An Integrated Framework for Firewall Testing and Validation

Mehdi Akiki

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

June 2009

© Mehdi Akiki, 2009

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-67146-7
Our file Notre référence
ISBN: 978-0-494-67146-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

ABSTRACT

An Integrated Framework for Automated Firewall Testing and Validation

Mehdi Akiki

In today's global world, most corporations are bound to have an Internet presence. This phenomenon has led to a significant increase in all kinds of network attacks. Firewalls are used to protect organizational networks against these attacks. Firewall design is based on a set of filtering rules. Because of the nature of these rules, and due to the rising complexity of security policies, an increasing number of mistakes are found in configurations. A reliable and automated technique for testing firewall configuration is becoming necessary to ensure the full functionality of the firewall.

In this thesis, a new approach to fully test a firewall has been developed using a white box approach that takes into account its inner implementation. Also—thanks to the information provided by the network information file—the environment where the firewall will be deployed is considered, ensuring a better accuracy and performance than previous work. Moreover, the method uses a combination of algorithms that remove common misconfigurations widely present in current firewall configurations [1] and guarantees a coverage that is greater than previous methods for generating test sets with a novel test set generation approach.

The developed framework is fully automated and contains the full steps to get testing done, from the parsing of the firewall file to the generation of the test set based on the actual configuration of the firewall to the correction of the error in the firewall file,

avoiding all types of errors of omission and misconfiguration that occur during a manual configuration.

Keywords: *Firewall, Policy Language, Conflict Free Rules, Rule Set, White Box Testing, Misconfiguration Errors, Configuration, Rule Update*

ACKNOWLEDGEMENTS

First, I would like to express my gratefulness to my mentor and supervisor, Dr. Abdeslam En-nouaary, whose guidance, encouragement, support and kindness have made this thesis possible. I wholeheartedly appreciate his suggestions and the inspiring discussions we enjoyed.

I also want to thank all the professors who have taught me invaluable lessons while I was a student at Concordia University.

I dedicate this work to my father, my mother, my wife Houda, my brother, and my friends.

Table of Contents

List of Figures ... vi

List of Tables ... viii

Chapter 1: Introduction ... 1

1.1 Objectives of This Thesis ... 2

1.2 Organization of this Thesis ... 3

Chapter 2: Network Security ... 4

2.1 Introduction to Networking ... 4

2.2 TCP/IP: The Language of the Internet ... 6

2.3 Risk Management: The Game of Security ... 13

2.4 Types and Sources of Network Threats ... 14

2.5 Firewalls ... 20

2.5.1 Firewalls Definitions ... 20

2.5.2 Types of Firewalls ... 23

2.6 Summary ... 27

Chapter 3: Firewall Testing ... 29

3.1 Passive Testing ... 29

3.1.1 Vulnerability Testing ... 30

3.1.2 Real Time Testing ... 31

3.1.3 Formal Network Building ... 33

3.1.4 Algorithmic Approach ... 35

3.1.5 Query Engines ... 39

3.2 Active Testing ... 40

3.2.1 Graphical Analysis ... 40

3.2.2 Formal Policy Writing 42
3.3 Summary 44
Chapter 4: A New Approach for Automated Firewall Testing and Validation 45
4.1 General Presentation of the Framework 46
4.2 The Different Modules of the Framework 48
4.2.1 Firewall Parser 48
4.2.2 Misconfiguration Removal 49
4.2.3 Test Set Generation 53
4.2.4 Test Execution 58
4.2.5 Log File Parser 60
4.2.6 Results Analyzer 61
4.2.7 Rule Updater 62
4.3 Summary 64
Chapter 5: Implementation and Case Study 65
5.1 Implementation 65
5.1.1 Software and Tools Used 65
5.1.2 UML Class Diagram of the Framework 67
5.2 Case Study 69
5.3 Summary 78
Chapter 6: Conclusions and Summary 80
6.1 Achievements 80
6.2 Future Work 81
References 82

List of Figures

Figure 2.1: The OSI Model ... 5

Figure 2.2: The IP header ... 7

Figure 2.3: IP Spoofing Attack ... 8

Figure 2.4: IP Hijacking ... 9

Figure 2.5: 3-Way TCP Handshake ... 11

Figure 2.6: Denial of Service Attack ... 15

Figure 2.7: Bastion Host ... 21

Figure 2.8: DMZ with one firewall ... 22

Figure 2.9: Proxy server architecture ... 23

Figure 2.10: Example of an application level firewall ... 24

Figure 2.11: Packet Filtering Firewall ... 25

Figure 3.1: Blowtorch Packet Flow ... 31

Figure 3.3: Vigna's Firewall Testing Approach ... 34

Figure 3.4: Different Rule Configurations ... 36

Figure 3.5: Policy Segmentation Technique ... 37

Figure 3.7: Fang Query Engine Data Flow ... 39

Figure 3.8: Graphical User Interface ... 41

Figure 3.10: Or-BAC Model in XML ... 43

Figure 4.2: Testing Framework Schema ... 45

Figure 4.2: Misconfiguration Algorithm ... 51

Figure 4.3: Test Generation Algorithm ... 53

Figure 4.4: Test Bench Schema 58

Figure 4.5: Correction Algorithm's Pseudo code 61

Figure 5.1: UML Diagram of the test framework... 67

Figure 5.2: Our GUI interface for Firewall testing 74

List of Tables

Table 3.1: Rule Anomaly Types 36

Table 4.1: Format of the Parsed Rule Set 48

Table 4.2: Correction to the Current Algorithm Technique 52

Table 4.3: Network Information File 54

Table 4.4: Iptables Log Entry 59

Table 4.5: Useful log file information 60

Table 4.6: Error Detection and Correction Example 62

Table 5.1: Specifications of the Tested Firewall 68

Table 5.2: Input configuration file 69

Table 5.3: Parsing Results of the Firewall File 70

Table 5.4: Results of the Misconfiguration Removal Algorithm 71

Table 5.5: Generated Packets per Rule 72

Table 5.6: Faulty packets with their decision 74

Table 5.7: Updated Rules 75

Table 5.8: Output configuration file 76

Table 5.9: Comparison with other testing approaches 77

Chapter 1: Introduction

The Internet has dramatically transformed the manner in which organizations and individuals conduct business. The increased communication potential and efficiency of network technology have made e-commerce, web-based business-to-business operations, and global connectivity integral components of successful business strategies. For an ever-increasing number of enterprises, the online presence is crucial. At the same time, however, this global network has created a variety of problems such as intrusions, both automated and manual, which cause damage that can cost organizations enormously in terms of money, efficiency, and resources. Thus, organizations must discover methods to achieve their mission goals in relation to the Internet, while simultaneously keeping their websites secure from attacks. Given the unsafe nature of Internet protocol[1], as well as the increasing sophistication of attacks to which organizations are exposed, secure and reliable access to the Internet must be a priority. In effect, a system that protects organizations while letting them operate in a normal capacity is critical.

Firewalls play that role in the sense that they act as network devices that filter the ingoing as well as outgoing traffic that passes through them, preventing various Internet attacks and intrusions [2]. In other words, they constitute a bridge between trusted and untrusted networks. In this respect, firewalls are the cornerstone of a corporation's general security policy. Thus, these devices have to work properly by correctly translating the security policy of a given corporation or organization. In this sense, the

testing phase of a firewall is the most essential step towards ensuring that the right functionality of the device is deployed, thus avoiding network leakage and security holes.

Firewalls are placed between a private network and the Internet. They often represent the single point of failure in an organization's network security, since all connections are established through the firewall policing the network traffic for a given organization. Moreover, firewalls are only as efficient as their configuration. Configuration is thus a vital task, if not the most significant factor, in the security firewalls provide. According to [1], most corporate firewalls are poorly configured. The main reasons for this deficiency are:

- The rising number of rules needed to enforce more and more protectionist policies
- The low level, archaic configuration language used to enforce a given policy

The problematic of testing to validate a system is being able to test a system covering all cases while keeping it feasible. In an ideal world, exhaustive testing would be conducted. Unfortunately, due to the duration of testing, this objective would take a considerable amount of time. Random testing can be an alternative, but is just too inaccurate to inspire confidence. Given the aforementioned situation, there is a clear need to test and correct each security policy in order to produce a rule set that accurately maps it. An intelligent method is needed to select the packets used for testing in order to save time on exhaustive testing and allow for more accurate results than with a random testing approach.

1.1 Objectives of this Thesis

Having grasped the ongoing need for testing, this thesis presents an automated framework to fully test and update a given firewall configuration, using an intelligent test set selection and correction process. This thesis will describe an approach that takes as an input a given firewall configuration file, tests it, and then corrects it, ultimately producing an error free configuration file.

The automated testing approach is based on a white box testing approach thanks to a set of algorithms that take into account the inner configuration of the firewall, allowing it to obtain an accurate and complete test set. The test set obtained can then be applied against the firewall. Once the test is complete, a comparison module detects errors as it tests results against expected values. The last step is to correct and update the firewall file.

Moreover, the approach starts by translating the rules into a formal language from the configuration file. Then, a misconfiguration algorithm is applied in conjunction with a test set generator module that takes into account both addresses and services in the network to produce test packets, taking into account the environment where the firewall is tested. This step will then result in a test set that maps the inner functioning of the firewall in its environment. Next, the test is applied against the configured firewall. Finally, the obtained results are analyzed for correctness; a firewall updater module detects the inconsistencies and corrects them.

This method is based on a combination of algorithms and real-time testing to produce a reduced yet accurate test set. Due to the increasing number of

misconfigurations in actual firewalls, this approach uses the advantages of the algorithmic approach by removing all misconfiguration errors and optimizing the test set generation. At the same time, it takes into account the configuration of the firewall as well as its environment to provide a realistic set of tests, while keeping the test length reasonable. Also, the test is applied in real time, ensuring the normal functioning of the device, as opposed to simulation-only approaches where the functionality of the device is not taken into consideration. In short, this approach is optimized, complete, and real-time.

1.2 Organization of this Thesis

The structure of this thesis starts with an introduction to computer and network security. Beginning with an introduction to networking, Chapter 2 discusses these concepts in detail and introduces more advanced topics like TCP/IP and sources of network threats. A brief introduction to the role of firewalls in a network is also discussed. After that, Chapter 3 presents a literature review of testing methods used to solve the complex issues related to firewall testing. Chapter 4 introduces a new approach for testing a firewall configuration. The chapter highlights the different modules composing the framework and its functioning as a whole. Chapter 5 describes the implementation of the testing framework using Java programming language as well as a case study to showcase the use of the tool. The case study showcases the use of a firewall configuration from the Linux firewall iptables. The chapter ends with a comparison of the framework against existing tools. To conclude, Chapter 6 presents a summary of the research and presents propositions that can be useful to improve the solutions for this problem in future work.

Chapter 2: Network Security

In today's increasingly digital world, there exists a greater need to grasps the fundamental concepts of network security and risks involved. The notion of network security raises a number of complex problems. While these concerns have traditionally resided in the domain of trained specialists, this chapter provides an overview of the relevant issues, along with some background on networking and Internet protocols. Factors like risk management, network threats, and protection tools like firewalls are considered. A thorough understanding of network security is essential to comprehend the growing need to have reliable firewalls in organizations of all sizes.

2.1 Introduction to Networking

A network consists of two or more computers that are linked in order to share resources such as printers and CD-ROMs, exchange files, or allow electronic communications. The computers on a network may be linked through cables, telephone lines, radio waves, satellites, or infrared light beams. An example of a network is a Local Area Network (LAN), a computer network that spans a relatively small area.

The International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model is an abstract description of communications and network protocol designed according to layers. The model consists of seven layers, from highest to lowest: Application, Presentation, Session, Transport, Network, Data Link, and Physical layers. Each of the layers, as illustrated in Figure 2.1, is dependent upon the ones below, with the physical network hardware—like network adapters and connecting wires—at the bottom.

OSI Model			
	Data Unit	**Layer**	**Function**
Host Layers	Data	7. Application	Network process to application
		6. Presentation	Data representation and encryption
		5. Session	Inter host communication
	Segment	4. Transport	End-to-end connections and reliability (TCP)
Media Layer	Packet/Datagram	3. Network	Path determination and logical addressing (IP)
	Frame	2. Data Link	Physical addressing (MAC & LLC)
	Bit	1. Physical	Media, signal and binary transmission

Figure 2.1: The OSI Model

This model can be readily understood through the familiar metaphor of telephone communication. The telephone is a device that allows us to speak over a great distance—the application layer. However, a telephone by itself is useless without being able to convert the sound of speech into an electronic signal that can be carried over a telephone line—the functionality made possible by the layers beneath the application layer. Ultimately, we reach the physical layer—the hardware and infrastructure that connects the phones to a switch within the system's larger network of switches. Making a telephone call involves lifting the receiver and dialing a number, which directs the request to a central dispatch. From there, the central office specifies the appropriate phone to start ringing. The session begins when that phone is answered and the two parties commence speaking. This dynamic is, in essence, exactly how a computer network works. For its functionality, each layer of the ISO/OSI Reference Model depends on the operation of the layer directly beneath.

With this basic conceptual understanding of networking in place, it is useful to consider a specific example of the world's most widely used network. The Internet

functions as a meta-network—the biggest network of all the networks in the world. To access an online resource, like a website, does not actually involve connecting directly to the Internet. An entity connects to a sub-network that ultimately links up with the Internet backbone, consisting of vast numbers of interconnected, high-speed, and high-capacity network components like core routers that can only function with a language protocol. TCP/IP is the most used and common language protocol and is often described as the language of the Internet. A throughout description of this protocol is provided as follows.

