channel.

RTSP does not have the notion of connection. A server maintains a session labeled by an identifier. An RTSP session is in no way tied to a transport level connection such as a TCP connection. During an RTSP session, an RTSP client may open and close many underlying transport connections to the server to send RTSP requests. The underlying transport protocol can be either connection oriented

or connectionless.

RTSP is intentionally designed to have similar syntax and operations to HTTP/1.1 so that extension mechanisms to HTTP can be seamlessly added to RTSP. RTSP differs only in a number of aspects from HTTP/1.1 as shown below. RTSP provides no security mechanism other than basic HTTP authentication.

- RTSP provides a number of new methods and has a different protocol identifier.
- An RTSP server needs to maintain state by default in almost all cases, as opposed to the stateless nature of HTTP.
- Both RTSP server and client can issue requests.
- Data is carried out-of-band by a chosen transport protocol, in most cases.
- RTSP is defined to use ISO 10646 (UTF-8) rather than ISO 8859-1.
- The Request-URI always contains the absolute URI. HTTP/1.1 carries only the absolute path in the request and puts the host name in a separate header field for backward compatability.

The following operations are supported by the RTSP protocol.

- Retrieval of media from media server, i.e. a client can request a presentation description via HTTP formatted URI.
- Invitation of a media server to a conference, i.e. a media server can be "invited" to join an existing conference, either to play back media into the presentation or to record all or a subset of the media in a presentation.
- Adding media to an existing presentation, particularly for live presentations. This is useful if server can notify clients when additional media becomes available.

2.1.2 Session Initiation Protocol

Session Initiation Protocol (SIP) [67] is another IETF defined application level protocol designed to deliver streaming content, however mainly voice data, over multicast or unicast network. Similar to RTSP, SIP does not mandate the use of a specific transport protocol. It is up to the negotiation process to select TCP, UDP or RTP.

SIP is not a transport but a signalling protocol used to establish sessions for VoIP applications. Functions and characteristics that SIP offers include:

- Adaptability. SIP is able to adjust seamlessly to various network infrastructures and to provide non-interrupted and reliable service.
- Scalability: The magnitude of serving capability can be easily increased and also designed in such a way as to provide full redundancy architecture.
- Text based programming: This simplifies the task of application support. Due to its simple presentation using plain text, SIP services can easily be implemented and diagnosed.
- Similarity with HTTP: SIP bears much similarity with HTTP in syntax, such as cause codes and message header syntax.
- End-devices control support: SIP carries most intelligence at the end devices. Session control is initialed from end devices. As a result, end devices participate in and have control over the session establishment and tear down setup.
- Accommodation for new services: SIP can seamlessly support new session control requirement by new services. SIP can transport almost any service requirements, such as the underlying transport other protocol to use.
- Mobility support: SIP can effectively support mobile and ported users and services. Its interaction with server farms supports mobile users by providing location-based services.

SIP has been proposed for quite a few years. However only until the last couple of years has SIP been adopted by the VoIP community as the dominating protocol for signaling [118]. A session could be a simple two-way IP telephony call or it could be a collaborative multi-media conference session. The following list summarises session forms that SIP supports.

- two-party sessions
- multi-party sessions
- multi-cast sessions

The ability to establish these sessions enables SIP to host innovative services such as voice-enabled e-commerce, Instant Voice Messaging (IVM) with selected buddies lists, and IP-Central services.

The design philosophy of SIP is to specify merely what needs to be specified and nothing more. SIP is purely developed as a mechanism to establish between VoIP applications. It does not know the details of the established session. It is only responsible for initiating, terminating and accepting modification to sessions. As such, SIP is scalable and extensible, and is able to work with various system architectures and deployment scenarios.

SIP is a request-response protocol that bears resemblance to many other Internet protocols, e.g. RTSP and HTTP. As a result, SIP sits comfortably alongside other web applications. Using SIP, telephony becomes another web application and integrates seamlessly into other Internet services. SIP is a simple protocol which service providers can use to deliver integrated voice and multimedia services. Parameters of services are negotiated and agreed by all participating parties during session establishment. SIP itself does not control the negotiation process - it merely is a negotiation protocol for the participating parties to agree on a common set of features and parameters.

SIP is clearly a vital protocol to VoIP and is increasingly widely deployed. It is a catalytic protocol that delivers key signaling elements. SIP can be used to deliver VoIP and multi-party conferencing services based on IP network, which is capable of delivering next generation integrated services. SIP is powerful, and yet simple. This is by the philosophy of "doing what it does best", and playing nicely with other protocols in the integrated protocol sandbox.

SIP shares many characteristics with RTSP. However SIP is designed for a different purpose than RTSP, which is designed to control the media stream during delivery. SIP is not directly involved in controlling media streams.

2.1.3 H.323

Another popular VoIP protocol is ITU-developed H.323 [40]. Unlike SIP which is a simple and self-contained protocol, H.323 is a protocol suite rather than a specific protocol. It contains a range of specific protocols for call setup, data transmission and capabilities negotiation. The H.323 model considers inter-networking between IP voice network and traditional PSTN network. It consists of a gateway which

is responsible for connecting PSTN networks into IP-based voice network. The gateway needs to understand both PSTN protocols and H.323 protocol suite. Also within an H.323 zone, a gatekeeper may exist to manage end-users registration and admission control within a given zone of operation. End users in this model are referred to as terminals.

The main protocols within the H.323 protocol suite are:

- RAS (Registration/Admission/Status): defined as H.225 (RAS) protocol in the standard. RAS messages are originated by the gateway at the moment a terminal initiates a call. These messages are transported using UDP to the gatekeeper.

- H.225: also known as H.225.0 - call control signaling. This protocol specifies the use and support of Q.931 signaling messages. This protocol is responsible for establishing and releasing connections.

- H.245: used after connection establishment between two gateways to negotiate parameters of a call such as codecs to be used, video or conference support, timer values, and etc. Moreover, IP logical channel ports are exchanged for RTP sessions and eventually transmission of data and/or voice traffic.

Although both are VoIP protocols, SIP and H.323 differ in many aspects.

- SIP is a self-contained simple protocol. It does nothing more than assist participating parties to negotiate and agree on call parameters. On the other hand, H.323 is a monolithic suite of protocols that cover everything including encoding/decoding, call control, conferencing, and many other functions in one vertically integrated stack.

- SIP is designed to work only with IP based VoIP network. On the other hand, H.323 was proposed at early days when IP based voice transmission had not gain its popularity. As a result, H.323 design also considers interaction and collaboration between IP based and conventional PSTN networks. The H.323 model consists of a gateway which is responsible for connecting PSTN networks into IP network. The gateway needs to understand both PSTN protocols (e.g. MTP3) and the H.323 protocol suite.

- SIP is a publicly open and free protocol developed by IETF, while H.323 is more or less proprietary by ITU.

- SIP is easy to extend to work with other protocols and applications. H.323 cannot easily support new services, especially those that require mobility and location-awareness. In general H.323 is not open to extensions.

H.323 and RTSP are complementary in function. H.323 is suitable for delivering streaming content in moderately sized peer-to-peer groups, whereas RTSP is suitable for setting up large-scale broadcast streaming applications.

2.1.4 Real Time Transport Protocol

RTSP, SIP and H.323 are all defined at application layer. They rely on transport (network) layer protocols to actually transmit streaming data, e.g. voice, video, multi-party conferencing or gaming data. TCP and UDP have been long established in the transport protocol family. TCP features connection oriented and reliable data delivery. It offers functions such as sequence number and loss recovery to ensure data is delivered reliably. However it has been proved that TCP is not suitable for the task of delivering streaming content [4]. This is because in the streaming content delivering scenario, minor data loss is more tolerable than transmission delay. Waiting for retransmission of one packet may introduce perceptible delay to the streaming session and is not desirable. UDP is generally used to deliver streaming data but lacks the ability to deliver critical control signals such as parameters negotiated between participating parties.

To combine the "streaming content favored" features of TCP and UDP, Real-Time Transport Protocol (RTP) [37] has been proposed and standardized to provide end-to-end delivery services for data with real-time characteristics, such as interactive audio and video data, which can be transmitted over multicast or unicast network services. RTP usually runs on top of UDP to deliver audio and video data to achieve timely delivery. It may also run on top of TCP to reliably transmit controlling signals. RTP may also be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network. RTP consists of two closely-linked parts: the real-time transport protocol (RTP) and the RTP control protocol (RTCP). RTP carries data that has real-time properties. RTCP is the controlling aspect of RTP and defines the controlling segments.

RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services, usually UDP,

to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence. However the sequence validation scheme within RTP has a security flaw and can be used to compromise system security. This will be presented in Chapter 3.

RTP itself has no built-in security mechanisms. To overcome this problem, Secure RTP [61] is developed to provide confidentiality, message authentication, and replay protection to the RTP traffic and to the control traffic for RTP, i.e. RTCP. SRTP provides a framework for encryption and message authentication of RTP and RTCP streams, defines a set of default cryptographic transforms, and allows new transforms to be introduced in the future. SRTP can achieve high throughput and low packet expansion. It proves to be a suitable protection scheme for RTP in heterogeneous environments, i.e. mix of wired and wireless networks. These features are based on an additive stream cipher for encryption, a keyed-hash based function for message authentication, and an "implicit" index for sequencing/synchronization based on the RTP sequence number for SRTP and an index number for Secure RTCP (SRTCP).

A number of RTP capturing and monitoring tools are publicly available in the research community. Among these are *rtpdump* [97] and *rtpmon* [18]. *rtpdump* parses on a specified address and port pair for RTP and RTCP packets, and generates report to output files. *rtpmon* monitors RTP sessions by viewing packet loss rate and jitter information presented in RTCP feedback packets.

Putting all the above streaming protocols together, we depict the typical protocol stack and data packet format of streaming applications in Figure 2.1. Note that although RTP may run over TCP and others may run over UDP, it is not a typical scenario which is beyond the scope of the thesis.

The industry has also invented various streaming protocols, mostly proprietary, for audio, video, or Internet telephony application. One of the most notable examples is Skype [100]. Details of such protocols are mostly unknown due to their proprietary nature. Although there has been work to reverse-engineer Skype [88, 106], it remains mostly a black box because of two reasons. Firstly, Skype and its protocol evolves over time, and secondly the reverse-engineering work is only empirical and lacks formal verification.

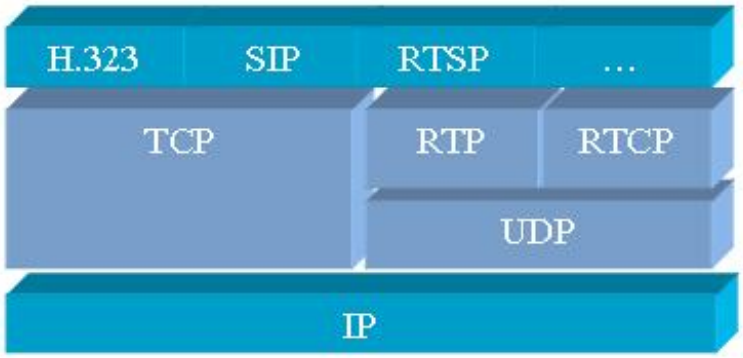

Figure 2.1: Typical Protocol Stack of Streaming Applications

2.2 Other Literature

Not much work has been done on securing delivery of streaming content. On the contrary, a fair amount of work has been conducted to bypass firewall checking in delivering streaming content, with Skype being the most famous (and infamous) example. Baset *et al.* [88] provided details on techniques that Skype uses to bypass firewall checking. This causes problems for system administrators in the case where Skype traffic is not desired. Techniques Skype uses to bypass firewall checking are summarized in the following list. Chapter 4 further details on how Skype bypasses conventional firewall checking.

- Skype is based on peer-to-peer technology. It transports data on a dynamically negotiated port with other participants. This can bypass conventional firewalls IP address and port checking.
- Skype encrypts its voice data and prevents content inspection by firewalls.
- Skype uses a proprietary protocol for call setup and encryption negotiation. It is difficult to analyze the protocol to work out port and encryption key it uses to bypass firewall inspection.

Little has been done in the area of enhancing conventional firewall technology to handle streaming content. Fung *et al.* [31, 50] proposed a transport-level proxy to secure multimedia streams. An extended SOCKS UDP binding model with appropriate socket calls is proposed to provide complete support for UDP-based, multimedia streaming applications. However, they mainly analyzed streaming application at application and UDP layer, without considering the in-between RTP layer. As a

result, packet injection and session hijacking are only prevented at the UDP layer, i.e. based on IP addresses and port numbers. Gupta *et al.* [80] performed an in-depth analysis of VoIP protocols. Potential security flaws in various streaming protocols are found vulnerable to replay attack and man-in-the-middle attack. The paper demonstrated how to cause the SRTP protocol to repeat the keystream used for datagram encryption. This enables the attacker to obtain the XOR of plaintext datagrams or even to completely decrypt them. It also demonstrated an attack on the SRTP protocol that allows an attacker to convince SRTP session participants that they have lost the shared secret. Additionally several minor weaknesses and potential vulnerabilities to denial of service in other streaming protocols are analyzed. It is worth noting that the paper only presents the potential attacks against streaming protocols without providing curing methods.

Much research has been done on identifying peer-to-peer traffic. These have the potential to be employed by conventional firewalls to detect violations such as using Skype. Sen *et al.* [96] presented a signature-based approach to identifying traffic generated by five popular P2P applications. They derived TCP signatures for each application mainly by examining packet-level traces and some available documents. Karagiannis *et al.* [108] extended this approach to nine P2P applications. Furthermore, they proposed two non-payload based heuristics to identify P2P applications, namely "TCP/UDP IP pairs" and "{IP, port} pairs". The "TCP/UDP IP pairs" heuristic identifies source-destination IP pairs that use both TCP and UDP transport protocols. It is based on the observation that most file-sharing P2P applications use UDP and TCP to transfer, respectively, query or query responses and actual data. The "{IP, port} pairs" heuristic utilizes the connection patterns of P2P applications. In P2P networks, a peer advertises its IP and port on the network so that other peers can connect. As a result, the number of distinct IPs connected to the advertised <IP, port> pair will be roughly equal to the number of distinct ports used to connect to it. It should be noted that the research effort mentioned above aims at detecting general peer-to-peer traffic, other than to detect and block streaming traffic based on peer-to-peer technology to by pass firewall checking.

There is also considerable research interest in detecting Skype traffic [1] [115] [52]. When either caller or callee or both are behind NATs or firewalls that prevent direction connection, Skype traffic must be relayed via a third node. This situation is referred to as Skype traffic relaying. Suh *et al.* [52] proposed to detect relaying of Skype traffic by an end-host in a monitored network based on the correlation

of packet sizes and bit rates between the incoming and outgoing (relayed) Skype traffic flows. They take the perspective of the operator of a large network, who is monitoring incoming and outgoing traffic at an access link. The goal is to determine whether Skype traffic is being relayed through an end-host belonging to the network. Their detection heuristic is based on the facts that 1) two flows carrying Skype-relayed traffic must have opposite directions (one entering, the other leaving the network), have the same end-host (same IP address) within the network being monitored, and have different end-hosts (different IP addresses) outside the monitored network; 2) Skype-relayed traffic is voice traffic and poses strict constraints on maximum delays and minimum bit rates; and 3) patterns of packet sizes and bit rates in Skype-relayed flows are well preserved during the relaying process by the relaying node. That being said, outgoing (relayed) Skype flow demonstrates similar patterns of packet sizes and bit rates to that of incoming Skype flow. Putting these together, two flows satisfying 1) with a relatively small delay in time are considered to be relaying Skype traffic if enough correlation can be found in their packet size and bit rate patterns.

In industry, although several commercial products [1] [115] claim the ability to detect or block Skype traffic, their details have not been made public and their source code is not available. Hence, we have not had the opportunity to evaluate the mechanisms used by these products.

2.3 Conclusion

In general, an important piece of work for securing streaming content delivery has been missing from open literature. Providing a mechanism for conventional firewalls to inspect streaming content will benefit system administrators in two aspects. First, firewalls can be used to block malicious content mixed in streaming content into a protected network. Second, firewalls can be used to block undesired traffic from entering or leaving a controlled network. Moreover, streaming content often is delivered using dynamic port numbers. Configuring firewall policies to enable streaming content is difficult. In the rest of the thesis, we will be looking at these specific problems.

Chapter 3

Preventing Malicious Streaming Traffic Into A Secured Network

3.1 Introduction

Internet streaming applications such as live video or audio broadcasting, IP telephony and teleconferencing have gained increasing popularity in recent years. Advancement of several technologies have combined to drive the growth of streaming applications. Examples of such technologies are personal computer performance improvement, increasingly available residential broadband access, and virtual-reality technology advancement. With further advancement of these technologies, growth of streaming applications is expected to continue.

Similar to other Internet applications, applications involved in processing streaming content may be vulnerable to network attacks. It is highly desirable to have a firewall that can filter out malicious content in streaming traffic. However, streaming applications behave fundamentally different than conventional Internet applications. Whereas conventional Internet applications are mostly based on Transmission Control Protocol (TCP), streaming applications largely rely on Real Time Transport Protocol (RTP) for content delivery. Typically, RTP uses User Datagram Protocol (UDP) as the underlying transport mechanism. TCP can be used too, however only for tunneling through firewalls at the cost of timely delivery. This behavior of streaming applications has created a new threat against conventional firewalls which are mainly designed to handle connection-oriented, TCP-based Internet applications. Due to connectionless nature of UDP protocol, conventional firewalls have technical difficulties in detecting malicious content injected into streaming applications.

The rest of this Chapter first explains the reasons why conventional firewalls have difficulties in filtering streaming applications. An analysis of the streaming protocol stack is then provided. The streaming protocol stack shows where reliable

information to distinguish good traffic from injected or malicious content can be obtained. This information is not utilised effectively in current systems and results in vulnerabilities which an attacker can exploit to "hijack" a streaming session. In this case, an attacker can replace content in a streaming session with arbitrary payload without the receiver detecting the replacement. Then a novel and effective inspection scheme is provided to enhance conventional firewalls to deal with such attacks. In the end, experimental results are presented as a proof of concept to demonstrate the effectiveness of the proposed traffic inspection scheme. Part of this Chapter appeared in [53].

3.2 Preliminaries

3.2.1 Streaming Content Overview

Before streaming applications gained popularity, people had to fully download a media file to local storage devices before they could begin to play the file. It could take from seconds to hours to download a media file depending on the file size. With streaming technology, media files can be played almost immediately after download starts or after a short buffering time. This is achieved by packetising a media file into a large number of much smaller portions. Each small portion then streams like little drops of water through the network pipe. Streaming technology also enables users to select a particular section of a media file such as a certain episode of a TV series. Communication with other peers in realtime is also made a reality with streaming technology. Major applications of streaming technology include IP telephony, teleconferencing, live video/audio broadcast and stock monitoring. The only disadvantage of streaming technology from a user's perspective is that received content cannot be archived or redistributed easily.

Streaming application protocols such as Real Time Steaming Protocol (RTSP) [35], Session Initiation Protocol (SIP) [67], and H.323 [40] are defined at the application layer. Typically, these protocols use TCP to reliably deliver session control messages such as setup, manipulate, and tear down commands. The data stream containing the streaming content itself is typically delivered using a UDP based protocol, because timeliness is typically more important to data delivery than reliability and waiting for retransmission of a lost frame would have worse impact on viewers' impression than dropping it.

However, as UDP itself does not provide sequence reconstruction, packets arriving out of order cannot be reconstructed to their original sequence. To solve this problem, IETF developed the *Real Time Transport Protocol (RTP)* [37] (formerly as RFC 1889 [36]), which was also published by ITU-T as *H.225.0*. Same as UDP and TCP, RTP is also defined at the transport layer. It provides protocol elements necessary for the delivery of streaming content such as

- Sequence number: a 16-bit number that increments by one for each RTP data packet sent for the receiver to detect packet loss or to restore packet sequence.
- Timestamp: a 32-bit number reflecting the sampling instant of the first octet in the RTP data packet.
- Payload type: a 7-bit number indicating RTP payload format and its interpretation by the application.

RTP is typically run on top of UDP which provides the end-to-end delivery for RTP data packets. *RTP Control Protocol (RTCP)* [37] can be optionally used to provide feedback on the quality of the data distribution.

3.2.2 Incapacity of Conventional Firewalls to Handle Streaming Content

The three types of conventional firewalls today are packet filtering firewalls, application gateways [117] and stateful firewalls [17]. As stated previously, conventional firewalls are built on technology fundamentals which have not considered the capability to work with streaming content applications.

Packet filtering firewall is the most basic, fundamental type of firewall. Access control is governed strictly by a statically predefined rule set mainly consisting of IP addresses and port numbers. Hence, it is unsuitable for inspection of streaming content, which sends/receives data via dynamically negotiated ports.

Application gateways are firewalls that provide filtering functions at the application layer in addition to access control at the network and transport layer. It has numerous advantages over packet filtering firewalls, such as user authentication, application logging, and content filtering. However, in order to scan application headers and payloads apart from IP and TCP headers, application gateways need to spend a fair bit of effort and time copying packets from system kernel to users

space. Hence, they are "are not generally well suited to high-bandwidth or real-time applications [46]".

Similar to packet filtering firewalls, stateful firewalls also enforce access control at the network and transport layer. They are more effective than packet filtering firewalls in that they enforce the notion of "stateful connection", i.e. incoming packets are passed through only if they belong to an established session. Compared with the above 2 types of firewalls, stateful firewalls have the following advantages in inspecting streaming content:

- They are able to open dynamic ports for packets that belong to an established session.
- Working at the transport and network layer, they are suitable for high-bandwidth and real-time applications.

However, stateful firewalls identify a session only by session-specific protocol fields such as IP address and port numbers, they cannot prevent injecting malicious packets with forged IP address and port number into streaming content. In the rest of this Chapter, we will look into injection of forged content into a streaming session by exploiting vulnerabilities in current traffic sanitizing schemes in streaming protocols, when data delivery is accomplished by UDP. We will also present an inspection scheme that can be used to filter out such injected content. The presented scheme relies on protocol fields defined at the transport layer of streaming content, and thus can be easily incorporated into stateful firewall kernels.

3.3 Injection of Malicious Traffic

TCP traffic contains more stateful information about an established connection than UDP traffic does. This includes TCP flags and sequence and acknowledgement numbers. Stateful firewalls thus can handle TCP traffic effectively. On the other hand, inspection of UDP traffic is usually based merely on IP addresses, port numbers, and a virtual timer [17]. These fields do not change over time in an established session and are easy to forge. Consequently, injection into a continuous UDP traffic sequence is straight forward if the IP addresses and port numbers are known by an attacker. Therefore, although possessing some advantages in scanning streaming content, stateful firewalls cannot prevent injection of packets when streaming applications use UDP for data delivery, which is often the case for timely delivery.

Although sequence numbers that change over time are also provided at the transport layer of RTP-based streaming applications, their behavior is very different to TCP sequence numbers. Most notably, sequence numbers in an RTP session may form a loose sequence with missed sequence numbers. Sequence numbers can be missed if a packet is dropped because UDP does not provide for retransmission of dropped packets. Packets with duplicate sequence numbers are not expected to be received before sequence numbers wrap around for the same reason.

Therefore, sequence numbers of RTP cannot be used in the same way in which TCP sequence numbers are employed in stateful firewalls. Schulzrinne *et al.* [37] proposed a sequence number validation scheme, where a sequence number is considered valid only if it is no more than $MAX\_DROPOUT$ ahead of the max sequence number ever received nor more than $MAX\_MISORDER$ behind, where $MAX\_DROPOUT$ and $MAX\_MISORDER$ are constants determined by an administrator. Additionally, a stream is considered restarted if two consecutive packets with neighboring sequence numbers, both considered invalid, are received. This scheme utilises RTP sequence numbers to a certain extent, but leaves a number of exploitable vulnerabilities unsolved:

- Packet Injection: Legal packets and injected packets are placed into a "fairly-competing" situation. A packet is considered legal as long as it hits the acceptable range of sequence numbers. Currently, the RFC for RTP defines a 16-bit sequence number, resulting in a good chance of randomly hitting this range. Moreover, this acceptable range may be "pushed" forward by an injected packet, resulting in successive legal packets being rejected for falling behind the acceptable range. Consequently, an attacker may take over the entire session if he can completely "push" the acceptable range off the legal stream. To increase the chance of a successful injection, an attacker can inject a sequence of packets whose sequence numbers increase by more than 1, or at a faster rate than a legal sequence.

- Session Restart: The validation scheme considers that a session is restarted if two consecutive packets with neighboring sequence numbers, both considered invalid, are received. An attacker may exploit this scheme to take over a session by injecting packets at a faster rate than a legal stream. As long as two consecutive injected packets arrive in the gap between two legal packets, the streaming session will be restarted, and consequently the next legal packet

may be rejected for falling out of the acceptable range set by the injected packets.

- Use of Magic Numbers: Use of $MAX\_DROPOUT$ and $MAX\_MISORDER$ is a trade-off between fault-tolerance and effectiveness of sequence validation. Administrators may have difficulties in selecting appropriate values of such numbers as their effectiveness is connection dependent. For instance, by assuming a maximum misordering time of 2 seconds at 50 packets/second and a maximum dropout of 1 minute, H. Schulzrinne el al [37] set $MAX\_DROPOUT$ and $MAX\_MISORDER$ to 3000 and 100 respectively. That being said, any injected packet with a random sequence number has about 4.7% (3100/65535) success rate. We can see that although setting a maximum dropout of 1 minute allows a certain degree of fault-tolerance, it also increases the success rate of random packet injection to a non-trivial level.

3.4 Streaming Content Modelling and the Inspection Scheme

In this Section, we provide an inspection scheme against vulnerabilities presented above. Our inspection scheme is based on modelling the arrival process of streaming content. Packet arrival process in the past was often assumed to be Poisson process because such process has attractive theoretical properties [110]. However, some recent research has pointed out failures of Poisson Process in modelling packet arrival process of both wide-area and local-area network traffic [82, 33, 34, 77, 112]. Hence, we have not assumed arrival process of RTP-based streaming traffic to be Poisson process. Our inspection scheme uses the *Central Limit Theorem* [47], which does not require that the population follow a particular statistical distribution.

3.4.1 Arrival Process Modelling of Streaming Content

A streaming session essentially delivers a large number of packets of media content from a server to a receiver. Packets arrive sequentially at the receiver with non-overlapping time intervals. Let P denote the set of all packets in a particular streaming sequence, and p_i denote the ith packet that arrives in sequence, then P can be represented by an ordered set of sequential packets $\{p_0, p_1, \ldots, p_n\}$, where

p_{i+1} arrives after p_i. Each packet p_i essentially consists of a 4-tuple: $\{seq, time, ordered, rate\}$, meanings of which are explain in the following.

$p.seq$ denotes the sequence number, and $p.time$ denotes the arrival time (wall clock time) of a packet p. p is accepted if $p.seq$ falls in the acceptable range at $p.time$.

Packets may arrive out of order. We represent whether a packet p arrives out of order with $p.ordered$, which is formally defined as follows,

$$p.ordered = \begin{cases} false, & \text{if } p.seq < MAX\_SEQ \\ true, & \text{if } p.seq > MAX\_SEQ \\ null, & \text{if } p.seq = MAX\_SEQ \end{cases}$$

where MAX_SEQ denotes the maximum valid sequence number previously received. In practice, sequence numbers can wrap around. Hence, when comparing $p.seq$ and MAX_SEQ, we may need to add a multiple of 65536 to $p.seq$. Unless otherwise stated, we will refer to $p.seq$ as $p.seq + k * 65535$ in the following discussion, where k denotes the number of times that a sequence has wrapped around.

$p.rate$ denotes a packet p's arrival rate which is defined as the ratio between sequence number increment and arrival time difference from the last ordered packet. Formally, assume we have a sequence of packets $P = \{p_0, p_1, \ldots, p_n\}$, then $\forall p_i \in P$ where $i > 0$, we define the arrival rate of a packet p_i as follows,

$$p_i.rate = \begin{cases} \frac{p_i.seq - p_j.seq}{p_i.time - p_j.time}, & \text{where } p_j.seq = MAX\_SEQ \ (p_i.ordered = true) \\ \frac{p_i.seq - p_j.seq}{p_i.time - p_j.time}, & \text{where } \forall k, 0 \leq k \leq i-1, \text{ and} \\ & 0 < p_i.seq - p_j.seq < p_i.seq - p_k.seq, \\ & \text{that is, } p_j.seq \text{ is nearest to } p_i.seq \text{ in any sequence} \\ & \text{numbers less than } p_i.seq \\ & (p_i.ordered = false) \end{cases}$$

Confining packet arrival rate to a reasonable range is essential to filtering out injected malicious traffic from a legal stream. This prohibits an attacker to increase the chance of success by sending a stream whose sequence numbers increase faster than those of the legal stream, as we discussed in section 3.3.

Due to characteristics of streaming content, the following restrictions apply to P:

- $\forall i, j$, if $i \neq j$, then $p_i.seq \neq p_j.seq$. This is because streaming applications do not resend lost packets.
- $\forall i, j$, if $i > j$, then $p_i.time > p_j.time$
- $\forall i$, if $p_i.ordered = true$, then $\forall j$ where $0 \leq j < i$, $p_j.seq < p_i.seq$
- $\forall i$, if $p_i.ordered = false$, then $\exists j$ where $0 \leq j < i$, $p_j.seq > p_i.seq$
- $\forall i$, $p_i.rate > 0$

3.4.2 Application of The Central Limit Theorem

Before the detailed discussion on our inspection scheme, we provide a review of the relevant statistical background.

Assume we have a sample $S = \{s_1, s_2, \ldots, s_n\}$ gathered from a population P, regardless of what statistical distribution P follows. Let $\bar{S}$ and $\bar{P}$ denote the average of S and P respectively, and σ_P denote the standard deviation of P. The *Central Limit Theorem* tells us that distribution of

$$\frac{\bar{S} - \bar{P}}{\sigma_P / \sqrt{n}} \tag{3.1}$$

is increasingly well approximated by the normal distribution $N(0, 1)$ as the sample size, n, gets larger. Since we have the statistical table for this normal distribution, we can figure out the probability that $\frac{\bar{S}-\bar{P}}{\sigma_P/\sqrt{n}}$ lies in a particular interval (a, b) using the equation

$$\lim_{n\to\infty}[a \leq \frac{\bar{S} - \bar{P}}{\sigma_P / \sqrt{n}} \leq b] = \alpha \tag{3.2}$$

where $\alpha = \int_a^b \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du$. This means that, with the specified probability α, we believe that

$$a \leq \frac{\bar{S} - \bar{P}}{\sigma_P / \sqrt{n}} \leq b \tag{3.3}$$

which can be transformed to

$$\bar{S} - b * \frac{\sigma_P}{\sqrt{n}} \leq \bar{P} \leq \bar{S} - a * \frac{\sigma_P}{\sqrt{n}} \tag{3.4}$$

This enables us to estimate the range of $\bar{P}$ based on the value of $\bar{S}$ with a certain confidence level α. The meaning of α can be explained intuitively as follows: if, for example, we set $\alpha = 0.95 = 95\%$, then the possibility that any $\bar{S}'$ lies in the estimated interval of $\bar{P}$ is 95% on average.

3.4.3 Inspection Scheme

Typically, packetised streaming content arrives at a relatively constant rate. Change or fluctuation of arrival rate usually takes place gradually over a non-trivial period of time [76]. Even though there exists on-demand media encoding techniques such as *Variable Bitrate (VBR)*, they are more applicable to the application layer than the transport layer. Moreover, there exist smoothing techniques which can smooth and reduce imposed burstiness [86].

For this reason, we can assume that, in a particular streaming session, arrival rates of successive packets follow the same distribution as preceding packets. An empirical justification of this assumption was provided by media traffic captured using *mmdump* [44]. By the *Central Limit Theorem*, we can estimate the interval in which arrival rates of successive packets will lie based on arrival rates of preceding packets. Assuming that we have a sample of legal packets $P = \{p_0, p_1, \ldots, p_n\}$ collected at the beginning of a streaming session, we are then able to estimate the acceptable range of arrival rate for packets in this stream using equation (3.4), where a and b are specified by equation (3.2) given a certain confidence level α. Successive packets are accepted if their arrival rates lie in the corresponding estimated interval, or otherwise dropped.

The algorithm that enforces our inspection scheme is presented in algorithm 1. Because of the fact that arrival rate may change gradually in the long run, we estimate the acceptable interval of arrival rate for packet p_{n+1} from the arrival rates of the most recent n packets preceding p_{n+1}.

Validating RTP-based streaming packets on the basis of their arrival rates has the following advantages over the validation scheme based on sequence numbers [37]:

- It is more difficult to forge a packet's arrival rate than its sequence number. Unlike real fields such as IP addresses, port numbers and sequence numbers, packet arrival rates are dynamically computed using non-static sequence numbers and packet arrival times. To successfully inject a packet, an attacker must be able to choose a suitable sequence number and launch the injection at the right time for the selected sequence number. More subtly, even though an attacker may be able to launch an injection at a precisely specified time, he still has difficulties in foreseeing the arrival time which is governed by the *Round Trip Time (RTT)* between the receiver and himself.

Algorithm 1: Validate($p_0 - p_n$, p_{n+1})

```
/* The function infers the validity of a packet p_{n+1} based on n
   preceding packets p_0 - p_n that have been assumed or verified as
   valid.                                                          */
/* p_{n+1}: the packet to be validated.                            */
/* p_0, ..., p_n: n+1 packets that precede p_{n+1} and have been
   assumed or verified as valid                                    */
/* r_1, ..., r_{n+1}: arrival rates of packet p_1, ..., p_{n+1}    */

begin
    /* the confidence level associated with the estimation        */
    const CONFIDENCE_LEVEL;

    /* set up sample parameters                                   */
    for l = 1 to n+1 do
        r_l = p_l.rate

    set r̄ = (Σ_{l=1}^{n} r_l) / n
    set σ_r = sqrt( Σ_{l=1}^{n} (r_l - r̄)^2 / (n-1) )
    set r_max = r̄ + CONFIDENCE_LEVEL * σ_r / sqrt(n)
    set r_max = r̄ - CONFIDENCE_LEVEL * σ_r / sqrt(n)

    /* set up parameters of the validated packet                  */
    set r̄' = (Σ_{l=1}^{n} r_l) / (n+1)

    if r_min ≤ r̄' and r̄' ≤ r_max then
        accept p_{n+1} ;
    else
        drop p_{n+1} ;
```

- Administrators are relieved from the task of selecting magic numbers such as $MAX\_DROPOUT$ and $MAX\_MISORDER$. This is a difficult task as the effectiveness of $MAX\_DROPOUT$ and $MAX\_MISORDER$ is connection-dependent. For example, smaller $MAX\_DROPOUT$ and $MAX\_MISORDER$ would be expected in a stable connection. In our scheme, we just need to select the confidence level, the value of which determines the effectiveness of the inspection scheme. This is achieved by the underlying statistical model.

- Packet arrival rate is a relatively stable measurement. Packets arriving after a network disruption period, regardless of how long it is, are expected to present

arrival rates similar with previously received packets, and be accepted. Hence, fault-tolerance ability is increased without trading off effectiveness, as opposed to the use of $MAX\_DROPOUT$ and $MAX\_MISORDER$ which represents a trade-off between fault-tolerance and effectiveness.

3.5 Experiments and Results

3.5.1 Experimental Setup

To send and receive RTP-based streaming content, we employed the *Java Media Framework (JMF)* [103] which enables audio, video and other time-based media to be added to applications built on Java technology.

For packet injection, we employed *Nemesis* [45] which is a command-line network packet crafting and injection utility. We modified its source code and enabled it to craft RTP packets with some configurable RTP parameters as follows,

- Sequence Hop (-h): Sequence hop between a pair of crafted RTP packets. For example, we can send a sequence of crafted packets with sequence number $\{1, 2, 3, ...\}$, or $\{100, 200, 300, ...\}$, where h = 1 or h = 100 respectively.
- Injection Interval (-i): The interval between which two RTP packets are crafted and injected, measured in milliseconds.
- Packet Count (-c): Number of crafted packets to be injected.

Setup of our experiments is depicted in Figure 3.1, where arrival rates of crafted RTP packets vary in the following 3 categories,

- Fast: Sending crafted RTP packets without specifying an injection interval, which means no substantial interval between crafting and sending two packets. Arrival rate of an injected sequence is around hundreds of thousands of packets per second.
- Medium: Sending crafted RTP packets with a substantial interval as specified by the "-i" parameter. We are then able to inject a sequence of packets with an arrival rate similar to the legal stream, typically around 30-60 packets per second, i.e. i $\approx$ 16-33.

- Slow: Sending only one crafted RTP packet each time when *Nemesis* is invoked. Number of injected packets in this case is governed by a for-loop script which repeatedly creates a new process to invoke *Nemesis*. The "-c" parameter (packet count) is implicitly set to be 1, and need not be specified. Typically, arrival rate is around 5 packets per second, as process creation and invocation takes quite a bit time.

3.5.2 Result on Packet Injection

We send a clip of music using *JMF* representing the legal sequence, and inject using *Nemesis* a clip of human conversation representing the illegal sequence. The illegal sequence is a live dump of the clip being replayed normally. *Nemesis* is used to read packets from the live dump and inject the packets by controlled parameter described earlier in the section.

When we use fast injection with sequence hop (the "-h" parameter) 1, we are always able to hijack the legal session and replace its content with the selected conversation. This is caused by the restarting mechanism as we analysed in section 3.3. As we can see from Figure 3.1, any two consecutive crafted packets that arrive between a pair of legal packets can take over the streaming session and "push away" the acceptable range of sequence numbers so that packets belonging to the legal stream are not accepted.

When we use slow or medium injection with a large sequence hop (e.g. 500), we can inject some noise being voice fragment into a legal stream, as sequence numbers of the injected stream travel faster and can "catch up" the acceptable range. However, we have not been able to hijack an entire session. Although an injected packet that accidently falls in the acceptable range of sequence numbers can push it away, any two consecutive legal packets that arrive between a pair of injected packets can take over the streaming session and "pull back" the acceptable range of sequence numbers so that packets belonging to the injected stream are dropped. On the contrary, two injected packets that arrive consecutively at the receiver cannot "pull back" the acceptable range since their sequence numbers are not consecutive.

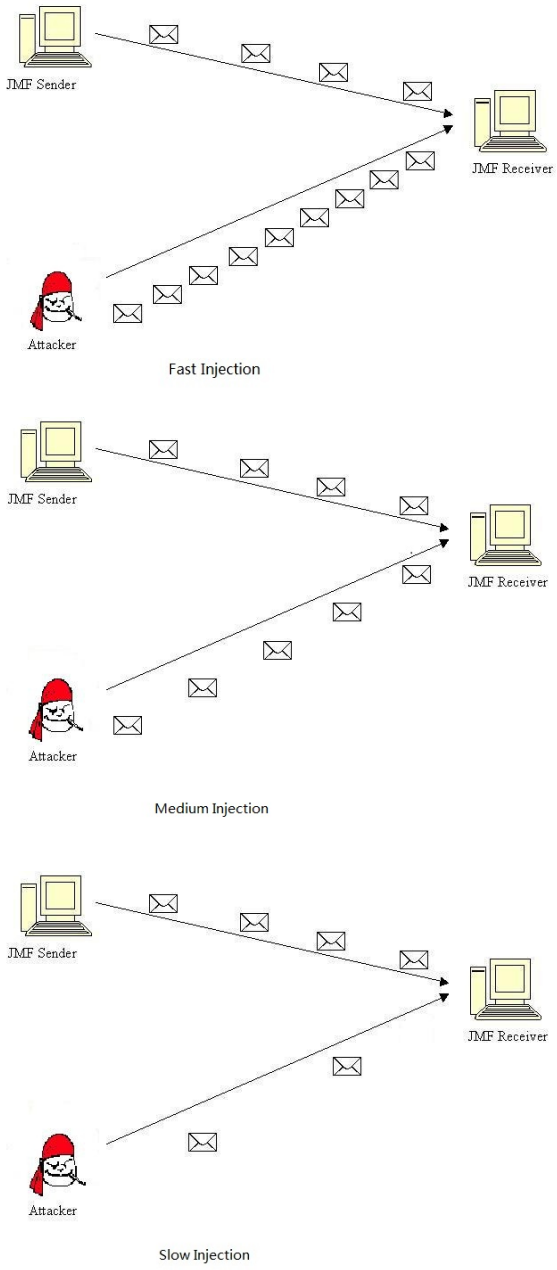

Figure 3.1: Experimental Setup

3.5.3 Effectiveness of The Inspection Scheme

With a statistical approach, it is always possible that legal packets are dropped or injected packets are accepted. We refer to these cases as false-positive and false-negative respectively. A low false-positive rate guarantees system usability, while a low false-negative rate assures legitimacy of received streaming content. To measure effectiveness of our inspection scheme by false-positive and false-negative rates, we performed an offline analysis on RTP-based streaming content from various sources, including web radio stations (lexp web radio[1] and webtalk radio[2]), Microsoft *MSN* live conversation, and *JMF*-generated streaming content.

Result On Passing Through Legal Streaming Content

We first performed an experiment on normal streaming content, i.e. streaming content that is delivered without injection. We have included experiments during which the network cable was disconnected for 10 to 61 seconds to simulate network disruption. Similarly, we have included silent periods in *MSN* conversations to simulate burstiness in VoIP.

The experimental result is summarised in Table 3.1.

	lexp radio	webtalk radio	JMF	MSN
bit rate	128kbps	28kbps	not specified	not specified
number of packets	47135	36471	54023	55738
number of bytes	64.9M	28.9M	11.9M	42.8M
duration	4021.1s	6819.2s	1832.6s	1474.6s
false-positive	0.004%	0.0%	0.06%	0.2%
false-negative	N/A	N/A	N/A	N/A

Table 3.1: Experimental result under confidence level $\alpha = 0.98$

Figure 3.2 demonstrates variation of packet arrival rates in the above experiments, as well as that of the estimated acceptable range. It illustrates that a very small portion of legal packets are dropped because of dramatic fluctuations in their arrival rates. Apart from that, arrival rates of most packets belonging to a particular stream do not fluctuate greatly.

[1] www.lexp.org
[2] http://www.webtalkradio.com/

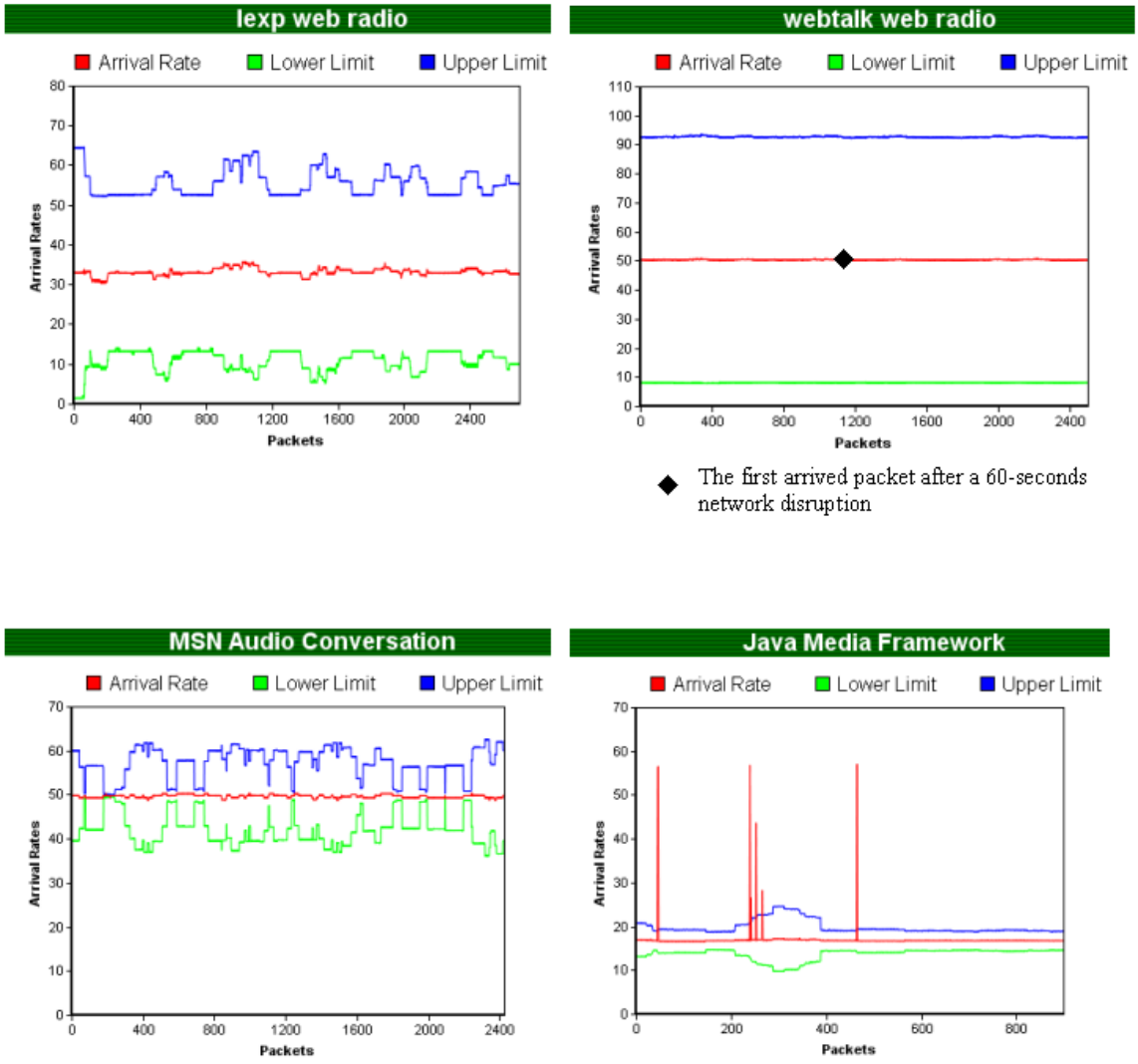

Figure 3.2: Variation of Packet Arrival Rates: Legal Traffic

Result On Filtering Out Injected Streaming Content

Our experimental environment is separated from the outside world by a firewall not under our control that prohibits incoming UDP traffic, so that we have not had the opportunity to test the inspection scheme with injection against streaming content received from a commercial streaming server on Internet, such as the lexp or webtalk server. Instead, we performed experiments on filtering out injected packets with streaming content generated by *JMF* which is open source and free to use. The legal dataset used in the experiment is part of the JMF dataset as used in Table 3.1. Only part of the dataset is used because the injected clip is shorter than the legal clip.

We performed experiments for all 3 categories of packet injection methods as discussed in section 3.5.1. The results are summarised in Table 3.2 to Table 3.4. Note that the situation that h=1 does not apply to Table 3.3 and Table 3.4 which represent slow and medium speed injection. For all types of injection, big hops (100 and 500 respectively) are selected to ensure injected sequence have a even distribution into the legal stream sequence where each packet hops by 1. The overall packet rates range from 40 to 70 packets per second which is larger than normal streaming rates ranging from 30 to 60 packets per second because injected packets are mixed in.

fast injection	-c=10000, -h=1	-c=10000, -h=100
number of packets	22761	24746
number of bytes	17.0M	18.7M
duration	329.2s	405.7s
false-positive	0.067%	0.17%
false-negative	0.0%	0.0%

Table 3.2: Experimental results with confidence level $\alpha = 0.98$ for fast injection

medium speed injection	-c=1000, -h=500
number of packets	28085
number of bytes	20.9M
duration	461.1s
false-positive	0.033%
false-negative	0.0%

Table 3.3: Experimental results with confidence level $\alpha = 0.98$ for medium speed injection. Interestingly, this is the case when crafted traffic is injected at a similar rate to that of legal traffic.

slow injection	-c=10000, -h=100
number of packets	29986
number of bytes	22.9M
duration	738.0s
false-positive	0.012%
false-negative	0.0%

Table 3.4: Experimental results with confidence level $\alpha = 0.98$ for slow injection

Figure 3.3 demonstrates variation of packet arrival rates of legal and injected packets. We can see that the arrival rates of injected packets significantly deviate from those of legal packets.

The experiment shows a remarkable result against packet injection and streaming session hijacking. The results of all three configurations (slow, medium and fast injections) consistently demonstrate 0% false-negative rate indicating that all injected packets are successfully identified. That is, variation in attack traffic rate (controlled by the parameter h) has no adverse impact on the proposed filtering scheme, the effectiveness of which is based on the fact that sequence numbers of injected packets have to increase significantly faster than those of legal packets in order to "catch up" a legal stream.

The results of all three configurations also consistently demonstrate a low false-positive rate. That is, in all configurations a small portion of legal packets are misidentified as injected traffic This is because by accident arrival rate of these packets significantly deviated from that of other packets in the legal stream. However due to the redundant nature of streaming content, dropping a small portion of packets is usually tolerable.

3.6 Conclusion

This Chapter first provides an in-depth analysis of the typical protocol stack for delivering streaming content, followed by discussion on deficiencies in conventional firewalls that lead to incapability of securing streaming content. After that, an inspection technique that makes use of sequence numbers at the transport layer of streaming applications is presented. We believe that, when armed with this technique, conventional stateful firewalls will be able to handle streaming content more effectively and securely. In the end, this Chapter presents substantial amount of experiments carried out to demonstrate the effectiveness of the proposed technique.

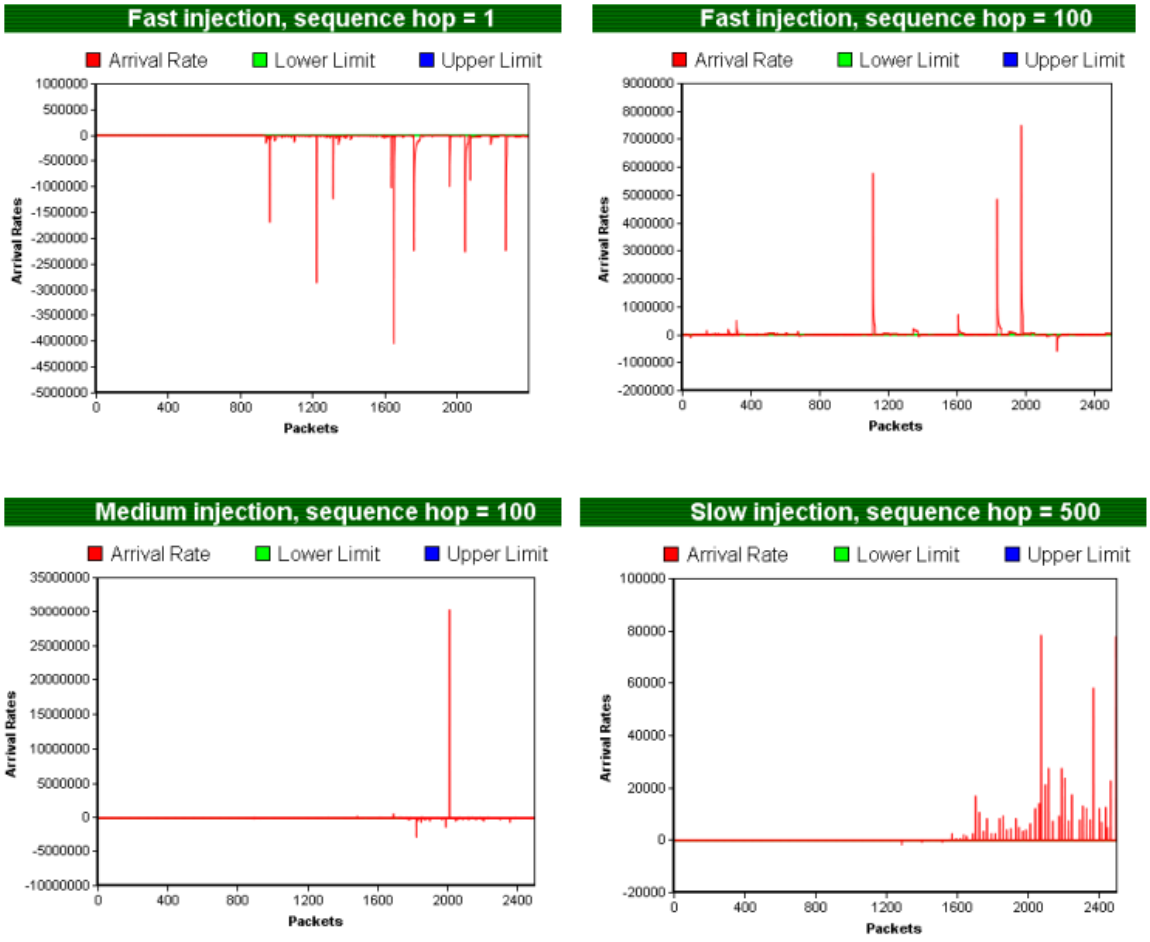

Figure 3.3: Variation of Packet Arrival Rates: Legal and Injected Traffic

A piece of practical follow-on would be to investigate how variation of the controlled parameters affect the effectiveness of the inspection scheme. More experiments are to be conducted with different parameters to allow analysis on connection between controlled parameters and effective of the inspection scheme.

Blocking malicious content off a secured border is only one aspect of firewall functionalities. The other aspect is to prevent unwanted traffic type from leaving the border. This may be due to legal, privacy, or performance issues. Capability of conventional firewalls in this aspect is also challenged by emerging streaming content technologies. In the next Chapter, we will look into this area and present our contribution.

Chapter 4

Preventing Streaming Traffic From Flowing Out Of A Secured Network

In addition to blocking unwanted content from entering secured network, firewalls are also often used to prevent certain types of traffic from leaving a network. There may be performance, legal or privacy issues that traffic must be confined within a private network. This Chapter looks into challenges that conventional firewalls are facing to fulfil this requirement, with focus on capability to filter outbound traffic generated by emerging streaming applications. In particular, we look in-depth into Skype, an Internet telephony application that has gained its reputation for capability to tunnel through firewalls. We examine the technical details that Skype relies on to bypass firewall inspection, and provide a counter-measure for firewalls to effectively detect Skype traffic.

4.1 Introduction

Recent years have seen increasing application and deployment of Voice over IP (VoIP) - the transmission of voice over traditional packet-switched IP networks. Compared to conventional Public Switched Telephone Network (PSTN) service, VoIP offers lower cost, greater flexibility and easier integration with value-added services such as call monitoring and auditing if traffic is not encrypted.

Among VoIP applications, Skype has gained a reputation for ease of use, superior sound quality and secure communication. Skype offers seamless penetration through Network Address Translations (NAT) devices and firewalls. It also provides better voice quality than other major free-to-use VoIP applications such as I-Seek-You (ICQ), Yahoo AIM and Microsoft MSN [8]. Moreover, Skype voice calls are encrypted "end-to-end" to offer protection against call tapping. Finally, being built on a decentralised overlaying peer-to-peer network allows Skype calls to be related

through an intermediate node in the situation when both end nodes are behinds NAT devices or firewalls which prevent direct connections. Skype hence claims to offer a higher call completion rate than any other VoIP applications [8].

However, there may be circumstances where the ability of Skype to penetrate firewalls or encrypt calls is not desirable. For example, phone calls are of a personal nature but not business-related. In this case it may be desired to prevent if possible this abuse of network access. Also, phone calls to certain organisations such as government agencies or financial institutions may need to be recorded for training or tracking purposes, and therefore must not be encrypted. Additionally, from personal computer users' point of view, Skype running on a personal computer may route calls for other Skype users silently in the background. In all the above situations, the capability of Skype to intelligently tunnel through firewall and to encrypt calls increases the difficulty to enforce access policy in a managed network. The ability to detect and/or block Skype traffic is therefore highly desirable in order to simplified the task of network management.

Unfortunately, conventional firewalls are not suitable for the tasks of detecting or blocking Skype traffic. Conventional network layer firewalls are designed for traditional Internet applications, most of which run on pre-defined, well-known ports. The ability to tunnel through firewalls using random ports increases the difficulty of detecting or blocking Skype traffic at the network layer without blocking some other required applications. For example, blocking ports greater than 1024 to incoming data packets in order to block incoming Skype calls also throttles some other streaming protocols [9] [71] that may be required by an organisation. Application layer firewalls do not offer much help either as Skype traffic is encrypted. The above factors have combined to make detecting or blocking Skype a difficult task for firewalls or intrusion detection systems. To the best of our knowledge, there has not been a systematic approach described in an open literature to detect Skype traffic from other use of streaming content such as real-audio or gaming traffic.

Our research focuses on detecting the use of Skype for phone call purposes in a managed network. In other words, our objective is to identify end hosts that participate in phone calls using Skype. We study both payload and non-payload based schemes to identify Skype traffic. We start with an empirical, payload based scheme relying on signatures collected from captured Skype call traces. Hosts are considered to have participated in Skype calls if they generate traffic that contains such signatures. After that, we looked into a non-payload based scheme which is

based on realtime characteristics and connection pattern of Skype traffic. Realtime characteristics that are used to determine if traffic is generated by Skype include packet size pattern, byte rate pattern, and packet inter-arrival time pattern. Accuracy and effectiveness of the non-payload based scheme is evaluated by comparing results from running the payload based approach against the same sample traces. The experimental results reveal that, at least to a certain extent, the effort by Skype to avoid content analysis using encryption and randomised ports can be overcome.

It is not always practical to be able to collect signatures from traffic traces. There may be legal or ethical constraints on monitoring and analysing traffic dump. Moreover, Skype is built on a proprietary protocol which has not been made publicly available. Signatures of Skype traffic are only empirically derived from our observations on traffic generated by a particular version of Skype. The signatures may change and become more elusive as Skype evolves. These have combined to make payload based identification a fragile detection scheme that is not resistant against change of protocols and software upgrade. Our research therefore focuses on the non-payload based scheme for Skype identification, and uses the payload based scheme mainly to verify its non-payload based counterpart. Our results demonstrate that the non-payload based identification scheme is more resistant to Skype version upgrade than the payload based scheme.

Skype tends to use UDP for data transfer as much as it can [88]. However, there are circumstances where Skype has to fall back to TCP. We leave this situation as an open research subject.

4.1.1 Skype Overview

Skype is a proprietary but free peer-to-peer VoIP application developed by N. Zennstre and J. Friis, the creators of KaZaA. It allows users to participate in voice calls, send text messages and files to other users. Skype has the reputation for working seamlessly across firewalls and NATs, and providing better voice quality than Microsoft MSN and Yahoo AIM [88]. Skype provides data confidentiality using end-to-end 256-bit AES encryption. Symmetric AES keys are negotiated using 1024 bit RSA [101].

Before a Skype user can participate in phone calls with other Skype users, he or she must first join the Skype peer-to-peer network. This process is referred to as the *login* stage. It is during this process that Skype determines the type of NAT and

firewall it is behind, authenticates its user name and password with the login server, and advertises its presence to other peers and its buddies [88]. After login, users can make calls to other Skype clients through *call establishment* stage. During this stage a connection between caller and callee is created and a session key is generated if one does not exist already [106]. Voice media is packetised and transferred between participating peers after that. Baset *et al.* [88] observed that call establishment was always signalled by TCP, while voice media was transferred over UDP as much as possible. In the following discussions, we refer to traffic regarding login, connectivity establishment, and call setup/teardown as *signalling messages*, and packetised voice data as *voice traffic*.

Skype has attracted many users because of its advantages, such as seamless NATs and firewalls traversal, superior voice quality, and higher call completion rate. One year after Skype being launched, it had more than 9.5 million users, with more than 1.5 million connections per day and 500,000 people connected at any time [90].

4.1.2 Related work

Although interesting and related, Suh *et al.*'s detection heuristic cannot be applied in the situation discussed in this Chapter [52]. On one hand, Skype flows entering and leaving the network in which the end-host participating in a Skype phone call resides have the same destination (IP address) outside the network. This is different to the situation with Skype-relayed flows. Moreover, patterns of packet sizes and bit rates of the incoming and outgoing flow may be considerably dissimilar due to transmission delays on Internet (see Section 4.3). These issues combined to invalidate Suh *et al.*'s detection heuristic. On the other hand, more metrics are available in our situation for identification of Skype traffic. For example, the peer-to-peer connection pattern of Skype can be used to distinguish its traffic from non peer-to-peer softwares and other VoIP applications which are mostly based on standard protocols such as SIP or H.323. In addition, as a software providing realtime communication, Skype emits packets at a considerably smaller time interval than most other applications. This pattern is typically expected to be well preserved in local network, and can be used to distinguish Skype from other applications, including most peer-to-peer file sharing applications, which are not operating under strict time constraints. Applications that can be distinguished from Skype by these metrics, which are not available to relaying nodes, represent a significant portion of false-positives in Suh *et al.*'s

heuristic in detecting Skype-relayed traffic.

Branch *et al.* provided a machine learning technology to identify Skype traffic based on learning from realtime Skype traffic characteristics, including packet length and packet inter-arrival time [76]. However the peer-to-peer nature of Skype has not been considered. As a result, possibilities are that other streaming application may demonstrate similar realtime characteristics with Skype and are incorrectly identified as Skype traffic.

In the following of this Chapter, Section 4.2 introduces our payload based detection techniques. Section 4.3 analyses the realtime characteristics of Skype voice traffic. Section 4.4 presents our non-payload based detection techniques of Skype traffic, and Section 4.5 provides experimental results. A conclusion is provided in Section 4.6. Part of this Chapter appeared in [55].

4.2 Payload Based Detection

We now describe our Skype traffic detection techniques. This section focuses on payload based techniques, and the section following presents non-payload based techniques. In the rest of this Chapter, all observations and experiments are performed for Skype version 1.3, unless otherwise stated.

Our payload-based identification of Skype traffic is based on characteristic signatures, i.e. bit strings, observed in packet payload, which potentially represent Skype signalling messages such as login or call establishment traffic. As Skype uses a proprietary protocol, we empirically derived a set of signatures by observing TCP and UDP traffic to and from Skype nodes. Note that because voice payloads are encrypted, the signatures are derived only from signalling payloads. Traffic is captured and recorded using Ethereal [25] on end-point computers.

We define two signature types — simple signatures and composite signatures. A simple signature is certain characteristics that a single packet presents, and a composite signature represents characteristics that are presented collectively by a number of consecutive packets. We empirically derived a set of simple and composite signatures by observing traffic generated by Skype. We repetitively carry out independent experiments and observations over months by varying a number of factors, including caller ID, callee ID, caller IP type, callee IP type, date and time and duration of call, to ensure the effectiveness and stability of our signature

set. Note that because of limitation by payload based detection scheme stated earlier, it is mainly developed as a verification approach against the non-payload based detection scheme, which is the focus of this research.

4.2.1 Notations and Preliminaries

Let [a,b] denote the set of numbers x such that $a \leq x \leq b$. Let $SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, $Protocol$, $Payload$, Dir denote source IP address, destination IP address, source port number, destination port number, transport protocol, packet payload, and direction of the packet respectively, where $SrcIPAd$, $DestIPAd \in [0, 2^{32} - 1]$, $SrcPort$, $DestPort \in [1, 2^{16} - 1]$, $Protocol \in \{TCP, UDP\}$, $Dir \in \{in, out\}$, and $Payload$ represents a byte sequence of variable length. Note that protocol can be TCP because signalling payloads are also considered. We use $|Payload|$ to denote the length of byte sequence represented by $Payload$. We also use $Payload[i-j]$, where i, $j \in N$, to denote the sub-sequence that begins with the ith and ends with the jth byte of $Payload$.

We assume that a packet is convertible to the following 7-tuple: $\{SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, $Protocol$, $Payload$, $Dir\}$. Packets can then be classified into flows, defined by the 5-tuple $\{SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, $Protocol\}$. A packet p belongs to a flow f if they have the same $SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, and $Protocol$. In the following, we use $p.X$ or $f.X$ to denote a particular field X that belongs to packet p or flow f. To simplify our notation, we also use $p[i-j]$ to denote $p.Payload[i-j]$.

4.2.2 Simple Signatures

For a packet $p = \{SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, $Protocol$, $Payload$, $Dir\}$, let $ByteVal$ denote a sub-sequence of $Payload$, i.e. $\exists i, j \in N$, such that $ByteVal = Payload[i-j]$, and we use Idx to represent the pair $\{i, j\}$. Then we define a simple signature as a 5-tuple $\{ByteVal$, Idx, $|Payload|$, $Protocol$, $Dir\}$. A signature does not have a timeout component as only the start of Skype flow is of our concern. In any case, The payload based scheme is only used to verify effectiveness of non-payload based scheme.

For example, assume outgoing UDP packets of length 18 having the third byte $0x02$ are observed frequently, then the simple signature for it can be denoted as $\{0x02$, {2, 2}, 18, UDP, out$\}$.

Table 4.1 summarises the repetitively occurring simple signatures that we identified during the observation period on Skype traffic. It is worth noting that occurrences of these signature are independent of firewall and NAT configurations, as Skype always starts with the same signalling messages in any attempt to connect to the outside world.

Number	$ByteVal$	Idx	$\|Payload\|$	$Protocol$	Dir
#1	$0x02$	{2, 2}	18	UDP	out
#2	$0x01$	{3, 3}	23	UDP	out
#3	$SrcIPAd$	{3, 6}	11	UDP	in

Table 4.1: Simple Signatures

4.2.3 Composite Signatures

The rationale behind composite signatures is that, although encrypted by a proprietary protocol, Skype traffic still presents some characteristics of its protocol. For example, a query/response pair may contain the same cipher text that is encrypted from a common value shared by the query/response messages.

Let f denote a flow which consists of packet sequence $\{p_1, p_2, \ldots, p_N\}$ in flow f. Let $SigLength$ denote the number of packets in f, i.e. N. Let $PktLengths$ denote a sequence of natural numbers representing payload length of each packet, i.e. $\{|p_1.Payload|, |p_2.Payload|, \ldots, |p_N.Payload|\}$. Let $BoolCondition$ denote a boolean condition $p_a[i_a - j_a] <|=|> p_b[i_b - j_b]$, where $a, b \in [1, N]$, $i_a, j_a \in [1, |p_a.Payload|]$, and $i_b, j_b \in [1, |p_b.Payload|]$, meaning that the numeric value of byte sequence consisted of the i_ath to j_ath byte of $p_a.Payload$ is greater than, equal to, or less than that consisted of the i_bth to j_bth byte of $p_b.Payload$. For example, assume $p_1.Payload = 1d500250ed4f9528$ and $p_2.Payload = 1d5007829528$, then we say $p_1[0-1] = p_2[0-1]$, $p_1[2-3] < p_2[2-3]$, and $p_1[6-7] = p_2[4-5]$. Let $Condition$ denote a set of boolean conditions, i.e. $Condition = \{BoolCondition_1, BoolCondition_2, \ldots, BoolCondition_K\}$. We then define a composite signature as a 5-tuple $\{SigLength, PktLengths, Condition, Protocol, Dir\}$.

For example, assume that flows containing the sequence of consecutive packets as depicted in Figure 4.1 has been observed frequently,

Then the signature of the above pattern can be represented by a composite signature as $\{3, \{18, 11, 23\}, \{p_1[0-1] = p_2[0-1] = p_3[0-1], p_1[6-7] = p_2[4-5] < p_3[9-10]\}$, UDP, {out, in, out}$\}$.

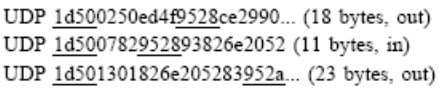

Figure 4.1: An example of composite signature

Table 4.2 summarises the repetitively occurring simple signatures that we identified during the observation period on Skype traffic. Note that the potential complexity of the process to generate the complete set of composite signatures over a large flow of packets is astronomic. It is not the intention of this research to generate the complete set of composite signatures. Only a small set of composite signatures has been empirically derived and is used mainly to verify the non-payload based detection scheme.

We have no intention of doing so and instead only empirically derived a small set of composite signature as we use payload based detection scheme only to verify the non-payload based detection scheme.

SigLength	*PktLengths*	*Condition*	*Protocol*	*Dir*
3	{18, 11, 23}	$p_1[2] = p_4[2] = 0x02$, $p_1[0-1] = p_2[0-1] = p_3[0-1]$, $p_2[8-10] = p_3[5-7]$	UDP	{out, in, out, in}
2	{18, 26}	$p_1[2] = p_2[2] = 0x02$	UDP	{out, in}

Table 4.2: Composite Signatures

4.3 Characterisation of Skype Traffic

We now introduce a systematic approach to identifying Skype voice traffic at the transport layer, i.e. based on the IP and TCP/UDP header and packet arrival times, without relying on packet payload.

As an Internet telephony application, Skype traffic demonstrates realtime streaming characteristics, i.e. smaller packet sizes and short packet inter-arrival time. On the other hand, being a peer-to-peer software, Skype presents almost identical connection patterns to other peers as do other P2P applications. In the following, we characterise the behavior of Skype traffic from these two aspects, i.e. realtime characteristics and connection patterns.

4.3.1 Realtime Characteristics

Real-time applications need to emit packets at a relatively small interval for the simulation of continuous and non-delaying effect. This is not a major concern for traditional non-realtime applications that are not operating under strict time constraints. As a result, real-time applications tend to produce smaller packets than traditional client-server applications such as HTTP or FTP. Additionally, for VoIP applications that transfer real-time traffic, UDP is usually preferred over TCP for its timely delivery and smaller header overhead.

We study the realtime characteristics of Skype voice traffic from 3 aspects, i.e. packet size, packet inter-arrival time, and bandwidth burstiness. We study these realtime characteristics by analysing traffic captured on end-point hosts running Skype. Skype traffic is generated with a number of independent and varying factors, including caller ID, callee ID, caller IP type, callee IP type, audio type, date and time, and duration of call. Experiments are repeated over months to ensure that the realtime characteristics observed are consistent and stable.

We illustrate the realtime characteristics of Skype by an example where calls are established between a host in a research lab and a host connected with shared residential ADSL and behind NAT. Call durations vary from 10 seconds to 30 minutes. In respect to the traffic capture point which is the host at the research lab in this example, voice traffic that is sent to/from the host behind residential ADSL is *outbound/inbound* traffic respectively.

Packet Size Characteristics

Figure 4.2 depicts distribution of packet sizes when call durations vary in 10 seconds, 30 seconds, 1 minute, 3 minutes, 10 minutes and 30 minutes. It can be seen that packet size distributions are self-similar over various call durations and time-scale independent. That being said, packet size distribution is consistent over varying durations, and mainly centers around 120 bytes. Packets that are longer than 50 bytes and shorter than 150 bytes constitute the major portion. We observed that packets falling outside this range are mainly captured at the call establishment stage, and thus believe that they are signalling messages instead of voice traffic.

Figure 4.3 depicts the same scenario with *cumulative density function* (CDF). It can be seen that cumulative density of packets with packet size around [50, 150] bytes (corresponding to the major portion in Figure 4.2) increases as calls last longer. This

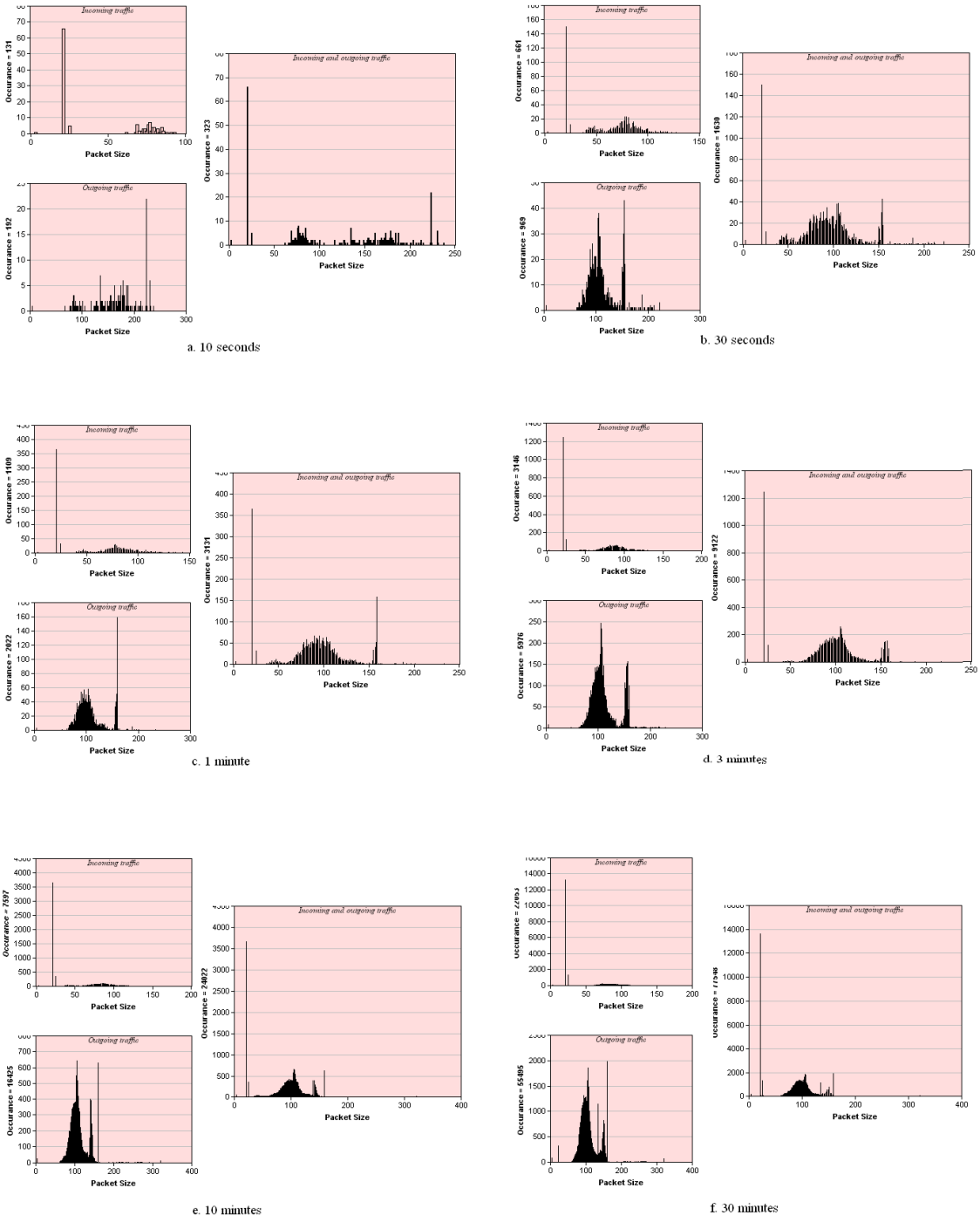

Figure 4.2: Packet Size Distribution Diagram

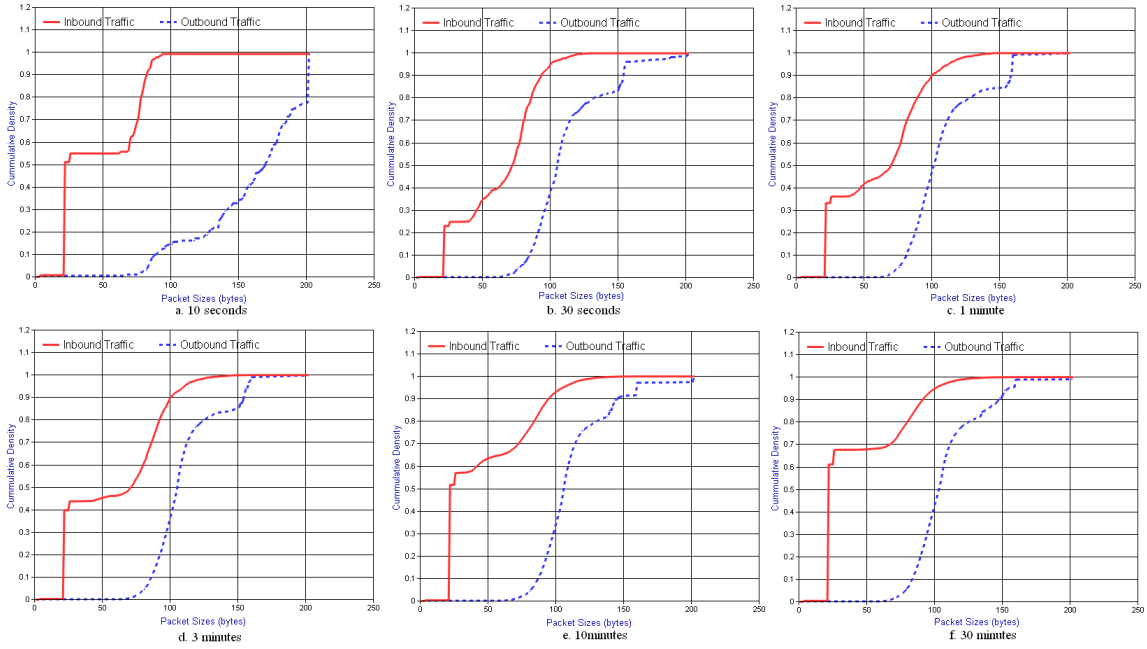

Figure 4.3: Packet Size Cumulative Density Function

justifies our hypothesis that packets longer than 150 bytes and shorter than 50 bytes are signalling messages and are emitted mainly during call establishment stage. It can also be observed that the inbound traffic contains many packets between 20 to 30 bytes. We attribute this to signalling messages that are related to NAT, as they are only observed when the host that we communicate with is behind NAT. CDFs in Figure 4.3 also demonstrates strong self-similarity over varying time scales — in all sub-diagrams, the curve between x-coordinate [50, 150] is highly similar to the second quadrant portion of the normal distribution curve with mean value 0, and its slope ($\frac{dy}{dx}$) peaks at around x=120.