2.2 TCP/IP: The Language of the Internet

The Internet functions using Transport Control Protocol/Internet Protocol (TCP/IP). Thus, fluency in TCP/IP is required to access the Internet. In terms of the ISO/OSI Reference Model, IP corresponds to the Network layer, and TCP to the Transport layer. Any host with TCP/IP functionality, such as an operating system, is able to support applications, such as web browsers, that make use of the network. Collectively, the suite of Internet protocols is known as TCP/IP, since the two were designed together and are inevitably found together, working in tandem.

Among the most significant elements of TCP/IP is not its technological character, but what might be termed its uniquely social one. Because this protocol is open, it can be implemented freely by anyone who wishes. Around the globe, members of the scientific community regularly contribute to the protocol design that drives the Internet, through their role in the Internet Engineering Task Force (IETF) work groups.

IP, as a protocol corresponding to the Network layer, enables the hosts to communicate with each other. IP ensures that devices with Internet connectivity are able

to identify and reach each other by carrying datagrams; mapping Internet addresses, such as 10.2.3.4, to physical network addresses, such as 08:00:69:0a:ca:8f; and routing. The IP header content is described in Figure 2.2.

<table>
<tr><td>4-bit Version</td><td>4-bit Header Length</td><td>8-bit Type of Service</td><td colspan="2">16-bit Total Length</td></tr>
<tr><td colspan="3">16-Bit Identification Number</td><td>3-bit Flags</td><td>13-bit fragment Offset</td></tr>
<tr><td colspan="2">8-Bit Time To Live (TTL)</td><td>8-Bit Protocol</td><td colspan="2">16-Bit Header Checksum</td></tr>
<tr><td colspan="5">32-Bit Source IP Address</td></tr>
<tr><td colspan="5">32-Bit Destination IP Address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

Figure 2.2: The IP header

There are several significant elements that contribute to the incredible flexibility and robustness of IP as a protocol, like routing and data encapsulation. However, IP was not designed to provide a reliable service. This means that the network makes no guarantee about packets arrival, which can cause security problems and this is the major weakness of the protocol.

IP is vulnerable to various kinds of attacks. Generally, attackers take advantage of the lack of a strong feature within IP for authentication to ensure that packets actually originate from their stated source. As a consequence, there is no pre-established way to determine the provenance of a given packet. While this anonymity does not constitute a

flaw as such, it is worthwhile to note that the service of host authentication must take place at a higher layer in the ISO/OSI Reference Model. This authentication is performed at the application layer, for instance, in contemporary applications, like cryptographic applications, that call for strong host authentication.

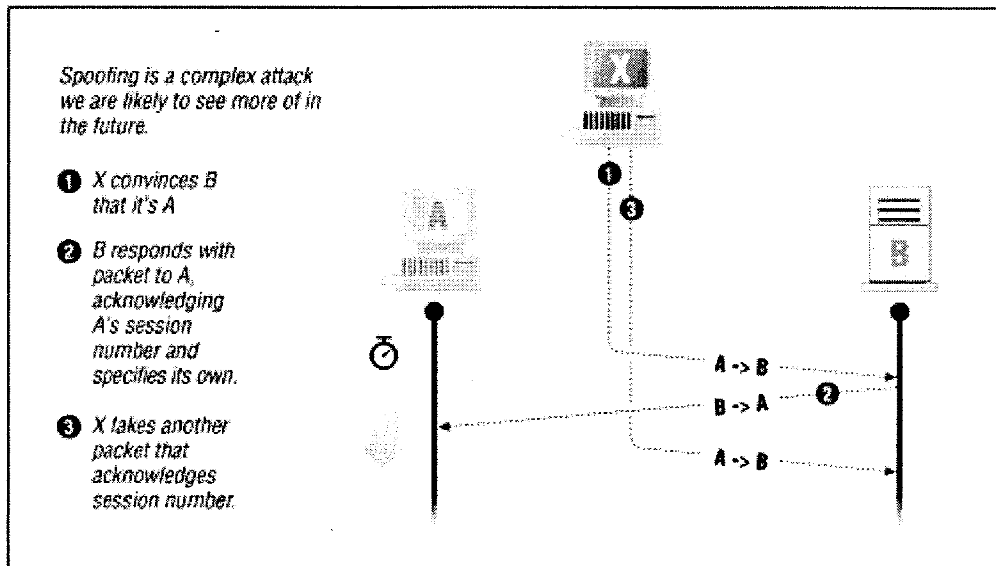


Figure 2.3: IP Spoofing Attack

IP spoofing occurs when a host claims to have a different host's IP address. A number of systems, like router access control lists, use the sender's IP address to determine the admissibility or inadmissibility of packets; IP spoofing is thus a valuable weapon in an attacker's arsenal. Packets can be sent to the host that will result in its taking a particular action. For some applications, login is allowed on the basis of the requesting party's IP address. These examples illustrate the way that undependable layers can result in weakened security. Figure 2.3 illustrates the different steps engaged in a typical IP spoofing attack. First, Host X disables Host A. Then, Host X impersonates

Host A by sending a connection request to host B. After that, Host B responds to host A along with a sequence number. Eventually, Host A then sends back an answer to Host B by guessing the sequence number and Host X is then able to fully communicate with Host B.

A comparatively more sophisticated attack is IP session hijacking, as first identified by Steve Bellovin [3]. Such an attack, nonetheless, poses an extremely large risk, particularly since the underground community now offers toolkits to assist otherwise untrained would-be perpetrators with this illicit execution. In this type of attack, a user's session is taken over and placed in the attacker's control. If the attacker interrupts a user in the process of sending emails, the attacker can now access all of the user's information and perpetrate functions in the user's place. Meanwhile, the user under attack will find that the session has been dropped and might proceed to simply login once more, potentially unaware that the attacker may still be logged in and executing commands.

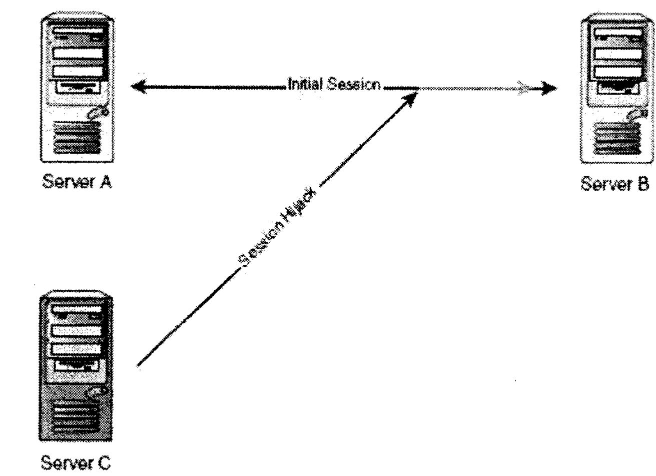


Figure 2.4: IP Hijacking

For a better understanding of the IP session hijacking attack, an example, in Figure 2.4 is described in the following. Host A is currently in the middle of a session with Host B. This occasion could be a telnet session, wherein a user is checking email or using a home-based account running UNIX. Meanwhile, lurking in the network between A and B, is Host C, run by a malefactor who observes the traffic between them. Host C can then implement a tool that imitates A to B, while simultaneously silencing A or even convincing it that B has left the network, as could occur in a crash or network outage. In the event of a successful attack, within only a few seconds, the user's session has been hijacked. Whatever actions the user could have performed are now possible for the attacker, while B remains completely oblivious to the infraction.

One possible solution to this scenario is to use encrypted versions of applications in place of their standard telnet-type counterparts. While the session could still be hijacked, because of the encryption, only garbled nonsense will be visible to the attacker, who lacks the necessary key or keys for decrypting the data stream from host G. As a result, the attacker will be rendered incapable of any actions during the session.

TCP, as the transport layer of the ISO/OSI Reference Model, must rest upon a network-layer protocol. TCP was intended to work in conjunction with IP, while IP was designed to carry TCP packets. Some significant features of TCP will be briefly reviewed.

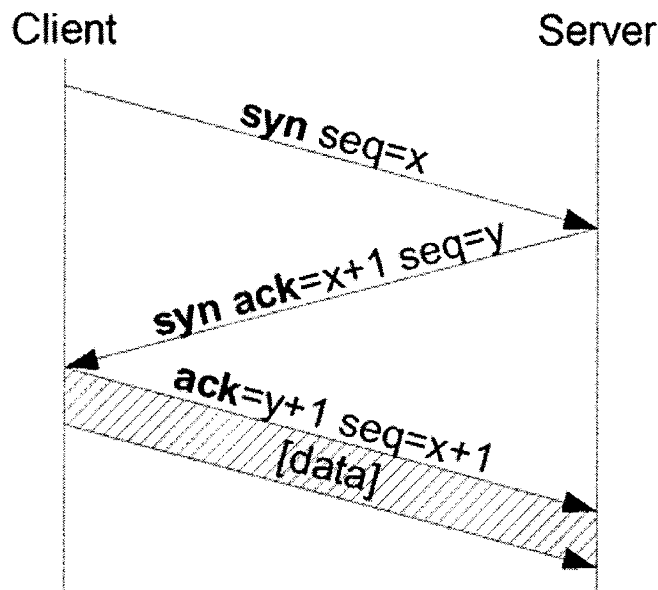


Figure 2.5: 3-Way TCP Handshake

Perhaps most notable among these features is guaranteed packet delivery. Host A, when transferring packets to host B, will anticipate acknowledgment of the receipt of each packet within a predefined time period. If the expected acknowledgement from B does not arrive, the packet will be resent by A.

The applications run by host B will expect a complete and properly ordered data stream from a TCP session. Whenever a packet is missing, as above, A will resend it. Any packets that arrive out of sequence are re-ordered by B before being passed to the application requesting the data. In that sense, TCP uses a handshaking technique to open connections. It is referred to as 3-way handshaking or as "SYN-SYN-ACK". The handshaking works as follows. Client sends a TCP SYN packet to server to initiate the session. Server, then, sends a SYN-ACK to the client along with a sequence number. Client, then, answers by sending the ACK flag with the incremented sequence number.

Finally, server receives ACK with the valid sequence number and the TCP connection is then, established. This mechanism, described in Figure 2.5, is designed so that two systems attempting to initiate a connection for communication can negotiate one connection at a time independently of each other.

For many applications, such as a telnet session, this system works very well. For a user, the key is to ensure that the remote host receives each keystroke and that every packet is received and sent back. The minor, intermittent slowdowns caused by resending or rearranging lost or out-of-order packets are a negligible problem in these cases.

On the other hand, when it comes to media-rich applications like streaming audio and video, this system is far from ideal. In such situations, the occasional dropped packet in a stream of hundreds is of minor matter. Of greater consequence are speed and the packets' timely arrival; slowdowns caused by resending lost packets will result in a break or pause of the data stream. Only after the lost packet is resent and received can it be slotted into the appropriate position in the data stream and accessed by the application.

A simple transport-layer protocol, User Datagram Protocol (UDP), provides fewer features than TCP. UDP is therefore not considered robust or reliable. Still, while UDP is not suitable for all applications, there are those for which it is better suited than the more sophisticated TCP.

A major advantage of UDP is the parsimony of this protocol. The fact that it need not track the packets' sequences or ensure their receipt results in a lower overhead compared to TCP. The protocol's simplicity contributes to its suitability for streaming-data applications, since it involves expending less effort for rearranging out-of-sequence packets or resending lost ones. After understanding how the Internet operates, especially

the TCP/IP protocol, and the threats that associated with it, an analysis of the risk management is developed, in other words, the level of security that we need to ensure against the access that we need to have and the compromises that should be done.

2.3 Risk Management: The Game of Security

When considering risk management, there are no one-size-fits-all solutions, nor is any one firewall necessarily the best for a given organization. In all cases, the extremes of total security and total access must be moderated. In practice, absolute security can only be attained in the case of a machine disconnected from any network and made physically inaccessible. Of course, this machine would then be of no practical value whatsoever. At the other extreme, absolute access—without any passwords, authorizations, or other security measures required—would have the virtue of absolute convenience. From a security standpoint, however, this increased access would be equally undesirable, given the risks associated with the Internet. Eventually, the damage caused by attacks or break-ins could render such a machine just as useless.

The same principle is constantly applied in everyday life: choices must be made about what level of risk is acceptable in a given situation. Every time someone starts a car or boards an airplane, the possibility of danger is introduced. We accept these risks in exchange for the convenience involved, while in other situations convenience is sacrificed in favor of greater security. Generally, we all operate within certain pre-defined boundaries that dictate which risks are acceptable and which are not. In situations where

the danger associated with an action clearly outweighs its convenience, most of us will choose the safer alternative.

In terms of organizational security policy, then, the same general rule applies. It must be decided what point on the spectrum between absolute security and absolute access is appropriate for that organization's needs. An effective security policy should clarify the acceptable level of risk and dictate the mechanisms used to enforce those boundaries in a consistent way.

2.4 Types and Sources of Network Threats

Having reviewed the relevant networking context, we can now consider the implications for network security. To begin with, the various kinds of security threats faced by networked computers will be described. Next, some of the precautionary measures that can be taken are outlined.