Both Figure 4.2 and 4.3 show that Skype sends considerably smaller packets than the typical Internet traffic, which is dominated by specific packet sizes and average out at around 400 bytes [68] [15].

Packet Inter-Arrival Characteristics

Figure 4.4 depicts packet inter-arrival time *cumulative density function* (CDF) of Skype voice traffic. It can be seen that most outbound packets, i.e. packets sent to the host behind residential ADSL, are sent in the range between 0.02 to 0.04 second. In addition, the percentage of packets arriving within 0.04 second increases as call

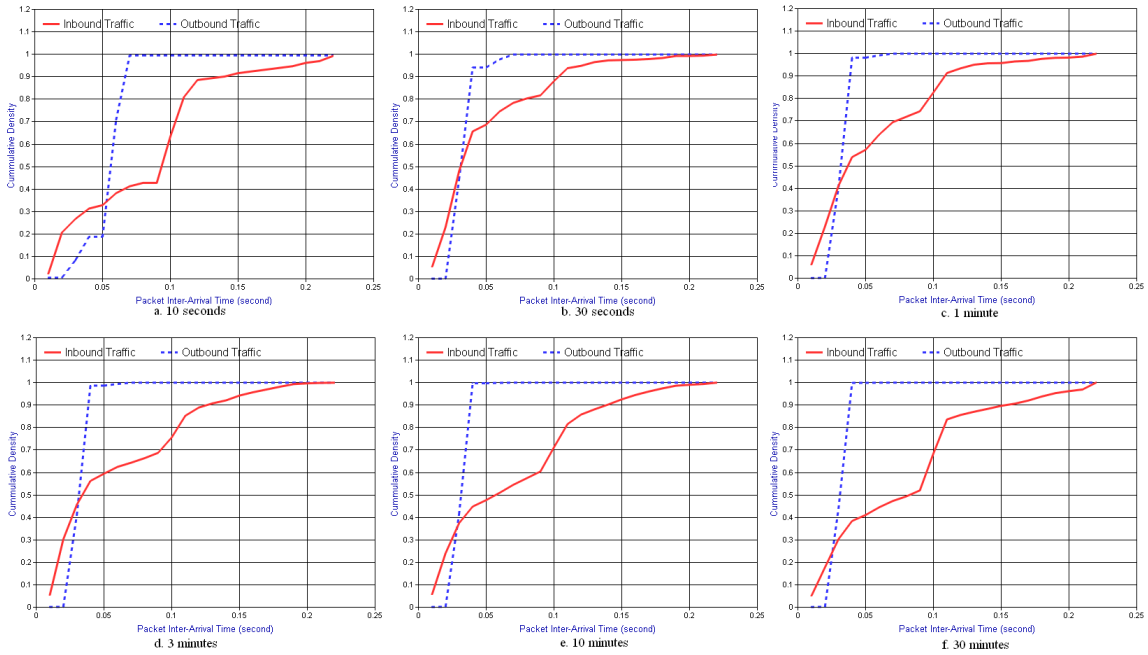

Figure 4.4: Packet Inter-Arrival Time Cumulative Density Function

duration becomes longer. We believe that packets with inter-arrival time greater than 0.04 second are signalling messages. For inbound traffic, packets arrive fairly uniformly over a range of 0.01 to 0.1 second. We attribute this spread of times to the delay imposed by the shared residential ADSL connection with very limited upload capability (128Kbps). To verify our hypothesis, we ran Ethereal on the host behind the residential ADSL, and the result is depicted in Figure 4.5. It can be seen that the discrepancy in packet inter-arrival time distribution between inbound and outbound traffic is significantly smaller. Similar to packet size distributions as depicted in Figures 4.2 and 4.3, both inbound and outbound traffic demonstrates similarity over varying call durations.

Bandwidth Burstiness Characteristics

We measure bandwidth burstiness by not only byte rate but also packet rate, i.e. the number of packets sent per time unit. Figure 4.6 and 4.7 depict byte rate and packet rate consumed by Skype voice traffic in the start-up 30 seconds and over a 30-minute period, respectively. It can be seen that, for outbound traffic, both packet rate and byte rate are fairly constant after a rise-up stage in the first few seconds. Packet rates stayed at 33 or 34 packets per second, and byte rate slightly fluctuated between 3

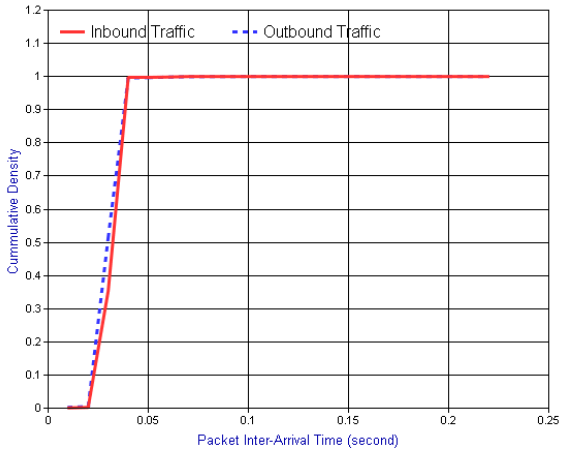

Figure 4.5: Packet Inter-Arrival Time Cumulative Density Function Captured at Residential ADSL

to 5 kilobytes per second. This observation is also supported by the empirical study at a larger scale in [91]. On the other hand, inbound traffic demonstrates strong fluctuation. We again attribute this fluctuation to delay and packet loss imposed by the shared residential ADSL connection which has very limited upload capability. To verify this hypothesis, we ran Ethereal on the host behind the residential ADSL, and observed that packet rate and byte rate are fairly constant when monitored at this point. Figure 4.8 presents the packet rate and byte rate demonstrated by traffic captured on the host behind the shared residential ADSL connection.

4.3.2 Connection patterns

Peer-to-peer traffic demonstrates certain connection characteristics. Karagiannis *et al.* [108] proposed two non-payload based heuristics, namely "TCP/UDP IP pairs" and "{IP, port} pairs", to identify peer-to-peer traffic from other traffic. However, it is worth noting that they can only distinguish peer-to-peer traffic from other types of traffic, but cannot distinguish peer-to-peer traffic generated by one particular application from another.

"TCP/UDP IP pairs" identifies source-destination IP pairs that operate TCP and UDP on the same port. Unfortunately, we have observed frequent departure from this pattern in our experiments, i.e. Skype uses only one protocol for each source-destination IP pair. Hence, we are not going to use this pattern in identifying Skype voice traffic.

"{IP, port} pair" utilises the fact that for the advertised {IP, port} pair of host

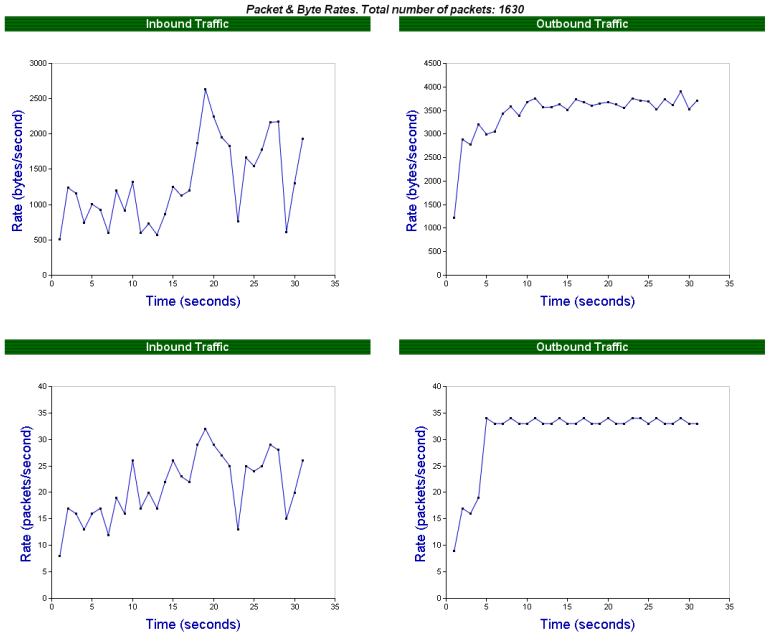

Figure 4.6: Bandwidth Burstiness of the start-up 30 seconds

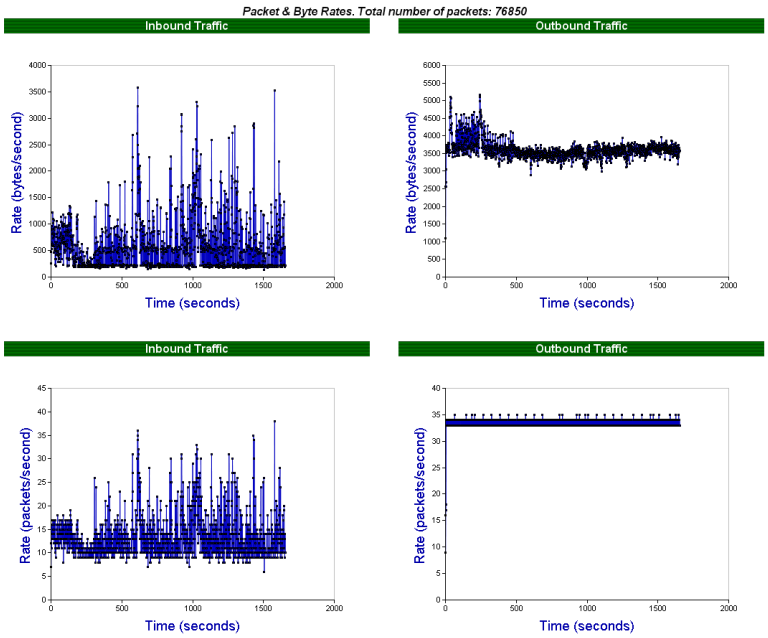

Figure 4.7: Bandwidth Burstiness of 30 minutes

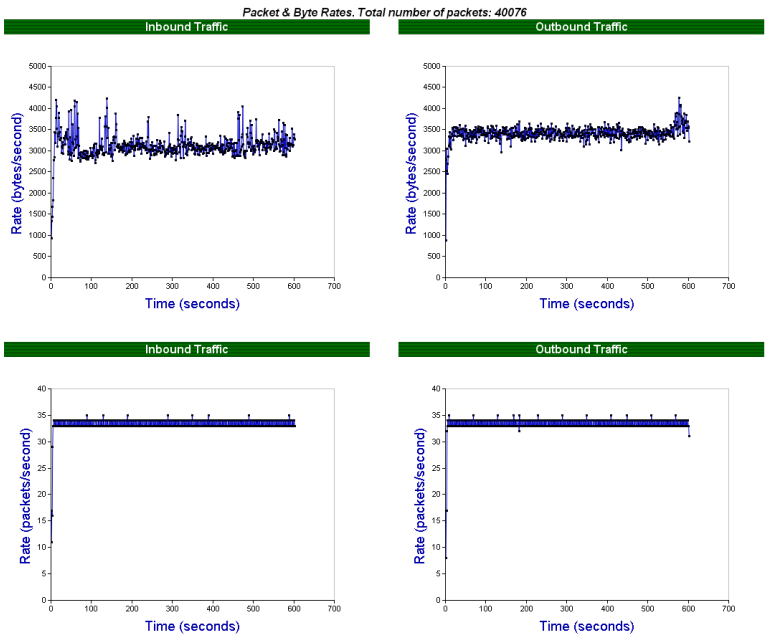

Figure 4.8: Bandwidth Burstiness Captured Behind Shared ADSL

A, the number of distinct IPs connected to it will be equal to the number of distinct ports used to connect to it. As do most peer-to-peer applications, the advertised port used by Skype can be configured by users. Change to the port will be applied the next time Skype is started. In our experiments, we observed that this advertised port is not only used as a destination port for incoming connection attempts, but also as a source port for outbound voice traffic.

4.4 Non-payload Based Detection Technique

In this section we will introduce our non-payload based technique to identify Skype voice traffic from other types of traffic. Our identification heuristic combines the realtime characteristics of Skype voice traffic as discussed in Section 4.3 and the "{IP, port} pair" heuristic. That being said, we consider a host has had Skype conversation if a port is identified to be an advertised peer-to-peer port by the "{IP, port} pair" heuristic, and traffic associated with this peer-to-peer advertised port demonstrates realtime characteristics as discussed in section 4.3. We only consider outbound traffic with respect to the monitoring point which is relatively close to the point where outbound traffic is generated, as it can be seen from Figures 4.4, 4.6 and 4.7 that realtime characteristics of inbound traffic may not be well preserved

due to transmission delay on Internet.

4.4.1 Conventional client-server applications and other peer-to-peer applications

Conventional client-server applications, such as HTTP and FTP, demonstrate substantially distinct characteristics from Skype voice traffic. They usually use predefined, well-known source and/or destination port numbers; they exclusively rely on TCP, and exhibit large packet sizes. We considered a flow was generated by conventional client-server applications if its traffic conforms to such characteristics.

Many peer-to-peer applications are used for file and music sharing. In such applications, while reducing transmission errors and header overhead is considered important, timely delivery of data is usually trivial and not relevant. Hence, they typically use TCP for actual data transfer but may use UDP for signalling. On the other hand, Skype was designed from the beginning to deliver data in realtime, and prefers the use of UDP for voice transmission as much as possible [88]. We hence distinguish Skype from other conventional peer-to-peer applications.

4.4.2 Realtime applications

We now focus on other types of realtime UDP-based applications. In the top 25 UDP application categories seen on Internet [68], Real Audio [72] accounts for the largest number of bytes among non-anonymous UDP applications. In addition to being a realtime application, Real Audio also demonstrates certain characteristics similar to peer-to-peer applications — it uses both TCP and UDP to transport data. To distinguish Real Audio from Skype traffic, we develop two heuristics as follows

- Real Audio traffic is dominated by specific packet sizes [69].
- Real Audio traffic is unidirectional. Volumes of inbound and outbound traffic are highly asymmetric.

Out of the top 25 UDP applications [68], 14 of them are Internet realtime strategic games, such as Starcraft and Half Life. Karagiannis's heuristic to identify gaming traffic is based on the viewpoint that Internet based realtime strategic games are inclined to employ packets dominated by specific packet sizes. This pattern is expected by Internet based games as each player sends out multiple copies of its

current state to each other player [64] [2] [116]. However, games and gaming traffic differ a lot from one another, and some may employ packets not dominated by specific size. Due to different gaming types, packet sizes may vary in a wide range which is hard to predict. We hence extend Karagiannis's heuristic by taking into consideration the periodicity of gaming traffic, i.e. frequencies by which each player sends update of its status to other players. Our viewpoint is based on that gaming traffic demonstrates this periodicity with bandwidth consumption fluctuation and burstiness [116] [48]. On the other hand, packet rate of Skype traffic is relatively constant as demonstrated in Figures 4.6, 4.7 and 4.8.

Besides Real Audio and gaming traffic, other VoIP applications such as Microsoft MSN or GnomeMeeting may also potentially demonstrate realtime characteristics that are not distinguishable from Skype. However, they are mostly built on standard-based protocols such as H.323 or SIP, and transfer voice traffic through a dynamically negotiated port. As opposed to the advertised port that Skype uses to transfer voice traffic with the chatting peer and to signal other peers simultaneously, this negotiated port is dedicated to transferring voice traffic by the chatting peers of a particular session. We can therefore distinguish Skype from other standard-based VoIP applications.

4.4.3 Final Algorithm

We use x and y_1, y_2, y_3 to denote, respectively, the $\{IP, Port\}$ *pair* connection pattern and the realtime characteristics that Skype presents as follows,

- x <IP, Port> pair heuristics.
- y_1 Packet sizes are relatively small and the majority follows normal distribution.
- y_2 Relatively constant packet and byte rate.
- y_3 Packet inter-arrival time mainly resides in between [0.02, 0.04] second.

We use the matrix as shown in Table 4.3 to summarise distinct characteristics of various applications. It can be seen that Skype can be distinguished from each of other applications by at least one differentiating characteristic. Note that here we assume the worst case for other VoIP applications, i.e. they present the same realtime characteristics as do Skype.

Application	x	y_1	y_2	y_3
Skype	✓	✓	✓	✓
Conventional Web Apps				
Other P2P Apps	✓			
Real Audio			✓	
Online Games	✓			✓
Other VoIP Apps		✓	✓	✓

Table 4.3: characteristics matrix

Combining the techniques of all previous sections yields our final non-payload based identification method for Skype voice traffic. Note that our algorithm is designed for analysis of passive traffic traces, allowing multiple passes over the data if necessary. Adapting our algorithm to detect and block Skype traffic dynamically is part of our ongoing work. Our final algorithm is presented in Algorithm 5.

4.4.4 Discussions

Traffic classification techniques based on pattern recognition, including signature based techniques, can be invalidated by applying variations to the distinctive patterns. In particular, our non-payload based technique is built upon realtime characteristics of Skype traffic such as packet size and bandwidth consumption characteristics in addition to the peer-to-peer connection pattern, and hence can be circumvented by applying changes and variations to such realtime characteristics. For example, "junk bytes" can be added to Skype packets to change packet sizes. This also changes the bandwidth consumed by Skype. However, Skype is designed to deliver packets in realtime as much as possible. Although increase in packet size or bandwidth consumption can circumvent our detection scheme, it could also severely degrade call quality of Skype. On the other hand, unless voice compression techniques can be significantly improved in the near future, reducing packet size is desirable but not quite practical. Moreover, downward compatibility must be considered for any change made to a widely deployed application. This requires significant efforts and takes a relatively long time. Therefore, realtime characteristics presented by Skype as it is now is expected to remain at least in the near future. We empirically justified our viewpoint by experimenting our non-payload based technique, developed from Skype 1.3, with the latest Skype version 2.0 in Section 4.5.2.

Algorithm 2: Nonpayload algorithm for Skype voice traffic identification

```
begin
    /* constants definition                                          */
    const FT = All Flows ;

    for each flow f in FT ;
    do
        if Use of <IP, Port> pair connection pattern then
            if Packet sizes are relatively small and the majority follows normal
            distribution then
                Packet_Size_Heuristic = true
            if Relatively constant packet and byte rate then
                Packet_Rate_Heuristic = true
            if Packet inter-arrival time mainly resides in between [0.02, 0.04]
            second then
                Inter_Arrival_Heuristic = true
            if Packet_Size_Heuristic and Packet_Rate_Heuristic and
            Inter_Arrival_Heuristic then
                SkypeCall.insert(f) ;
            else
                OtherP2P.insert(f);
        else
            NoneP2P.insert(f);
```

4.5 Implementation and Experiments

We implemented the algorithm and techniques presented in previous sections in Java. The experiments were performed on a Dell Optiplex GX280 with 1 GBytes of RAM and a 2.8GHz processor running Windows XP Professional.

We evaluate our schemes from two perspectives — false-positive and false-negative. False-positive evaluates accuracy of our schemes, i.e. likelihood that other non-Skype traffic are misclassified as Skype traffic. False-negative indicates the extent of misclassification where our schemes fail to identify Skype traffic.

4.5.1 False-Positive Evaluation

We first evaluate the false-positive, i.e. non-Skype traffic being identified as Skype traffic, of both payload and non-payload based schemes. We obtained from CAIDA [1] two OC-48 traffic traces that were captured in early 2003. They offered us large data sets that were expected to contain little or no Skype traffic because the initial version of Skype was released on August 29, 2003. All packets are truncated to 48 bytes, and hence UDP packets are preserved up to 16 bytes of data [2]. This is not an issue in our situation as the payload-based signatures examine packet headers and up to the $10th$ byte in payloads as shown in Table 4.2, and our non-payload based scheme does not rely on any payload. Details of the traffic traces are described by trace file #1 and #2 in table 4.4.

No.	Packets	Bytes	Start Time	Dur.	Flows	UDP Flows
1	84 Mil.	43G	5:00pm, Apr 24, 2003	60 min.	8136 K	1498 K
2	88 Mil.	62G	4:59am, Jan 16 2003	26 min.	3176 K	653 K
3	14 Mil.	4.2G	4:00am, Mar 16, 2006	1 week	154 K	77 K

Table 4.4: OC-48 Traffic Traces

On one hand, our non-payload based scheme identified 6 Skype flows in trace #1 and 1 Skype flow in trace #2. Therefore, it has resulted in at most 7 false-positives out of 2.1 million UDP flows, i.e. about 0.00003% flows are mis-identified as Skype calls.

On the other hand, the payload based scheme identified 916 simple signature #1 and 4 simple signature #2 identified in Table 4.1 from trace #1, and 844 simple

[1] http://www.caida.org

[2] Taking away 4 bytes Cisco HDLC header, 20 bytes IP header, and 8 bytes UDP header

signature #1 from trace #2. Therefore, it has resulted in at most 1764 false-positives out of 172 million packets, i.e. about 0.001% packets are mis-identified as simple signature. No packet sequence is mis-identified as composite signature.

4.5.2 False-Negative Evaluation

We next explore the extent of misclassification, where the non-payload based scheme fails to identify Skype traffic, by comparing the result from payload based scheme. We do not investigate the false-negative of our payload based scheme directly. Instead, we ensure that the signature set we derived is at least sufficient to the organisation and environment where the experiments are conducted by repetitively carrying out independent experiments over months with a number of varying factors including caller ID, callee ID, caller IP type, callee IP type, date and time and duration of call.

We monitored a week's traffic on a boundary firewall in our organisation. The experimental setup is illustrated in Figure 4.9. All hosts are Dell Optiplex series running Windows XP Professional and connected to Ethernet Switches. Trace file #3 in table 4.4 details the specification of traffic captured. We use Skype to call or receive calls from other users located on the other side of the firewall or on Internet, as indicated by dashed lines in Figure 4.9. As our non-payload based scheme heavily relies on packet inter-arrival timing, delay by traffic filtering may degrade its effectiveness and accuracy. Therefore, in order to study the effectiveness of our non-payload based scheme in the presence of filtering delay, we kept the firewall busy outside normal office hours by running random traffic generators. The randomly generated traffic traverses through the firewall as indicated by solid lines in Figure 4.9. During the experiment period, we also made a frequent use of other types of streaming applications when no Skype call is happening, including streaming video and audio, real time strategic online games, and other types of VoIP applications such as MSN, Yahoo! Messenger and Google Talk, which including Skype combined to generate 4.2 GBytes streaming traffic.

During a week's time, we made in total 80 calls with varying caller ID, callee ID, caller IP type, callee IP type, date and time and durations of call which vary from 1 to 106 minutes averaging at 28 minutes. Our non-payload based scheme successfully identified all 80 calls. Within the duration of each identified call, at least one simple or composite payload based signature is also identified. Note that

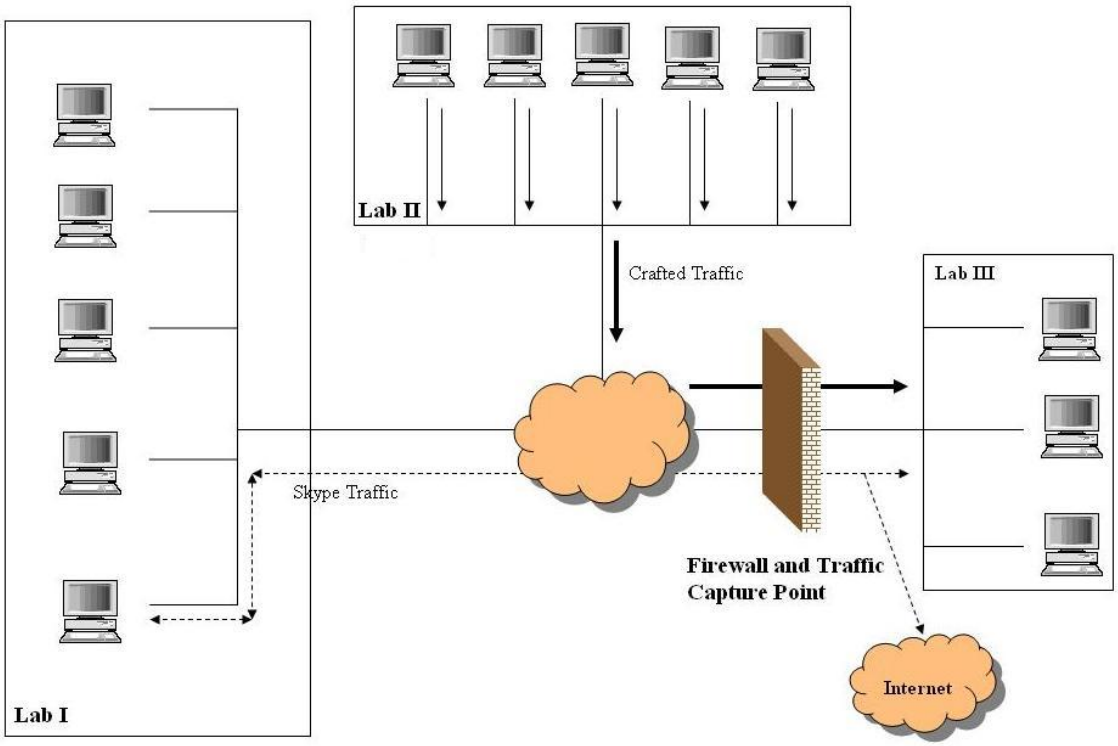

Figure 4.9: Experimental Setup

payload based signatures are also detected outside the 80 calls. However these are false-positives of only the payload based detection and is not of our concern. Figure 4.10 to Figure 4.16 present the results of both schemes. Each sub-figure represents a 24 hour period during the experiment, starting at 4am, March 16, 2006. X-axis denotes time elapsed in seconds since 4am on the day of capture, and Y-axis denotes source and destination of calls. Each diamond on the diagram denotes an occurrence of signature, and each horizontal red line denotes a detected call duration by our non-payload scheme. It can be seen from the consistency of the results that, although being encrypted, Skype traffic can still be effectively identified without looking at its encrypted payload, or at least as effective as its payload based counterpart.

Upgrade to Skype 2.0

All the experiments so far are performed with Skype 1.3. In order to test the resistance of our non-payload scheme against Skype version upgrade, we repeat the experiments in Section 6.2 with Skype 2.0. We made in total 60 calls with Skype 2.0 in a week's time. Call durations vary from 2 to 46 minutes, and average at 7

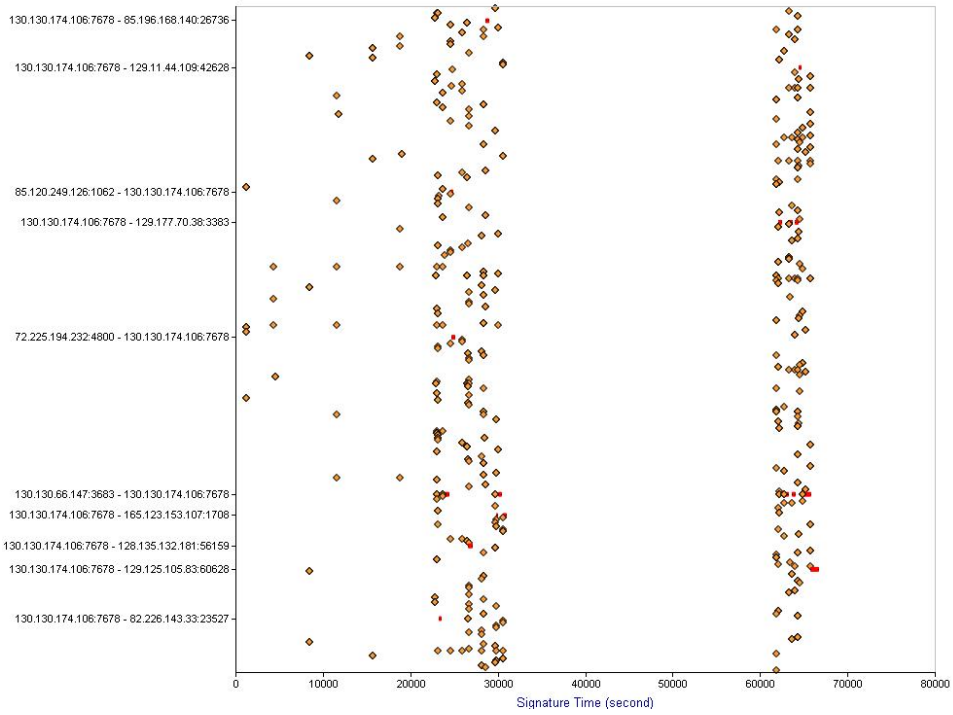

Figure 4.10: Experimental Results Day 1

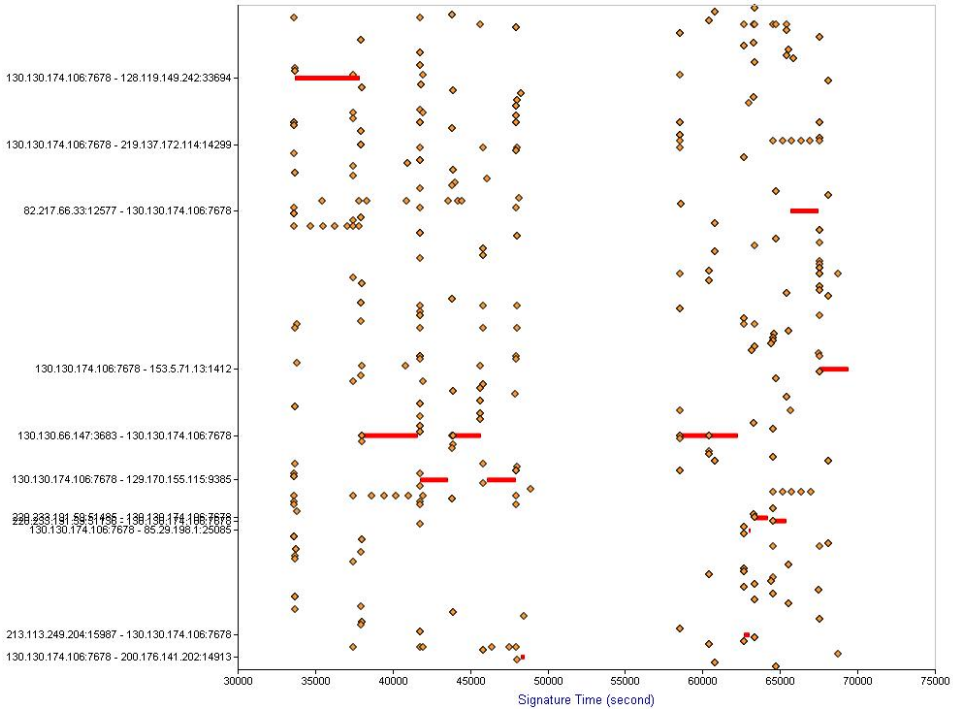

Figure 4.11: Experimental Results Day 2

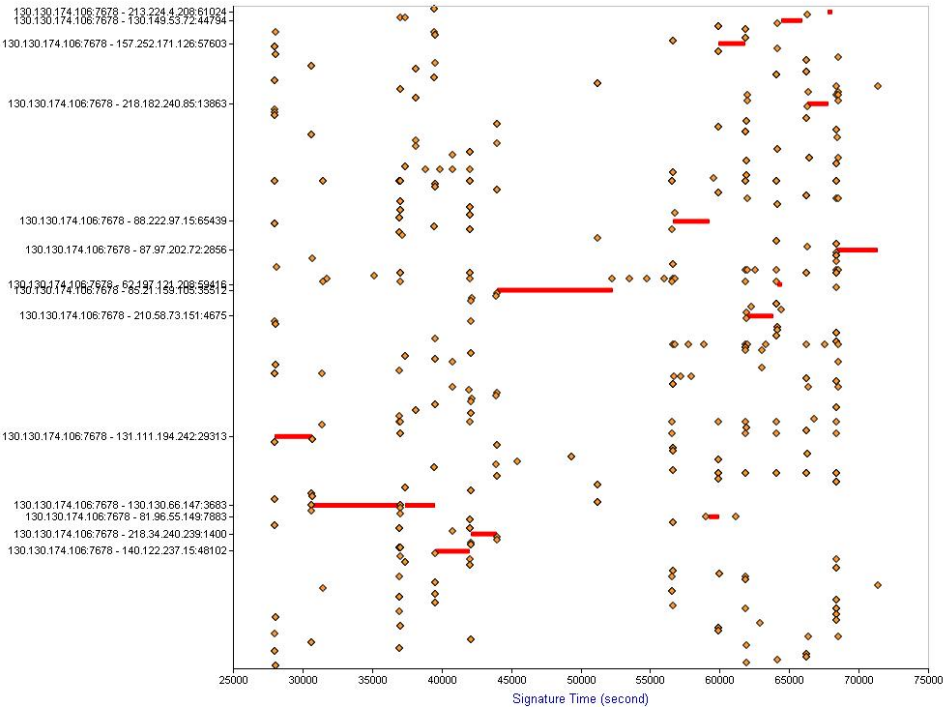

Figure 4.12: Experimental Results Day 3

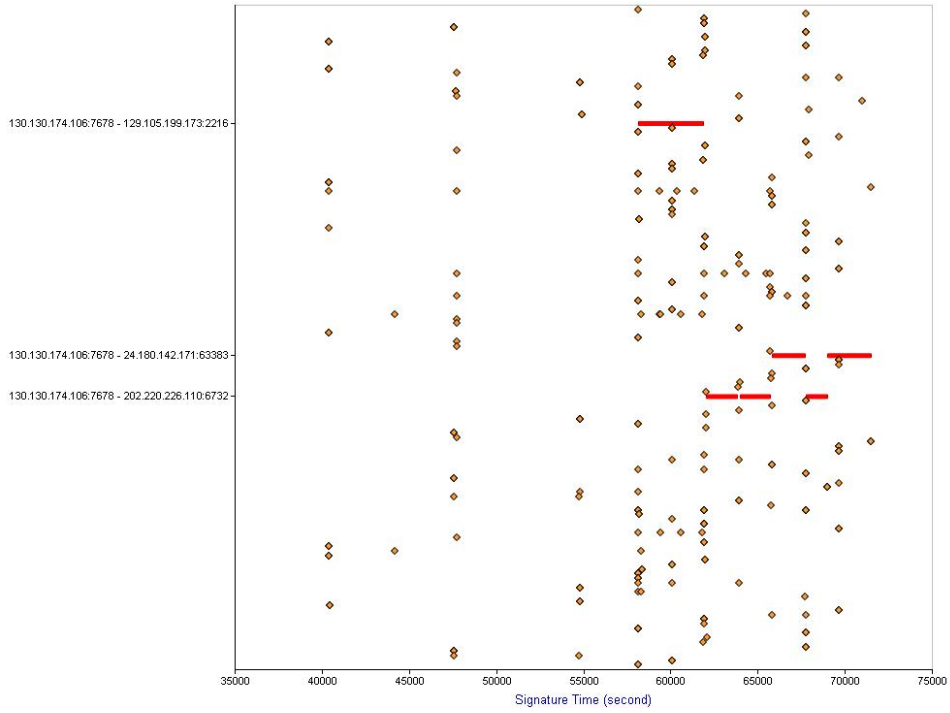

Figure 4.13: Experimental Results Day 4

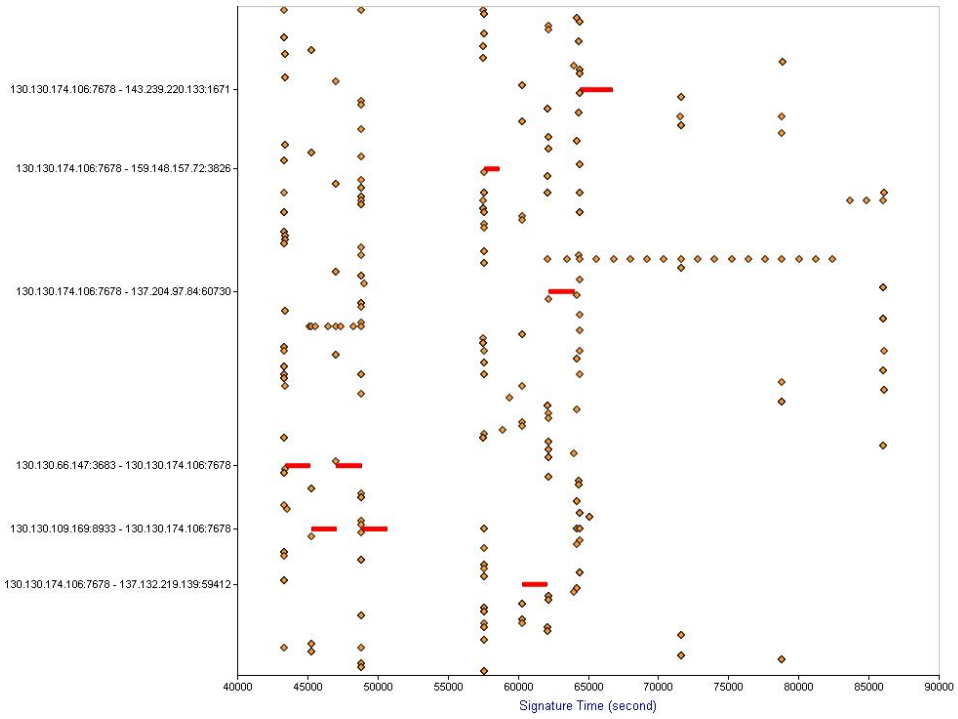

Figure 4.14: Experimental Results Day 5

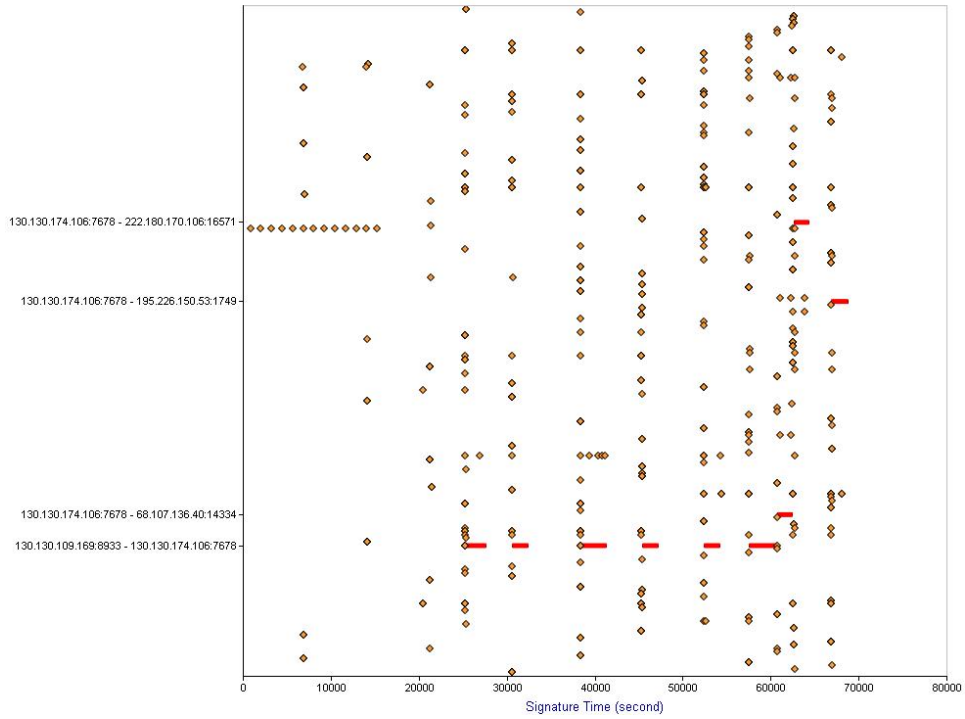

Figure 4.15: Experimental Results Day 6

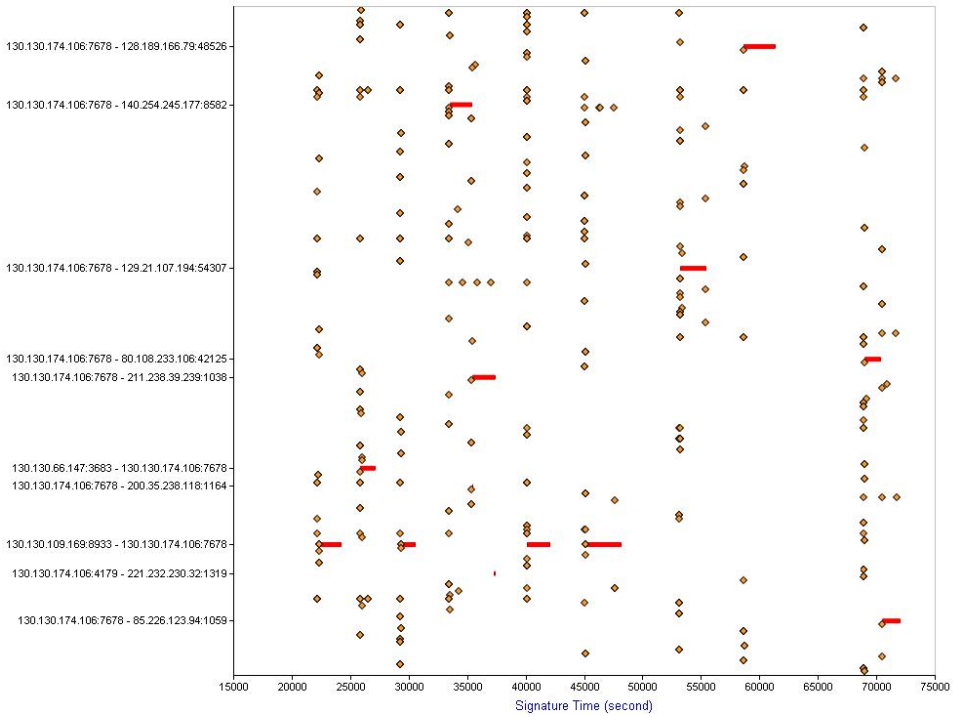

Figure 4.16: Experimental Results Day 7

minutes. Our non-payload based scheme successfully identified all 60 calls with no false-positive. It is worth noting that our payload based scheme has not detected any occurrence of simple signature #2 and composite signature #1, i.e. these two signatures do not survive from version upgrade. This result shows that, although our non-payload based scheme is built on pattern recognition that can be invalidated by future changes to Skype protocol or variations to Skype traffic patterns, it is at least more resistant to such changes and variations than its payload (signature) based counterpart. The experimental result also empirically justifies our hypothesis that realtime characteristics of Skype will remain within at least the near future.

4.6 Conclusion and Further Work

We presented both payload and non-payload based techniques for detecting Skype activities, especially Skype voice traffic. The presented technique can be used by firewalls to detect and block Skype traffic. Counter-measures can be taken on detection of breach of network access policies. A limitation of our technique is that it is based on empirical study of Skype traffic which may change over software version.

However, we believe it will only be a trivial task to re-collect and to sample new version of Skype traffic and apply the same technique presented in this Chapter.

4.6.1 A Related Problem

A related problem arises here. Streaming content like Skype and RTP tends to use dynamically negotiated ports. This makes translation of network access policies into firewall rule sets a task even more difficult. Translating a high-level security policy written in a natural language into firewall rule tables, a much lower-level description of the policy, is an error prone task particularly in the context of streaming content. Unlike traditional web services such as HTTP or FTP which have been studied thoroughly and have almost fixed pattern firewall configurations, defining firewall permit/deny rules to securely and effectively handle streaming content is still in the age of darkness and represents a serious challenge by system administrators.

The next Chapter presents our contribution to solving this problem. We provide a technique to compare and to analyse firewall rule tables. The proposed technique allows us to analyse firewall rule tables at a much fine-grained level, and to find out potentially incorrectly interpreted rules.

Chapter 5

Comparing Firewall Rules

One critical step towards enforcing network security with firewalls is to have correct filtering rules. However, developing firewall rules for large networks with complex security requirements is a difficult and error prone task. The situation is worsened when configuring firewall rules for complicated, dynamically port negotiating protocols such as RTP.

In this Chapter, we present our contribution to solving this problem. A formal method of representing firewall rules is developed to allow comparison of two sets of firewall rules. Then an algorithm is provided to determine if two rule sets are equivalent. The meaning of equivalencies is that the two sets of firewall rules have the same filtering effect on any packet. We demonstrate how the provided technique can assist system administrators to diagnose correctness of firewall rules translated from organisational access policies that are written in high level, human language.

5.1 Introduction

With the rapid growth of Internet-based information sharing in recent years, network security is gaining an increasingly strong focus in the research and industrial communities. In the battle against network attacks, firewalls have played one of the most important roles. A firewall protects secure networks against malicious attacks and penetration by filtering out unwanted network traffic from traffic entering the secure network. The filtering decision is made according to a set of ordered rules translated from a high level organisational security policy by an administrator [117].

Translating a security policy described in a high-level natural language into firewall rule sets, a very much lower-level expression of the security policy, is a difficult task. It is likely that, for example, an understanding of the assets and culture of

an organisation as well as technical proficiency with a firewall system will be required for a correct translation. Also, the security policy needs to be clearly and unambiguously written.

As an aid to increasing management's confidence that a security policy has been implemented correctly to the extent possible by a firewall rule set, two rule sets can be translated independently from a high level description of network policy and the results compared for equivalence. That is verifying both rule sets perform the same operation (pass or deny) on every packet. An implementation is adoptable if two rule sets translated independently from the same high level description of network policy are considered to be equivalent. In the case where equivalency is not found, it is highly desirable to be able to "debug" the rule sets. That is locating rules which permit packets denied by the other rule set. Such rules may exist for various reasons such as ambiguity in the policy description or inconsistent understanding of organisation assets. Being able to locate such rules therefore allows us to inspect closely problems in policy description and translation like the above.

A similar question arises when firewall rules need to be adjusted. Adding a new rule, or changing an existing rule, is often required because of a policy change or in response to a discovered security flaw. A new rule is added to a firewall to permit a new set of packets, or stop a set of permitted packets. The new rule may be added at the top of the table, or as an intermediate rule. The new set of packets, although may not have been explicitly specified by a single rule in the firewall rule set, might have already been implicitly covered by the combination of existing rules which precede the new rule. In this situation the new rule is redundant and unnecessary. This may indicate incorrect configuration of firewall by the existing rules, especially when the new rule is positive and specifies packets that, although not intended, have already been permitted by existing rules. Therefore, it is worth investigating rules which permit packets that should have been permitted "only" by the addition of the new rule, as well as the consequence of permitting packets which was not explicitly intended. In general, an administrator needs to be able to examine the effect of the new rule by determining the set of packets that are permitted (or denied) "only" because of the new rule.

This Chapter provides a systematical and provable approach that allows us to answer questions like the above. For a rule set T, starting from the first rule and sequentially going through each rule, we construct a new rule set T' consisting of only *positive, non-overlapping* rules. Non-overlapping means that each incoming

packet is matched by exactly one rule. Therefore in T', the set of packets permitted by the firewall is the union of a number of disjoint subsets. The order of rules in T is considered and used in the phase of constructing non-overlapping rules, and is irrelevant afterwards in T'. This is an important property that allows us to easily compare two rule sets, or analyse the effect of adding rules to a rule set. That is to compare two rule sets T_1 and T_2, we need to determine if $T_1' = T_2'$ where T_1' and T_2' are T_1 and T_2, respectively, written in the new form, i.e. positive non-overlapping rules. In the case equality does not hold, the new form of rule set can be used to find rules that pass packets not permitted by the other rule set (See Section 5.5). Also, to add a new rule r to the rule set T, we will write the extended table T_r (T enhanced with r) in this new form T_r' and if $T' = T_r'$ conclude the rule is redundant and not required. In this case, the new form of rule set can be used to find rules which combine to pass packets that should have been permitted "only" because of the new rule. We implemented in Java a GUI prototype of the above technique, and demonstrated in a case study its application for comparing and debugging firewall rule sets.

In the rest of this Chapter, Section 5.2 provides an summary of previous work related to the issues introduced above. Section 5.3 presents related background knowledge on firewall rules and rule tables. In Section 5.4, we give a formal yet intuitive definition of equivalence between firewall rule tables, as well as some other fundamental definitions. In Section 5.5, we present a theorem and a set of algorithms for determining whether two rule tables are equivalent, and locating the rules that permit packets denied by the other rule table in the case that inequality holds. In Section 5.6, we provide an GUI prototype written in Java, along with a complete example of applying the prototype to compare and debug firewall configurations. Section 5.7 provides a conclusion of the Chapter. Part of this Chapter appeared in [58].

5.2 Related Work

Management tools for developing firewall rule tables have found much attention in recent years. Modeling languages that are more expressive and closer to natural language than firewall rules alleviate the error-prone translation tasks [121, 107, 70]. On the other hand, less attention has been paid in particular to the problem of comparing firewall rules and rule tables.

Much research has been conducted on rule management problems. Two main threads of research [21] in this area are packet classification [109, 63, 3, 60, 79, 43, 26, 119, 114, 113] and detecting (and resolving) conflict between rules [21, 12, 107, 122, 19, 20]. Given a number of ordered filtering rules and an incoming packet, packet classification refers to the process of determining the appropriate action (pass or drop) on the packet as specified by the filtering rules. Rule conflict refers to the situation where two or more filtering rules cover a common set of packets, creating an ambiguity in packet classification.

Although previous works in the area provided techniques potentially applicable to comparing and analysing firewall rule sets, little attention has been paid to this particular area. Hazelhurst *et al.* provided an approach to comparing firewall rule sets using Binary Decision Diagrams (BDD) [93, 92]. A rule set is converted into a boolean expression, which can be compactly and uniquely represented by a BDD. The uniqueness of BDD is an important property that allows us to determine the equivalency of different rule sets by comparing their BDD representations. Moreover, as rule sets are represented as boolean expressions, extensive analysis can be made by performing symbolic boolean operations on boolean expressions representing rule sets. For example, suppose we have two rule sets represented respectively by boolean expressions T_1 and T_2, then we can perform a variety of analysis on T_1 and T_2 such as

- Are T_1 and T_2 equivalent, i.e. do they accept the same set of packets? — $T_1 = T_2$.
- Are all the packets accepted by T_1 also accepted by T_2? — $T_1 \Rightarrow T_2$.
- Are there packets accepted by T_2 but not by T_1? — $\overline{T_1} \wedge T_2$.

Although representing rule sets with BDDs provides a wide range of analysis on rule sets, the original form and order of rules are not preserved in the BDD representation. That being said, for example, although it is able to determine that T_1 and T_2 are not permitting the same set of packets, it cannot locate the rules that are permitting packets denied by the other rule set. This is important in the situation where further analysis is needed for debugging unexpected results, e.g. T_1 and T_2 are not permitting the same set of packets when they are expected and required to do so. For debugging purposes, it needs to be able to locate rules that are permitting packets denied by the other rule set, as such rules exist typically

because of ambiguous policy description or incorrect translation of security policy into firewall rules.

Being able to locate such rules is a significant advantage of our approach over the use of BDD. Given two rule sets T_1 and T_2, we can perform analysis on T_1 and T_2 such as

- Are T_1 and T_2 equivalent, i.e. do they accept the same set of packets? If not, which rules in T_1 permit packets denied by T_2, and vice versa.
- Are all the packets accepted by T_1 also accepted by T_2? If not, which rules in T_1 accept packets not permitted by T_2?
- Are there packets accepted by T_2 but not by T_1? If so, which rules in T_2 accept packets denied by T_1?

Al-Shaer *et al.* provided a tool to assist in inserting or editing rules in a rule table [24, 23]. An inserted rule is compared sequentially with each rule in the table to ensure that the set of packets it matches is not in the meantime already matched by any preceding rule such that the insertion would not have been necessary, which is usually considered an error. A similar approach is applied to rule editing. However, it is often the case that the packets matched by the new rule, although had not been explicitly matched by a single existing rule, may already have been matched by a combination of existing rules and so the new rule need not be added. The approach proposed in this Chapter detects such cases and so enhances Al-Shaer *et al.*'s work.

5.3 Formally Representing Firewall Rules and Rule Tables

In the following discussion, protocol means transport layer protocol such as TCP or UDP [1]. Let $SrcIPAd$, $DestIPAd$, $SrcPort$, $DestPort$, $Protocol$ denote source IP address, destination IP address, source port number, destination port number, and protocol respectively. Let $[a, b]$ denote the set of numbers x between a and b; that is $a \leq x \leq b$. Then $SrcIPAd$, $DestIPAd \in [1, 2^{32} - 1]$, $SrcPort$, $DestPort \in [1, 2^{16} - 1]$, and $Protocol \in \{TCP, UDP\}$. We also use [X] to denote a specific

[1] Note that our discussion focuses on only IPv4.

range of numbers that can be assigned to element X, e.g. [IP] represents the range of legal IP addresses. We assume a packet can be represented in the following form

$$(SrcIPAd, DestIPAd, SrcPort, DestPort, Protocol)$$

Let $\mathcal{P}$ denote the set of all packets. It can be seen that $\mathcal{P}$ is a finite set of size $|\mathcal{P}| = 2^{32} \times 2^{32} \times (2^{16} - 1) \times (2^{16} - 1) \times 2$.

5.3.1 Firewall Rules

A firewall filters packets based on fields defined in the network and/or transport layer, including source IP address ($[SrcIPAd]$), destination IP address ($[DestIPAd]$), and service (Srv).

$[SrcIPAd]$ and $[DestIPAd]$ specify an acceptable range of source/destination IP addresses respectively. Srv is the combination of protocol, source port number and destination port number. Note that the source port number is rarely significant, and commonly any source port is acceptable. Each rule is associated with an action (Act) field, the value of which is either "pass" or "drop", indicating whether a packet is passed or dropped when it is matched by this rule. Rules with the *pass*/*drop* action are referred to as *positive*/*negative* rules respectively.

There may also be other relevant fields, such as the rule number (rn) that identifies order of rules. In general, a firewall rule can be presented as a 4-tuple ($[SrcIPAd]$, $[DestIPAd]$, Srv, Act).

A rule r specifies a subset $\mathcal{P}_r \subseteq \mathcal{P}$ of packets that it matches. A packet p is matched by a rule r (and so $p \in \mathcal{P}_r$), if $SrcIPAd_p \in [SrcIPAd_r]$, $DestIPAd_p \in [DestIPAd_r]$ and the combination of $(Protocol_p, SrcPort_p, DestPort_p) = Srv_r$.

When a packet arrives at the firewall, it is mapped into the form of firewall rules and then matched against the firewall rule table. The first rule that matches the packet will determine the action on the packet. For example, if a packet p is matched by two rules r and r' where r' precedes r in the order, then r' will be applied to p but r will not. If p is matched by multiple rules preceding r, then which rule will be applied to p depends on the order of the matching rules. If no preceding rule matches p, then r will be applied to p.

We use $\{\mathcal{R}\}$ to denote the set of packets to which rule R is applied. These are the packets for which R is the first rule matched in the rule table. In general, $\{\mathcal{R}\} \subseteq \mathcal{P}_R$

— that is, the set of packets to which R is applied is a subset of the set of packets that R matches.

5.3.2 Firewall Rule Table

A *firewall rule table*, T, is an ordered set of rules, and specifies a set of packets $\mathcal{P}_T$ that are passed by the rule table. We use $(R_1, R_2, \ldots, R_n)$ to denote a rule table consisting of rules $R_1, R_2, \ldots, R_n$ in order.

Let $Match(p, T)$ denote a function that takes a packet p and a rule table T and returns the first rule that matches p. This rule, denoted by $r_{p,T}$, is the rule that will be applied to the packet p. There can be more rules that match p but only $r_{p,T}$ will be applied to p.

A firewall defaults to accept or drop packets not matched explicitly. In most situations, firewalls use "drop packet" as the default action so that only packets that are explicitly permitted are passed by the firewall [51], and this is the assumed default action in the following discussion. That being said, if $r_{p,T} = \phi$ where ϕ denotes the empty set, T will drop p. As a result, we need not use negative rules explicitly to drop prohibited packets. Explicit positive rules are needed to accept legitimate packets. Negative rules provide readability and compactness for firewall table by denying packets that are permitted by subsequent positive rules. For example, consider the case that connections from a large block of network addresses are permitted with the exception of a small range of addresses in the middle of the larger block. This can be implemented more compactly using a negative rule denying traffic from the small range of addresses followed by a positive rule that permits traffic from the larger block.

5.3.3 An Example

Consider the firewall rule table shown in Table 5.1. R_1 drops packets from 192.168.13.10 and to 192.168.13.11 for the RTP service so that R_2 can be written in a more compact and readable form.

The rule table presented in Table 5.1 can be written with only positive rules as shown in Table 5.2. It is seen that Table 5.1 is more compact and readable (allowing packets from 192.168.13.0/24 to go anywhere with the exception that packets from 192.168.13.10 are not allowed to reach 192.168.13.11), and is also more efficient

Rule #	Source IP range	Destination IP range	Service	Action
1	192.168.13.10	192.168.13.11	RTP	Drop
2	192.168.13.0/24	0.0.0.0-255.255.255.255	RTP	Pass

Table 5.1: Firewall rule table using both negative and positive rules

because a firewall does not have to try to match all the rules before a packet can be dropped.

Rule #	Source IP range	Destination IP range	Service	Action
1	192.168.13.10	0.0.0.0-192.168.13.10	RTP	Pass
2	192.168.13.10	192.168.13.12-255.255.255.255	RTP	Pass
3	192.168.13.0-192.168.13.9	0.0.0.0-255.255.255.255	RTP	Pass
4	192.168.13.11-193.168.13.255	0.0.0.0-255.255.255.255	RTP	Pass

Table 5.2: Firewall rule table using positive rules only

This example shows that negative rules can be used to drop a subset of packets matched by subsequent positive rules, and could make the rule table more compact and readable. A negative rule r will not affect the set of permitted packets if $\mathcal{P}_r$ has no intersection with (or has only empty intersection) with the sets $\mathcal{P}_{R_1}, \cdots \mathcal{P}_{R_i}$ where $R_1, \cdots R_i$ are positive rules following r. That is, a negative rule r will not affect $\mathcal{P}_T$ if $\forall R_j$, $\mathcal{P}_{R_j} \cap \mathcal{P}_r = \phi$, where R_j are positive rules following r.

5.4 Preliminaries

Definition 1 *Two firewall rule tables T_1 and T_2 are **equivalent** if $\mathcal{P}_{T_1} = \mathcal{P}_{T_2}$.*

We assume that firewalls drop all packets by default, and there is a "deny all packets" at the end of the firewall rule table. We only need to consider packets that can pass the firewall. This is because $\mathcal{P}$, the set of all packets, is a finite set, if $\mathcal{P}_{T_1} = \mathcal{P}_{T_2}$, then T_1 and T_2 will also drop the same set of packets since $\mathcal{P} - \mathcal{P}_{T_1} = \mathcal{P} - \mathcal{P}_{T_2}$.

A positive rule specifies a set of packets that are permitted by the rule. However, the set of packets that will be permitted by the rule will also depend on other rules and their order. To assist with Definition 2, consider a table with rule list

$(R_1, \cdots R_n)$. A packet p that matches a rule R_i in the table will be in one of the following categories.

1. p is matched by R_i and at least one rule R_j where $j < i$. In this case p will be dropped or passed depending on R_j being negative or positive respectively. In both cases R_i will not be applied.

2. p is matched by R_i and at least one rule R_j where $j < i$ and R_j is positive. In this case p is matched by both r and at least one preceding rule so that r will not be applied and p will be passed if the first rule that matches p is positive;

3. p is matched only by R_i but not any rule R_j where $j < i$. In this case R_i will be applied to p and $Match(p, T) = R_i$.

Note that in Point 1 and Point 2 above there may be multiple such R_j rules. In the following, we show how a positive rule can be divided into sub-rules to capture above cases. A rule R' is called a *sub-rule* of a rule R if $\mathcal{P}_{R'} \subset \mathcal{P}_R$.

Consider a firewall table with rule list $T = (R_1, \cdots R_n)$.

Definition 2 *For a rule R_i,*

- *R'_i is the **redundant part of** R_i **with respect to table** T if R'_i is a sub-rule of R_i, and for some positive rule $R_j \in T$, where $j < i$, $\mathcal{P}_{R'_i} \subseteq \mathcal{P}_{R_j}$. In this case R'_i will never be applied and $\{\mathcal{R}'_i\} = \phi$.*

- *R''_i is the **shadowed part of** R_i **with respect to table** T if R''_i is a sub-rule of R_i, and for some negative rule $R_j \in T$, where $j < i$, $\mathcal{P}_{R''_i} \subseteq \mathcal{P}_{R_j}$. In this case R''_i will never be applied and $\{\mathcal{R}'_i\} = \phi$.*

- *R'''_i is the **effective part of** R_i **with respect to table** T if R'''_i is a sub-rule of R_i, and matches packets that are not matched by any rule $R_j \in T$, where $j < i$. In other words, $\mathcal{P}_{R'''} = \mathcal{P}_R - (\mathcal{P}_{R'} \cup \mathcal{P}_{R''})$.*

Following Definition 2, it can be seen that any positive rule R can be re-written as three sub-rules R', R'' and R''' in general.

If R is written as ($[SrcIPAd]$, $[DestIPAd]$, Srv, $pass$), then R', R'' and R''' can be written as, respectively, ($[SrcIPAd1]$, $[DestIPAd1]$, Srv, $pass$), ($[SrcIPAd2]$, $[DestIPAd2]$, Srv, $pass$) and ($[SrcIPAd3]$, $[DestIPAd3]$, Srv, $pass$) where $[SrcIPAd1] \cup [SrcIPAd2] \cup [SrcIPAd3] = [SrcIPAd]$ and $[DestIPAd1] \cup [DestIPAd2] \cup$

$[DestIPAd3] = [DestIPAd]$. This implies that $\mathcal{P}_R = \mathcal{P}_{R'} \cup \mathcal{P}_{R''} \cup \mathcal{P}_{R'''}$. The *Venn Diagram* in Figure 5.1 illustrates the relation between R, R', R'', and R'''.

Al-Shaer *et al.* also used the concepts of shadowed and redundant rule in firewall rule tables [24]. Our definition differs in that a rule R_i is divided into three sub-rules and the redundant/shadowed relation is defined between one of the sub-rules and a preceding rule R_j where $j < i$, as opposed to between R_i itself and R_j.

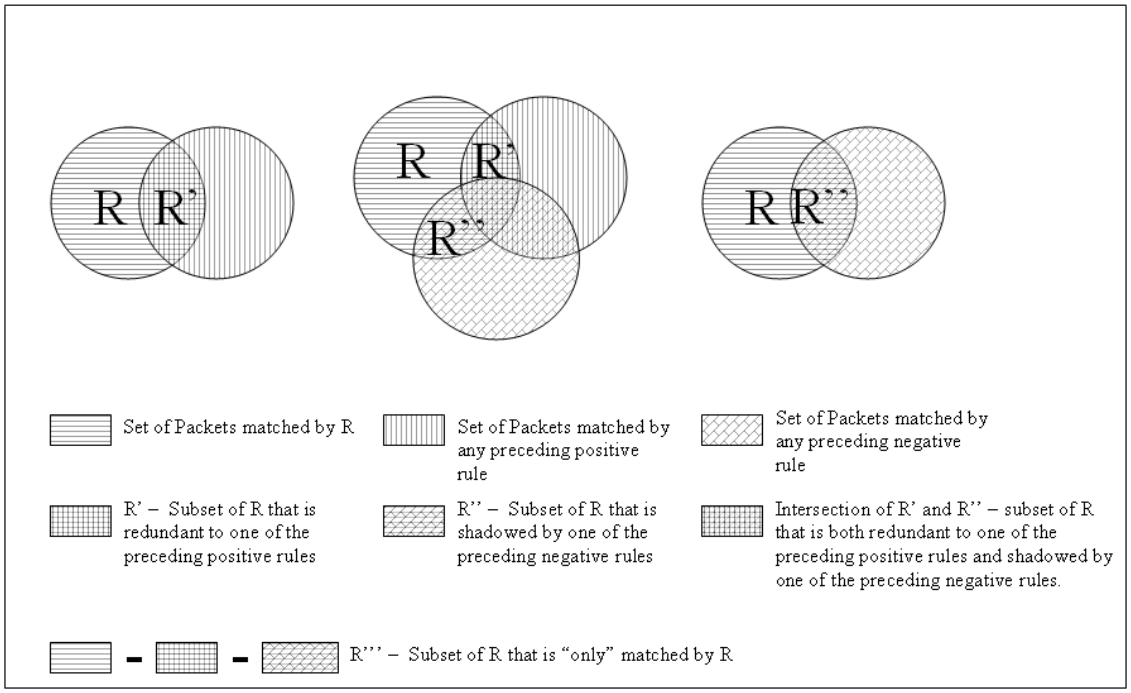

Figure 5.1: The Venn Diagram illustrating R', R'', and R'''. R''' is the set of packet "only" matched by R. Here it is assumed that R has overlap with earlier rules; R', R'', and R''' may be empty otherwise.

A packet p that is matched by R''' implies that p will not be passed or dropped by any preceding rule in the rule table and so R will be applied to p. A packet p that is matched by R' or R'' will be passed or dropped by a preceding rule in the rule table and so R will not be applied to it. We have the following properties for R', R'' and R'''.

A_1 A packet p that matches a rule table T, is matched by the effective part of exactly one rule R. That is, $p \in \mathcal{P}_T \Rightarrow (\exists R \in T$ such that $p \in \mathcal{P}_{R'''}) \wedge (\forall r \in T$ we have $r \neq R \Rightarrow p \notin \mathcal{P}_{r'''})$.

A_2 The effective part of a rule does not overlap with that of another rule in the same table. That is, $(\forall R_1, R_2 \in T,\ R_1 \neq R_2 \Rightarrow \mathcal{P}_{R_1'''} \cap \mathcal{P}_{R_2'''} = \phi)$.

A_3 It is possible to have $\mathcal{P}_{R'} \cap \mathcal{P}_{R''} \neq \phi$. However $\mathcal{P}_{R'} \cap \mathcal{P}_{R'''} = \phi$ and $\mathcal{P}_{R''} \cap \mathcal{P}_{R'''} = \phi$.

A_4 If $p \in \mathcal{P}_{R'''}$, then $Match(p, T) = R$.

A_5 $\forall R \in T,\ \mathcal{P}_{R'''} \subseteq \mathcal{P}_T$.

A_6 $\forall R \in T,\ \mathcal{P}_{R'''} = \{\mathcal{R}\} = \{\mathcal{R}'''\}$ and $\{\mathcal{R}'\} = \{\mathcal{R}''\} = \phi$.

Given a firewall rule table $T = (R_1 \cdots R_n)$, we call $T' = (r_1 \cdots r_n)$ the *effective representation* of T, where $r_i = R_i'''$ if R_i is a positive rule in T, otherwise $r_i = \phi$. Slightly abusing notations, if we denote the effective part of a negative rule by ϕ for negative rules do not enlarge the set of allowed packets, then T' can be written as $T' = (R_1''' \cdots R_n''')$.

Definition 3 *Consider two firewall rule tables T_1 and T_2 and let R denote a rule in T. Assume both tables are converted into their effective representations. A rule $R \in T_1$ has **equivalent rule set** in T_2 if there exists a subset $R_1, R_2 \ldots R_n \in T_2$ such that $\mathcal{P}_{R'''} \subseteq \mathcal{P}_{R_1'''} \cup \mathcal{P}_{R_2'''} \cup \ldots \cup \mathcal{P}_{R_n'''}$. Using property A_1 this implies that $\mathcal{P}_{R'''} \subseteq \mathcal{P}_{T_2}$. The equivalent rule set of a negative rule is defined to be ϕ.*

If $R \in T_1$ has an equivalent rule set in T_2 then packets in $P_{R'''}$ are also permitted by the equivalent rule set. If R does not have an equivalent rule set in T_2 then not all packets in $P_{R'''}$ are permitted by T_2. Note that Definition 3 applies to only positive rules.

5.4.1 An Example

We use an example to explain the above concepts in details. Consider the rule table shown in Table 5.3.

For R_5, source IP address range 192.168.13.0/24 = 192.168.13.1-20 ∪ 192.168.13.20-30 ∪ 192.168.13.31-255. Destination IP address range 192.168.13.0/24 = 192.168.13.30-40 ∪ 192.168.13.40-50 ∪ 192.168.13.1-29,51-255.

When packets from 192.168.13.1-20 and to 192.168.13.0/24, or from 192.168.13.0/24 and to 192.168.13.30-40 arrive at the firewall, they will be matched and passed by

Rule #	Source IP range	Destination IP range	Service	Action
1	192.168.13.1-20	any	RTP	Pass
2	192.168.13.20-30	any	RTP	Drop
3	any	192.168.13.30-40	RTP	Pass
4	any	192.168.13.40-50	RTP	Drop
5	192.168.13.0/24	192.168.13.0/24	RTP	Pass

Table 5.3: An example of rule table

R_1 and R_3. Although R_5 also matches them, it will not be applied. When packets from 192.168.13.20-30 and to 192.168.13.0/24, or from 192.168.13.0/24 and to 192.168.13.40-50 arrive at the firewall, they will be matched and dropped by R_2 and R_4. Although R_5 also matches them, it will not be applied. As a result, we can divide R_5 as shown in Table 5.4.

sub-rule	Source IP range	Destination IP range	Service	Action
R'	192.168.13.1-20	192.168.13.0/24	RTP	Pass
	192.168.13.0/24	192.168.13.30-40	RTP	Pass
R"	192.168.13.20-30	192.168.13.0/24	RTP	Pass
	192.168.13.0/24	192.168.13.40-50	RTP	Pass
R"'	192.168.13.31-255	192.168.13.1-29,51-255	RTP	Pass

Table 5.4: Dividing R_5 into sub-rules

Consider another example as shown in Table 5.5, it can be seen that the effective parts of rule R_1, R_3 and R_6 are themselves, i.e. they have no redundant or shadowed parts. Comparing Table 5.3 and Table 5.5, it can be seen that R_5 in Table 5.3 has equivalent rule set in Table 5.5, which are R_1, R_3 and R_6.

5.5 Comparing Firewall Rule Tables

In this section, we first prove theorems that specify the condition under which two rule tables are equivalent. We then use these results to give algorithms to determine if two rule tables are equivalent. If the two table are not equivalent, the algorithms will locate rules that are causing the conflict, that is rules that permit packets that are denied by the second rule table.

Rule Number	Source IP range	Destination IP range	Service	Action
1	192.168.13.31-255	192.168.13.1-29	RTP	Pass
2	10.1.1.0/24	10.1.2.0/24	RTP	Drop
3	192.168.13.31-100	192.168.13.51-255	RTP	Pass
4	10.1.2.0/24	10.1.1.0/24	RTCP	Pass
5	10.1.2.0/24	10.1.1.0/24	RTP	Pass
6	192.168.13.101-255	192.168.13.51-255	HTTP	Pass
7	any	any	any	Drop

Table 5.5: Another example of rule table

By the definition of two sets being equivalent, two firewall rule tables T_1 and T_2 are *equivalent* if $\mathcal{P}_{T_1} \subseteq \mathcal{P}_{T_2}$ and $\mathcal{P}_{T_2} \subseteq \mathcal{P}_{T_1}$. *Lemma 1* states the the condition under which packets permitted by T_1 is a subset of packets permitted by T_2, i.e. the $\mathcal{P}_{T_1} \subseteq \mathcal{P}_{T_2}$ condition.

Lemma 1 *Let $T_1 = (R_1, R_2, \ldots, R_n)$. If all R_i, $i = 1 \cdots n$ is either negative or has equivalent rule set in T_2, then $\mathcal{P}_{(R_1,R_2,\ldots,R_n)} \subseteq \mathcal{P}_{T_2}$*

Proof: The proof is by induction on n, the number of rules in T_1.

1. For the first rule R_1 in T_1 (situation when n=1)

   (a) Assume R_1 is positive

   As R_1 is the first positive rule, it is apparent that $\mathcal{P}_{R_1'} = \mathcal{P}_{R_1''} = \phi$, $\mathcal{P}_{R_1} = \mathcal{P}_{R_1'''}$

   Because R_1 has equivalent rule set in T_2, according to the definition of equivalent rule set in Section 5.4, it can be concluded that $\mathcal{P}_{R_1'''} \subseteq \mathcal{P}_{T_2}$. This implies $\mathcal{P}_{R_1} \subseteq \mathcal{P}_{T_2}$ as $\mathcal{P}_{R_1} = \mathcal{P}_{R_1'''}$.

   Therefore, when $n = 1, \mathcal{P}_{(R_1)} \subseteq \mathcal{P}_{T_2}$.

   (b) Assume R_1 is negative

   By definition, $\mathcal{P}_{(R_1)} = \phi$, which implies that $\mathcal{P}_{(R_1)} \subseteq \mathcal{P}_{T_2}$.

2. Assume this theorem is true for $n-1$ ($n \geq 1$), i.e. we assume that if $R_1, R_2, \ldots, R_{n-1}$ in T_1 all have equivalent rule set in T_2, then $\mathcal{P}_{(R_1,R_2,\ldots,R_{n-1})} \subseteq \mathcal{P}_{T_2}$.

Now we need to prove that the result is valid for n, i.e. we need to prove that if $R_1, R_2, \ldots, R_n$ all have equivalent rule set in T_2, then $\mathcal{P}_{(R_1,R_2,\ldots,R_n)} \subseteq \mathcal{P}_{T_2}$.