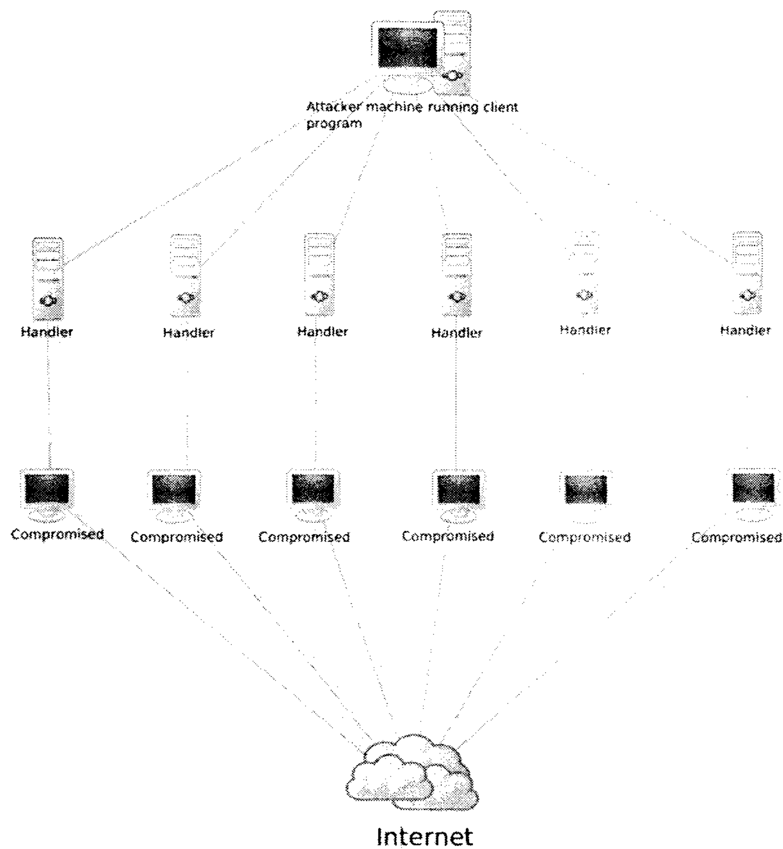


Figure 2.6: Denial of Service Attack

Perhaps the most pernicious **attacks** are Denial-of-Service (DoS) attacks. The ease of perpetrating such attacks, the difficulty or impossibility of tracking them, and the fact that an attacker's requests cannot easily be refused—without compromising service to legitimate users—make DoS attacks notoriously tricky to handle. DoS attacks, shown in Figure 2.6, operate according to a simple premise: more requests are sent to the machine than it can effectively process. In the underground community, toolkits circulate that simplify the process further. A program is run that can be instructed to target a specific host. Then, the program connects to a service port, sometimes using forged packet header information to misidentify its origin and then immediately drops the

connection. By sending a large number of requests per second to which the host is able to respond, the program renders the host unable to answer all of the requests. In the meantime, legitimate requests—like those from users trying to access a website on the server—will also be prevented from being serviced.

The risks of getting hit by such an attack can be mitigated in a few ways. One is not running visible-to-the-world servers too close to capacity. Packet filtering can be used to keep the network space clear of packets that are obviously forged—like those claiming to originate from one's own hosts, from addresses reserved for private networks as defined in RFC 1918 [4] and the loopback network (127.0.0.0). Finally, it is helpful to stay updated on security patches for hosts' operating systems.

The term unauthorized access refers, at a high level of abstraction, to various sorts of possible attacks. Overall, such attacks aim to access any resource that the computer should not grant the attacker. In the case of a web server, for example, the host should deliver the requested web pages to any user; it should not, however, provide command shell access to any user, except local administrators or anyone else confirmed to have the proper authorization.

Clearly, preventing unknown and untrustworthy parties from executing commands on server machines is essential to the integrity of the system. Generally, the severity of the problem is classified according to normal user and administrator access. Normal users can perform certain actions, like reading and mailing files, which would be undesirable for an attacker. In some cases, retrieving confidential documents may be all that an attacker hopes to accomplish. In other cases, an attacker might seek administrator access in order to make configuration changes to the host, resulting in even more

dramatic harm. Examples include changing the host's IP address or putting in place a start-up script to prevent the machine from starting up effectively.

According to the threat model, the key question to ask is: what is it that one is trying to protect oneself against? A company's sensitive documents or other data that can be accessed by a normal user may cause a great deal of harm if accessed by competitors, enemies, or the public. Thus, break-ins occurring on normal users' accounts can have significant implications for security, in the form of public relations disasters or the leaking of closely guarded information.

Many of these sorts of break-ins are simply the work of relatively harmless, if misguided, thrill-seekers. An attacker may have no interest in breaking into a company's network beyond the excitement of the deed itself. As discussed below, however, other attackers may have more malevolent intentions. Keep in mind that the former, thrill-seeking breed of trouble maker can be enlisted in the service of the latter, more pernicious enemy or competitor.

The most destructive kinds of attacks and break-ins can be divided into two classifications: data diddling and data destruction. Data diddling is, in some ways, the most harmful, since it can be more difficult to initially detect. A data diddler may modify spreadsheet data, alter key project dates and deadlines, change employee payroll deposit information, or otherwise interfere with relevant company data. Such tampering may not become apparent until weeks or months later, when a discrepancy finally turns up. By this point, tracking down the source of the problem will present enormous difficulties. Even then, the difficulty of establishing order in the face of unsafe data remains. Data destruction is often the work of perverse troublemakers who simply enjoy causing trouble

by deleting files. The resulting damage to an organization's computing infrastructure can be comparable to an arsonist setting a fire that destroys all of the physical equipment.

One obvious question is: how is it possible for attackers to gain access? The answer is that any link to the exterior world—be it through an Internet connection, a modem, or even staff members' physical use of equipment—represents a point of vulnerability. It only takes one unscrupulous employee to uncover sensitive information like passwords, contacts, and other data. Thus, all such vulnerabilities need to be carefully considered before security issues can be effectively tackled. All points of access to an organization's computing system need to be secured, in keeping with that organization's security policies.

Several steps can be implemented to reduce the risks of attack and mitigate their consequences, based on the types of common attacks identified above. These measures include both high-level practices to prevent security breaches and damage control initiatives that can be taken to minimize the harm caused by a successful attack.

The benefits of regularly and thoroughly backing up data are not limited to security implications. A backup policy, as dictated by the organization's operational needs, ought to be accompanied by a comprehensive recovery plan. In the case of disaster, such as fire or flooding, the backup policy and data recovery plan should function together to allow normal operations to be restored, even from a remote location if necessary. The same is true for electronic failures or malicious damage caused by attackers.

Despite the fact that this may seem like common sense, such details often escape the attention of many. Simply put, any data that need not be available to outsiders should not be made accessible. Neglecting to protect this information can needlessly compound the damage caused by break-ins.

An organizational security system is only as strong as its weakest link. If the entire system can be compromised by access through a single component, the organization's overall security will be weak as a result. Redundancy in the system can help eliminate some of the risk associated with break-ins.

Staying abreast of security-related advisories from the relevant dealers can prevent one of the most popular, and effective, means of attack: taking advantage of old bugs in order to break into the system. Somebody familiar with the system should stay closely updated on operating system patches. Besides information and advisories issued by commercial vendors, security organizations like CERT[5] and CIAC[6] can be a valuable source of information.

Within the organization, a good practice is to have at least one employee responsible for taking note of the latest developments in the field of security. This person can simply stay informed about advisories and other news related to security and become familiar with basic security protocols like the "do"s and "don't"s found in resources like the "Site Security Handbook". The person need not be a technical specialist or otherwise possess special computer expertise. Armed with knowledge of the latest issues, such as current software problems, this person can be a valuable resource to confer with on matters of organizational security.

2.5 Firewalls

The Internet and other computer networks represent a two-way communication pathway through which information both enters and leaves a connected organization. In some contexts, this bilateral flow of information can be detrimental. For instance, proprietary data may be easily accessible from inside an organization's Intranet. A firewall, a set of components designed to act as a fence between one network and another, are commonly used to form a partial barrier between internal and external networks.

2.5.1 Firewalls Definitions

Before we proceed, some terms associated with firewalls will be defined. This introduction is important to understand how the designs involved in securing a network work. The terms that we are going to define in the following are bastion host, router, access control list, demilitarized zone, and proxy.

A bastion host, depicted in Figure 2.7, is a computer on the network that regulates contact between an Internal network and any untrusted network, like the Internet. Designed to resist attacks, bastions are general-purpose computers dedicated to this purpose, generally running a customized version of a UNIX operating system. All other service functionality is removed, or limited as much as possible, in order to reduce the external threat to the system.

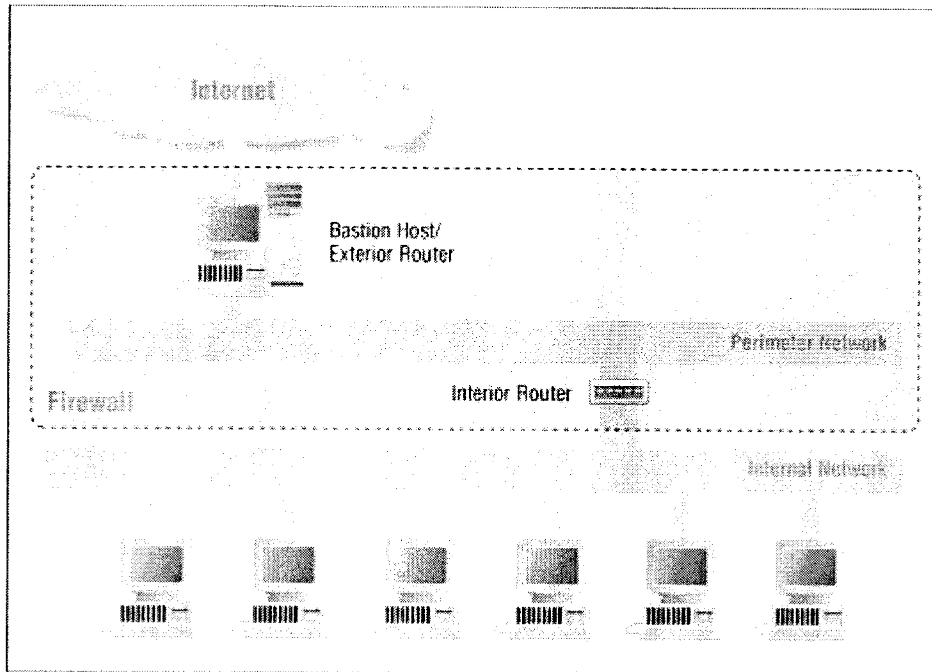


Figure 2.7: Bastion Host

A router is used to connect networks to each other. These special purpose computers are also responsible for functions like routing or traffic management on connected networks. By using an access control list (ACL), some routers and servers are able to identify permitted origination addresses, destination addresses, and destination service ports by analyzing various data about incoming and outgoing packets. Effectively, the ACL can allow the router to perform selective filtering.

The demilitarized zone (DMZ) is a network that intervenes between the trusted and untrusted network. It is not itself part of either one, but connects the two to each other. This function makes it a key element of the firewall, since the DMZ includes layers of protection between the network and anyone attempting to break in via the Internet.

Figure 2.8 presents architecture of single firewall DMZ. In this case, the firewall must have three network interfaces, one for the external network, the second for the internal network and the third for the DMZ. The firewall in this case, must be configured to handle the traffic on these three interfaces.

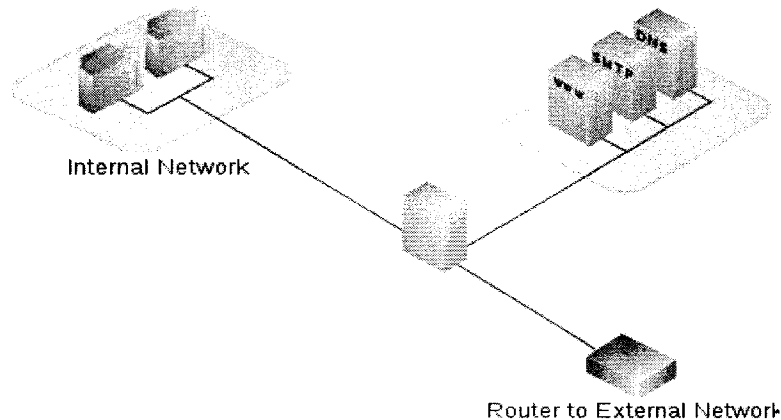


Figure 2.8: DMZ with one firewall

A proxy is one host acting in place of another host. As in Figure XX, They can allow hosts on an internal network to access Internet resources while being unable to directly connect to the Internet. The proxy server acts on the application layer of the OSI Model and controls traffic on that level as shown in Figure 2.9. For example, a host capable of requesting online documents from the Internet can be configured as a proxy server. Intranet-based hosts can be configured as proxy clients. Every time a host on the Intranet fetches a given webpage, the web browser connects to the proxy server to request the URL. In the following section, different kinds of firewalls are described.

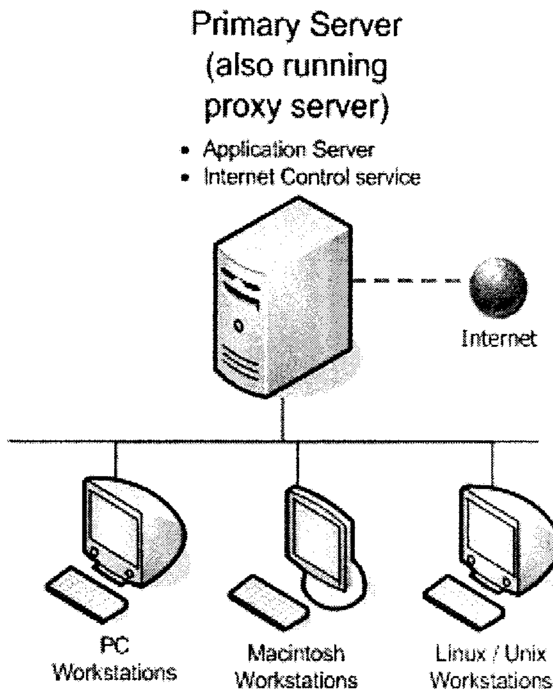


Figure 2.9: Proxy server architecture

2.5.2 Types of Firewalls

The next considerations are the three main types of firewalls— application gateways, packet filtering, and hybrid systems—and their features.