If R_n is negative, then adding R_n to the rule table does not enlarge the set of packets that are allowed to pass, i.e. $\mathcal{P}_{(R_1,R_2,\ldots,R_{n-1})} = \mathcal{P}_{(R_1,R_2,\ldots,R_n)}$. Thus according to the induction assumption, $\mathcal{P}_{(R_1,R_2,\ldots,R_n)} \subseteq \mathcal{P}_{T_2}$.

If R_n is positive, then

(a) For $\forall p \in \mathcal{P}_{R'_n}$ or $\forall p \in \mathcal{P}_{R''_n}$, if it can pass rule table consisting of $(R_1, R_2, \ldots, R_n)$, then according to *Fact 1* of R', R'' and R''', $\exists R'''_k$ $(1 \leq k \leq n-1)$ such that $p \in \mathcal{P}_{R'''_k}$. Because R_k has equivalent rule set in T_2 according to our assumption, then according to the definition of equivalent rule set, $\mathcal{P}_{R'''_k} \subseteq \mathcal{P}_{T_2}$, and hence p can also pass T_2.

(b) For $\forall p \in R'''_n$

If R_n has equivalent rule set in T_2, then according to the definition of equivalent rule set, $\mathcal{P}_{R'''_n} \subseteq \mathcal{P}_{T_2}$, and hence p can also pass T_2.

Finally, combining 1) and 2), we know that the theorem holds for $\forall n \in N$. □

Theorem 1 *Two firewall rule tables T_1 and T_2 are equivalent if (i) all positive rules in T_1 have equivalent rule sets in T_2; and (ii) all positive rules in T_2 have equivalent rule sets in T_1,*

Proof:

Let $T_1 = (R_1, R_2, \ldots, R_n)$. Since all of $R_1, R_2, \ldots, R_n$ have equivalent rule sets in T_2, using *Lemma 1* we have $\mathcal{P}_{T_1} \subseteq \mathcal{P}_{T_2}$.

Similarly, we can prove that $\mathcal{P}_{T_2} \subseteq \mathcal{P}_{T_1}$.

Hence, using the definition of equivalent firewall tables, we have that T_1 and T_2 are equivalent. □

Theorem 1 gives the condition under which two rule tables are equivalent. However, it is difficult to examine whether a rule has equivalent rule set in the second table because of implications discussed in Section 5.3.1.

Theorem 2 *A firewall rule table T and its effective representation are equivalent*

Proof:

Consider two rule tables $T_1 = (R_1, R_2, \ldots, R_n)$ and $T_2 = (r_1, r_2, \ldots, r_n)$, where $r_k = R_k'''$ if R_k is positive, otherwise $r_k = \phi$.

1. For each positive R_k, because $\mathcal{P}_{r_k'} = \mathcal{P}_{r_k''} = \phi$ (otherwise $\exists s < k$ such that $\mathcal{P}_{r_k} \cap \mathcal{P}_{r_s} \neq \phi$. i.e. $\mathcal{P}_{R_k'''} \cap \mathcal{P}_{R_s'''} \neq \phi$. This cannot be true according to *Fact 2* of R', R'' and R'''.), it can be seen that $r_k = r_k'''$, and $R_k''' = r_k = r_k'''$.

   Therefore, all positive rules in T_1 have equivalent rule set in T_2.

2. For each positive r_k, because $\mathcal{P}_{r_k'''} \subseteq \mathcal{P}_{r_k} = \mathcal{P}_{R_k'''}$, all positive rules in T_2 also have equivalent rule set in T_1.

   Thus according to the *Theorem 1*, T_1 and T_2 are equivalent. □

Theorem 2 can be used to compare two rule tables T_1 and T_2. To compare T_1 and T_2, we need to compare their effective representation T_1' with T_2'. Let $T_1 = (R_1, R_2, \ldots, R_n)$ and $T_2 = (r_1, r_2, \ldots, r_m)$. Then $T_1''' = (R_1''', R_2''', \ldots, R_n''')$ and $T_2''' = (r_1''', r_2''', \ldots, r_m''')$. Since $R_i''' \cap R_j''' = \emptyset$ for all i and j, for all rules in T_1, we need to determine if $\mathcal{P}_{R_i'''}$ is covered by a union of $\mathcal{P}_{r_j'''}$. That is if R_i''' has an equivalent rule set in T_2'. Similarly, for all rules in T_2 we need to determine if an equivalent rule set in T_1 exists.

5.5.1 Algorithms to Compare Firewall Rule Tables

In this section, we provide two algorithms where the first one determines the effective part of a rule, and the second one determines if two tables are equivalent and the rules that permit packets denied by the other table if inequality holds.

Algorithm 3 finds the effective part of a rule R_i in the rule table $T = (R_1, \ldots, R_n)$ where $n \geq i$. The Algorithm 3 finds R_i' and R_i'' and in the last step obtains R_i''' from R_i' and R_i''. The algorithm examines every R_j $(1 \leq j \leq i-1)$ and accumulates intersections $R_i \cap R_j$ $(1 \leq j \leq i-1)$ in R_i' and hence the part of R_i which is redundant to at least one preceding positive rule is put into R_i'. According to *Definition 2*, the resulting R_i' will be the *redundant part* of R_i. Similarly, the resulting R_i'' will be the *shadowed part* of R_i. Therefore, R_i''' will the *effective part* of R_i. Time complexity required for Algorithm 3 is $O(n) \times \beta$, where β denotes time complexity required for computation of $R_j \cap R_i$, which can be reduced to the problem of 2-dimensional rectangle intersection. A number of methods for calculation of d-dimensional rectangle

intersection is summarised in [65], the best time-complexity for the 2-dimensional case being $O(NlogN)$.

Algorithm 3: Finding the effective part of a positive rule

input : rule R_i — The positive rule whose effective part is to be computed
input : array of rules $(R_1 \dots R_{i-1})$ — The $i-1$ preceding rules in the same rule table as R_i
output: R_i''' — Effective part of R_i

set $R_i' = R_i'' = R_i''' = \phi$
for *each rule* R_j *(*$1 \leq j \leq i-1$ *)* **do**
  if R_j *is positive* **then**
    $R_i' = R_i' \cup (R_j \cap R_i)$
  else
    $R_i'' = R_i'' \cup (R_j \cap R_i)$
set $R_i''' = R_i - R_i' - R_i''$
return R_i'''

Algorithm 4 determines if two rule tables T_1 and T_2 are equivalent. First, using Algorithm 3 the rules in T_1 and T_2 are converted into their corresponding effective forms. Let $T_1 = (r_1, r_2, \dots, r_n)$ and $T_2 = (R_1, R_2, \dots, R_n)$. We first construct T_1' $= (r_1''', r_2''', \dots, r_n''')$ and $T_2' = (R_1''', R_2''', \dots, R_n''')$. Then for each rule r_i''' in T_1', we examine if it has an equivalent rule set in T_2'. This step is repeated for all rules in T_2' and if both steps produce a true results, we conclude the two tables are equivalent. Correctness of this conclusion follows directly from *Theorem 1* and *Theorem 2*. Time complexity in relation to the number of rules required for Algorithm 4 is $O(n^2)$.

If T_1 and T_2 are not equivalent, Algorithm 4 returns the sets of packets permitted only by T_1 or T_2 but not both, and also rules that permit these packets.

5.6 Implementation and a Complete Example

5.6.1 An implementation

We implemented a GUI prototype of the techniques presented above in Java, which is illustrated in Figure 5.2. It can be seen that the prototype can be used to analyse effective parts of rules in a rule table, equivalent rule set in another rule table, and whether two rule tables are equivalent. We will explain in details how this prototype can be used to compare rule tables and debug firewall configurations with a complete example presented next.

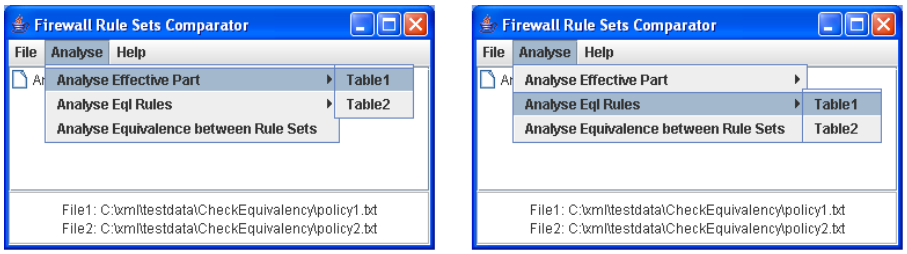

Figure 5.2: The Prototype

5.6.2 A Complete Example

Consider the campus network configuration illustrated in Figure 5.3. The network consists of 3 servers accessible from outside, and two student labs - the project lab and the security lab. The project lab provides a platform for students to carry out daily activities such as browsing Internet or writing assignments. The security lab is for students studying network security to perform experiments such as sniffing or packet spoofing. It is hence desirable to separate traffic of the security lab from that of the project lab.

The security policy is described in Table 5.6. Description of the policy might be ambiguous and not clear at this stage. This is often an important reason leading to incorrect translation of network policy into firewall rules. We will demonstrate later how the prototype can be used to clarify the ambiguity, and to elicit unstated requirement in policy description.

Assume there are a chief administrator and an associate administrator in the network department. The chief administrator has more experience and follows "who can access what" to translate the security policy into firewall rule table. On the other hand, the associate administrator has less experience and sequentially translates each item in the policy description into corresponding firewall rules. The resulting firewall rule tables are listed in Table 5.7 and 5.8 respectively.

Now we need to compare the rule tables to gain a confidence in their correctness, as well as to explore possible ambiguity in policy description and incorrect translation of the policy. To compare the rule tables, we first load them into the prototype, then select "Analyse" $\Rightarrow$ "Analyse Equivalence between Rule Tables" from the menu, as illustrated in Figure 5.4. The result shows that the rule tables are not equivalent, because rule 1,8,9,10 in rule table 1 and rule 1,3,10 in rule table

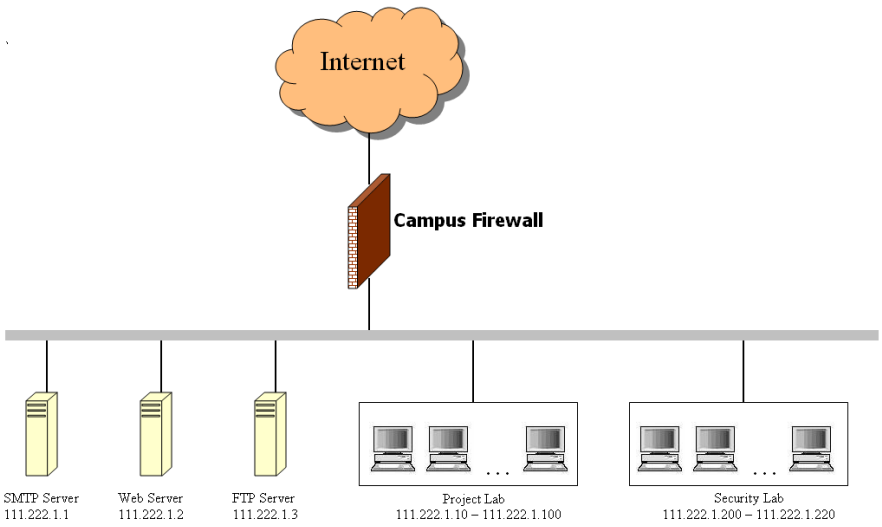

Figure 5.3: An example of network setup

2 have no equivalent rule set in the other rule table.

We then select "Analyse" ⇒ "Analyse Eql Rules" from the menu to analyse the packets applied to and permitted only by these rules but not by the other rule table. The prototype shows us, for each of the above rules, the set of packets that cannot pass the other rule table, as illustrated in Figure 5.5. By a close look at the rule tables and the results in Figure 5.5, we revealed a number of problems in the policy description as well as its translation into firewall rules.

- The policy does not explicitly grant the use of DNS, which in practise is a pre-requisite for using HTTP and FTP. The permission to use DNS should be added explicitly to the policy description to reduce ambiguity.

- The policy does not mention which hosts the servers can access. The chief administrator grants them access to the project lab, but the associate administrator does not. This conflict should be discussed further and addressed with department managers.

- The associate network administrator translates each item in the policy description into firewall rules in order. By a mistake in representing "external hosts" with "*", the negative rules that prohibit external hosts to access the servers also prohibit hosts in both student labs to access themselves. By a

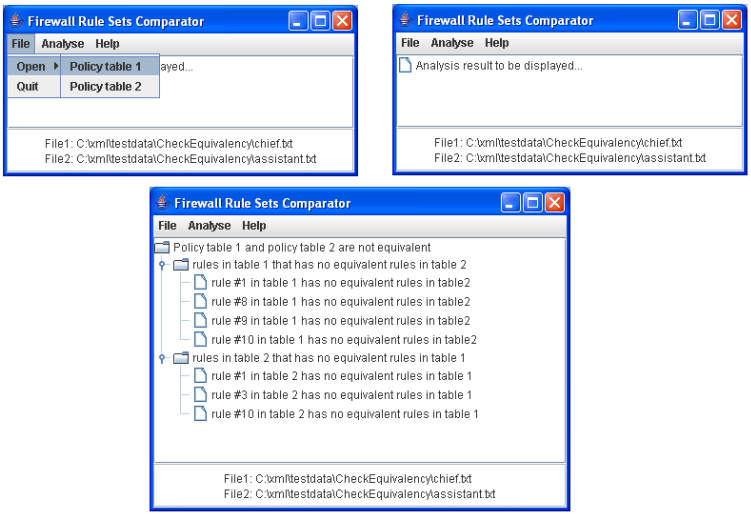

Figure 5.4: Rule table compare result

similar mistake, he also permits hosts in the security lab to access the SMTP and FTP server.

- The statement "hosts in the project lab can access everything" in policy description is ambiguous, as "everything" can be taken as "everything in the internal network" or "everything including both internal and external network". It can be seen that the chief administrator is more rigorous and take it as "everything in the internal network", while the associate administrator takes the other understanding.

Counter-measures against these problems can be taken effectively after they are revealed. The prototype can therefore assist in clarifying policy ambiguity and debugging firewall configurations by accurately comparing firewall rule tables and locating rules that cause inequality.

5.7 Conclusions and Further Work

In this Chapter, we presented an effective technique to compare firewall rule tables. Our technique can not only determine whether rule tables represented in different forms are equivalent, but also accurately locate the rules in their original form and order that are causing the inequality. Multiple rule tables written by different administrators to implement the same security policy can be compared to gain an

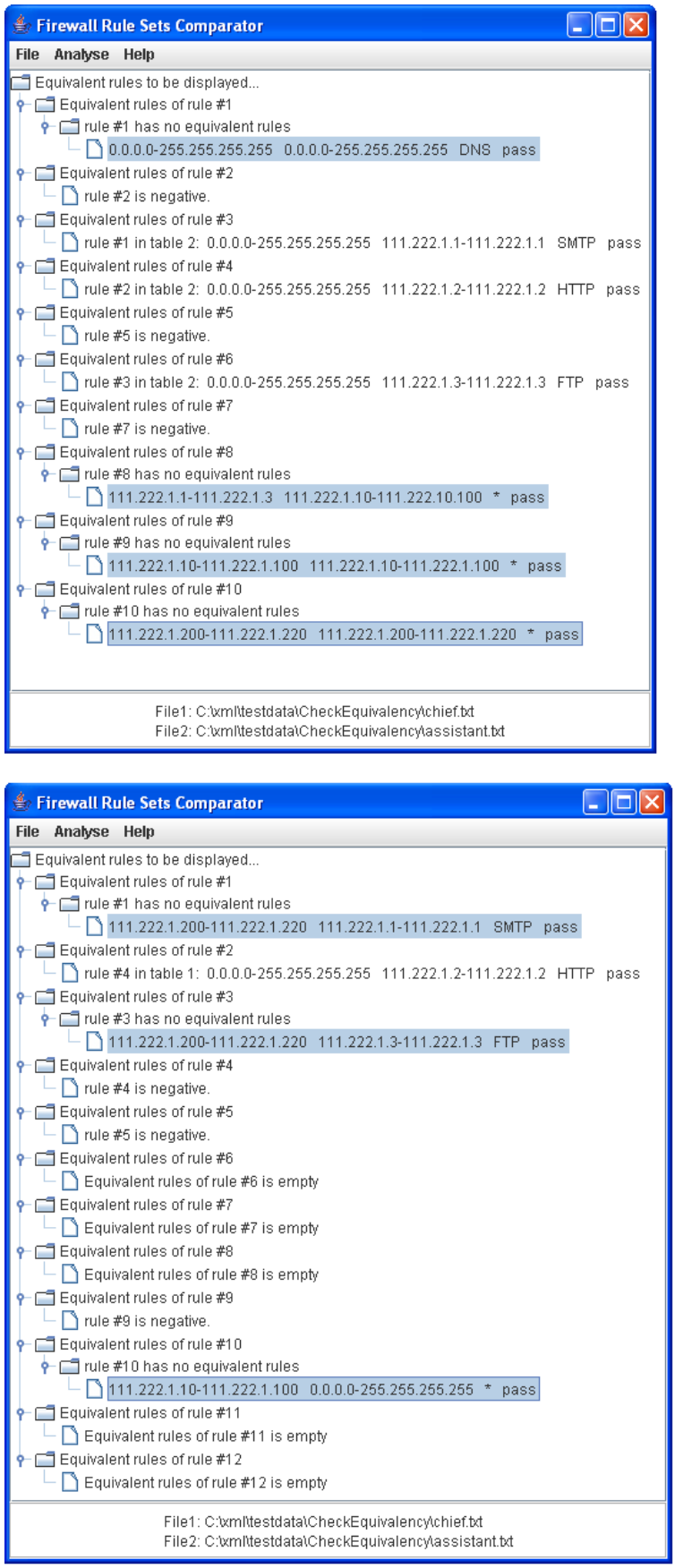

Figure 5.5: Rule table compare result

increased level of confidence in their correctness when the rule tables are equivalent; otherwise the rules causing conflicts between rule tables can be accurately located, as can assist in resolving conflicts between rule tables. Our technique can also be used to analyse changes to a rule table, and determine whether desired changes are made correctly by comparing the original rule table and the modified one. We also implemented our technique with a GUI prototype written in Java, and demonstrated a complete example of using the prototype to compare and debug firewall configurations.

5.7.1 Deficiency of Firewall Techniques and Further Work

Although techniques have been provided to handle streaming content, conventional firewalls still have inherent limitations like any other technologies. Defending against multi-stage attacks is a prominent example among intrusion scenarios that conventional firewalls are incapable to handle. In this scenario, each "atomic" action by an attacker does not trigger a direct violation of firewall rules, nor imposes a sensible threat to the system. In other words, each "atomic" action may only escalate the attacker's privilege by a such a small level that itself has only little impact on system security and is not detected by firewall. However combining several "atomic" actions an attacker may gain privileges to the system that is not supposed to be granted.

In the rest of the thesis, we will look at an emerging technology, namely attack graph, that supplements incapability of firewalls to defend against multi-stage attacks. The following Chapter covers technical details of attack graphs and our contribution to the area. An concrete example will also show how firewalls and attack graphs can be used in conjunction to fight against multi-stage attacks.

Algorithm 4: Check the equivalency of two rule tables T_1 and T_2

input : Rule table $T_1 = r_1 \dots r_{n_1}$, and $T_2 = R_1 \dots R_{n_2}$

output: A boolean value indicating whether T_1 and T_2 are equivalent, and if not, difference between T_1 and T_2 and rules that are causing the difference

```
/* Boolean value indicating if T1 and T2 are equivalent        */
```
set $isEquivalent = true$

```
/* The rules that permit packets denied by the other rule table
   */
```
set $DiffRules_1 = DiffRules_2 = \phi$

```
/* The set of packets permitted by only one rule table         */
```
set $Diff_1 = Diff_2 = \phi$

for *each* r_i *in* T_1 **do**
  $r_i''' =$ FindEffectivePart(rule r_i, rule[] $r_1 \dots r_{i-1}$)

set $T_1' = (r_1''', r_2''', \dots, r_n''')$

for *each* R_i *in* T_2 **do**
  $R_i''' =$ FindEffectivePart(rule R_i, rule[] $R_1 \dots R_{i-1}$)

set $T_2' = (R_1''', R_2''', \dots, R_n''')$

for *each* r_i''' *in* T_1' **do**
  for *each* R_i''' *in* T_2' **do**
    $r_i''' = r_i'''$ - R_i'''
  if $r_i''' \neq \phi$ **then**
    $isEquivalent = false$;
    $DiffRule_1$.insert(r_i) ;
    $Diff_1 = Diff_1 \cup tempr_i'''$

for *each* R_i''' *in* T_2' **do**
  for *each* r_i''' *in* T_1' **do**
    $R_i''' = R_i'''$ - r_i'''
  if $R_i''' \neq \phi$ **then**
    $isEquivalent = false$;
    $DiffRule_2$.insert(R_i) ;
    $Diff_2 = Diff_2 \cup tempR_i'''$

return $isEquivalent$, $DiffRules_1$, $DiffRules_2$, $Diff_1$, $Diff_2$

Policy Description
▶ External hosts can only access the servers
▶ External hosts cannot access any machine in either lab
▶ Hosts in the project lab can access everything except the security lab
▶ Hosts in the security lab can only access the web server and other hosts in the security lab

Table 5.6: The security policy

Rule #	Source IP range	Destination IP range	Service	Action
1	*	*	DNS	pass
2	111.222.1.200-111.222.1.220	111.222.1.1	SMTP	deny
3	*	111.222.1.1	SMTP	pass
4	*	111.222.1.2	HTTP	pass
5	111.222.1.200-111.222.1.220	111.222.1.3	FTP	deny
6	*	111.222.1.3	FTP	pass
7	111.222.1.200-111.222.1.220	111.222.1.10-111.222.1.100	*	deny
8	111.222.1-111.222.1.3	111.222.1.10-111.222.1.100	*	pass
9	111.222.1.10-111.222.1.100	111.222.1.10-111.222.1.100	*	pass
10	111.222.1.200-111.222.1.220	111.222.1.200-111.222.1.220	*	pass

Table 5.7: The rule table by the chief administrator

Rule #	Source IP range	Destination IP range	Service	Action
1	*	111.222.1.1	SMTP	pass
2	*	111.222.1.2	HTTP	pass
3	*	111.222.1.3	FTP	pass
4	*	111.222.1.10-111.222.1.100	*	deny
5	*	111.222.1.200-111.222.1.220	*	deny
6	111.222.1.10-111.222.1.100	111.222.1.1	SMTP	pass
7	111.222.1.10-111.222.1.100	111.222.1.2	HTTP	pass
8	111.222.1.10-111.222.1.100	111.222.1.3	FTP	pass
9	111.222.1.10-111.222.1.100	111.222.1.200-111.222.1.220	*	deny
10	111.222.1.10-111.222.1.100	*	*	pass
11	111.222.1.200-111.222.1.220	111.222.1.200-111.222.1.220	*	deny
12	111.222.1.200-111.222.1.220	111.222.1.2	HTTP	deny

Table 5.8: The rule table by the assistant administrator

Chapter 6

Using Attack Graphs to Analyse Network Security

6.1 Introduction

A large computer system is often build upon multiple platforms, runs different operating systems and software packages and has a complex connection policy to an open network. Despite the best efforts by system designers to deploy security measures such as firewalls, an adversary or an attacker is often able to gain a level of undesired access by system policy by exploiting bugs or design flaws in the system. The act of exploiting a single system bug or design flaw is referred to as an "atomic attack" or an "exploit". An atomic attack often has only insignificant impact on the system and may not be detectable by firewalls. However, an adversary may be able to construct a severe system intrusion by combining a series of atomic attacks, each escalating the privilege of the attacker to a slightly higher level. It can be seen that to evaluate the security level of a large computer system, the system administrator must deploy a more effective mechanism than firewalls to handle intrusion scenarios where an adversary may inflict severe damage to the system by combining a series of insignificant and undetectable atomic attacks.

Sheyner *et al.* proposed using attack models and attack graphs to provide a global view of system security against exploiting the combination of vulnerabilities [75] [99]. An attack model is a graph that consists of a set of nodes and edges. Each node represents a reachable system state and each edge represents an atomic attack that takes the system to another state where the attacker's privilege may be escalated. An attack graph is a sub-graph of attack model and contains only nodes on paths that eventually reach a state where the modeled system is considered compromised. With attack models and graphs, a global view on multi-stage attacks by combination of individual vulnerabilities can be obtained by administrators to

assist in implementation of effective security counter measures.

Another application of attack graph is to provide clues to questions such as "what happens if a particular vulnerability is patched" or "what happens if a firewall rule is altered, added or removed". These are often raised in an organisation before any planned network structural change takes place. Being able to answer such questions (without actually implementing the reconfiguration) is necessary for the network administrator to evaluate the results of the planned network changes. Clues to answering such questions can be obtained from analysing attack graph resulted from situation changes in the modeled system.

6.1.1 Related Work

Techniques for quantitative security measurement have been a strong focus in the research community [66, 11, 83, 14]. Dacier *et al.* [66] modeled a computer system using a *Privilege Graphs* exhibiting security vulnerabilities and convert it into a Markov chain corresponding to all possible successful attack scenarios. The Markov chain is then used to compute MTTF (mean time to failure) of the system, used as the quantitative measure of the security level of a system. Time and effort required by each type of attack is estimated from empirical and statistical data. Phillips *et al.* [22] present a framework for evaluating the most likely attack paths in the attack graph generated by an ad hoc algorithm. The framework requires attacker profiles and attack templates in order to compute the likelihood of each type of attack. Madan *et al* [11] proposed an approach to quantifying various security related attributes of a computer system such as system availability, MTTF, and probabilities of system failure. Quantification of security related attributes is by solving the Semi-Markov Process (SMP) model describing state transitions in the system. However, the proposed approach requires availability of a wide range of ad hoc model parameters, restricting the approach feasible only for systems of a small scale. Another related work presented in [83] provides a quantitative analysis of attacker behavior based on empirical data collected from intrusion experiments.

A limitation of using attack graphs is that the size and complexity of attack models/graphs usually greatly exceed human ability to visualise, understand and analyse. It is highly desirable to have a scheme to identify important portions of attack models/graphs. An effective method is to rank states in attack models/graphs based on factors like the probability of an intruder reaching the state. Important

portions of attack models/graphs can hence be identified by ranks of their states. Analysis of attack graphs can then be focused on the important section. Mehta *et al.* [111] propose to rank states of an attack model by the probability of an adversary reaching a state by a sequence of atomic attacks. Their ranking algorithm is based on the PageRank algorithm used by Google to measure importance of web pages on the World Wide Web. Given a system to be analysed, first an attack model formally describing the system is constructed. Then the ranking algorithm is applied to rank states of the obtained attack model. Meanwhile, an attack graph is generated using the attack graph generation tool [75], and is "projected" on the ranked attack model to obtain a ranked attack graph. The ranked states of an attack graph provide various security analysis for the system, such as measuring security of the system, evaluating effectiveness of counter-measures, and identifying important portion for visual analysis of the system.

Despite the similarity between ranking web pages and ranking system states, differences not considered by Mehta's scheme exist between the two scenarios. The World Wide Web model adopted by PageRank assumes that a random surfer has universally equal probabilities of following one of the links in a current page to the next page, and correspondingly Mehta's ranking scheme assumes that an attacker has equal probability of remaining undetected at all states of an attack model. However, the likelihood of an attacker remaining undetected at a state so as to exploit a vulnerability that takes the system to another state could be considered to be influenced by the number of steps required to reach the state from the starting position. Consider a scenario where a network has implemented some sort of defense-in-depth as an example. The more steps an attack has taken, the more likely that an attacker is discovered. Therefore, we assume the probability of an attacker remaining undetected at a state decreases with number of steps required to reach that state. The decreasing rate is not universal amongst all computer systems but determined by each system's intrusion detection ability. This affects state transitions in attack models and should be considered when ranking system states.

Moreover, the random transition model adopted by PageRank assumed that a WWW surfer navigates to the next page by selecting one of the available succeeding pages at random with equal probabilities. This assumption fits well with intrusions that use a brute-force probe-scan approach. However, an adversary may exploit vulnerabilities based on metrics such as cost, age, evaluation of probability of success and being detected. In this case, vulnerabilities are not selected at random and

different vulnerabilities have different probabilities of being exploited. The behavior of an adversary selectively exploiting vulnerabilities has considerable effect on the chance to reach the final goal, and therefore should be considered when ranking system states of attack models and graphs.

6.1.2 Our Contribution

In this Chapter, we propose a ranking scheme that addresses problems stated above. The proposed ranking scheme is adjusted from Mehta *et al.*'s scheme, but has the advantage of modeling variation in intrusion detection abilities amongst computer systems, and non-uniform distribution in probability that each vulnerability is exploited. First, in addition to modeling vulnerabilities in a system that could be exploited to have system states changed, our ranking scheme also models intrusion detection ability of computer systems defined as the system's effort to detect and prevent such state transitions by intruders exploiting vulnerabilities. Secondly, we provide an instantiation of the biasing idea in [73] by modeling adversaries' behavior in exploiting vulnerabilities probabilistically based on certain metrics as stated above but not by brute-force probing. The proposed ranking scheme produces more accurate ranks of attack graphs when certain system metrics, such as evaluation of system intrusion detection ability or adversaries' ability in relation to probabilistically exploit vulnerabilities, is available from for example empirical data or log statistics. The proposed scheme can also be applied to other areas such as network research or system design, e.g. determining minimum system intrusion detection strength required to protect against best effort by an adversary.

To evaluate the effectiveness of the proposed ranking scheme, we implemented a prototype of the scheme in Java. We experiment with the network example used by Mehta *et al.* [111] and have the results compared with their scheme. The experiments yielded promising results that demonstrated consistent ranks amongst varying parameters modeled by the proposed ranking scheme.

One mechanism to answer questions like "what happens if a particular vulnerability is patched" or "what happens if a firewall rule is altered, added or removed" is to vary the network model accordingly and regenerate the attack graph. The regenerated attack graph often needs to be re-ranked. A network may be re-engineered in many different forms, each resulting in generation of a new attack graph. The PageRank algorithm is not of linear time complexity and thus it may be difficult to

rank many attack graphs, each resulting from one of many possible changes in the modeled system [62].

To solve this problem, we consider an alternative scheme to estimate ranks of attack graphs based on the Graph Neural Network (GNN) [29], a new neural network model capable of processing graph structures. The applications of GNN have been successful in a number of areas where objects are naturally represented by a graph structure, the most notable example being the ranking of web pages [28]. In [28] it is shown that the GNN can be trained to simulate the PageRank function by using a relatively small set of training samples. Once trained, the GNN can rank large sets of web pages efficiently due to its ability to generalise over unseen data. In other words, the GNN learns the ranking function (whether explicit, as in the case of PageRank, or implicit) of web pages from a small number of training samples, and can then generalise over unseen examples, possibly contaminated by noise.

Because of the above stated properties, GNN may be considered suitable for the task of ranking attack graphs. Moreover, [27] provides a universal approximation theorem that the GNN is capable of learning to approximate any "reasonable" problem (reasonable in the sense that the problem is not a pathological problem) to any arbitrary degree of desired precision. The universal approximation theorem further justifies the suitability of ranking attach graphs using GNN. However on the other hand, some properties of attack graphs differ fundamentally from those of the web graph. The web graph is a single, large graph with a recursive link structure (web pages referring to themselves) whereas attack graphs can be numerous, are relatively small, and may be of strictly tree structured. This together with the observation that PageRank is a link based algorithm whereas the GNN is a node based approach in that it processes nodes more effectively than links, whether the GNN is suitable for the task of ranking attack graphs is unknown. One of the key questions that this Chapter wishes to answer is the suitability of GNN for the purpose of ranking attack graphs. There were numerous prior approaches to the processing of attack graphs and similar poli-instantiation problems. Research in this area was particularly active in the 1980s and early 1990s (see for example [81]). Most of these work were of an automated proof nature, desiring to show if an attack graph is vulnerable or not. Such automated proof concept can be expressed in terms of graph structured data [78]. However, any attempts to use a machine learning approaches required the pre-processing of the graph structured data to "squash" the graph structure into a

vectorial form as most popular machine learning approaches, e.g. multilayer perceptrons, self organising maps [38], take vectorial inputs rather than graph structured inputs. Such pre-processing steps can result in the loss of contextual information between atomic components of the data. The GNN is a relatively recent development of a supervised machine learning approach which allows the processing of generic graphs without first requiring the "squashing" of graph structured data into vectorial form. It is shown in [27] that the GNN algorithm is guaranteed to converge, and that it can model any set of graphs to any arbitrary degree of precision. Hence, to the best of our knowledge, this work is the first reported one on ranking attack graphs (without pre-processing) using a machine learning approach.

Training a GNN can take a considerable period of time. It is shown in [29] that the computational burden of the training procedure can be quadratic. However, once trained, a GNN is able to produce output in linear time. This is an improvement over $O(N\log\frac{1}{\epsilon})$, the computational complexity of PageRank, where N is the number of links and ϵ is the expected precision [62]. Moreover, GNN is able to learn the ranking scheme from a small number of training examples and then generalise to other unseen attack graphs. For reasons stated above, using GNN may be more suitable than the PageRank algorithm in the case where it requires ranking many attack graphs, each resulting from one of many possible changes in the modeled system. A network administrator is hence able to better evaluate changes made to the system by focusing on important portions of "prospective" attack graphs.

The rest of this Chapter first present an attack graph ranking scheme that addresses situations which PageRank does not take into consideration in attack graph ranking scenario. Experiments are then conducted and results presented to demonstrate improvement upon PageRank. Finally an alternative ranking scheme using GNN is provided. Part of this Chapter appeared in [54, 56].

6.2 Modeling Adversary and Intrusion Detection Capability in Ranking Attack Models

6.2.1 Background and Preliminaries

Attack Models and Attack Graph

Sheyner *et al.* first formally defined the concept of *Attack Model* and *Attack Graph* [75]. An *Attack Model* is a formal description of security related attributes of the attacker, the defender and the modelled system using graph representation. Nodes represent the states of the system, such as the attacker's privilege level on individual system components. Transitions correspond to actions taken by the attacker which lead to a change in the state of the system. Note that not necessarily every transition is an exploit; however any unauthorised transition in state is still undesired by system administrators. As such we do not distinguish transition in state from exploit in our discussion. The starting state of the model denotes the state of the system where no damage has occurred and the attacker is looking for an entry point to enter the system. As an example, if we consider the case of a computer network attack model, a state represents the state of the attacker, the running services, access privileges, network connectivity and trust relations. The transitions correspond to actions of the attacker such as exploiting vulnerabilities to obtain elevated privileges on the computer system. Formally,

Definition 4 [75] *Let AP be a set of atomic propositions. An Attack Model is a finite automaton* $M = (S, \tau, s_0, l)$*, where* S *is a set of states in relation to a computer system,* $\tau \subseteq S \times S$ *is the transition relation,* $s_0 \in S$ *is the initial state, and* $l : S \to 2^{AP}$ *is a labelling of states with the set of propositions true in that state.*

The negation of an attacker's goal in relation to an attack model can be used as *security properties* that the system must satisfy in a secure state. An example of a security property in computer networks would be "the intruder cannot gain root access on the database server". States in an attack model where the *security properties* are not satisfied are called *error states*. Given an attack model and the attacker's goal, an *Attack Graph* is a subgraph of the attack model which contains only paths leading to one of the error states. Formally,

Definition 5 [75] *Let AP be a set of atomic propositions. An Attack Graph is a finite automaton* $G = (S, \tau, s_0, S_s, l)$*, where* S *is a set of states in relation to a*

computer system, $\tau \subseteq S \times S$ is the transition relation, $s_0 \in S$ is the initial state, $S_s \subseteq S$ is the set of error states in relation to the security properties specified for the system, and $l : S \rightarrow 2^{AP}$ is a labelling of states with the set of propositions true in that state.

Given an attack model and the associated security properties, model checking techniques can be used to generate attack graphs automatically [94].

Web Graph and PageRank

Web Graph and Notations For a web model consisting of N nodes (pages), notations used in the our discussion are defined in Table 6.1. Consider the web graph shown in Figure 6.1 as an example. Node 1 has three out links (node 2, 3 and 4) and hence $h_1 = 3$. Node 4 is pointed to by two nodes (node 1 and 3) and hence $pa[4] = \{$node 1, node 3$\}$. Equation 6.1 represents the transition matrix W in relation to the web graph. It is noticeable that not all nodes have out links, and we refer to these nodes as *dangling nodes*. When reaches at a web page that has no out links, a web surfer is often assumed to select a random page to continue surfing [89] [6]. To model this, a *dangling node* in a web graph is typically assumed to be pointing to all other nodes with equal probabilities.

Notation	Meaning
h_j	Number of nodes (pages) pointed to by node (page) j
$pa[j]$	Set of nodes (pages) pointing to node (page) j.
d	Probability that a random surfer continues surfing by navigating to one of the pages linked by the current page, usually referred to as *damping factor*. Correspondingly, $1 - d$ represents the probability that a random surfer continues surfing and navigates to a random page
W	$W = \{w_{i,j}\}$ is a transition matrix such that $w_{i,j} = \frac{1}{h_j}$ if there is a link from node j to node i, otherwise $w_{i,j} = 0$. An important property of W is that $\forall j$, $\sum_{i=1}^{N} w_{i,j} = 1$.
Π_N	[1, ..., 1]', i.e. transpose of the N-dimension unit vector

Table 6.1: Web Model Notations

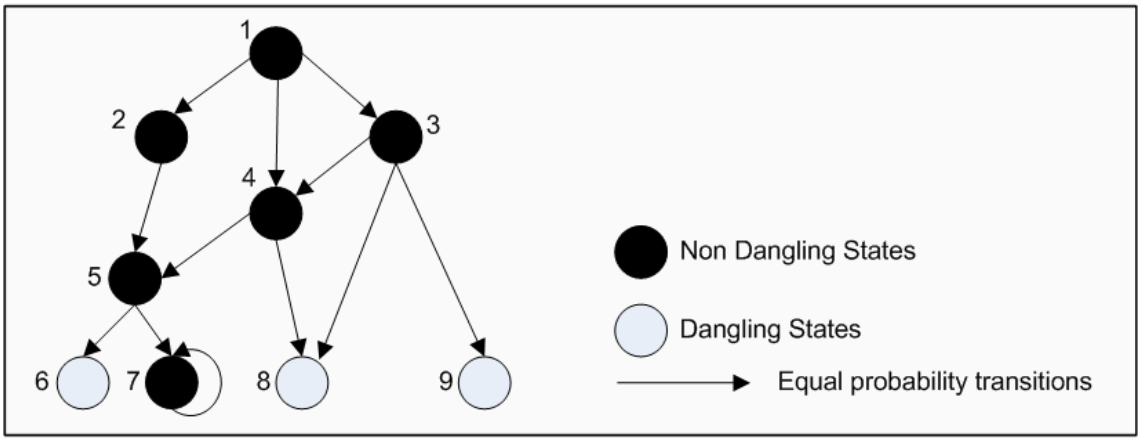

Figure 6.1: An example of web graph

$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.1}$$

PageRank Algorithm PageRank [89] is the algorithm used by Google to determine the relative importance of web pages on the World Wide Web. PageRank is based on the behavior model of a random surfer in a web graph. It assumes that a random surfer starts at a random page and keeps clicking on links and eventually gets bored and starts on another random page. To capture the notion that a random surfer might get bored and restart from another random page, a *damping factor* d is introduced, where $0 < d < 1$. The transition probability from a state is divided into two parts: d and 1 - d. The d mass is divided equally among the state's successors. Random transitions are added from that state to all other states with the residual probability 1 - d equally divided amongst them, modelling that if arrives at a dangling page where no links are available, a random surfer is assumed to pick another page at random and continue surfing from that page. The computed rank of a page is the probability of a random surfer reaching that page. That is, consider

a web graph with N pages linked to each other by hyperlinks, the PageRank x_p of page (node) p is defined as the probability of the random surfer reaching p, formally

$$x_p = d \sum_{q \in pa[p]} \frac{x_q}{h_q} + \frac{1-d}{N} \tag{6.2}$$

When stacking all the x_p into a vector $\mathbf{x}$, it can be represented as

$$\mathbf{x} = dW\mathbf{x} + \frac{1}{N}(1-d)\Pi_N \tag{6.3}$$

Using iterative expression, Equation 6.3 can be represented as

$$\mathbf{x}(t) = dW\mathbf{x}(t-1) + \frac{1}{N}(1-d)\Pi_N \tag{6.4}$$

The computation of PageRank can be considered a Markov Process, as can be seen from Equation 6.4. It has been proved that after multiple iterations, Equation 6.4 will reach a stationary state where each x_p represents the probability of the random surfer reaching page p [62].

Mehta et al's Ranking Scheme

Given an attack model $M = (S, \tau, s_0, l)$, the transition probability from each state is divided into d and 1-d, modelling respectively that an attacker is discovered and isolated from the system, or that the attacker remains undetected and proceeds to the next state with the intrusion. Similar to PageRank, the rank of a state in an attack model is defined to be the probability of the system being taken to that state by a sequence of exploits. The ranks of all states are computed using the method for computing PageRank described in Section 6.2.1. Breadth first search starting from the initial system state s_0 is then performed for each atomic attack in τ to construct the transition matrix W. The only adjustment from PageRank, where a transition from each state pointing to all other states with probability 1-d equally divided amongst all other states, is that a transition from each state pointing back to the initial state with probability 1-d is added to model the situation where an attacker is discovered and has to restart the intrusion from the initial state.

6.2.2 Modelling Adversary and Intrusion Detection Capability in Ranking Attack Models

Recall the discussion in Section 6.1. Unlike the web graph model adopted by PageRank where the probability that a random surfer follows a link to the next page is

state independent, the likelihood of an attacker remaining undetected at a state so as to exploit a vulnerability that takes the system to another state could be considered to be influenced by the number of steps required to reach the state from the starting position. Therefore we assume the probability of an attacker remaining undetected at a state decreases with number of steps required to reach that state. The decreasing rate is not universal amongst all computer systems but system specific as determined by each system's intrusion detection ability. Another important dissimilarity between a web surfing scenario and a system intrusion scenario is that exploits taking a computer system from one state to another may be "selected" not at random but based on the adversary's evaluation on metrics such as cost, effort, probability of success and being detected, whereas links taking a web surfer to the next page is always selected at random with equal probabilities. These factors affect an adversary's chance to reach the final goal, and therefore should be considered when ranking states of attack models and graphs.

In this section, we propose an adversary aware and intrusion detection aware ranking scheme that addresses problems stated above. Being adversary aware, the proposed scheme considers how an adversary selectively exploiting vulnerabilities affect the chance to compromise the system. Being intrusion detection aware, the proposed scheme considers system intrusion detection ability and how it affects an adversary's chance to reach the final goal.

6.2.3 Web Graph Adjustment

The transition model of web graph needs to be adjusted to provide a more accurate simulation of computer system state transitions in relation to system intrusion scenario. As in Mehta *et al.*'s ranking scheme, we add a transition from each state pointing back to the initial state with probability 1-d, modelling the situation where an intrusion is detected and needs to be restarted from initial state. Furthermore, our ranking scheme differs from Mehta *et al.*'s scheme in that

1. We assume that the probability of an attacker remaining undetected at a state decreases with the number of steps required to reach that state, which in an attack model can be represented as length of the intrusion path to reach that state. The decreasing rate is determined by each system's intrusion detection ability. In general, well-protected systems such as systems implementing "defense-in-depth" have better ability to detect intrusions at earlier stages

and can be simulated with greater decreasing rates. As it is difficult to predict the actual intrusion path, we simplify the situation by assuming that at each state s_j the probability of an attacker remaining undetected decreases at a rate proportional to $l(s_0, s_j)$, length of the shortest path between state s_j and initial state s_0. That is, the probability of an attacker remaining undetected at state s_j exponentially decays with length of the shortest path from s_0 to s_j. Consequently, transition probability from each state s_j is divided into d_j and 1-d_j representing respectively the situation where an attacker remains undetected and is able to take the system to another state, or where the attacker is discovered and has to restart the intrusion. d_j is the value of d exponentially decaying with $l(s_0, s_j)$ where d is the usual *damping factor*.

2. In a system intrusion scenario, it is more likely that an adversary has the ability to prioritise and exploit "promising" vulnerabilities based on past experience and knowledge, other than probing the target network with brute-force attack. This is modelled in our ranking scheme by assigning a separate probability to each type of exploit. We divide each d_j among state s_j's successors according to the probability that each type of exploit is selected to take s_j to one of its successors. The probability distribution can be obtained from empirical data or other sources [84]. By doing so, that the adversary probes the system with brute-force attack can be modelled by assigning equal probabilities to all exploits. Similarly, intrusions by an experienced attacker who exploits vulnerabilities selectively to maximise the chance of success can be modelled by assigning higher probabilities to critical exploits.

3. We add a transition from each dangling state pointing back to the initial state s_0 with probability 1, modelling the situation that an adversary has come to a state where the adversary cannot proceed with the intrusion and has to restart from initial state.

Consider the graph illustrated in Figure 6.1 as an example. Assume we have some empirical data that enables us to estimate that whenever the system is in s_1, on average it will take the transition to s_2, s_3 and s_4 2, 5 and 3 times, respectively, out of ten. We can then place probabilities 0.2, 0.5 and 0.3 on these transitions. Similarly, assume that the empirical data enables us to place probability 0.3, 0.4 and 0.3 to the transitions taking s_3 to s_4, s_8 and s_9 respectively, probability 0.8 and 0.2 to the

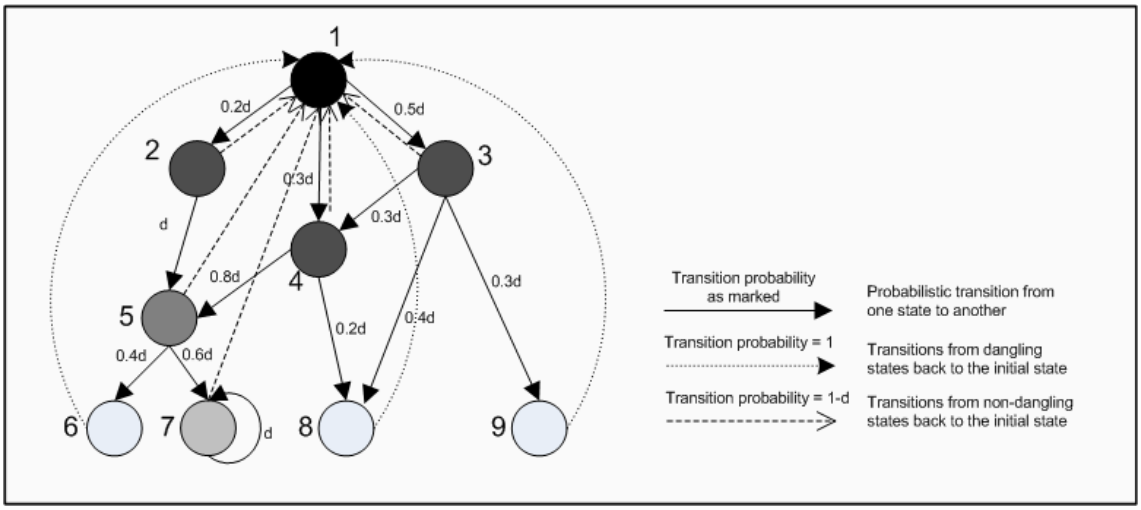

Figure 6.2: Transitions in attack models

transitions taking s_4 to s_5 and s_8 respectively, and probability probability 0.4 and 0.6 to the transitions taking s_5 to s_6 and s_7 respectively. Figure 6.2 illustrates the graph with adjusted transition model from web graphs, which is a more accurate simulation of computer system state transition in an intrusion scenario. The intensity of color for each state s_j visualise the probability d_j that an intrusion is not detected at that state.

6.2.4 Transition Matrix Construction

To rank an attack model $M = (S, \tau, s_0, l)$, we need to construct the *transition matrix* $\overline{W} = \overline{w}_{ij}$, the matrix representation of state transitions in an attack model, where $\overline{w}_{ij}$ is the probability of the system being taken to state s_j from state s_i. Let $\tau(s_j \rightarrow s_i)$ denote the proportion between the number of exploits that take the system from s_i to s_j and the total number of exploits applicable to s_i and $l(s_i, s_j)$ denote the length of the shortest path between s_i and s_j, a concrete algorithm for constructing $\overline{W}$ is presented in Algorithm 5. Depth-first-search or model checker such as NuSMV [73] is first used to construct the $N \times N$ *adjacency matrix* AM where N is the number of reachable states in M, such that $AM[i, j] = 1$ if state s_j is one of the successor states of state s_i, otherwise $AM[i, j] = 0$. Then the *transition matrix* $\overline{W}$ is constructed following the above stated adjustment to state transitions in web graphs.

Reconsider the web graph illustrated in Figure 6.1 as a example. We now construct the *transition matrix* $\overline{W}$ according to the adjustment illustrated in Figure

6.2 using Algorithm 5. The generated $\overline{W}$ is shown in Equation 6.5 where each $d_i = d \times e^{-\lambda l(s_1, s_i)}$, d being the usual damping factor used in PageRank.

$$\overline{\mathbf{W}} = \begin{pmatrix} 0 & 1-d_2 & 1-d_3 & 1-d_4 & 1-d_5 & 1 & 1-d_7 & 1 & 1 \\ 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0.3d_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0.8d_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4d_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6d_5 & 0 & d_7 & 0 & 0 \\ 0 & 0 & 0.4d_3 & 0.2d_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3d_3 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.5}$$

6.2.5 Ranking Attack Models

Following Mehta *et al.*'s definition, we define the rank for each state s_j in an attack model as the probability that s_j is reached from the initial state s_0. This can be recursively represented as

$$x_p = \sum_{q \in pa[p]} \overline{w}_{qp} \times x_q \tag{6.6}$$

When stacking all x_p into one vector $\mathbf{x}$, Equation 6.6 can be represented as

$$\mathbf{x} = \overline{W}\mathbf{x} \tag{6.7}$$

$\mathbf{x}$ in Equation 6.7 can be computed by multiple iterations through the following equation until a stationary state is reached.

$$\mathbf{x(t)} = \overline{W}\mathbf{x(t\text{-}1)} \tag{6.8}$$

If Equation 6.8 reaches a stationary state, i.e. $x(t) = x(t-1)$, after a long run of computation, all states in attack graph can be ranked. However, Equation 6.8 may or may not reach a stationary state after a long run of computation. Moreover, the result after multiple iterations may not be interesting (for example, the stationary state $\lim_{t\to\infty} \mathbf{x(t)}$ may be a vector of all 0s). A detailed proof of **Theorem 1** is provided in the following to justify that Equation 6.8 constructed as above can always reach a non-trivial stationary state after multiple iterations.

Lemma 2 *Each column of the transition matrix $\overline{W}$ constructed by Algorithm 5 sums to 1, i.e. $\sum_{i=1}^{N} \overline{w}_{i,j} = 1$.*

Proof of the above lemma follows directly the way by which $\overline{W}$ is constructed.

Theorem 3 *Equation 6.8 converges at a non-trivial vector x^* where $\sum_i x_i^* = 1$ after multiple iterations.*

Proof: Consider a linear transformation of x_p defined in Equation 6.2. Let

$$x'_p = c_1 \times x_p + c_2 = c_1 \times \sum_{q \in pa[p]} x_q \times \overline{w}_{pq} + c_2 \tag{6.9}$$

where $c_2 = \frac{1-c_1}{N}$. Stacking all x'_p into one vector x' and using iterative expression, Equation 6.9 is represented as

$$\mathbf{x(t)'} = c_1 \overline{W} \mathbf{x(t\text{-}1)'} + c_2 \Pi_N \tag{6.10}$$

A well-known theory states that the condition that $MX(K+1) = NX(K) + b$ converges at $(M-N)^{-1}b$ is $\rho(M^{-1}N) < 1$ [32].

Here we have $M = I$ and $N = c_1\overline{W}$. Therefore $\rho(M^{-1}N) = \rho(c_1\overline{W}) = c_1\rho(\overline{W})$. Assume x is an eigenvector of $\overline{W}$ and λ is the associated eigenvalue, then $\overline{W}x = \lambda x$, i.e. $\forall i$, $\sum_{j=1}^{N} \overline{w}_{i,j} x_i = \lambda x_i$. Extracting the common factor x_i, this can be written as $x_i(\sum_{j=1}^{N} \overline{w}_{i,j} x_i - \lambda) = 0$. As x is an eigenvector, there exist non-zero x_i. Therefore, $\lambda = \sum_{j=1}^{N} \overline{w}_{i,j} x_i$. Following lemma 1, $\sum_{i=1}^{N} \overline{w}_{i,j} = 1$, we know that $\lambda = \sum_{j=1}^{N} \overline{w}_{i,j} x_i = 1$. Therefore, $\rho(\overline{W}) = 1$. On the other hand, $0 < c_1 < 1$. As a result, $\rho(M^{-1}N) = c_1\rho(\overline{W}) < 1$, and hence Equation 6.10 converges at a stationary state $\lim_{t\to\infty} \mathbf{x(t)'}$.

We then prove by induction on t that the stationary state $\| \lim_{t\to\infty} \mathbf{x(t)'} \|_1$ of Equation 6.10 is a unit vector, i.e. $\| \lim_{t\to\infty} \mathbf{x(t)'} \|_1 = 1$.

1. For $t = 0$, Let $\mathbf{x(0)'} = \frac{1}{N}\Pi_N$; hence $\|\mathbf{x(0)'}\|_1 = 1$.

2. Let $t > 0$ and assume by induction that $\|\mathbf{x(t)'}\|_1 = 1$. Then, based on the definition of stochastic matrices,

$$\begin{aligned} \|\mathbf{x(t+1)'}\| = \Pi'_N \mathbf{x(t+1)'} &= c_1 \Pi'_N W \mathbf{x(t)'} + c_2 \Pi'_N \Pi_N \\ &= c_1 \Pi'_N \mathbf{x(t)'} + (1 - c_1) = 1 \end{aligned} \tag{6.11}$$

We hence proved that with the initial *unit vector* $\mathbf{x(0)'} = \frac{1}{N}\Pi_N$, $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1 = 1$. As stationary solution of Equation 6.10 is independent of the initial value $\mathbf{x(0)'}$ [32], it can be concluded immediately that $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1 = 1$ with any initial vector $\mathbf{x(0)'}$. Note that it can be seen from Equation 6.9 that $x'_p > 0$; hence $\|\mathbf{x(t)'}\|_1 = \sum_{p=1}^{N} x'_p = 1$

The stationary state $\mathbf{x(t)}$ of Equation 6.2 can be retrieved from $\mathbf{x(t)'}$ with linear conversion $\mathbf{x(t)} = \frac{(\mathbf{x(t)'} - c_2)}{c_1}$. $\mathbf{x(t)}$ is not trivial, because

$$\sum_{p=1}^{N} x_p = \sum_{p=1}^{N} \frac{x'_p - c_2}{c_1} = \frac{\sum_{p=1}^{N} x'_p - N \times c_2}{c_1} = \frac{1 - N \times c_2}{c_1} = 1 \tag{6.12}$$

That is, the stationary state $\mathbf{x(t)}$ is a unit vector. □

Given an attack model and empirical data that enables us to evaluate probabilities of different vulnerabilities being exploited, we first construct the *transition matrix* $\overline{W}$ as presented in Algorithm 5. We then assign random initial value to the rank of each state, and run Equation 6.8 for multiple iterations until it reaches the stationary state, guaranteed to exist by **Theorem 3**.

6.3 Implementation and Experiments

To evaluate the effectiveness of the proposed ranking scheme, we developed a toolkit in Java that ranks attack models with the proposed scheme. We ran the toolkit on the network example used by Mehta *et al* [111] and have the results compared with their ranking scheme. In this section, we first provide implementation details of the toolkit, then present the network model and the experimental results.

6.3.1 Implementation

The implementation toolkit is developed in Java but relies on NuSMV [73] for model checking functionalities, such as generating the complete set of reachable states given an initial state and the set of allowed state transitions. We made a minor modification to the source code of NuSMV (see below) to achieve interaction with our Java-based implementation toolkit. In the following, we provide details on the architecture of our implementation toolkit and its interaction with the modified NuSMV.

Toolkit Architecture

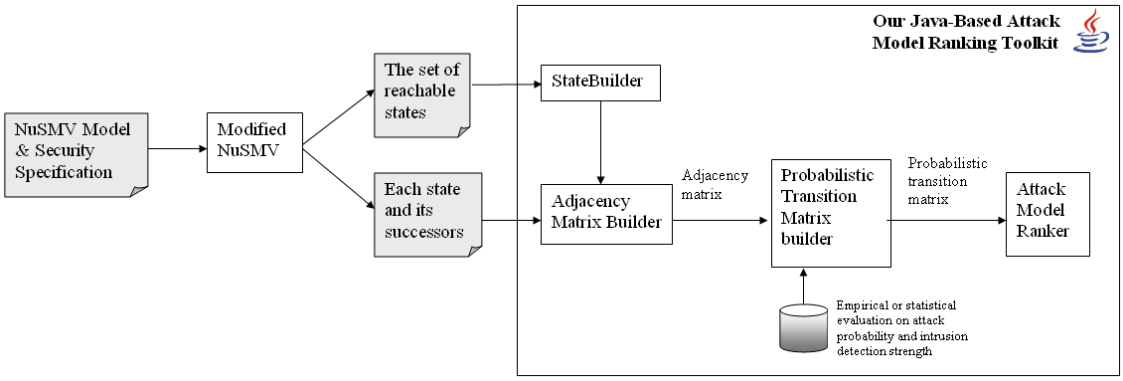

Figure 6.3: Toolkit Architecture

The architecture of our Java-based attack model ranking toolkit is illustrated in Figure 6.3. A network model along with the security specification written in NuSMV modelling language are fed to NuSMV. NuSMV then generates the complete set of reachable states S in the given model. We also modified NuSMV so that for each state $s \in S$ it generates the set of successors. The results generated as above are then saved as files, and feeded to the implementation toolkit to construct the adjacency matrix for states in the model. Combining the adjacency matrix, the empirical evaluation on the probability that each type of vulnerability is exploited, and the evaluation on the system's intrusion detection ability, the implementation toolkit generates the *transition matrix* $\overline{W}$ and ranks the states in the attack model as described in Section 6.4.2.

Toolkit Components

State Builder With the NuSMV command *print_reachable_states -v*, we generate the set of reachable states from the specified system initial state which are saved to a text file. The State Builder then reads the set of reachable states into Java-specific representation from the generated text file.

Adjacency Matrix Builder We modified NuSMV such that it generates and saves into a text file the successor states of a given state with the *-st* command line option. Iteratively using the *-st* option for each reachable state, the Adjacency Matrix Builder generates an $N \times N$ adjacency matrix AM where N is

the number of states in the attack model such that $AM[i,j] = 1$ if state j is one of the successor states of state i, otherwise $AM[i,j] = 0$.

Transition Matrix Builder Combining the adjacency matrix, the empirical evaluation on the probability that each type of vulnerability is exploited, and the evaluation on the system's intrusion detection ability, the Transition Matrix Builder follows Algorithm 5 to generate the *transition matrix* $\overline{W}$.

Attack Model Ranker Given the *transition matrix* $\overline{W}$, the Attack Model Ranker computes the ranks for all reachable states in the attack model using Equation 6.8 iteratively until the stationary is reached. A non-trivial stationary state is guaranteed to exist after multiple iterations by **Theorem 3**.

6.3.2 The Network Model for Experiments

Our experiments are based upon a network model similar to the one used by Mehta *et al* for comparison with their ranking scheme. The network model is illustrated in Figure 6.4. There are two target hosts ip_1 and ip_2, and a firewall separating them from the rest of the Internet. As shown each host is running two of three possible services (ftp, sshd, database). We model the same 4 types of atomic attacks summarised in Table 6.2 as in [95] [111] for comparable results. A detailed explanation of each attack follows.

The intruder launches an attack starting from an external machine ip_a that lies outside the firewall. The eventual goal is to gain access to the database. For that, the attacker needs root access on the database server ip_2.

We construct a finite state model of the network such that each state represents the system state including trust relation, connectivity, and adversary privilege on each machine, and each state transition corresponds to a single atomic attack which takes the system from one state to another.

Connectivity and Trust Relation Connectivity models the connection between two machines. We denote the connectivity by a binary relation $Reachable \subseteq Host \times Host$, where $Reachable(h_1, h_2) = 1$ if h_1 can connect to h_2, otherwise $Reachable(h_1, h_2) = 0$, i.e. either there is no physical link between h_1 and h_2, or the link is blocked by the firewall. Assuming the firewall policy is that the ftp server (ip_1) is publicly accessible while the database server (ip_2) can only be accessed internally, the connectivity relation is shown in Table 6.3. Similarly, we

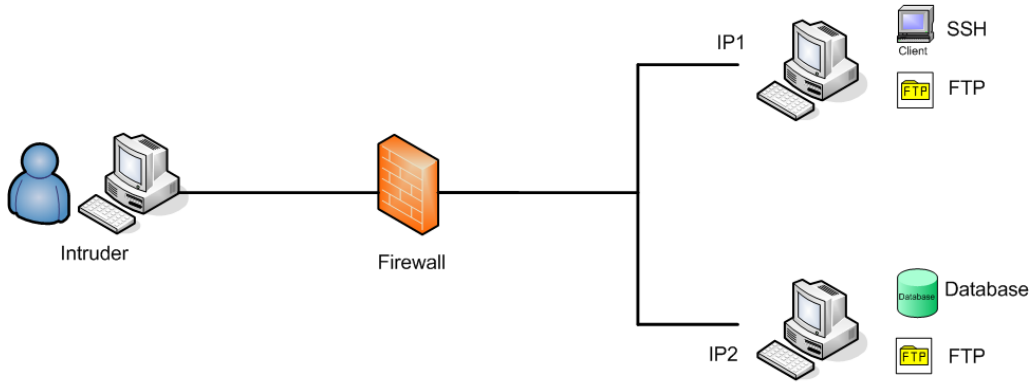

Figure 6.4: Network

denote trust relation between machines by a binary relation $Trust \subseteq Host \times Host$, where $Trust(h_1, h_2) = 1$ if a user on h_1 can login to h_2 remotely without specifying a password, $Trust(h_1, h_2) = 0$ otherwise. The trust relation is summarised in Table 6.4.

The Adversary and Privilege Privileges are $\{none, user, root\}$. There is an ordering of privileges: $none < user < root$. The adversary has $root$ on ip_a and no privileges on other machines initially. We use the function $plvl_A(H) : Hosts \rightarrow \{none, user, root\}$ to denote the level of privilege that intruder A has on machine H.

Vulnerabilities and Atomic Attacks We model the same 4 types of attacks as in [95] [111], each taking the modelled network from one state to another as described by the "effect" section of the attack. An attack is only applicable when both the network precondition and intruder precondition are satisfied. Throughout the following description, we denote source and target machine by S and T. To simplify the notations, we use ssh_H, ftp_H and $local_H$ to denote the presence of a vulnerability by running ssh service, ftp service and a `setuid root` executable respectively on host H.

1. sshd buffer overflow: Some versions of ssh services are vulnerable to a buffer overflow attack that allows an intruder to obtain a root shell on the target machine. Formally,

attack sshd-buffer-overflow **is**

intruder preconditions

[User-level privileges on host S]

$plvl_A(S) \geq user$

network preconditions

[Host T is running a vulnerable version of ssh service]

ssh_T

[Host T is reachable from S]

$Reachable(S, T) = 1$

intruder effects

[Root-level privileges on host T]

$plvl_A(T) = root$

end

2. ftp .rhosts: With a writable home directory and an executable command shell assigned to anonymous ftp users, an intruder can modify the .rhosts file in the ftp home directory, so as to create a remote login trust relationship between the attacker's machine and the target machine. Formally,

attack ftp-rhosts **is**

intruder preconditions

[User-level privileges on host S]

$plvl_A(S) \geq user$

network preconditions

[Host T is running a ftp service in a writable directory,

which gives a user shell to ftp users]

ftp_T

[Host T is reachable from S]

$Reachable(S, T) = 1$

network effects

[Trust relation between the intruder's machine and the target]

$Trust(S, T) = 1$

end

3. remote login: Using an existing remote login trust relationship between two machines, the intruder can login from the attacker's machine to the target and obtain a user shell without supplying a password. Although remote login is usually considered a legitimate operation by regular users, it is however an atomic attack from an intruder's viewpoint. Formally,

   attack remote-login **is**

   intruder preconditions

   [User-level privileges on host S]

   $plvl_A(S) \geq user$

   network preconditions

   [Host T trusts S]

   $Trust(S, T) = 1$

   [Host T is reachable from S]

   $Reachable(S, T) = 1$

   intruder effects

   [User-level privileges on host T]

   $plvl_A(T) = user$

   end

4. local buffer overflow: The attacker exploits a buffer overflow vulnerability in a `setuid root` executable to gain root access. Formally,

   attack local-buffer-overflow **is**

   intruder preconditions

   [User-level privileges on host T]

   $plvl_A(T) \geq user$

   network preconditions

[Host T runs a vulnerable version of a `setuid root` *executable]*

$local_T$

intruder effects

[Root-level privileges on host T]

$plvl_A(T) = root$

end

A NuSMV Implementation A NuSMV model that implements the above network model is as follows.

```
MODULE main

  VAR
    ip1 : machine;
    ip2 : machine;
    c   : Connection;
    t   : Trust;

    attackIP1ssh : process AttackMachineSSH(ip1 ip2 c.conn t.trust);
    attackIP1ftp : process AttackMachineFTP(ip1 ip2 c.conn t.trust);
    attackIP1remote : process AttackMachineRemote(ip1 ip2 c.conn t.trust);
    attackIP1local : process AttackMachineLocal(ip1 ip2 c.conn t.trust);
    attackIP2ssh : process AttackMachineSSH(ip2 ip1 c.conn t.trust);
    attackIP2ftp : process AttackMachineFTP(ip2 ip1 c.conn t.trust);
    attackIP2remote : process AttackMachineRemote(ip2 ip1 c.conn t.trust);
    attackIP2local : process AttackMachineLocal(ip2 ip1 c.conn t.trust);

  ASSIGN
    init(ip1.id) := 2;
    init(ip2.id) := 3;

    init(ip1.vulnerability[1]) := 1;
    init(ip1.vulnerability[2]) := 1;
    init(ip1.vulnerability[3]) := 1;
```

```
    init(ip1.vulnerability[4]) := 1;

    init(ip2.vulnerability[1]) := 0;
    init(ip2.vulnerability[2]) := 1;
    init(ip2.vulnerability[3]) := 1;
    init(ip2.vulnerability[4]) := 1;

    init(ip1.access) := none;
    init(ip2.access) := none;

    init(ip1.exploit[1]) := 0;
    init(ip1.exploit[2]) := 0;
    init(ip1.exploit[3]) := 0;
    init(ip1.exploit[4]) := 0;
    init(ip2.exploit[1]) := 0;
    init(ip2.exploit[2]) := 0;
    init(ip2.exploit[3]) := 0;
    init(ip2.exploit[4]) := 0;

  SPEC AG !(ip2.access = root)


MODULE machine

  VAR
    id : {1 2 3};
    access : { none user root };
    vulnerability : array 1..4 of boolean;
    exploit : array 1..4 of boolean;

  ASSIGN


MODULE Connection
```

```
  VAR
    conn : array 1..3 of array 1..3 of boolean;

  ASSIGN
    init(conn[1][1]) := 1;
    init(conn[1][2]) := 1;
    init(conn[1][3]) := 0;

    init(conn[2][1]) := 1;
    init(conn[2][2]) := 1;
    init(conn[2][3]) := 1;

    init(conn[3][1]) := 1;
    init(conn[3][2]) := 1;
    init(conn[3][3]) := 1;


MODULE Trust

  VAR
    trust : array 1..3 of array 1..3 of boolean;

  ASSIGN
    init(trust[1][1]) := 1;
    init(trust[1][2]) := 0;
    init(trust[1][3]) := 0;


    init(trust[2][1]) := 0;
    init(trust[2][2]) := 1;
    init(trust[2][3]) := 1;

    init(trust[3][1]) := 0;
    init(trust[3][2]) := 1;
    init(trust[3][3]) := 1;
```

```
MODULE AttackMachineSSH(targetCharacter‘route conn trust)

  VAR

  ASSIGN

    next(target.exploit[1]) :=
      case
       target.vulnerability[1]
       & (
           ( target.id = 2 & (conn[1][2] | (route.id=3 &
                (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2])) ) |
           ( target.id = 3 & (conn[1][3] | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3])) )
  )
& !target.exploit[1]
   :  1;
1  : target.exploit[1];
      esac;

    next(target.access) :=
      case
        target.vulnerability[1]
        & (
            ( target.id = 2 & (conn[1][2] | (route.id=3
                & (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2])) ) |
            ( target.id = 3 & (conn[1][3] | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3])) )
       )
```

```
    & !target.exploit[1]
          :  root;
       1 : target.access;
     esac;

  next(target.id) := target.id;

  next(target.vulnerability[1]) := target.vulnerability[1];
  next(target.vulnerability[2]) := target.vulnerability[2];
  next(target.vulnerability[3]) := target.vulnerability[3];
  next(target.vulnerability[4]) := target.vulnerability[4];

  next(conn[1][1]) := conn[1][1];
  next(conn[1][2]) := conn[1][2];
  next(conn[1][3]) := conn[1][3];
  next(conn[2][1]) := conn[2][1];
  next(conn[2][2]) := conn[2][2];
  next(conn[2][3]) := conn[2][3];
  next(conn[3][1]) := conn[3][1];
  next(conn[3][2]) := conn[3][2];
  next(conn[3][3]) := conn[3][3];

  next(trust[1][1]) := trust[1][1];
  next(trust[1][2]) := trust[1][2];
  next(trust[1][3]) := trust[1][3];
  next(trust[2][1]) := trust[2][1];
  next(trust[2][2]) := trust[2][2];
  next(trust[2][3]) := trust[2][3];
  next(trust[3][1]) := trust[3][1];
  next(trust[3][2]) := trust[3][2];
  next(trust[3][3]) := trust[3][3];

FAIRNESS
  running;
```

```
MODULE AttackMachineRemote(target route conn trust)

  VAR

  ASSIGN

    next(target.exploit[3]) :=
      case
       target.vulnerability[3]
       & (
           (target.id = 2 & ( (conn[1][2] & trust[1][2]) | (route.id=3
                & (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2] & trust[3][2]) ) ) |
           (target.id = 3 & ( (conn[1][3] & trust[1][3]) | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3] & trust[2][3]) ) )
  )
& !target.exploit[3]
   :  1;
1  : target.exploit[3];
      esac;

    next(target.access) :=
      case
        target.vulnerability[3]
        & (
           (target.id = 2 & ( (conn[1][2] & trust[1][2]) | (route.id=3
                & (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2] & trust[3][2]) ) ) |
           (target.id = 3 & ( (conn[1][3] & trust[1][3]) | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3] & trust[2][3]) ) )
       )
     & !target.exploit[3]
```

```
        :  user;
       1 : target.access;
     esac;

    next(target.id) := target.id;

   next(target.vulnerability[1]) := target.vulnerability[1];
   next(target.vulnerability[2]) := target.vulnerability[2];
   next(target.vulnerability[3]) := target.vulnerability[3];
   next(target.vulnerability[4]) := target.vulnerability[4];

   next(conn[1][1]) := conn[1][1];
   next(conn[1][2]) := conn[1][2];
   next(conn[1][3]) := conn[1][3];
   next(conn[2][1]) := conn[2][1];
   next(conn[2][2]) := conn[2][2];
   next(conn[2][3]) := conn[2][3];
   next(conn[3][1]) := conn[3][1];
   next(conn[3][2]) := conn[3][2];
   next(conn[3][3]) := conn[3][3];

   next(trust[1][1]) := trust[1][1];
   next(trust[1][2]) := trust[1][2];
   next(trust[1][3]) := trust[1][3];
   next(trust[2][1]) := trust[2][1];
   next(trust[2][2]) := trust[2][2];
   next(trust[2][3]) := trust[2][3];
   next(trust[3][1]) := trust[3][1];
   next(trust[3][2]) := trust[3][2];
   next(trust[3][3]) := trust[3][3];

FAIRNESS
   running;
```

```
MODULE AttackMachineFTP(target route conn trust)

  VAR

  ASSIGN

    next(target.exploit[2]) :=
      case
        target.vulnerability[2]
        & (
            (target.id = 2 & (conn[1][2] | (route.id=3
                & (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2]) ) ) |
            (target.id = 3 & (conn[1][3] | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3]) ) )
          )
        & !target.exploit[2]
            :  1;
        1  : target.exploit[2];
      esac;

    next(trust[1][2]) :=
      case
target.vulnerability[2]
& (
            (target.id = 2 & (conn[1][2] | (route.id=3
                & (route.access=user | route.access=root)
                & conn[1][3] & conn[3][2]) ) )
          )
       & !target.exploit[2]
          :  1;

     1 : trust[1][2];
      esac;
```

```
    next(trust[1][3]) :=
      case
target.vulnerability[2]
& (
            (target.id = 3 & (conn[1][3] | (route.id=2
                & (route.access=user | route.access=root)
                & conn[1][2] & conn[2][3]) ) )
          )
       & !target.exploit[2]
          :  1;

     1 : trust[1][3];
      esac;

    next(target.id) := target.id;

    next(target.vulnerability[1]) := target.vulnerability[1];
    next(target.vulnerability[2]) := target.vulnerability[2];
    next(target.vulnerability[3]) := target.vulnerability[3];
    next(target.vulnerability[4]) := target.vulnerability[4];

    next(conn[1][1]) := conn[1][1];
    next(conn[1][2]) := conn[1][2];
    next(conn[1][3]) := conn[1][3];
    next(conn[2][1]) := conn[2][1];
    next(conn[2][2]) := conn[2][2];
    next(conn[2][3]) := conn[2][3];
    next(conn[3][1]) := conn[3][1];
    next(conn[3][2]) := conn[3][2];
    next(conn[3][3]) := conn[3][3];

    next(trust[1][1]) := trust[1][1];
    next(trust[2][1]) := trust[2][1];
    next(trust[2][2]) := trust[2][2];
```

```
    next(trust[2][3]) := trust[2][3];
    next(trust[3][1]) := trust[3][1];
    next(trust[3][2]) := trust[3][2];
    next(trust[3][3]) := trust[3][3];

  FAIRNESS
    running;


MODULE AttackMachineLocal(target route conn trust)

  VAR

  ASSIGN

    next(target.exploit[4]) :=
      case
       target.vulnerability[4]
& !target.exploit[4]
& target.access = user
   :  1;
1  : target.exploit[4];
      esac;

    next(target.access) :=
      case
       target.vulnerability[4]
& !target.exploit[4]
& target.access = user
   :  root;
1  :  target.access;
      esac;

    next(target.id) := target.id;
```

```
    next(target.vulnerability[1]) := target.vulnerability[1];
    next(target.vulnerability[2]) := target.vulnerability[2];
    next(target.vulnerability[3]) := target.vulnerability[3];
    next(target.vulnerability[4]) := target.vulnerability[4];

    next(conn[1][1]) := conn[1][1];
    next(conn[1][2]) := conn[1][2];
    next(conn[1][3]) := conn[1][3];
    next(conn[2][1]) := conn[2][1];
    next(conn[2][2]) := conn[2][2];
    next(conn[2][3]) := conn[2][3];
    next(conn[3][1]) := conn[3][1];
    next(conn[3][2]) := conn[3][2];
    next(conn[3][3]) := conn[3][3];

    next(trust[1][1]) := trust[1][1];
    next(trust[1][2]) := trust[1][2];
    next(trust[1][3]) := trust[1][3];
    next(trust[2][1]) := trust[2][1];
    next(trust[2][2]) := trust[2][2];
    next(trust[2][3]) := trust[2][3];
    next(trust[3][1]) := trust[3][1];
    next(trust[3][2]) := trust[3][2];
    next(trust[3][3]) := trust[3][3];

  FAIRNESS
    running;
```

6.3.3 Experimental Results Analysis and Evaluation

Let the security property be "intruder cannot gain root access on ip_2". We ran our attack model ranking toolkit presented in Section 6.3.1, and visualised the results with the *graphViz* package [10]. Figure 6.5 illustrates the result obtained as such. For each state, the intensity of color is proportional to the rank of that state. Any path in the graph from the root node to a leaf node represents a sequence of exploits

with which the intruder can achieve the final goal. It can be seen that *local buffer overflow* and *remote login* are critical exploits as each path from the root node to a leaf node has exploited them at least once. After fixing either *local buffer overflow* or *remote login*, NuSMV asserts security property "intruder cannot gain root access on ip_2" to be true. On the other hand, *ftp.rhost* and *ssh buffer over flow* are non-critical exploits as an intruder can still reach the final goal without either of them.