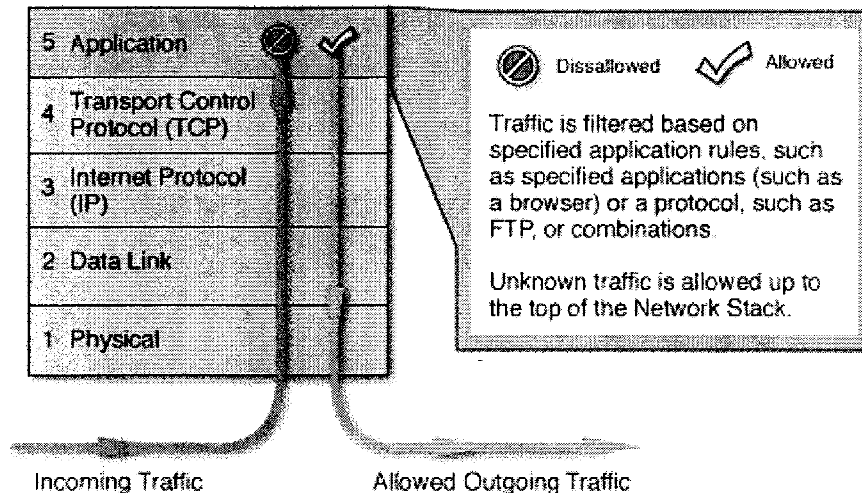


Figure 2.10: Example of an application level firewall

Early firewalls functioned as application gateways, or proxy gateways. The application gateways consist of bastion hosts running software designed to let them function as a proxy server. In terms of the ISO/OSI Reference Model, the software operates at the Application layer. All clients behind the firewall need to be configured to use the proxy before they can access resources and services on the Internet. In terms of security, because this kind of firewall prevents any default access to traffic, they are considered the strongest. The security offered by these systems, however, comes at the cost of speed. Generally, the number of processes required to return a request make them the slowest. Figure 2.10 shows how the filtering occurs in incoming and outgoing traffic in an application firewall.

With packet filtering, a router uses access lists (ACL) to regulate incoming traffic. The default setting allows all traffic to pass through, with no restrictions. ACLs allow

security policies to be implemented in terms of access between the internal and external networks.

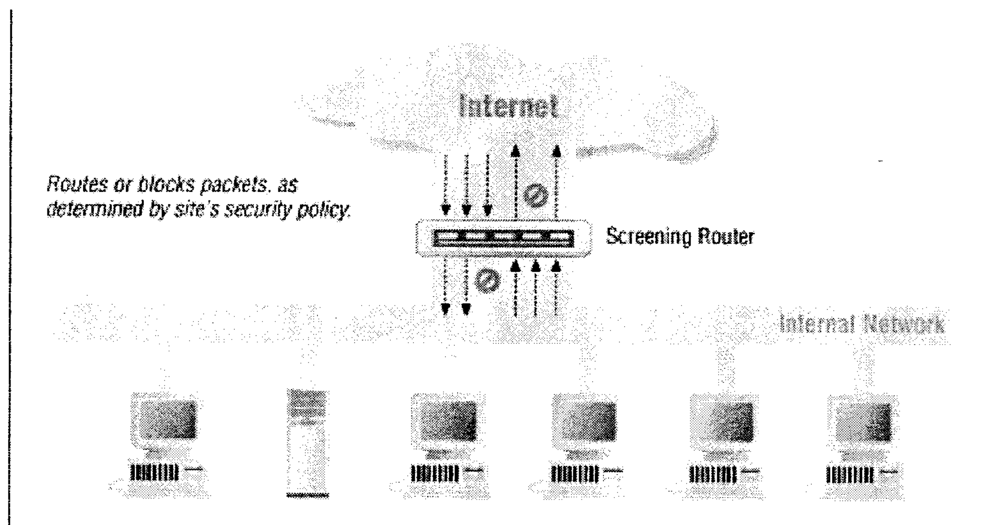


Figure 2.11: Packet Filtering Firewall

Packet filtering, Figure 2.11, compared to application gateways, involves comparatively low overhead. This property is due to the fact that access control takes place at a lower level, on the ISO/OSI Reference Model, generally the transport or session layer. For this reason, and because the routers employed by packet filtering are special purpose machines designed to function optimally in a network environment, packet filtering is also usually a faster system than application gateways.

Moreover, since this filtering takes place below the application level, new applications can be supported with relative ease. The support of new applications either

happens automatically or involves specifying particular types of packets that are allowed to pass.

However, this technique is not without its flaws. As noted above, TCP/IP does not include any mechanism for authenticating the origin of a given packet. Restricting traffic thus involves employing layers of packet filters, and all that can be ascertained for certain is the network from which the packet originated. By using two layers of packet filters, it is possible to distinguish packets originating from the Internet and those from within the internal network. The host itself cannot be authenticated.

Some systems have been developed that seek to combine the strengths of each of these approaches; thus, retaining the stronger security of an application layer gateway, while offering the greater flexibility and speed associated with packet filtering. This integration involves a two-stage process, in which authentication and approval is first granted to new connections at the application layer with the rest of the connection relegated to the session layer. The connection is then monitored by packet filters, ensuring that packets which have already been authenticated and approved at the application layer are allowed to pass.

Another option is to combine packet filtering with proxies at the application layer. This technique offers the security advantages of protecting machines, like web servers, which provide Internet resources, while at the same time functioning as an application layer gateway for an internal network. This approach also has the advantage of setting up more obstacles through which an attacker must penetrate to access the internal network. In order to break in, an attacker must get past the access router, bastion host, and choke router.

In order to choose the most suitable solution from among the many options available, organizations are well advised to consult with a security expert who is familiar with their security policies. Based on those policies, firewall architecture can be designed and built to optimize security in keeping with stated objectives, while also taking into consideration other factors like the required services, ease of use, and scalability.

Often, the term firewall is used to describe a single component, like a bastion host, that can ostensibly be used all by itself to keep networks from harm's way. In fact, a firewall is a series of components working in conjunction to protect the system from attack. In the case of a single component, of course, there is only one barrier that an attacker needs to contend with in order to break into the internal network.

2.6 Summary

This chapter has presented some basic issues involved in network security, along with some of the considerations that networked organizations need to keep in mind in order to minimize the risk of attacks and break-ins. The nature of TCP/IP protocol, which lacks any inherent means for host authentication, entails a certain level of vulnerability for any machine connected to the Internet. Network security testing is thus a crucial step for mitigating the inevitable risks. The most common types of attacks have already been outlined. As we have seen, the harm caused by DoS attacks or unauthorized access at the user or administrator levels can be devastating to an organization. Firewalls, as collective assemblages of security components, can provide solutions tailored to an organization's needs—effectively translating security policy into practice. However, certifications and other quantifications can be unreliable indicators of a firewall's practical efficacy. More

comprehensive methods for firewall testing are key to ensuring the functionality of these systems. In that sense, firewalls are as good as the policy they are configured to implement. When testing firewalls, we are essentially seeking to determine whether packet filtering has taken place effectively. To accomplish this goal, an efficient and complete testing method should be implemented.

In the next chapter, the different methods that have been developed and tested to date in the field of firewall testing are reviewed and compared. As discussed next, the area of firewall testing can be divided into active and passive testing methods. Active methods help system administrators in the writing of the policy while passive methods take existing firewall configurations and correct them.

Chapter 3: Firewall Testing

Firewall development and implementation is constantly being improved to accommodate higher security and performance standards. In contrast, the testing of firewalls has for a long time not been taken as seriously as it should [1]. In the last couple of years, however, more research in the field of firewall and network security testing has been taking place. Methods have evolved from vulnerability testing of firewalls using tools, like SATAN [7], to more advanced techniques that take into account the firewall rules to produce a corresponding test.

To fully grasp the issues involved in firewall testing, the basics of network security testing and the most popular methods used to test a network for security breaches and functionality must first be discussed. Then, the latest state-of-the-art for firewall testing will be explored in greater depth. Previous work in firewall testing can be divided into two categories. On the one hand, passive methods use a given firewall configuration and develop various methods to test it. Some methods use a predefined test set while others, more advanced, use the actual firewall configuration and extract tests from it. Active methods, on the other hand, look for ways to avoid errors during the writing process of the configuration rules by developing algorithms and formal languages on top of low level firewall rules or by using graphical interfaces that are more user-friendly.

3.1 Passive Testing

One of the two major methods for firewall testing, namely passive testing, takes an already-configured firewall device and performs operations on this same

configuration, whether to remove mistakes by analyzing the configuration or by performing a test and analyzing the output, be it log file analysis or program output.

3.1.1 Vulnerability Testing

The first real-time testing technique ever used was based solely on vulnerability testing using tools like SATAN [8]. SATAN is used to identify network security vulnerabilities and misconfigurations. Administrators frequently use this publicly available tool to identify weaknesses in a network's security—however, it can be used by attackers for the same purpose. In article [7], SATAN is used to test two popular firewalls, namely, TIS [9] and SOCKS [10]. Test cases include known vulnerabilities and hence do not take into account the firewall rule set. Moreover, the test does not ensure that our network is protected against new types of vulnerabilities. Finally, for this test, a network needs to be already set up; testing comes afterwards, leaving the private network vulnerable for a certain time window. This type of testing is time-consuming and can demand costly resources. In article [11], a CASE tool is used to derive test sets after formally modeling the network surrounding the firewall. A mechanical approach is used to derive test-cases that check for common network vulnerabilities and threats. This method takes into account the network topology as it goes further than SATAN. Unfortunately and just like SATAN, the approach uses simple check lists for vulnerabilities without taking into account the particular configuration of the firewall.

3.1.2 Real Time Testing

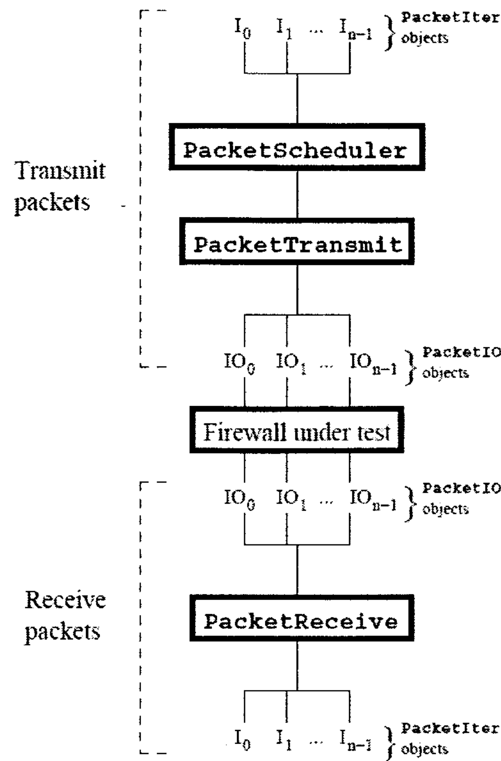


Figure 3.1: Blowtorch Packet Flow

Blowtorch in article [12] is another example of real-time testing. Blowtorch is a C++ framework designed for testing firewall rule sets in the process-control environment, where the cost of failure is high and extensive testing is justified. Blowtorch tests the firewall in isolation, connected only to test equipment. It has more capabilities as it includes a packet generation module, supports handshaking as well as allows for traffic capture and replay. The C++ implementation of the framework is based on the packet flow described in Figure 3.1. In that sense, the flow is being inspired from the real

functioning of the firewall and simulates the packets entering and exiting the latter. As the tool is specifically designed for process control environment, its main focus is in the real time part of testing. Unfortunately, it lacks an important feature: it does not generate tests based on the internal specification of the firewall. The test generated remains then incomplete.

Further improvement has come with PBit [13]. PBit is a pattern-based testing framework for iptables. It contains a collection of test templates. For example, to test the protocol options for iptables, we use the following template:

ProtocolTest(rule-proto, test-proto, direction)

The input domains for this test template are:

rule-proto: {tcp, udp}
test-proto: {tcp, udp, icmp}
direction: {inbound, outbound}

Figure 3.2 is an example of a multiport template in PBit as well as the options available for generating a test set. From the interface, you can select protocols, input source and destination ports as well as common ports. Then, it uses regression testing with the help of parameterized test cases that can be configured in the user interface to reduce the test overhead. Just like the previous tool, PBit does not include tests based on the firewall specification and thus cannot produce accurate test cases based on the its functioning.

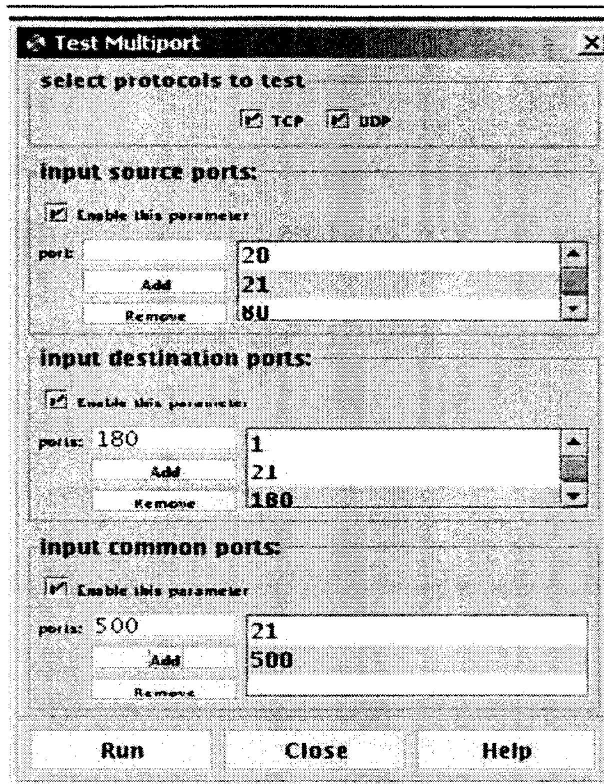


Figure 3.2: PBit multiport template

3.1.3 Formal Network Building

In formal network building approaches, we model the network as well as its elements to test a firewall configuration. Computer networks are composed of hosts connected by communication links. Hosts are connected to the communication links by interfaces, through which messages are sent. The previously described network model can be used to model vulnerabilities as well—for example IP spoofing. This attack usually happens when a host tries to masquerade as another host. In article [14], a formal

method similar to what was just described is presented. In order to test the firewall, the following steps are presented in Figure 3.3.

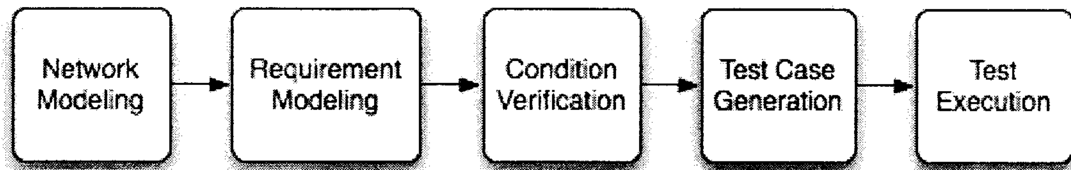


Figure 3.3: Vigna's Firewall Testing Approach

The proceeding measures must be taken to test firewalls. First, a program models the network using network topology typified by hypergraphs and trust relations using trust vectors. Requirement modeling then specifies the types of vulnerabilities against which the network has to be protected. Condition verification is used to ensure that sufficient conditions satisfy the requirements of the given firewall. In test case generation, the set of messages that should be obtained is derived for each requirement. Then, using monitors, whether the configuration matches the policy can be determined. During the test execution phase, the messages that correspond to the generated test cases are injected and verified for correctness.

The main drawback of this method is that it is only a model and fails to take into account the real functioning of the firewall. Also, this formal representation is not intuitive and requires the administrator time to become accustomed to the formal language. Moreover, this approach disregards the inner functioning of the firewall as it

tests for common vulnerabilities. The test generation also generates one test per case, making it inaccurate and not necessarily covering all cases.

3.1.4 Algorithmic Approach

Work in algorithmic research for firewall testing and error detection has been conducted in several projects to date. On the one hand, we have tools [15,16] that are used to detect anomalies as depicted in Table 3.1 and Figure 3.4 and remove them using one or more algorithms. The most advanced work in this field can be found in article [15]. The approach is can be considered the most general and the most simple. They consider any misconfiguration a redundancy or a shadowing. Two algorithms are used in this approach: an algorithm that detects and removes shadowing and a second one that detects and removes redundancy. In this way, when applying both algorithms sequentially, we get misconfiguration error free rules. However, these tools do not guarantee an error-free rule set, since they do not take into account typographical or policy errors.

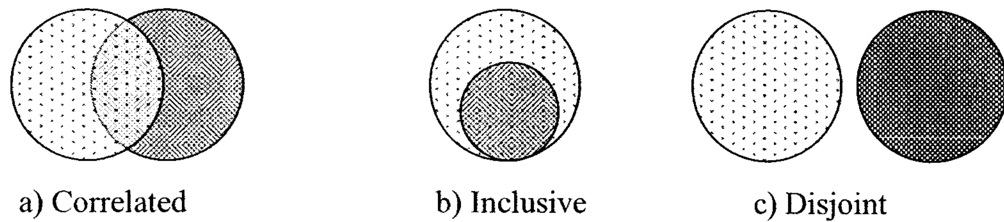


Figure 3.4: Different Rule Configurations

Shadowing anomaly: R2 subset of R1, R1 decision ≠ R2decision
Correlation anomaly: Some of R2 subset of R, vice versa, R1decision ≠ R2 decision
Generalization anomaly: R1 subset of R2, R1 decision ≠ R2decision
Redundancy anomaly: R1 subset of R2, R1 decision = R2decision

Table 3.1: Rule Anomaly Types

On the other hand, we have algorithms that analyze rules and policies to generate an efficient test. In Table 3.1, we describe the four types of anomalies, namely shadowing, correlation, generalization and redundancy. Shadowing consists of covering the whole address space of a rule by another rule, the consequence is that the shadowed rule is never used. Correlation, on the other hand consists of having some address space of one rule that are covered by some address space of the other rule, the covered address space is never used in this case. Generalization anomaly consists of having a rule that is a subset of another rule. The consequence is that the address space of the subset rule contained in the other rule never gets triggered. Redundancy is only different from Generalization in the sense that both rules have the same decision. As in article [16], a segmentation approach, in Figure 3.5, is used to intelligently select packets to be used for

the test. In that sense, a system is used that is capable of smart selection of test packets using information about the network and the policy. After that, packets are selected to cover the whole space of possible packets, in the best possible manner. To that end, weights are used to select packets based on the segmentation and the precedence of rules in the rule set. The different steps to get to the intended result are the following. First, the address space is partitioned into segments based on the policy, as depicted in Figure 3.5. In fact, rule address space is partitioned into segments with each segment covering a unique address space. After that, the importance of each segment is calculated. Then, the rules from each segment from the firewall are extracted. Finally, by injecting these test packets, the output can be logged and analysed.

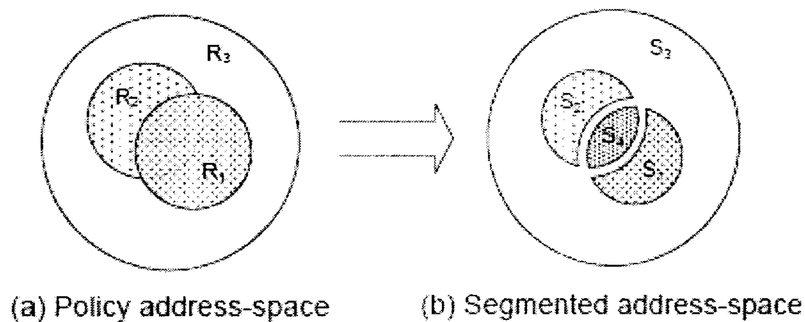


Figure 3.5: Policy Segmentation Technique

This packet selection method guarantees a much better and more accurate test than the previous methods. One major problem with this method is its latency, using a large set of rules when dynamic firewalls are used. In fact, the complexity of the algorithm is dependant of the initial address space for segmentation. Moreover, the

testing time can increase with an exponential factor. In fact and due to the recursive loops, see the algorithm in Figure 3.6, the complexity of the algorithm is $O(2^n)$.

1: $SEGLIST \leftarrow \Lambda$
2: **AddSegment** (InitDomain,Λ ,Λ , defAct)
3: **for all** rules: $i = 1$ to n **do**
4:    **for** segments: $j = SEGLIST.Count$ downto 1 **do**
5:       $S = SEGLIST[j]$
6:       $IncSeg \leftarrow S.AS \wedge AS(R_i)$ {Included part}
7:       $ExcSeg \leftarrow S.AS \wedge \neg AS(R_i)$ {Excluded part}
8:       **if** $IncSeg \neq Seg.AS$ **then** {Segment not contained in the Rule's AS}
9:          **if** $IncSeg \neq \Phi$ **then**
10:             **AddSegment** ($IncSeg$, $S.R_{in} \cup \{R_i\}$, $S.R_{out}$, $S.R_{eff} \cup \{R_i\}$)
11:             **AddSegment** ($ExcSeg$, $S.R_{in}$, $S.R_{out} \cup \{R_i\}$, $S.R_{eff} \cup \{R_i\}$)
12:          **else** {no intersection of rule and segment}
13:             **AddSegment** ($ExcSeg$, $S.R_{in}$, $S.R_{out} \cup \{R_i\}$, $S.R_{eff} \cup \{R_i\}$)
14:          **end if**
15:       **else** {Segment is inside the Rule's AS}
16:          **AddSegment** ($IncSeg$, $S.R_{in} \cup \{R_i\}$, $S.R_{out}$, $S.R_{eff}$)
17:       **end if**
18:       SEGLIST.Delete (Segment j) {delete original segment}
19:    **end for**
20: **end for**
21: **return** SEGLIST

Figure 3.6: Policy Segmentation Algorithm

3.1.5 Query Engines

Query engines like those in [17] answer questions about the firewall's configuration and its network. Figure 3.7 shows the data flow of a typical query engine. They take as input different firewall configuration files. Topology definition is then used to parse the configuration files into a language that the analyzer can process. The user thus interacts with the query engine in a query-and-answer session, taking place at a high level of abstraction. For example, the tool is able to answer such questions as which machines can reach the DMZ and with what services. This tool therefore serves a complementary role with respect to existing vulnerability analysis tools. It can be implemented prior to the deployment of the security policy; it functions at a more intuitive level of abstraction; and it deals with several firewalls simultaneously.

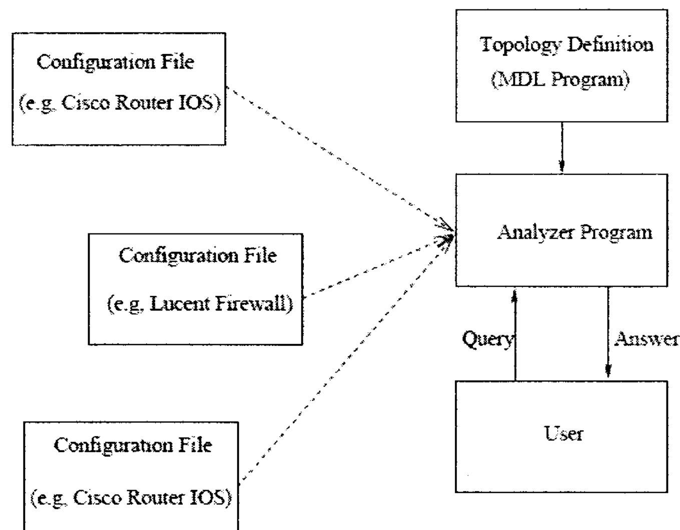


Figure 3.7: Fang Query Engine Data Flow

The main disadvantage of query engines, however, is that they can only manually detect errors. The system administrator's job is to formulate these queries. The method is not automated and lacks precision. It depends too much on human factors, the most significant being experience.

3.2 Active Testing

Active testing methods take the reverse approach. Given that most errors are the consequence of a faulty translation of the high-level policy into low-level rules, tools and methods to avoid these errors have been developed. In the field of active testing, two main approaches have been developed: visual graphical interfaces that are intuitive and easy to use and formal languages that are closer than the high level policy that reduce errors in the process.

3.2.1 Graphical Analysis

In articles [18, 19], for example, intuitive graphical interfaces are used. A graph, Figure 3.8, is used in article [18], to detect overlapping mistakes as well as masking mistakes when writing the rules. Colors are also used to help correct and spot those errors. There is also an editor that is used so the system administrator can manually correct mistakes.

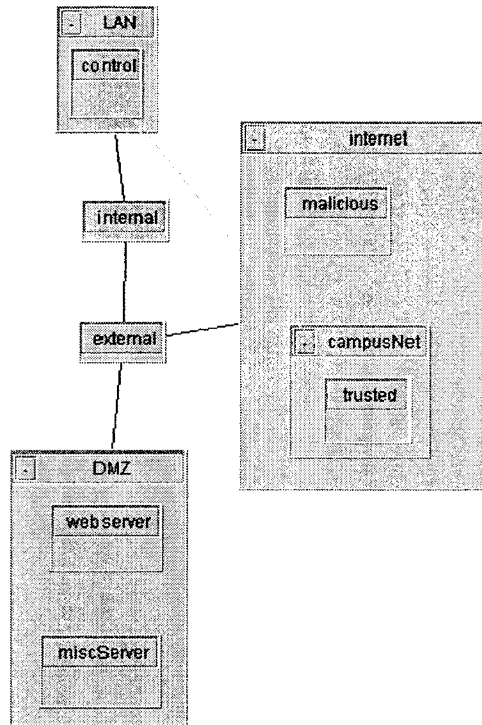
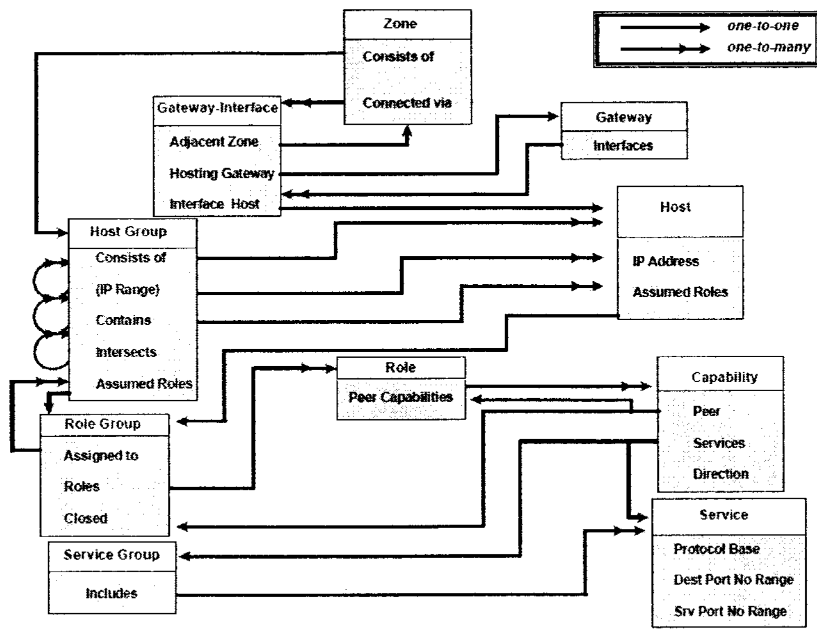


Figure 3.8: Graphical User Interface

On the other hand, an entity-relationship model in [20], illustrated in Figure 3.9 is used to derive firewall rules from that diagram. The entity-relationship model contains, in a unified form, global knowledge of the security policy and the network topology. A model compiler is used to derive this model into low level rules. The model is not complete as it neither covers all cases nor takes into account the all redundancies and shadowings leaving the final rule set error-prone.