To investigate how an attacker selectively exploiting vulnerabilities affects the chance of compromising the system, we vary the probability assigned to each type of exploit and have other exploits divide the remaining probability equally. Changes to the ranks of states resulting from varying the probabilities of the exploits reflects how an attacker selectively exploiting vulnerabilities affects the chance to compromise the system. We also set the rate by which probability of an intrusion remaining undetected decays with the shortest path to 0, so that changes to the rank of a state are the result only of varying the probabilities of the exploits. The experimental result is plotted in Figure 6.6 where the Y-axis represents the total rank of error states, i.e. the probability of an adversary reaching the final goal. It can be seen that the total rank of error states increases as the attacker prioritises critical exploits *local buffer overflow* and *remote login*, modelled by assigning higher probabilities to the two attacks. Similarly, the adversary's chance to succeed decreases as the adversary prioritise non-critical exploits *ftp.rhosts* and *sshd buffer overflow*. In general, our scheme produces a higher rank when the attacker prioritise critical exploits and hence has better chance to succeed. The rank produced by our scheme joins the rank by Mehta's scheme at the equal probability point, i.e. where all exploits are assigned equal probabilities.

To investigate the effect that probability of an intrusion remaining undetected decays at a rate proportional to length of the shortest path from initial state, we vary the decaying rate λ while assigning equal probabilities to all exploits. The experimental result is plotted in Figure 6.7. It can be seen that the total rank of error states increases as the decaying rate decreases. This corresponds to the fact that an attacker has less chance of success on well-protected systems such as systems implementing "defense-in-depth" which at each step of the intrusion and thus on the whole has a higher probability of being able to discover and thwart the intrusion. The ranks produced by our ranking scheme consistently remain lower than the rank by Mehta's scheme, resulting from the decaying of probability that an adversary remains undetected and is able to proceed.

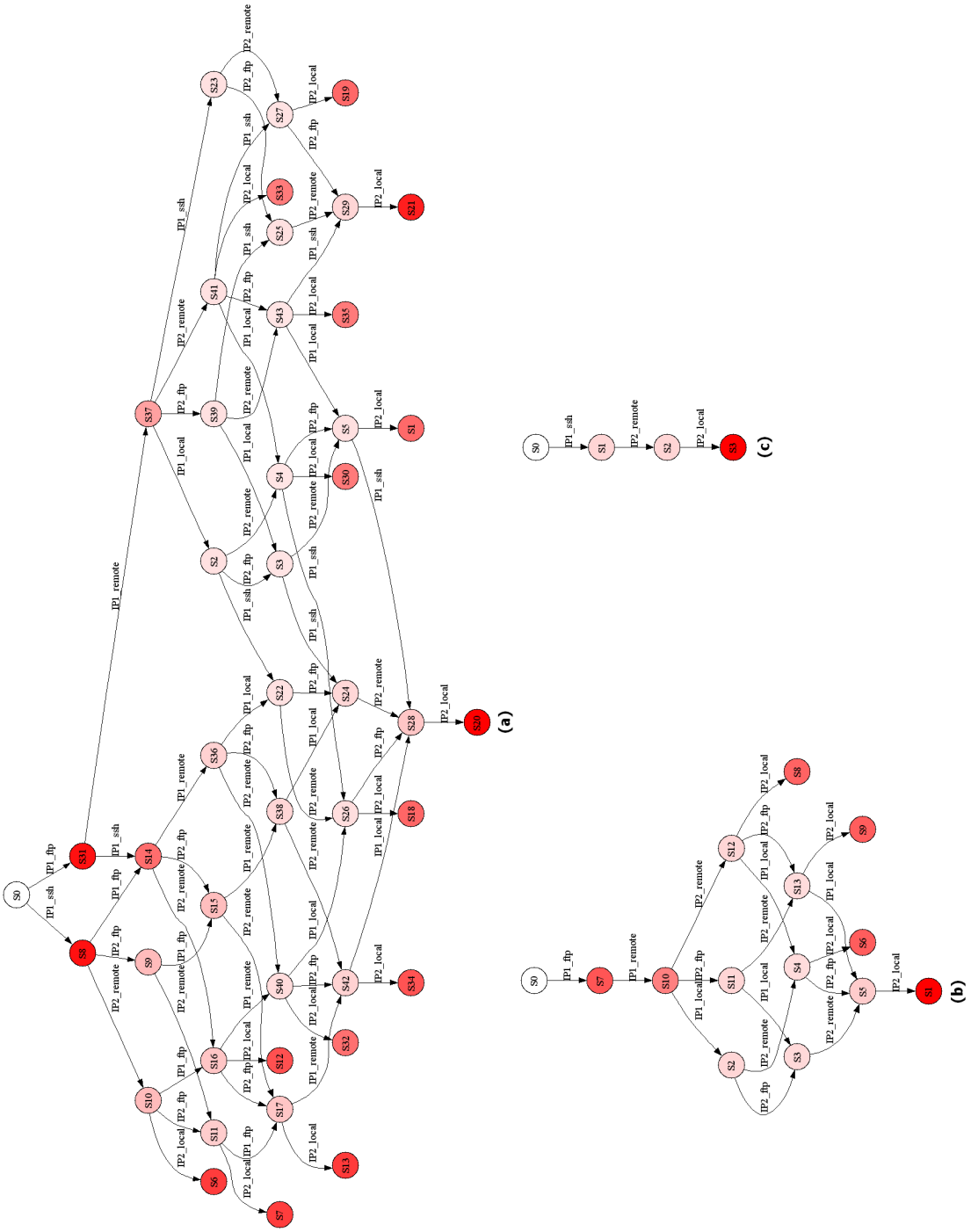

Figure 6.5: Comparison of Ranked Attack Models. (a) The complete ranked attack model (b) Attack Model after fixing up the SSH vulnerability (c) Attack Model after fixing FTP vulnerability

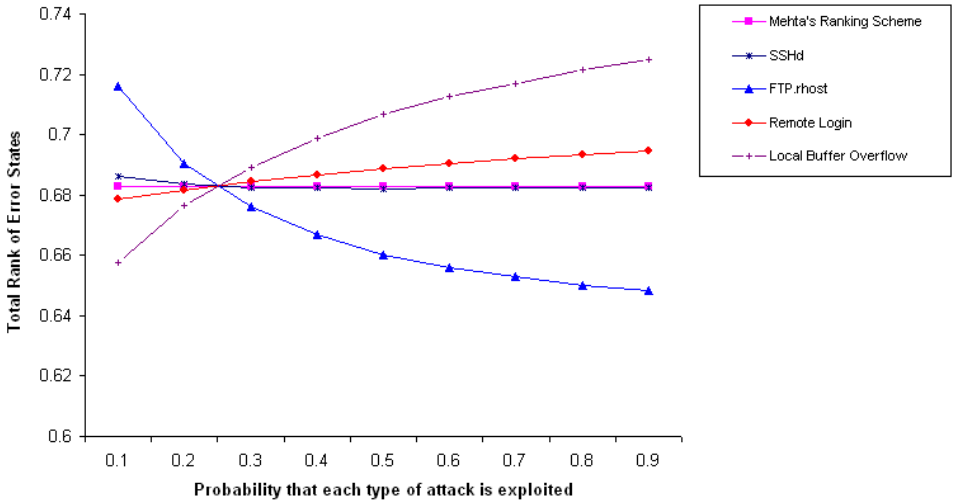

Figure 6.6: Rank varies with attack probabilities

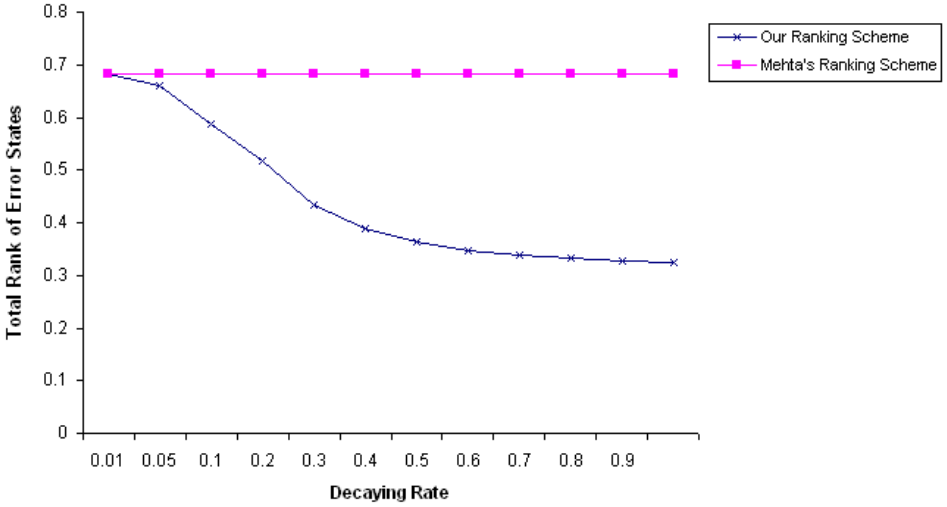

Figure 6.7: Rank varies with decaying rate

Figure 6.8 plots the experimental result by the overall effect of various decaying rates and varying probability assigned to each type of exploit (still other exploits divide remaining probability equally). It can be seen that ranking of system states is dominated by decaying of probability that intrusion remains undetected. Variation in probability assigned to each type of exploit only affects ranking of states to a minor extent. It can also be see that, the greater the decaying rate is, the less variation in probability assigned to each type of exploit affects ranking of states. The result reveals that deployment of well-protected system offsets experienced intruder's strategy in selectively exploiting vulnerabilities to maximise the chance of success, lowering the chance of success to no more than that of brute-force type of attack.

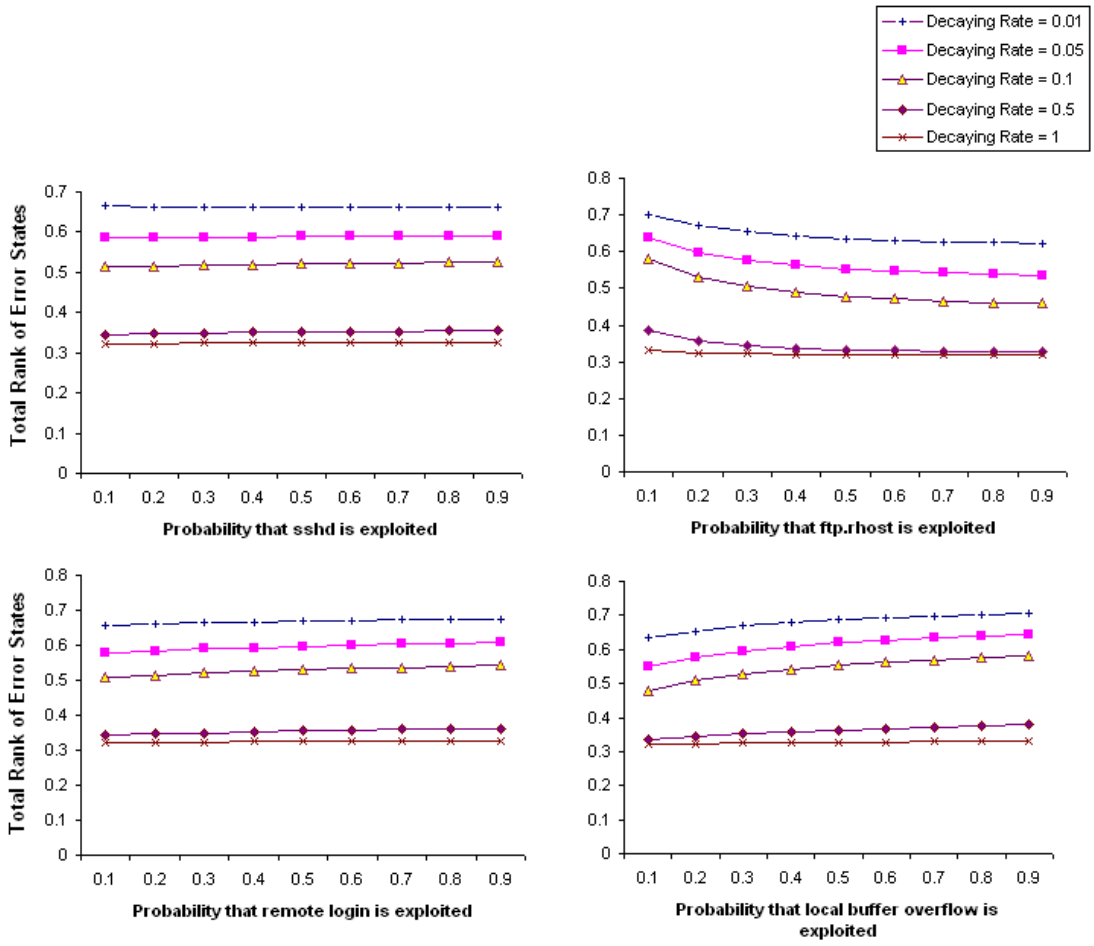

Figure 6.8: Rank varies with decaying rate and attack probabilities

The experimental results and analysis presented above demonstrates the advantage and application of the proposed ranking scheme. Firstly, it considers the effect on ranking of system states by an intruder selectively exploiting vulnerabilities to

maximise the chance of success. Secondly, it is able to model the effect on ranking of system states by system intrusion detection ability that aims at thwarting exploits of vulnerabilities that take the system to another state where the intruder gains an elevated privilege. Therefore, the proposed ranking scheme can rank attack models more accurately, and provide more realistic evaluation on the probability that a system is in a compromised state. Intuitively, the probability of a system being in a compromised state increases with the probability that an intruder is able to prioritise critical exploits, and with weakening system intrusion detection ability; however, our ranking scheme provides a quantitative measure for the increase. The proposed scheme can also assist network researchers and architects in network design and analysis, e.g. determining the minimum intrusion detection strength required to thwart the best effort in selectively exploiting vulnerabilities by intruders.

6.4 Ranking Attack Graphs with Graph Neural Network

6.4.1 Preliminaries

Multi-Layer Perceptron Neural Networks

The application of neural networks [38] for pattern recognition and data classification has gained acceptance in the past. In this Section, *supervised* neural networks based on *layered* structures are considered. Multilayer perceptrons (MLPs) are perhaps the most well known form of supervised neural networks [38]. MLPs gained considerable acceptance among practitioners from the fact that a single-hidden-layer MLP has a universal approximation property [49], in that it can approximate a non-pathological nonlinear function to any arbitrary degree of precision.

The basic computation unit in a neural network is referred to as a *neuron* [38]. It generates the output value through a parameterised function called a *transfer function* f. An MLP can consist of several layers of neurons. The neuron layer that accepts external input is called the input layer. The dimension of the input layer is identical to the dimension of the data on which an MLP is trained. The layer that generates the output of the network is called the output layer. Its dimension is identical to the dimension of the target values. A layer between the input layer and the output layer is known as a hidden layer. There may be more than one hidden

layer but here for simplicity we will only consider the case of a single hidden layer. Each layer is fully connected to the layer above or below[1]. The input to each hidden layer neuron and the output layer neuron is the weighted sum from the previous layer, parameterised by the connecting weights, known as synaptic weights. The multiple layered structure described here has given MLP its name.

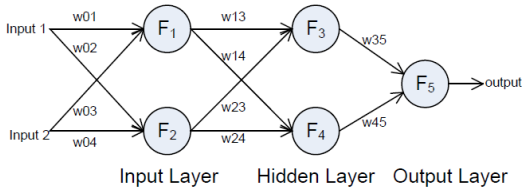

Figure 6.9: An example of a multi-layered perceptron neural network, where F_1, and F_2 form the input layer, F_3 and F_4 form the hidden layer, while F_5 forms the output layer.

Figure 6.9 illustrates an example of a single-hidden-layered MLP. The transfer function associated with each of the neurons is $F_j = f(\sum_{i=0}^{n} w_{ji}x_i)$, where n is the total number of neurons in the previous layer, $x_0 = 1$, x_i is the $i-$th input (either a sensory input or the output F_j from a previous layer), and w_{ji} is a real valued weight connecting neuron (or input) i with neuron j. The transfer function $f(\cdot)$ often takes the shape of a hyperbolic tangent function. The MLP processes data in a *forward* fashion starting from the input layer towards the output layer.

Training an MLP [38] is performed through a *backward* phase known as error back propagation, starting from the output layer. It is known that an MLP can approximate a given unknown continuously differentiable function [2] $g(x_1, \ldots x_n)$ by learning from a *training data set* $\{x_{i1}, \ldots x_{in}, t_i\}$, $1 \leq i \leq r$ where $t_i = g(x_{i1}, \ldots x_{in})$. It computes the output for each input $X_i = \{x_{i1}, x_{i2} \ldots x_{in}\}$ and compares the output with the target t_i. Weights are then adjusted using the gradient descent algorithm based on the error towards the best approximation of the target t_i. The forward and the backward phases are repeated a number of times until a given prescribed accuracy (stopping criterion) is met *e.g.* the output is "sufficiently" close to the target.

Once the stopping criterion is met, the learning stage is finished. The MLP is

[1]Fully connected layers are the most commonly studied and used architectures. There are some neural network architectures which are connected differently.

[2]The function does not need to be continuously differentiable. However for our purpose in this Section, we will assume this simplified case.

then said to emulate the unknown nonlinear function $g(\cdot, \ldots, \cdot)$. Given any input X_i with unknown target t_i, it will be able to produce an output from g or it is said to be able to generalise from the training set to the unseen input X_i.

The application of MLP networks has been successful in many areas [29, 28, 38]. The inputs to the MLP would need to be in vectorial form, and as such, it cannot be applied to graph structured inputs or inputs which relate to one another.

Graph Neural Network

In several applications including attack graphs and web page ranking, objects and their relations can be represented by a graph structure such as a tree or a directed acyclic graph. Each node in the graph structure represents an object in the problem domain, and each edge represents relations between objects. For example, an attack graph node represents a particular system state, and an edge represents that the attacker can reach one state from the other. Graph Neural Network (GNN) is a recently proposed neural network model for graph-based learning environments [29]. It is capable of learning topological dependence of information representing an object such as its rank relative to its neighbors, *i.e.* information representing neighboring objects and the relation between the object and its neighboring objects. The use of GNN for determining information on objects represented as nodes in a graph structure has been successfully shown in several applications [28, 30].

The GNN is a complex model. A detailed description of the model and its associated training method cannot be described in the thesis. Hence, this Section only summaries the key ideas behind the approach as far as it is necessary to understand the method. The interested reader is referred to [29].

In order to encapsulate information on objects, a vector $s_n \in \mathbb{R}^s$ referred to as *state*[3], which represents information on the object denoted by the node, is attached to each node n. This information is dependent on the information of neighboring nodes and the edges connecting to its neighbors. For example, in the case of page rank determinations, the *state* of a node (page) is the rank of the page. Similarly, a vector e_{ij} may also be attached to the edge between node n_i and n_j representing attributes of the edge [4]. The state of a node n is determined by states of its neighbors

[3]For historical reasons this is called a "state", which carries slightly different meaning to the meaning of "state" in the attack graph literature. However, there should not be any risk of confusion as this concept of "state" in GNN is used in the training of the GNN model.

[4]GNN can also accept labels on nodes; however this will not be used in this Chapter and is omitted here.

and the labels of edges that connect it to one of its neighbors. Consider the GNN shown in Figure 6.10, the state of node n_1 can be specified by Equation (6.13).

$$s_1 = f_w(s_2, s_3, s_5, e_{21}, e_{13}, e_{51}) \tag{6.13}$$

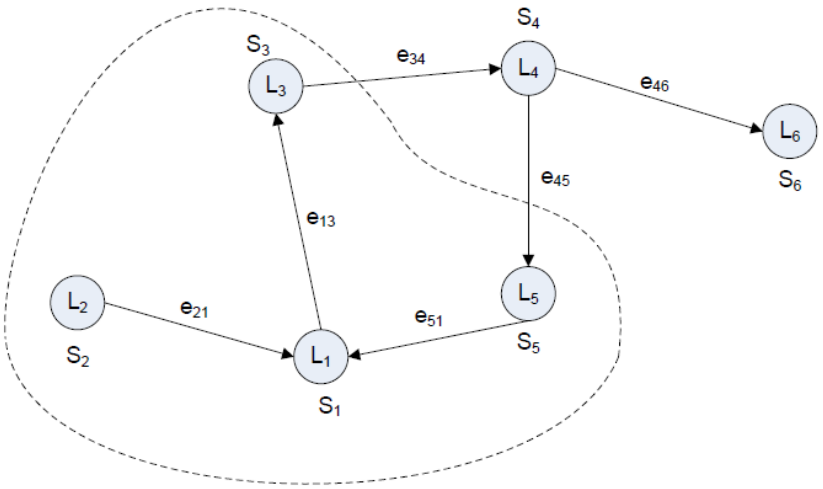

Figure 6.10: The dependence of state s_1 on neighborhood information

More generally, let $\mathbb{S}(n)$ denote the set of nodes connected to node n and $f_{\mathbf{w}_1}$ be a function parameterised by $\mathbf{w}_1$ that expresses the dependence of information representing a node on its neighborhood, then the state s_n is defined by Equation (6.14) where $\mathbf{w}_1 \in \mathbb{R}$ is the parameter set of function f which is usually implemented as an MLP described in Section 6.4.1.

$$s_n = f_{\mathbf{w}_1}(s_u, e_{xy}), u \in \mathbb{S}(n), x = n \vee y = n \tag{6.14}$$

The state s_n is then put through a parameterised *output network* $g_{\mathbf{w}_2}$, which is usually also implemented as an MLP parameterised by $\mathbf{w}_2$, to produce an output (*e.g.* a rank) o_n

$$o_n = g_{\mathbf{w}_2}(s_n) \tag{6.15}$$

Equations (6.14) and (6.15) define a function $\varphi_w(G, n) = o_n$ parameterised by weights $\mathbf{w}_1$, and $\mathbf{w}_2$ that produces an output for the given graph G. The parameter sets $\mathbf{w}_1$ and $\mathbf{w}_2$ are adapted during the training stage such that the output φ is the best approximation of the data in the learning data set $\mathcal{L} = \{(n_i, o_i)|1 \leq i \leq q\}$, where n_i is a node and t_i is the desired target for n_i. Approximation to the targets

can be measured by the quadratic error function

$$\mathcal{J}_w = \sum_i (t_i - \varphi(G, n_i))^2 \tag{6.16}$$

Computing the states

The solution to Eqs. 6.14 and 6.15 can be obtained by iterating through the following equations where r denotes the number of nodes in the given graph,

$$s_n(t+1) = f_w(s_u, e_{xy}), 1 \leq n \leq r, u \in \mathbb{S}(n), x = n \vee y = n \tag{6.17}$$

$$o_n(t+1) = g_w(s_n(t+1)) \tag{6.18}$$

Learning from a graph structure

To learn from a graph structure, an *encoding network* needs to be built as follows: each node of the graph is replaced by a unit that implements the transfer function f_w and g_w described in Eqs. 6.14 and 6.15. It is worth noting that the same MLPs are used to implement f_w and g_w in all units. Each unit stores the current state $s_n(t)$ and computes $s_n(t+1)$ using Eq. 6.17, taking input from the states stored in its neighboring units. The procedure repeats until it comes to a stable point, i.e. $s_n(t) = s_n(t+1)$. This procedure is referred to as the *feed-forward phase* of GNN. Fig. 6.11 illustrates how to build the encoding network for a given graph structure.

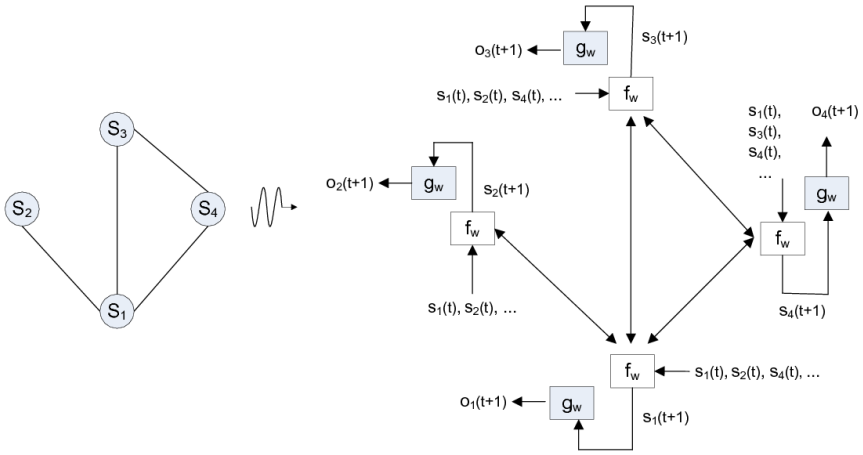

Figure 6.11: The encoding network

Once a GNN has come to a stable point through the *feed-forward phase*, the stable output o_n are compared against target value t_n and the gradient $\frac{\vartheta J_w(T)}{\vartheta w}$ is computed.

The weights w are adapted towards the best approximation of the training data set according to a gradient descent strategy based on the quadratic error function shown in Eq. 6.15. This procedure is referred to as the *back-propagation phase* of GNN.

While the *feed-forward phase* moves the system to a stable point, the *back-propagation phase* adapts the weights to change the outputs towards the desired target. These two phases are repeated until the outputs are sufficiently close to the targets, indicating that the GNN has finished the *training stage* and can be used to simulate the target function. Once trained, the GNN only needs to run through the *feed-forward phase* to produce an output for any given input graph.

A GNN described above provides an artificial approach to learning the pattern by which, in several real world example, information (such as ranks) on objects are structurally determined by information on their neighbors. It can learn the pattern from given training set $\mathcal{L} = \{(n_i, o_i)|1 \leq i \leq q\}$ where n_i is a node and t_i is its desired target, and generalise the result by applying the learnt pattern to data not included in the training set.

6.4.2 Ranking Attack Graph using GNNs

The use of GNN for determining information on objects represented as nodes in a graph structure has been shown in several applications such as ranking web pages. In this Section, we develop an implementation of the GNN model as described in Section 6.4.1 applied to ranking attack graphs. For attack graphs, nodes represent computer system states and edges represent the transition relations between system states. Hence, the *state* of a node is determined by states and outgoing links of its parent nodes. Symbolic values 0 or 1 are also assigned to edge labels to distinguish between incoming and outgoing links. The function $f_{\mathbf{w}_1}$ that expresses the dependence of the state s_n of node n on its neighbors can be represented by Equation (6.19) where $\mathbb{S}(n)$ denotes the set of nodes connected to node n,

$$s_n = f_{\mathbf{w}_1}(s_u, e_{uv}), u \in \mathbb{S}(n) \tag{6.19}$$

The output function $g_{\mathbf{w}_2}$ with input from the states produces the rank of node n.

$$o_n = g_{\mathbf{w}_2}(s_n) \tag{6.20}$$

The two computing units $f_{\mathbf{w}_1}$ and $g_{\mathbf{w}_2}$ are implemented by MLPs [13]. Inputs to the MLPs are the variables in Eqs. (6.19) and (6.20), *i.e.* states of neighboring

nodes and labels of connecting edges. We use the *sigmoid* function [13] as the transfer function of $f_{\mathbf{w}_1}$. The *sigmoid* function is commonly used in back-propagation networks. The transfer function of the output network $g_{\mathbf{w}_2}$ is a linear transfer function that stretches the output from $f_{\mathbf{w}_1}$ to the desired target. $f_{\mathbf{w}_1}$ and $g_{\mathbf{w}_2}$ are thus parameterised by weight sets $\mathbf{w}_1$ and $\mathbf{w}_2$ of the underlying MLPs. The adaption of weights sets for the best approximation of training data is through a back-propagation process [13].

It is known that the computational complexity of the forward phase of MLP is $\mathrm{O}(n)$, where n is the number of inputs [22]. When considering that the functions $f_{\mathbf{w}_1}$ and $g_{\mathbf{w}_2}$ can be implemented as MLPs, it becomes clear that the computational complexity of the forward phase of GNN is also $\mathrm{O}(n)$. Note that a direct comparison of the computational complexity of PageRank is difficult since the computational complexity of PageRank depends on the number of links rather than on the number of nodes.

The GNN implemented is initialised by assigning random weights to the underlying MLPs. The training data is generated using Mehta *et al.*'s ranking scheme on a set of attack graphs. The trained GNN can take as input an attack graph and output through the trained network ranks of each node.

6.4.3 Experiments and Results

The objectives of the experiments are to verify the effectiveness of the proposed ranking approach: (1) if GNN can learn a ranking scheme applied to attack graphs, and (2) if it can generalise the results to unseen data. In particular we consider the PageRank scheme used by Mehta *et al.* to rank attack graphs.

It has been shown that GNN can learn PageRank on very large graphs [30, 28]. Our objective is to ascertain if it can learn the ranking scheme when applied to attack graphs. We note that the WWW is a single very large graph and the aim of learning has been to generalise the results to this large graph by learning from selected small sub-graphs of WWW. However, in the case of ranking attack graphs, many small graphs of various sizes exist. For a small network with few vulnerabilities, it can be a small graph with few nodes and edges, while for more complex networks, it can have hundreds of nodes and edges. One question is: what type/size of graph should be used for training? It would be ideal to train GNN with a set of attack graphs generated from networks of various complexity. However, while attack graphs can

be generated quite easily from real networks, generation of realistic attack graphs of artificial examples requires significant manual labor as well as computational effort [75]. It is, in general, difficult to generate a large number of real attack graphs on artificial examples as training samples. To solve this problem, we experiment with training GNN with a set of automatically generated pseudo attack graphs that have similar shapes and node connection patterns to those of manually generated real attack graphs. Details for pseudo attack graph generation are provided in this section further below.

Note that it is time consuming to generate *artificial* attack graphs which resemble the type of attack graphs one would encounter in the real world. Real world attack graphs can be easily generated (the process may even be automated). However, due to a lack of access to large scale distributed system we were unable to obtain real world attack graphs. Instead we use the time consuming task of generating artificial data. We did not attempt to simulate attack graphs for small systems since the resulting graphs would be rather small. Existing approaches to ranking attack graphs are sufficient for small scale systems whereas the advantage of the proposed approach is most pronounced on larger graphs.

A related question is how we determine the effectiveness of the proposed ranking approach. GNN output produces a set of ranks for nodes of an attack graph. There are many ways to define the "accuracy" of a set of ranks compared to the set of ranks that is calculated by another scheme. In particular one may use the average distances of the sets, or the maximum distance over all elements of the set. For our particular application we use *Relative Position Diagram (RPD)* and *Position Pair Coupling Error (PPCE)* as described in this section further below. The PPCE measures the agreement between the proposed approach and the ordering imposed on the nodes in an attack graph by PageRank. Hence, this ordering compares the proposed approach with an existing approach. No attempt is made to assess whether the PageRank order by itself makes sense.

Performance Measure

A ranking scheme can be denoted as a function that takes as input an attack graph (AG) and outputs the ranks, i.e. $f(AG) = R$ where R is a real vector of the same size as the number of nodes in AG. A naive performance measure is to compare the output ranks by GNN $f_G(AG) = R_G$ and that produced by Mehta's ranking scheme

$f_M(AG) = R_M$. However, as ranks are used to identify important portions of an attack graph, ordering of ranks are considered more important than their actual numerical values. We therefore devised two performance measures to evaluate how well rank orderings are preserved.

`Relative Position Diagram (RPD):` This visually illustrates how well rank orders are preserved, and is obtained by sorting R_M and R_G and plotting the order of each node in R_M against that in R_G. The X-axis and Y-axis represent the order of ranks in R_M and R_G respectively. Therefore, nodes that have the same rank orders by both schemes are plotted along the diagonal.

`Position Pair Coupling Error:` An RPD only intuitively shows how well rank orders are preserved. We also provide a quantitative measure on rank order preservation as follows. Let r_i^M and r_i^G denote the i-th element in R_M and R_G respectively. For each pair of nodes, if r_i^M and r_j^M are not in the same order as r_i^G and r_j^G, then a position-pair-coupling-error (PPCE) is found. Performance of the GNN ranking scheme can therefore be quantitatively measured by the PPCE rate.

Experimental Results

We found that the computational demand of the attack graph generation method in [75] is high. The generation of a single real world attack graph took several hours, and hence, we were not able to produce more than 14 graphs within a reasonable time frame for the experiments. The 14 attack graphs were generated using the same computer network example as in a previous work [57], with variations created by adding, removing, or varying vulnerabilities or network configurations. Eight of the attack graphs were randomly selected to form the training data set of the GNN. The remaining six attack graphs are used as the testing data set so as to examine the GNN's generalisation ability in different situations. The GNN used for our experiments consists of two hidden layers, and each hidden layer has 5 neurons. The number of external inputs is also set to 5. Such a GNN has been shown to be successful when applied to ranking web pages [30, 28], and it produces a sufficiently small network that allows for a fast processing of the data. However, this GNN was used successfully on a problem involving much larger number of nodes, and hence, it may be possible to use a somewhat smaller network when dealing with attack graphs. This approach is not attempted in the Chapter and may be considered as a topic for future work.

We varied the number of training epochs from 500 to 20,000 and repeated the experiment 6 times with randomized initial weights of the underlying MLPs. The PPCE rate results are plotted in Fig. 6.12. It can be observed that when trained for a sufficiently large number of epochs (around 10,000) the PPCE rate reduces asymptotically to an optimal value. Improvement by further training can be observed but is not significant. Hence the number of training epochs is fixed to be 10,000 in the rest of the experiments for an appropriate effort/time tradeoff. However when applied to ranking attack graphs in practice the training epoch can be set to 20,000 or larger for more accurate results.

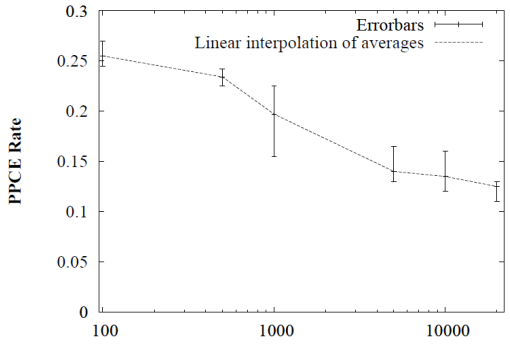

Figure 6.12: Effect of number of training epochs

From the above experimental results, it can be observed that the average PPCE rate can be reduced to around 10% when GNN is trained with sufficient number of training examples and training epochs. That being the case on an average the proposed GNN-based ranking scheme sacrificed around 10% accuracy in terms of relative position preservation. However, once trained, a GNN can produce output at an O(N) time complexity where N is the size of input. This may be an improvement compared with the O($N\log\frac{1}{\epsilon}$) computational complexity required by PageRank [89]. Here PageRank is used as an established algorithm. Its maths background such as under what circumstances it achieves the computation complexity is beyond the scope of this thesis. Moreover, the generalisation ability of GNN allows us to train GNN on only a small number of attack graphs and apply the results to unseen attack graphs.

Figure 6.13 plots the RPDs resulting from ranking graphs from the testing data set with GNN to visualise the effectiveness of relative position preservation. Each subgraph represents the resulting RPD for one of the 6 attack graphs in the testing data set, and hence it is of different scale to other subgraphs. It can be observed that

the order of ranks are preserved to a reasonable degree. Although only a relatively small portion of nodes remains at exactly the same position as the PageRank scheme, most nodes remain centered around the diagonal and as a result important nodes remain important and unimportant nodes remain unimportant. The GNN-based ranking scheme therefore allows a system administrator to focus on most of the important nodes in an attack graph.

Finally, we experimented with reducing the number of training samples. The number of training attack graphs in the training data set is reduced from 8 to 2. The effect on reducing the number of attack graphs used in the training data set is plotted in Figure 6.14. It can be observed that the number of attack graphs in the training data set has a significant impact on the accuracy in terms of relative positions on the training results. Sufficient attack graphs must be provided in the training data set to simulate the ranking scheme effectively.

Training with Pseudo Attack Graphs

Generating a large number of attack graphs for the training data set is difficult due to significant manual labor and computational effort that it requires [75]. In the following, we provided an alternative scheme for automated training set generation, and train GNN with automatically generated pseudo attack graphs that have similar shapes and node connection patterns to those of manually generated realistic attack graphs.