3.2.2 Formal Policy Writing

Firewall configuration languages currently lack any well founded semantics. Article [21] tackles the lack of founded semantics in current firewall configuration languages and suggests a high level configuration language for network access policy. One of the results of this is the difficulty in managing network access control policies. Most firewalls, in fact, are incorrectly configured. In Figure 3.10, an access control language based on XML syntax is presented with the access control model Or-BAC (Organization Based Access Control) used to interpret its semantics. This language can be used to dictate high-level network access control policies and automatically derive practical access control rules with which specific firewalls, by way of a translation process, can be configured. This approach offers clear semantics for specifying network

security policy; greatly increases the ease of policy management for administrators; and guarantees portability between different firewalls.

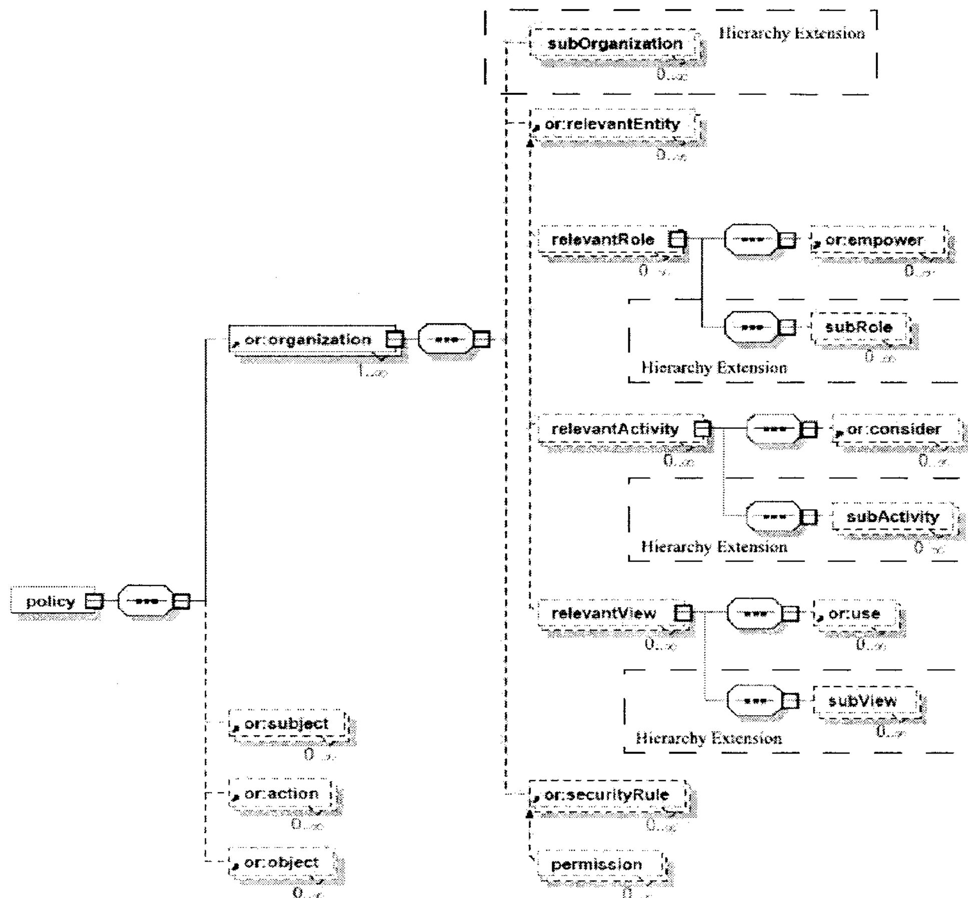


Figure 3.10: Or-BAC Model in XML

In the case of an already configured firewall, the formal policy writing approach does not actively test the firewall and requires the system administrator to become accustomed to the language and rewrite the entire rule set.

3.3 Summary

Real time testing tools assess the integrity of firewalls in a realistic environment. Creating an accurate environment to perform the tests; however, is sometimes unfeasible and can be costly. On the other hand, query engine tools exist that avoid having to set up a testing environment by modeling the network topology. They can then answer any query typed by an administrator. However, these tools are lacking in their ability to detect and correct mistakes. Their functioning is also dependent on human intervention. Furthermore, the algorithmic approach is convenient because we can automatically detect and remove all firewall anomalies. Finally, the active method testing tools currently used do not take into account the existing firewalls and require the network administrator to rewrite the rules in order for the policy to be applied.

The approach presented in this thesis uses the advantages of the algorithmic approach to remove misconfigurations, responsible for security leaks. As described in [1], most corporate firewalls are not correctly configured due to mainly complex policies that have to be deployed. It combines along with the algorithmic approach, a real time testing approach that takes into account the network environment where the firewall is deployed as well as the rules that are enforcing the policy. This way, it ensures that all misconfigurations errors as well as functioning and policy enforcement errors are detected and removed. The removal process is done using an update module that automatically detects and corrects the firewall configuration and produces a new configuration file.

Chapter 4: A New Approach for Automated Firewall Testing and Validation

As discussed, in previous chapters, by taking advantage of both passive and active firewall testing methods, a better solution to address the problems associated with testing has emerged. This chapter begins by presenting an overview of the entire framework for this approach. Next, each module of the framework is described in depth, discussing the motivation behind its use, as well as its inputs and outputs and its inner functioning and algorithms.

4.1 General Presentation of the Framework

Figure 4.1 shows an overview of the testing framework. It takes as input the firewall configuration file as well as the network information of the network where the firewall will be deployed and produces an error free rule set.

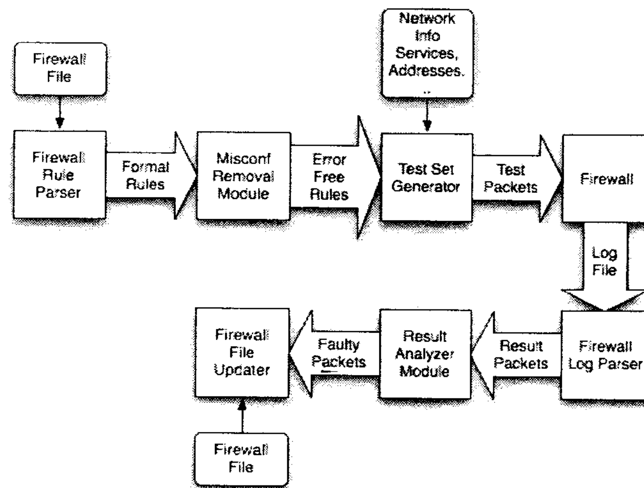


Figure 4.1: Testing Framework Schema

The automated testing approach is based on a set of algorithms that take into account the network environment as well as the number of rules in the configuration file to produce an accurate test set. The obtained test set is applied against the firewall. Then, the last step consists of comparing the intended results against the obtained ones and correcting the detected errors. The goal is to obtain an error free firewall configuration file.

The framework starts by taking as input the configuration file. The syntax of this language corresponds to the iptables. Given the complexity of the language, the first step is to translate this language into a formal language that the algorithm can process. The algorithm will then search for misconfiguration errors in the file, remove the faulty rules; and outputs a rule set that is misconfiguration error free—this step is mainly used to reduce the number of test packets used to test the given firewall; then, this number is shrunken to the minimum. The number of test packets is reduced, and only the packets that will trigger a rule in the rule set are left.

The next step is to generate a test set based on the translated rules. Since a full test set is nearly impossible to execute—testing all cases can take a great number of years to complete in some cases, as shown in article [16]—an intelligent way to cover most cases while keeping the size of the test set computable in a reasonable amount of time must be devised. To that end, network information as well as the number of packets for each rule are utilized. By employing all of this information a test set is then generated.

At this point, the different packets in the set are tested against the firewall in a local environment. An analysis of the log file is then completed. After that, the obtained results are compared to the intended results. If inconsistencies between the two results are

obtained, the original configuration file must be reconsidered to find the faulty rule and correct it.

This method uses the white box testing approach in the way that the test set is generated using the information in the configuration file. The main advantage of this method is the fact that the test is reduced, automated, and complete. The inner working of the system allows an intelligent testing approach, making it easier to design test cases.

4.2 The Different Modules of the Framework

To fully understand our approach, the framework will be divided into seven separate modules. The firewall configuration file is taken as input of the framework and its data is being transformed at each step of the process. For each module, a throughout description of the inputs and outputs as well a discussion are conducted.

4.2.1 Firewall Parser

The first step in the approach is to redefine the low level rules contained in the firewall configuration file. To proceed, the file is parsed into another format description of the language that the algorithm can understand and compute. The parser will conduct a lexical analysis of the file by dividing the strings into components that are stored in another format. This formal format is the same for all firewalls, making our approach suitable for testing firewalls of any kind.

The parser works as follows: every time a low level rule from the configuration file is encountered, it translates and stores it in a *Rule* object, with each rule composed of

a class *Condition* and a variable *Decision.* Each *Condition* is composed of *Protocol, Source Address*, *Source Port*, *Destination Address*, and *Destination Port.* Each element of a Condition is of class *Interval.* Each parsed *Rule* is then stored in a container of type Vector of *Rules*. Table 4.1 shows an example of a configuration file parsed into the formal format.

Order	***Condition***					***Decision***
	Protocol	*SrcAddress*	*Sport*	*DestAddress*	*Dport*	
1	[0, 1]	[20, 50]	[80,80]	[1, 50]	[10, 90]	Accept
2	[0, 2]	[20, 50]	[20, 20]	[10, 70]	[10, 90]	Drop
3	[1, 1]	[30, 100]	[1,80]	[5,75]	[21, 21]	Accept
4	[2, 2]	[30, 30]	[1,80]	[1,10]	[1, 100]	Accept
5	[0, 2]	[1, 90]	[1,80]	[1,75]	[1, 100]	Drop
6	[1, 2]	[1, 100]	[1,51]	[40,90]	[50,80]	Drop

Table 4.1: Format of the Parsed Rule Set

The parser used in this step takes into account the different fields of a firewall static rule. An addition should be to add another field for TCP/IP packets to make it compatible with stateful firewalls as well. After translation is complete, the next step, where misconfigurations are removed from the rules, can be undertaken.

4.2.2 Misconfiguration Removal

Now that we have parsed the rules into a Vector of *Rule*, which corresponds to the formal language described in the previous section, we will be using it to derive error free rules. In fact and as discussed in Chapter 3, misconfiguration errors are very common and widespread in current firewall configurations. This situation is due to the low level archaic firewall configuration language that causes the system administrator to not fully understand the functioning and the configuration of the firewall.

The method used to remove misconfigurations, namely shadowing and redundancy, is taken from the work done in [15], which is the most advanced work to date in the field of misconfiguration errors. The authors simplified and generalized the errors. In that sense, all misconfigurations can be classified as either shadowing or redundancy as in [15, 22]. For a better understanding of these errors, a description is given in the following.

In a first match policy, shadowing is an error responsible for masking a whole rule with different decision making. An example of shadowing is explained below:

R1: [0, 1] [10, 20] [10, 20] [10, 20] [10, 20] ACCEPT
R2: [0, 1] [15, 20] [10, 20] [10, 20] [10, 20] DROP

In this case, R2 is shadowed by R1 as R2 will never be matched. When faced with this kind of shadowing, the rule set is rewritten by simply removing R2 and still keeping the same decision making:

R1: [0, 1] [10, 20] [10, 20] [10, 20] [10, 20] ACCEPT
R2: ø Shadowing=True

Redundancy relates to masking part of or a whole rule, with both rules having the same decision. An example of redundancy is described below:

R3: [0, 2][1, 10][1, 80][1, 100][1, 80] ACCEPT
R4: [0, 2][1, 100][1, 80][1, 100][1, 80] ACCEPT

In this case, R3 is redundant. When we face this kind of redundancy, the rule set is rewritten by simply removing R3, as this will not modify the decision making:

R3: ø Redundancy=True
R4: [0,2][1,100][1,80][1,100][1, 80] ACCEPT

The algorithm both detects shadowing and redundancy and corrects rules, removing errors and pointing to where they were found. The removal method uses two algorithms: the first one removes shadowing from the rules and the second one removes redundancy. We have to use two separate algorithms because the removal and detection of redundancy does not obey to the same logic as shadowing and is not as trivial. So, the complete detection algorithm uses two simple algorithms sequentially. Figure 4.2 shows the steps that the algorithm goes through in the detection and removal process of both types of misconfiguration.

```
/* Phase 1 */
for i ← 1 to (count(R) − 1) do
    if testRedundancy(R, i) then
        R_i[redundancy] ← true;
    end
    if R_i[redundancy] then
        for j ← (i + 1) to count(R) do
            if R_i[decision] ≠ R_j[decision] then
                R_j ← exclusion(R_j,R_i);
            if R_j[condition] = ∅ then
                R_j[shadowing] ← true;
        end
    else
        for j ← (i + 1) to count(R) do
            R_j ← exclusion(R_j,R_i);
            if R_j[condition] = ∅ then
                R_j[shadowing] ← true;
        end
    end
end
/* Phase 2 */
for i ← (count(R) − 1) to 1 do
    if R_i[redundancy] then
        if testRedundancy(R, i) then
            R_i[condition] ← ∅;
        else
            R_i[redundancy] ← false;
            for j ← (i + 1) to count(R) do
                if R_i[decision]=R_j[decision] then
                    R_j ←exclusion(R_j,R_i);
                if R_j[condition] = ∅ then
                    R_j[shadowing] ← true;
            end
        end
    end
end
```

Figure 4.2: Misconfiguration Algorithm

This step is fundamental in the sense that most firewalls contain this type of error [15]. With this step, we ensure that rules cover the minimum address space, allowing us to build a minimal and yet complete test set. In other words, we avoid testing redundant or shadowed rules.

Finally, the algorithm complexity is at most $O(n^2)$ as it uses two separate algorithms and both algorithms have a maximum complexity of two loops inside each

other. This reduced algorithmic complexity will allow us to apply the framework against large rule sets without experiencing performance problems as in article [16].

A correction to the original algorithm has been performed. In fact, the actual algorithm does not work for *Intervals* that start with value 0. The correction was to simply add a value of one to all intervals that start with 0 and then remove the one after the algorithm has been executed. The steps are described as follows in Table 4.2.

1) Interval = [0, 20]
2) Before the Algorithm add one: Interval' = Interval + 1 = [1, 21]
3) Execute the Misconfiguration Algorithm
4) After the Algorithm subtract one: Interval = Interval' – 1 = [0, 20]

Table 4.2: Correction to the Current Algorithm Technique

4.2.3 Test Set Generation

The purpose of this step is to obtain a test set that is as complete as possible. Unnecessary shadowed and redundant packets have already been removed in the previous step. Now, the obtained rules to generate a test set using the internal specification of the firewall, as well as relevant network information, are used to derive a test set, as depicted in Figure 4.3.

```
for each Rule in errorFreeRules
  for each Condition in Rule
     Compute Weight
     totalWeight += Weight
  end for
end for
for each Rule in errorFreeRules
  for each Condition in Rule
    Compute numberOfTests
    for each numberOfTests
       Extract testPacket
    end for
  end for
end for
testPacket
```

Figure 4.3: Test Generation Algorithm

The network information notifies the administrator what addresses and services are actually used in the network. For a better understanding of the practical use of the network information, an example is described as follows. A local area network (LAN), L1, 12.1.1.0/24 contains hosts 12.1.1.2 and 12.1.1.3. Another LAN, L2, 12.1.2.0/24 has the hosts 12.1.2.2, 12.1.2.3, 12.1.2.5 and 12.1.2.20 that are setup. For each LAN, a rule is responsible for filtering the traffic to L1 and L2 respectively. As L2 contains more addresses that are actually used, the test intensity should be greater for L2 than for L1. The same reasoning can be made for services used in a specific network or host.

Table 4.3 is the transcript of a sample file that describes the format of network information.

File: Network Info.txt

tcp	192.168.1.23	20	192.1.1.1	80:100
udp	195.1.2.5	51	192.1.1.0/24	70
tcp	200.1.1.9	21	192.1.1.0/24	80
udp	121.1.1.10	22	192.1.1.0/24	51
udp	200.200.1	80	192.1.1.0/24	22
udp	121.1.1.2	8080	192.168.1.0/16	22
tcp	121.1.1.3	21	163.45.2.1	75,78,101
udp	121.1.1 .4	22	192.1.1.0/24	80

Table 4.3: Network Information File

To define test intensity for each rule, two frequencies to help us in the test set selection process are employed. They take into account network information previously defined as well as rule's largeness of address space.

Frequency1: *The more address space is covered by a rule, the less specific this rule is, and the fewer test segments are selected. This test frequency is inversely proportional to the total space covered by a rule.*

$$\textbf{Frequency1} = \frac{1}{CoveredRules}$$

Frequency2: *The more addresses used in the network that match a rule, the more critical these rules are. Test intensity increases proportionally with the ratio of the used address space to the total number of addresses covered by a given rule.*

$$\textbf{Frequency2} = NumberOfMatche$$

Given the definition of the frequencies used to compute the weights, the test intensity or weight for each rule is then computed using a simple multiplication of Frequency1 and Frequency2:

$$\mathbf{wi} = Frequency1 * Frequency2$$

Now that the weight computation for each rule is defined, we can extract the number of tests per rule. To do so, we have to define another constant that is called testLength. testLength, as its name describes, defines the total number of packets that will be generated during a test. This let the end user shorten or lengthen the test for its needs. If we need more precision, we can choose large values or on the other hand we need to compute a short test for maintenance, in this case smaller values of testLength will be preferred.

The formula to compute the number of test packets per rule is as follows:

$$\mathbf{numOfTests} = testLength * \frac{wi}{W}$$

$$Where\ W = \sum_{i=0}^{N} wi, \quad N = Number\ of\ rules$$

After calculating the number of packets generated for each rule, a method for selecting packets should be derived. Two methods can be applied. On the one hand, if the number of tests that exceeds or equals the packet coverage of a rule, all packets covered by the given rule are selected. On the other hand, the most obvious method is to randomly pick packets from each rule. Each interval in condition is taken and a random function is applied to pick a value from the interval. The second method is to take each interval, divide it evenly, and take values accordingly. The value by which the interval is

divided depends on the number of tests applied. Finally, given the fact that we have intervals, we could use the limit values method to select values. The smallest and largest values are selected, and one value in between, for each interval. In this manner, the limit values are covered—a critical element of the test. The weight or test intensity for each rule is then computed using a simple multiplication of Frequency1 and Frequency2. Now that the intensity has been computed, packets from each rule space must be selected. In the interest of simplicity and efficiency, a random approach is employed. Each packet is selected according to a random value from each field in the *Condition* part of the *Rule*. As we are computing a random function for each field in condition, the likelihood of having two packets with the same values is reduced. Randomly selected duplicated packets are simply removed from the list. An alternative approach would be to cut intervals and select values from these intervals. Network information as well as test coverage ensure that test packets will test the behavior of the firewall as it is operating and deployed in the network. In other words, the test translates the real operation of the firewall.

Using data mining could further improve the process by selecting only the most frequent cases for testing. Data mining refers to a selection process used to extract pertinent information from a large quantity of data. Although this practice is often associated with business intelligence and financial analysis, it is also increasingly employed in the scientific community. Using data mining can help scientists comb through the vast data sets that emerge from experimental and observational research. In enterprise resource planning, the process involves logically and statistically analyzing large databases in order to identify recurring patterns. Data mining could be integrated

into this test set generation approach by storing statistics on the firewall rules during a fixed, predefined time span, for example 1 week, 2 weeks, or 1 month.

The relevant information for each rule is:

- Number of matches
- Typical time of the day when this rule is triggered
- Packets that trigger this rule in the case of a rule that matches more than one packet

Given this information, the actual calculation of the test intensity can be leveraged by a data mining frequency. The new weight calculation with Frequency3 representing the data mining can simply be:

$$\mathbf{wi'} = Frequency1 * Frequency2 * Frequency3$$

After successfully extracting packets, the next module is going to take these packets and execute the test against the initial firewall configuration.

4.2.4 Test Execution

The goal in this step is to simulate the test process. To perform the test, three methods are used, from a simple software simulation of the expected results to a fancier real time simulation using a test bench in a real environment. In the following, we are going to describe the three methods in-depth.

In the case of a simple software simulation, the generated packets are used in conjunction with the rule set and by looping over the rules.

In the case of a distributed simulation, we use client server architecture to simulate two separate entities, similar to have a real test bench with a test and a machine

under test, the firewall. The simulated firewall acts as the real one by accepting the packets it should let pass and logging the dropped packets.

The last method used consists of a real time simulation using a test bench. This method is the one that is the closest to the real functioning conditions of the firewall.

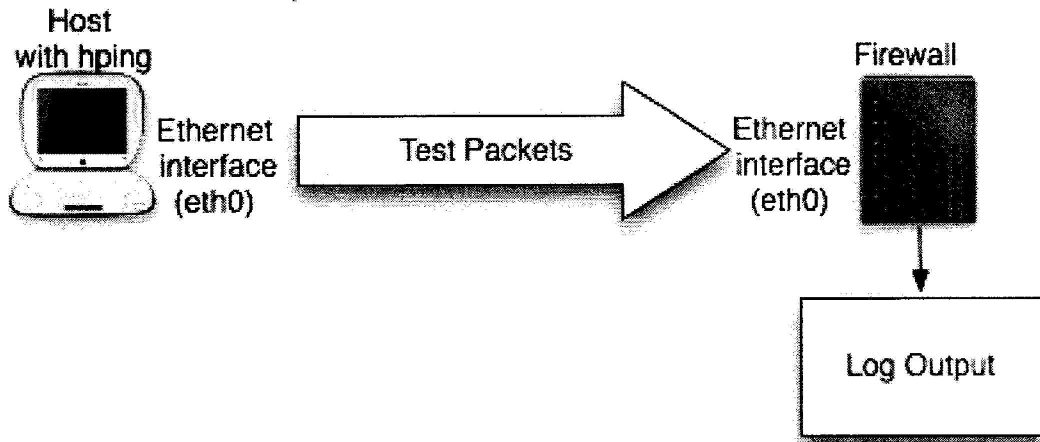


Figure 4.4: Test Bench Schema

The test bench illustrated in Figure 4.4 is employed. It is composed of a computer that crafts and sends packets to the machine where the firewall is deployed. This packet generation is made possible thanks to the fact that we are using a packet crafter that can craft packet header fields; there is no need to have the real deployment module to apply it against.

There are several benefits to using such a configuration over cited methods and testing in real situation:

- **Conformance**: Packets are crafted enabling for a real life testing before really deploying the firewall.
- **Speed**: The packets are crafted and sent using the same machine, thus the test can be completed faster
- **Cost**: Crafting packets is cheaper than reproducing a real network environment for the testing. Plus, this makes no difference from a firewall standpoint.
- **Flexibility**: Test of several configurations can be performed by simply uploading a different firewall configuration file.

Once the test is run, the results can be analyzed by reviewing the log file where the results of the packets sent against the firewall are contained.

4.2.5 Log File Parser

The results of the previous test run are found in the firewall log file. The best way to illustrate the functioning of the parser is by showing an example of a popular firewall log format, namely iptables. Table 4.4 shows a log entry in the iptables log file.

Date: Jun 19 Time: 15:24:16 DROP portmap IN=eth0 SRC=192.168.1.4 DST=192.168.1.2 PROTO=TCP SPT=33926 DPT=111 SYN

Table 4.4: Iptables Log Entry

To obtain the significant fields in the log file, each entry in the log file is parsed and stored in a container of type Vector *Rule*. In this container, only useful information is

kept. In this way, only the most relevant information for additional processing can be maintained, and the rest discarded. The relevant information allows the building of packets with their decisions and putting them into a container for further analysis. In the example described above, the useful information is as in Table 4.5.

Source Address: 192.168.1.4
Destination Address: 192.168.1.2
Source Port: 33926
Protocol: TCP
Destination Port: 111

Table 4.5: Useful log file information

As the packets logged are only the dropped packets, all logged packets are associated with *Decision* **DROP**. The unlogged packets will be associated with *Decision* **ACCEPT**. Storing packets allows the analysis of the results in a convenient GUI interface and comparison for the right decision. The next steps give more insight on how the analysis is performed.

4.2.6 Results Analyzer

This module is used for post-test analysis. It is employed after the test is executed in a way to validate the given results. To do so, each resulting packet with its decision is displayed for the operator to analyze the results.

This is a semi-automated method, as the expected results are known from the system administrator. The latter will use those results and his knowledge of the security

policy to analyze the resulting packets. In that sense, they are used in this step to validate the results obtained from the firewall file.

After the comparison is complete, the packets whose decisions were different from the expected decision are stored in a container of type *List*. The faulty packets are then used in the rule set corrector to update the firewall file.

4.2.7 Rule Updater

The correction algorithm, in Figure 4.5, goes through the following steps. First, the configuration file is read line-by-line. Whenever a line is read that is a rule, this rule is compared against all faulty packets. When matching occurs, the expected decision is compared with the obtained decision. If these are not the same, the decision in the file is changed.

```
for each Line in the firewall file
  if Rule in the firewall file
    Compare Rule with all Faulty Packets
    if Matching occurs
      Compare Expected and Obtained Decision
      if Expected Decision != Obtained Decision
        Create Rule' with Decision
        Apply Misconf Algorithm
        Write Misconf Rules in the new file
      else
        Write Rule in the new file
      end if
    end if
  else
    Write Line in the new file
  end if
end for
```

Figure 4.5: Correction Algorithm's Pseudo code

The verification process is as follows. For each erroneous packet and after detecting the rule responsible for the faulty packet, a new rule is created and added at the beginning of the current rule set to match the packet with the right decision. Finally, the misconfiguration removal algorithm is applied against this rule set to remove potential misconfiguration errors that have been introduced after this addition. The updated
Table 4.6 presents a simple example to better understand the process.