That attack graphs are non-cyclic, tree shaped graphs resulting from a procedure as follows: at the initial system state, the intruder looks for an entry point and reaches the first layer of nodes in the attack graph by exploiting one of the vulnerabilities applicable to the initial state. With the escalated privilege by the initial exploit, the intruder obtains more intrusion options for the next intrusion step. This procedure repeats and keeps expanding the attack graph until the intruder can reach the intrusion goal. Leaf nodes in the attack graph are thus produced, and the attack graph begins to contract until all attack paths reach the intrusion goal. We generate pseudo attack graphs by simulating this procedure. A pseudo attack graph begins with the root node representing the initial state. It then expands with increasing number of nodes at each layer to simulate the expanding process of realistic attack graphs. This repeats until presumably the intruder begins to reach the final goal. Then the pseudo attack graph contracts till all paths are terminated at a leaf node.

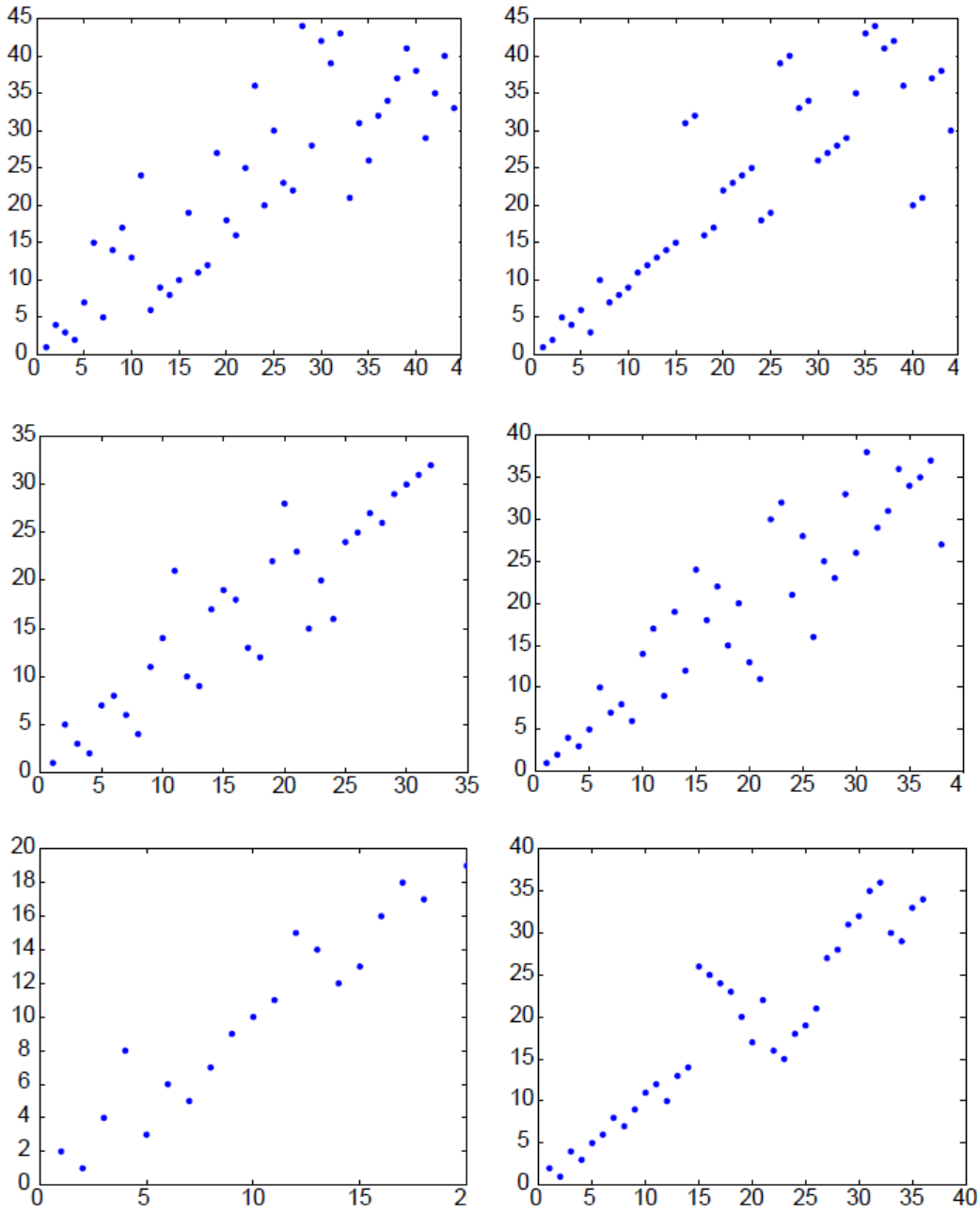

Figure 6.13: Relative Position Diagram when trained on real-world attack graphs

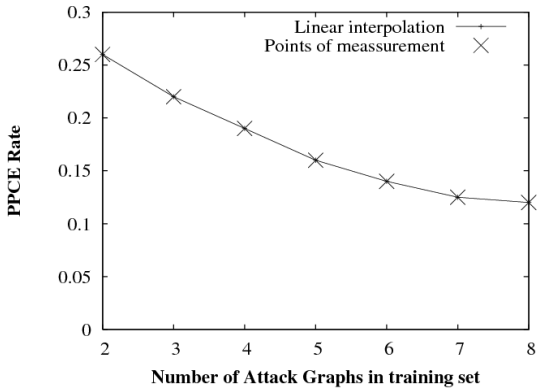

Figure 6.14: Effect of number of attack graphs used in the training data set

We repeat the experiments on the same 6 testing attack graphs but using pseudo attack graphs generated by the above procedure as the training data set. In total, 40 pseudo attack graphs are included in the training data set, and GNN is trained for 10,000 epochs for the asymptotic optimal result. The resulting Relative Position Diagrams are as shown in Figure 6.15. It can be observed that orders of ranks are preserved also to a reasonable degree, but these results are not as accurate as those obtained when using real attack graphs in the training data set. Table 6.5 lists maximum and minimum PPCE rate by repeating each experiment 3 times. The average PPCE is between 12% to 16%, indicating that on an average the proposed GNN-based ranking scheme sacrificed around 12% to 16% accuracy in terms of relative position preservation. Therefore, using pseudo attack graphs decreases the accuracy by around 2% to 4% compared with using real attack graphs for training, but on the other hand it significantly reduces the effort required to generate real attack graphs.

6.5 Conclusion

As the size and complexity of attack models/graphs usually greatly exceed human ability to visualise, understand and analyse, ranking of states is often required to identify important portions of attack models/graphs. Mehta *et al* proposed a ranking scheme based on the PageRank algorithm used by Google to measure importance of web pages on World Wide Web. We extend their scheme by modelling an attacker selectively exploiting vulnerabilities to maximise the chance of compromising the system, and intrusion detection ability of computer systems detecting and preventing

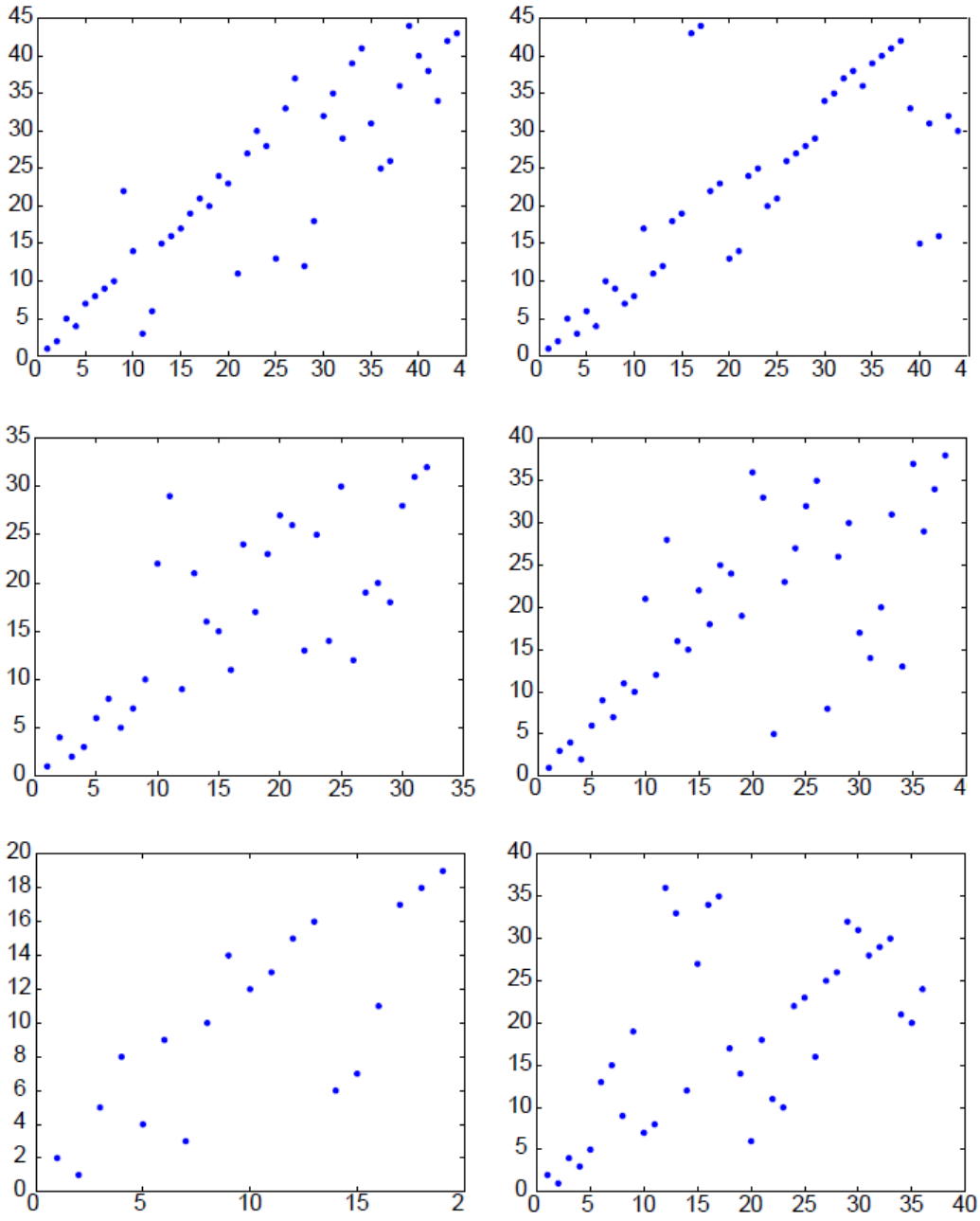

Figure 6.15: Relative Position Diagram when training on pseudo attack graphs

attackers to exploit system vulnerabilities. With the proposed ranking scheme, evaluation of system intrusion detection ability or attackers' ability in relation to probabilistically exploit vulnerabilities, when available from for example empirical data or log statistics, can be used to obtain more accurate ranks of computer system states modelled by attack models and attack graphs.

We also produced evidence that the GNN model is suitable for the task of ranking attack graphs. GNN learns a ranking function by training on examples and generalises the function to unseen data. It thus provides an alternative and time-efficient ranking scheme for attack graphs. The training stage for GNN, which is required only once, may take a relatively long period of time compared with *e.g.* PageRank scheme. However, once trained the GNN may compute ranks faster than that required by the PageRank scheme.

Another advantage of using a machine learning approach is their insensitivity to noise in a dataset. This is a known property of GNN which is based on the MLP architecture. Existing numeric approaches such as PageRank or rule based approaches are known to be sensitive to noise in the data.

The GNN provides much more flexible means for ranking attack graphs. A GNN can encode labels that may be attached to nodes or links, and hence, is able to consider additional features such as the cost of an attack or time required for an exploit. This is not possible with the PageRank scheme which is strictly limited to considerations of the topology features of a graph. We plan to investigate how the ability to encode additional information can help to rank attack graphs in the future.

Despite the proven ability of the GNN to optimally encode any useful graph structure [27], the result in this Chapter showed that the performance of the GNN has a potential for improvements. We suspect that this may be due to the choice of training parameters, or the chosen configuration of the GNN. A more careful analysis into this issue is left as a future task.

Algorithm 5: Generate$\overline{W}(M)$

```
/* The function generates the adjusted transition matrix W̄ from
   the given attack model M.                                    */
/* Input: M = (S, τ, s_1, L): the attack model where s_1 is the
   initial state of M.                                          */
/* Output: W̄ = w̄_ij, where w̄_ij represents the probability of an
   adversary exploiting the vulnerability that takes the system
   from state s_j to state s_i.                                 */

begin
    AM = Construct_Adjacency_Matrix_From_Model(M)

    /* Set the probabilities of transitions to and from the
       initial state                                            */
    for i = 1 to N do
        if AM[1, i] = 1 then
            w̄_i1 = τ(s_1 → s_i)
        else
            w̄_i1 = 0
        if ∀j, AM[i, j] = 0 then
            w̄_1i = 1
        else
            w̄_1i = 1 − d × e^{−λl(s_1, s_i)}

    /* Set the probabilities of transitions to and from other
       states                                                   */
    for i = 2 to N do
        for j = 2 to N do
            if AM[j, i] = 0 then
                w̄_ij = 0
            else
                w̄_ij = d × τ(s_j → s_i) × e^{−λl(s_1, s_j)}
```

Attack	Vulnerability Exploited
▶ sshd buffer overflow	Some versions of ssh are vulnerable to buffer overflow
▶ ftp.rhosts	Exploiting the vulnerability resulting from a writable ftp home directory
▶ remote login	Remote trust relation between machines
▶ local buffer overflow	Some `setuid root` executables are vulnerable to buffer overflow

Table 6.2: Atomic Attacks Modelled in the Sample Network

Reachable	ip_a	ip_1	ip_2
ip_a	1	1	0
ip_1	1	1	1
ip_2	1	1	1

Table 6.3: Connectivity

Trust	ip_a	ip_1	ip_2
ip_a	1	0	0
ip_1	0	1	1
ip_2	0	1	1

Table 6.4: Trust Relation

	AG-1	**AG-2**	**AG-3**	**AG-4**	**AG-5**	**AG-6**	**Avg**
Min PPCE	0.16	0.11	0.14	0.12	0.12	0.11	0.12
Max PPCE	0.21	0.15	0.19	0.17	0.15	0.15	0.16

Table 6.5: Position Pair Coupling Error

Chapter 7

Concluding Remarks

7.1 Thesis Contribution

In this thesis, we looked into how conventional firewalls are challenged by new Internet applications for streaming content delivery. The challenges mainly fall into two areas, that is to block malicious streaming traffic into a secured network, and to prevent unwanted streaming traffic from getting out of a controlled network.

The thesis first provided a detailed analysis of deficiencies in conventional firewalls that lead to incapability of handling streaming content. Following this, we investigated and presented a vulnerability from the built-in security mechanism for the Realtime Transport Protocol (RTP). By exploiting the vulnerability presented, an attacker can fully bypass conventional firewall inspection to inject arbitrary content into an RTP stream into a secured network. We then presented a technique to solve this problem so that conventional firewalls are able to effectively handle streaming content. Compared with conventional firewalls that do not consider and are unable to handle streaming traffic, the presented technique is fully aware of the RTP protocol and uses state information in RTP to effectively filter out injected malicious traffic.

The thesis then looked into the other area in streaming content handling, that is to block unwanted streaming content using conventional firewalls. In particular, we investigated the Internet telephony application *Skype* which is well-known for its capability to intelligently tunnel through firewalls by selecting customised ports and encrypting traffic to evade content based filtering. Although the capability to by pass firewall filtering may give some convenience to *Skype* users, it increases the difficulty of managing firewalls to filter out unwanted traffic. We proposed two different schemes, namely payload-based and non-payload based, for identification of *Skype* traffic. As payload based identification is not always practical due to legal,

privacy, performance, protocol change and software upgrade issues, we focus on the non-payload based scheme, and use the payload based scheme mainly to verify its non-payload based counterpart. Our research results reveal that, at least to a certain extent, effort by Skype to bypass firewall filtering can be overcome.

Then we looked at a related problem. Streaming content like Skype and RTP tends to use dynamically negotiated ports, and hence network access polices cannot be easily translated into rule sets that firewalls strictly enforce. Translating a high-level security policy written in a natural language into firewall rule tables, a much lower-level description of the policy, is an error prone task particularly in the context of streaming content. We provided a technique to compare and analyse firewall rule tables. With the provided technique, firewall rule tables can be analysed at a much fine-grained level. This can assist in finding and locating potentially incorrectly interpreted rules.

Although techniques have been provided to handle streaming content, conventional firewalls still have limitation like any other technologies. Another scenario that conventional firewalls are unable to handle is multi-stage intrusions. As an effort to complement firewall capabilities in detecting multi-stage intrusions, attack graphs are proposed and have gained much attention in the research community. However, size and complexity of attack graphs usually greatly exceed human ability to visualise, understand and analyse. To identify important portions of attack graphs, Mehta *et al.* proposed to rank attack graphs and highlight states that have higher ranks. The proposed ranking scheme is based on similarity, but has not considered differences, between attack graphs and web graphs. We extend Mehta et al.'s scheme by taking into consideration differences between web surfing scenarios and computer system intrusion scenarios. We experiment with the extended ranking scheme using a network model similar to what is used in Mehta et al.'s scheme, and compared the results with Mehta et al.'s scheme. The experiments yielded promising results where consistent and meaningful ranks are yielded for a range of parameters.

A network may be re-engineered into many different forms. This may result in re-generation and re-ranking of attack graphs. We provided an alternative neural network based ranking scheme to attack graphs in an effort to reduce computation required for re-ranking of attack graph. Once trained, a neural network can rank large sets of attack graphs efficiently due to its ability to generalise over unseen data. In our experiments, the neural network based ranking scheme successfully demonstrate the ability to generate similar ranks to that of the other ranking scheme

proposed earlier in the thesis.

7.2 Limitations

Due to certain restrictions in our experimental environment, the technique provided in Chapter 3 to filter out injected streaming content has only been applied to offline traffic traces analysis. It has not been applied to live traffic scanning and hence application of the presented technique may not suffice performance requirement for live traffic processing.

Similarly, the Skype traffic detection scheme provided in Chapter 4 has not been experimented with live Skype traffic. The experimental results are only derived from scanning offline traffic traces. Application of the detection scheme may therefore be limited by factors such as performance. Moreover, both detection schemes (payload and non-payload based) are based upon empirical observation on captured Skype traffic. Therefore, the presented techniques are vulnerable to situations such as change of network configuration, or upgrade of software version.

The firewall rule set compare technique proposed in Chapter 5 is only applicable to the packet filtering type of firewalls. It cannot be applied to more advanced types of firewalls such as application gateways. Moreover, it can only be used to compare rule sets which consists of IPv4 addresses.

Despite the proven ability of the GNN to optimally encode any useful graph structure, the result in Chapter 6 shows potential room for improvement on the presented GNN based ranking scheme. We suspect that this may be due to the choice of training parameters, or the chosen configuration of the GNN.

7.3 Open Problems

Although the technique provided to filter out malicious streaming content has only be tested with offline traffic traces, it has the potential to be applied to live traffic scanning and filtering. Performance will need to be investigated and be experimented before the provided technique can be applied to live traffic scanning. A practical piece of future work is to build the presented technique into a firewall and to experiment with the situation of live traffic filtering. It is also worth to investigate how varying parameters affect the effectiveness of the presented filtering technique.

Similarly, the Skype traffic detection scheme has not been applied to live traffic scanning. It would be interesting to implement the Skype detection technique into a firewall and experiment with live Skype traffic. This will reveal any performance issue as well as other potential issues with the presented technique. Also improving robustness of the presented technique against situations such as change of network configuration or upgrade of software version will be an interesting topic of future research.

It can be seen that performance of the proposed GNN ranking scheme for attack graphs has room to improve. This may be achieved by adjusting training parameters or choose different configuration of the GNN. Therefore, it will be interesting to experiment with different training parameters and different GNN configuration and compare the performance with what is achieved in this thesis.

Bibliography

[1] http://www.websense.com.

[2] A. Pescape A. Dainotti and G. Ventre. A Packet-level Traffic Model of Startcraft. In *Proceedings of the 2005 Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 33–42. IEEE, 2005.

[3] A. Feldmann and S. Muthukrishnan. Tradeoffs for Packet Classification. In *Proc. INFOCOM*, volume 3, pages 1193–1202, March 2000.

[4] A. Grant, J. H. Meadows. *Communication Technology Update and Fundamentals.* Elsevier, 2008.

[5] A. Wool. Architecting the Lumeta firewall analyzer. In *10th USENIX Security Symposium*, Washington, D.C., 2001. USENIX.

[6] A. Y. NG, A. X. Zheng, and M. I. Jordan. Link analysis, eigenvectors and stability. In *Proceedings of International Conference on Research and Development in Information Retrieval (SIGIR 2001)*, New York, 2001. ACM.

[7] Anonymous. *IP Multicast Initiative (IPMI).* Stardust Forums Inc., 1998.

[8] Anonymous. Why is Skype better than Net2Phone, ICQ, AIM, MSN, etc? http://support.skype.com/index.php?_a=knowledgebase &_j=questiondetails&_i=70&nav2=General, 2004.

[9] Anonymous. Using RealOne Player or RealPlayer with Firewalls. http://service.real.com/firewall/rplay.html, 2004.

[10] ATT Research. http://www.graphviz.org, 2004.

[11] B. B. Madan, K. G. Popstojanova, K. Vaidyanathan, and K. S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. In *Dependable Systems and Networks-Performance and Dependability Symposium*, number 167-186, 2004.

[12] B. Hari, S. Suri and G. Parulkar. Detecting and Resolving Packet Filter Conflicts. In *IEEE INFOCOM*, 2000.

[13] Bhadeshia, University Of Cambridge. Neural Networks in Materials Science, 2006.

[14] C. A. Phillips and L. P. Swiler. A graph based system for network vulnerability analysis. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2000.

[15] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-Level Traffic Measurements from the Sprint IP Backbone. *IEEE Network*, 17(6):6–16, 2003.

[16] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall PTR, 2002.

[17] Cisco. Stateful Inspection Technology. Technical report, CheckPoint Software Technologies Ltd, 2004.

[18] D. Bacher, A. Swan, and L. A. Rowe. rtpmon: A Third-Party RTCP Monitor, 1996. http://bmrc.berkeley.edu/people/drbacher/projects/mm96-demo/index.html.

[19] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router Plugins: A Modular and Extensible Software Framework for Modern High Performance Integrated Services Router. Wucs-98-08, Washington University, February 1998.

[20] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router Plugins: A Software Architecture for Next-generation Routers. *IEEE Transaction on Networking*, 8(1):2–15, February 2000.

[21] D. Eppstein and S. Muthukrishnan. Internet Packet Filter Management and Rectangle Geometry. In *12th Annual ACM_SHAM Symposium on Discrete Algorithms (SODA)*, 2001.

[22] E. Mizutani, S. E. Dreyfus. On complexity analysis of supervised MLP-learning for algorithmic comparisons. In *International Joint Conference on Neural Networks (IJCNN)*, volume 1, 2001.

[23] E. S. Al-Shaer and H. H. Hamed. Discovery of Policy Anomalies in Distributed Firewall. In *IEEE INFOCOM, Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2605 – 2616, March 2004.

[24] E. S. Al-Shaer and H. H. Hamed. Firewall Policy Advisor For Anomaly Discovery And Rule Editing. In *IEEE/IFIP 8th Int. Symp. Integrated Network Management*, 2004.

[25] Ethereal. Ethereal. http://www.ethereal.com, 2006.

[26] F. Baboescu and G. Varghese. Scalable packet classification. *IEEE/ACM Transactions on Networking*, 13(1):2–14, February 2005.

[27] A. C. Tsoi M. Hagenbuchner F. Scarselli, M. Gori and G. Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, volume 20(number 1):81–102, January 2009.

[28] M. Gori M. Hagenbuchner A. C. Tsoi F. Scarselli, S. L. Yong and M. Maggini. Graph neural networks for ranking web pages. In *Web Intelligence Conference*, pages 666–672, 2005.

[29] F. Scarselli, A. C. Tsoi, M. Gori, and M. Hagenbuchner. A new neural network model for graph processing. Technical Report DII 1/05, University of Siena, Aug 2005.

[30] F. Scarselli, S. L. Yong, M. Hagenbuchner, A. C. Tsoi. Adaptive Page Ranking with Neural Networks. In *WWW (Special interest tracks and posters)*, pages 936–937, 2005.

[31] K. P. Fung. SOCKS5-based Firewall Support for UDP-based Applications. Master's thesis, The Hong Kong Polytechnic Univ., Dept. of Computing, Hong

Kong, PRC, http://www2.comp.polyu.edu.hk/~csrchang/MSc/Billy.pdf, 1999.

[32] G. H. Golub and V. Loan. *Matrix computation*. The Johns Hopkins University Press, 1993.

[33] R. Gusella. A Measurement Study of Diskless Workstation Traffic on an Ethernet. *IEEE Transactions on Communications*, 38(9):1557–1568, 1990.

[34] H. Fowler and W. Leland. Local Area Network Traffic Characteristics, with Implications for Broadband Network Congestion Management. *IEEE JSAC*, 9(7):1139– 1149, 1991.

[35] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2336, Apr 1998.

[36] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, Jan 1996.

[37] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Jul 2003.

[38] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., 866 Third Avenue, New York, New York 10022, 1994.

[39] HLFL. HLFL—high level firewall language. http://www.hlfl.org/, 2002.

[40] ITU-T. Recommendation H.323: Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Nonguaranteed Quality of Service. ITU-T, 1996.

[41] J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of the Second IEEE International Information Assurance Workshop*, 2004.

[42] J. Homer, A. Varikuti, X. Ou and M. McQueen. Improving Attack Graph Visualization through Data Reduction and Attack Grouping. In *Visualization for Computer Security, LNCS 5210*, Hamburg, Germany, September 2008. Springer.

[43] J. Lunteren and T. Engbersen. Fast and scalable packet classification. *IEEE Journal on Selected Areas in Communications*, 21(4):560– 571, May 2003.

[44] J. Merwe, R. Caceres, Y. Chu and C. Sreenan. mmdump: a tool for monitoring internet multimedia traffic. *ACM SIGCOMM Computer Communication Review*, 30:48–59, 2000.

[45] J. Nathan. http://nemesis.sourceforge.net/, 2004.

[46] J. Wack, K. Cutler, and J. Pole. Guidelines on Firewalls and Firewall Policy. Technical report, National Institute of Standards and Technology, 2002.

[47] R. A. Johnson and D. W. Wichem. *Applied Multivariate Statistical Analysis*. Upper Saddle river, New Jersey: Prentice Hall, 1998.

[48] K. Chen, P. Huang, C. Huang, and C. Lei. Game Traffic Analysis: An MMORPG Perspective. In *Proceedings of NOSSDAV*, 2005.

[49] M. Stinchcombe K. Hornik and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[50] K. P. Fung and Rocky K. C. Chang. Secure media streaming & secure adaptation for non-scalable video. *ICIP*, 3:1763 – 1766, 2004.

[51] K. Strassberg, R. Gondek, and G. Rollie. *Firewalls: The Complete Reference*. McGraw-Hill/Osborne, 2002.

[52] K. Suh, R. Figueiredo, J. Kurose, D. Towsley. Characterizing and Detecting Skype-Relayed Traffic. In *Proceedings of IEEE Infocom*, Barcelona, Apr 2006.

[53] J. Horton L. Lu, R. Safavi-Naini and W. Susilo. On securing rtp-based streaming content with firewalls. In *CANS*, pages 304–319, 2005.

[54] J. Horton L. Lu, R. Safavi-Naini and W. Susilo. An adversary aware and intrusion detection aware attack model ranking scheme. In *ACNS*, pages 65–86, 2007.

[55] J. Horton L. Lu, R. Safavi-Naini and W. Susilo. Transport layer identification of skype traffic. In *ICOIN*, pages 465–481, 2007.

[56] M. Hagenbuchner W. Susilo J. Horton S. L. Yong L. Lu, R. Safavi-Naini and A. C. Tsoi. Ranking attack graphs with graph neural networks. In *ISPEC*, pages 345–359, 2009.

[57] L. Lu, J. Horton, R. Safavi-Naini, and W. Susilo. An Adversary Aware and Intrusion Detection Aware Attack Model Ranking Scheme. In *Proceeding of 5th International Conference on Applied Cryptography and Network Security (ACNS 07)*, Zhuhai, China, Jun 2007. Springer-Verlag.

[58] L. Lu, R. Safavi-Naini, J. Horton and W. Susilo. Comparing and Debugging Firewall Rule Tables. *IET Information Security*, 1(4):143–151, 2007.

[59] L. Wang, T. Islam, T. Long, A. Singhal and S. Jajodia. An Attack Graph-Based Probabilistic Security Metric. In *Data and Applications Security XXII, LNCS 5094*, Hamburg, Germany, July 2008. Springer.

[60] M. A. Ruiz-Sanchez, E. W. Biersack and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, 5(2):8–23, Mar/Apr 2001.

[61] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711, Mar 2004.

[62] M. Bianchini, M. Gori, and F. Scarsell. Inside PageRank. *ACM Transactions on Internet Technology*, 5(1):92–118, Feb 2005.

[63] M. buddhikot, S. Suri, and M. Waldvogel. *Space Decomposition Techniques for Fast Layer-4 Switching*, pages 25–41. Protocols for High SpeedNetworks IV, 1999.

[64] D. LaPoint M. Claypool and J. Winslow. Network Analysis of Counter-strike and Starcraft. In *Proceedings of 22nd IEEE International Performance, Computering and Communication Conference(IPCCC)*, Phoenix, Arizona, USA, April 2003. IEEE.

[65] M. D. Petty and A. Mukherjee. Experimental Comparison of d-Rectangle Intersection Algorithms Applied to HLA Data Distribution. In *Proceedings of the 1997 Distributed Simulation Symposium*, pages 13–26, Orlando FL, September 1997.

[66] M. Dacier, Y. Deswarte, and M. Kaaniche. Quantitative assessment of operational security: Models and tools. Technical Report 96493, LAAS, May 1996.

[67] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543, Mar 1999.

[68] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Pattern: A View from Ames Internet Exchange. In *Proceedings of ITC Specialist Seminar on Measurement and Modeling of IP Traffic*, pages 1–11. Cooperative Association for Internet Data Analysis (CAIDA), September 2000.

[69] A. Mena and J. Heidemann. An Empirical Study of Real Audio Traffic. In *Proceedings of the IEEE Infocom*, pages 101–110, Tel-Aviv, Israel, March 2000. IEEE.

[70] N. Damianou, N. Dulay, E. Lupu and M. Sloman. The Ponder Policy Specification Language. In *Workshop on Policies for Distributed Systems and Networks, LNCS*, volume 1995, pages 18–38. Springer, 2001.

[71] D. Nelson. Firewall Information for Windows Media Services 9 Series. http://www.microsoft.com/windows/windowsmedia/serve/firewall.aspx#PortAllocation, 2007.

[72] Real Networks. Real audio. http://asia-en.real.com/guide/radio/list.html, 2004.

[73] NuSMV. NuSMV: a new symbolic model checker. http://nusmv.irst.itc.it/, 2007.

[74] B. Wayne O, Xinming and A. McQueen. A scalable approach to attack graph generation. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, New York, NY, USA, 2006. ACM.

[75] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[76] A. Heyde P. A. Branch and G. J. Armitage. Rapid identification of skype traffic flows. In *NOSSDAV '09: Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, pages 91–96, New York, NY, USA, 2009. ACM.

[77] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, and D. Estrin. An Empirical Workload Model for Driving Widearea TCP/IP Network Simulations. *Internetworking: Research and Experience*, 3(1):1–26, 1992.

[78] M. Gori P. Frasconi and A. Sperduti. A general framework for adaptive processing of data structures. 9(5):768–786, September 1998.

[79] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, Mar/Apr 2001.

[80] P. Gupta and V. Shmatikov. Security Analysis of Voice-over-IP Protocols. In *Computer Security Foundations Symposium*. IEEE, 2007.

[81] M. Catherine R. A. Kemmerer and J. K. Millen. Three system for cryptographic protocol analysis. *Cryptology*, 7(2):79–130, 1994.

[82] R. Jain and S. Routhier. Packet Trains - Measurements and a New Model for Computer Network Traffic. *IEEE JSAC*, 4(6):986–995, 1986.

[83] Y. Deswarte R. Ortalo and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *Software Engineering*, 25(5):633–650, 1999.

[84] R. Ortalo, Y. Deshwarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. In *IEEE Transactions on Software Engineering*, pages 71–79, 1999.

[85] R. Sawilla and X. Ou. Identifying Critical Attack Assets in Dependency Attack Graphs . In *13th European Symposium on Research in Computer Security, LNCS 5283*, Malaga, Spain, Octorber 2008. Springer.

[86] R. Zimmermann, K. Fu, C. Shahabi, and M. Jahangiri. A Multi-Threshold Online Smoothing Technique for Variable Rate Multimedia Streams. *Multimedia Tools and Applications*, 28(1):23–49, 2006.

[87] R.P. Lippmann and K.W. Ingols. An Annotated Review of Past Papers on Attack Graphs. Technical Report ESC-TR-2005-054, Lincoln Laboratory, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2005.

[88] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. Technical report, Department of Computer Science, Columbia University, New York, 2004.

[89] S. Brin, L. Page, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical Report 1999-66, Stanford University, 1999.

[90] S. Garfinkel. Can 9 Million Skype Users Be Wrong? http://www.csoonline.com/read/030105/machine.html, 2004.

[91] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of IPTPS'06*, Santa Barbara, CA, Feb 2006.

[92] S. Hazehurst. Algorithms for Analyzing Firewall and Router Access Lists. Technical report trwitscs-1999, Department of Computer Science, University of the Witwatersrand, South Africa, 1999.

[93] S. Hazelhurst, A. Attar and R. Sinnappan. Algorithms for Improving the Dependability of Firewall and Filter Rule Lists. In *International Conference on Dependable Systems and Networks (DSN 2000)*, 2000.

[94] S. Jha and J. Wing. Survivability analysis of networked systems. In *23rd International Conference on Software Engineering(ICSE01)*, number 03-07, 2001.

[95] S. Jha, O. Sheyner, and J. Wing. Two Formal Analysis of Attack Graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 49. IEEE, 2002.

[96] S. Oliver S. Subhabrata and W. Dongmei. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. In *Proceedings International WWW Conference*, New York, USA, 2004.

[97] H. Schulzrinne. rtpdump. http://www.cs.columbia.edu/~hgs/rtp/rtpdump.html, 1999.

[98] O. Sheyner. *Scenario Graphs and Attack Graphs*. PhD in Computer science, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 2004. CMU-CS-04-122.

[99] O. Sheyner and J. Wing. Tools for generating and analyzing attack graphs. In *Formal Methods for Components and Objects, LNCS 3188*, pages 344–371. Springer Berlin / Heidelberg, 2004.

[100] Skype. http://www.skype.com, 2002.

[101] Skype FAQ. http://www.skype.com/help_faq.html, 2004.

[102] W. Stallings. *High Speed Networks, TCP/IP and ATM design principles*. Prentice-Hall Inc., 1998.

[103] Sun Microsystems Inc. Java Media Framework, 2005.

[104] H. Susan. Policy-based management: Bridging the gap. In *ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference*, page 209, Washington, DC, USA, 1999. IEEE Computer Society.

[105] L. P. Swiler, C. Phillips, and D. Ellis. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2001.

[106] T. Berson. Skype Security Evaluation. Technical report, Anagram Laboratories, Palo Alto, CA 94301, USA, Oct 2005.

[107] T. K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu, and N. Dulay. Compiling policy descriptions into reconfigurable firewall processors. In *Systems, Man and Cybernetics, IEEE International Conference*, 2003.

[108] M. Faloutsos T. Karagiannis, A. Broido and K. Claffy. Transport Layer Identification of P2P Traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, Taormina, Sicily, Italy, 2004. ACM Press.

[109] T. Woo. A Modular Approach to Packet Classification: Algorithms and Results. In *IEEE INFOCOM*, 2000.

[110] V. Frost and B. Melamed. Traffic Modeling for Telecommunications Networks. *IEEE Communications Magazine*, 32(3):70–80, 1994.

[111] V. Mehta, C. Bartzis, H. Zhu, E. Clarke and J. Wing. Ranking Attack Graphs. In *Proceeding of the 9th International Symposium On Recent Advances In Intrusion Detection*, Hamburg, Germany, September 2006. Springer.

[112] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking (TON)*, 3(3):226 – 244, 1995.

[113] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *ACM Computer Communication Review*, 28(4):191–202, September 1998.

[114] V. Srinivasan, S. Suri, and G. Varghese. Packet Classfication using tuple space search. *ACM Computer Communication Review*, pages 135 – 146, September 1999.

[115] Verso Technologies. Verso Netspective Enterprise. http://www.verso.com/enterprise/netspective/netspectiv e_brochure.pdf, 2004.

[116] W. Feng, F. Chang, W. Feng, and J. Walpole. A Traffic Characterization of Popular On-Line Games. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 13(3):488–500, 2005.

[117] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security, Repelling the Wily Hacker*. Addison-Wesley, 1994.

[118] WikiPedia. List of SIP softwares. http://en.wikipedia.org/wiki/List_of_SIP_software, 2010.

[119] X. Sun, S. K. Sahni, and Y. Q. Zhao. Packet classification consuming small amount of memory. *IEEE/ACM Transactions on Networking*, 13(5):1135–1145, 2005.

[120] K. Nissim Y. Bartal, A. Mayer and A. Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, 2004.

[121] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A Novel Firewall-Management Toolkit. In *IEEE Symposium on Security and Privacy*, 1999.

[122] Z. Fu, F. Wu, H. Huang, K. Lob, F. Gong, I. Baldine and C. Xu. IPSec/VPN Security Policy: Correctness Conflict Dection and Resolution. In *Policy'2001 Workshop*, 2001.