Rules before Detection	**Rules after Detection**
Faulty Packet: 45 DROP	Faulty Packet: 45 DROP
Rule 1: [0,60] ACCEPT	Rule 1 : [45,45] DROP
Rule 2: [70,80] DROP	Rule 2 : [0,44] ACCEPT
	Rule 3: [46,60] ACCEPT
	Rule 4: [70,80] DROP

Table 4.6: Error Detection and Correction Example

The updater module output is a firewall configuration file without misconfigurations errors, i.e. shadowing and redundancy, nor policy faults enforcing the wrong decision on a rule. Figure 4.5 presents the detailed pseudo-code used for the detection and correction algorithm.

Finally, the firewall configuration file that we obtain after this step has the following characteristics. It contains neither redundancy nor shadowing errors and hence the set of rules are completely independent. The readability is augmented and the packet processing is enhanced as we can move to the top of the rule set rules that are used more often. The test set generation and execution allows us to correct faulty decision, which can result from mistyping errors or wrong policy enforcement.

4.3 Summary

This chapter described our framework for testing firewalls and its inner functioning. An in-depth understanding of each of the seven modules that compose the framework was presented. First, a parser is used to obtain formal rules that are used for misconfiguration removal as well as test set generation. Once the test packets are extracted using rule set and network information, packets are sent against the firewall using one of the proposed simulation methods depending on the level of accuracy researched. Then, firewall log results are analyzed and a correction process updates the rule set by adding rules and removing misconfiguration and errors detected during the post-test analysis phase.

In the next chapter, and after understanding the functioning of our testing approach, the implementation of the approach can now be addressed. A case study is described at the end of the chapter to validate the implementation.

Chapter 5: Implementation and Case Study

The current chapter is divided into three parts. The first section presents details of the implementation of the framework, namely the tools used as well as its UML diagram. The second part presents a case study using an actual iptables 1.4.1 configuration file to validate our approach. The chapter ends with a summary that presents the different results with a comparison of existing tools.

5.1 Implementation

The implementation of the seven modules of our framework is based on a mix of Java 1.5 programming as well as parser generator using JavaCC [23]. An overview of the software and tools used is presented as well as a more detailed description of the UML structure of our implementation.

5.1.1 Software and Tools Used

The implementation of the seven modules of our framework is based on a mix of Java programming as well as a parser generator using JavaCC. The first module of the framework which is the firewall configuration file parser uses JavaCC as the parser generator. The misconfiguration removal algorithm, which has been developed in [15], is built using Java. The next step which is the test set generation module uses Java to derive the test set. After generating the test set, Hping2 [24], is used to craft packets and send them in the network.

Iptables [25] is the latest entry in Linux firewalls. The choice of iptables is due to the fact that it is one of the most popular firewalls and the one with the most active

communities supporting it, with features added on a regular basis. The rules are defined using commands in the command line. Those commands facilitate packet filtering; network address translation (NAT); packet mangling in the most recent Linux versions, 2.4 and above; and logging. A complete description of the syntax for iptables can be found in their official website [25]. In the following, an explanation of the different features of iptables is given.

Packet Filtering, as described in detail in Chapter 2, is the selective passing or blocking of data packets by analyzing and matching the header of these packets. Network address translation is the process of converting an Internet Protocol address into another Internet Protocol address. Packet Mangling is the ability to alter passing data packets before or after rooting occurs. Logging, commonly named data logging, is the process of recording sequential data, chronologically. Iptables is capable of logging data packets, often the dropped ones.

In our implementation, the focus has been placed on the packet filtering capabilities of iptables, but the efforts to offer a framework for packet filtering can be adapted by adding attributes to *Condition* for the features and the functionalities of iptables.

The Java Compiler Compiler (JavaCC) is a parser generator written in Java. Instead of writing code to parse a data stream, JavaCC can be used to write the parser. As we are using Java 1.5 language as our implementation language, JavaCC was the most natural choice as it has syntax similar to Java. In our implementation, JavaCC has been used to build a parser for the iptables file, the iptables configuration file and log file, as well as the network information file.

The tool used to craft packet headers in order to perform the test is called Hping2. Hping2 is a command-line oriented TCP/IP packet assembler. It is similar to the well-known ping UNIX command, but in addition to ICMP echo requests, Hping2 supports TCP, UDP, and RAW-IP protocols as well.

Compared to other tools, Hping2 has been used for its overall ease of use as it does not require any programming like other packet crafting tools that are built with low level C programming. In our framework, Hping2 commands are triggered using a wrapper, SysCommandExecutor class, around the Java command line executor method. Appendix A provides the code of the implementation of the seven modules of the testing framework.

5.1.2 UML Class Diagram of the Framework

The UML class diagram shows the different packages and classes of the framework and their interrelationships. This way, we have a holistic picture of the design and the implementation.

In the following, we will enumerate and explain the three packages that compose our framework.

Package *com* contains the following sub packages. *com.acl* contains the parser used for the Access List syntax used mainly in Cisco firewalls. *com.iptables* contains the parser used for the iptables Linux firewall syntax. It also has the following sub packages. *com.iptables.log* contains the parser for the iptables log file. *com.iptables.gui* contains the graphical user interface (GUI) used for firewall testing and the main function contained in class FirewallAnalyzerFrame. *com.tester* contains the classes responsible for test set

generation and execution as well as correction. *com.misconf* contains the class for the misconfiguration removal algorithm.

Package *obj* contains classes for the different objects relative to firewalls and networking, respectively *Rule*, *Condition*, *Packet*, *Address*, and *Host*.

Package *util* contains a number of utility functions like the parser for network information, the wrapper for Java command line execution method, and other methods for test set generation.

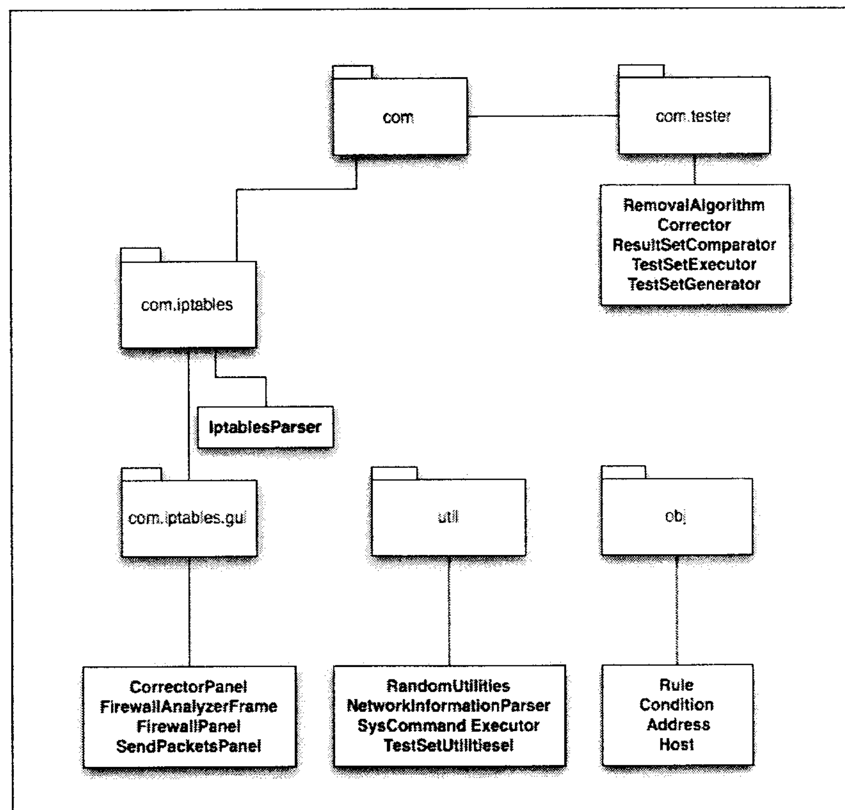


Figure 5.1: Structure Diagram of the Test Framework

Figure 5.1 is the representation of the different packages and classes of the implementation of the firewall updater framework presented in this thesis.

5.2 Case Study

Type of Firewall	First Match Policy
Default Security Policy	Deny-everything
Number of Rules	11
Name of Firewall	iptables 1.4.1

Table 5.1: Specifications of the Tested Firewall

To validate the methodology presented in this thesis, an iptables configuration file is used as an input for our framework. Table 5.1 states the specifications of the tested firewall.

To perform the test, two hosts are used and connected through an RJ 45 Ethernet crossover cable. The test setup configuration is inspired from [13]. This way, we can simulate an Internet connection and have both Ethernet interfaces up for the test purpose. As we are only looking to send the generated packets against the interface, all other traffic on both interfaces is disabled, such as Address Resolution Protocol (ARP). As a result of removing all other traffic, the output log checking and analysis is made much simpler. Both hosts run Fedora Linux 7 and are running iptables 1.4.1. The tested firewall is installed and run in one host, called the system under test.

```
$IPT --flush
$IPT -t nat --flush
$IPT -t mangle --flush

$IPT -A INPUT -i lo -j ACCEPT
$IPT -A OUPUT -o lo -j ACCEPT

$IPT --policy INPUT DROP
$IPT --policy OUTPUT DROP
$IPT --policy FORWARD DROP

$IPT -A INPUT -i eth0 -s 10.0.0.0/8 -j DROP
$IPT -A INPUT -i eth0 -s 172.16.0.0/12 -j DROP
$IPT -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP

$IPT -A INPUT -i eth0 -s 192.200.21.0/24 -j DROP

$IPT -A INPUT -i eth0 -p udp -m multiport --sport 20:50 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0 -p udp -m multiport --sport 60:70 -d 192.200.21.0/24 -j ACCEPT

$IPT -A INPUT -i eth0 -p udp -m multiport --sport 1:1024 -d 192.200.21.0/24 -j ACCEPT

$IPT -A INPUT -i eth0 -p tcp -s 170.1.1.0/24 --sport 80 --dport 55 -j ACCEPT

$IPT -A INPUT -i eth0 -p tcp -s 170.1.0.0/16 --sport 80 --dport 55 -j ACCEPT

$IPT -A INPUT -i eth0 -p tcp -s 192.1.1.0/24 --dport 80 -j ACCEPT
$IPT -A INPUT -i eth0 -p tcp -s 192.1.1.20  --dport 80 -j DROP
```

Table 5.2: Input configuration file

At first, the configuration file firewall.conf, in Table 5.2 is parsed by the iptables parser generator. The results of this step are the rules parsed in the format that can be understood by the misconfiguration detection and removal algorithm as described in Chapter 4. The parser will also keep all the information needed like variable declarations, loop back interface rules, default policy and the name of the interfaces, in order to recover in the last step of the process. Table 5.3 contains the parsed results and their format.

Rule 1: [0, 2] [167772160, 184549375] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 2: [0, 2] [2886729728, 2887778303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 3 : [0, 2] [3232235520, 3232301055] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 4: [0, 2] [3234338048, 3234338303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 5: [2, 2] [0, 4294967295] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 6: [2, 2] [0, 4294967295] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 7: [2, 2] [3234332928, 3234333183] [1, 1024] [3234338048, 3234338303] [0, 65534] DROP
Rule 8: [1, 1] [2852192512, 2852192767] [80, 80] [0, 4294967295] [55, 55] ACCEPT
Rule 9: [1, 1] [2852192256, 2852257791] [80, 80] [0, 4294967295] [55, 55] ACCEPT
Rule 10: [1, 1] [3221291264, 3221291519] [0, 65534] [0, 4294967295] [80, 80] ACCEPT
Rule 11: [1, 1] [3221291284, 3221291284] [0, 65534] [0, 4294967295] [80, 80] DROP

Table 5.3: Parsing Results of the Firewall File

The parser covers all stateless syntax of iptables. The main reason for not covering all cases remains in the fact, that in this work, the most important aspect is to validate the approach using a simple stateless firewall. The development of a more sophisticated parser that can be used to cover all iptables rules can be derived. This new parser will generate more fields in *Condition* like *tcpFlag* and *Time Intervals*. Those added fields will have no consequences in the next modules of the framework. Consequently, if the method works for this case, it will work for any firewall no matter the number of features.

After running the misconfiguration removal algorithm module, a summary of the errors found is presented in Table 5.4. Rule 8 and Rule 11 from last step have been removed because they were redundant and shadowed respectively.

Rule 1: [0, 2] [167772160, 184549375] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 2: [0, 2] [2886729728, 2887778303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 3: [0, 2] [3232235520, 3232301055] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 4: [0, 2] [3234338048, 3234338303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 5: [2, 2] [0, 167772159] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 6: [2, 2] [184549376, 2886729727] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 7: [2, 2] [2887778304, 3232235519] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 8: [2, 2] [3232301056, 3234338047] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 9: [2, 2] [3234338304, 4294967295] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 10: [2, 2] [0, 167772159] [60, 70] [3234338048, 3234338303] [0, 65534] DECISION: ACCEPT
Rule 11: [2, 2] [184549376, 2886729727] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 12:[2, 2] [2887778304, 3232235519] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 13: [2, 2] [3232301056, 3234338047] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 14: [2, 2] [3234338304, 4294967295] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 15: [2, 2] [3234332928, 3234333183] [1, 19] [3234338048, 3234338303] [0, 65534] DROP
Rule 16: [2, 2] [3234332928, 3234333183] [51, 59] [3234338048, 3234338303] [0, 65534] DROP
Rule 17: [2, 2] [3234332928, 3234333183] [71, 1024] [3234338048, 3234338303] [0, 65534] DROP
Rule 18: [1, 1] [2852192256, 2852257791] [80, 80] [0, 4294967295] [55, 55] ACCEPT
Rule 19: [1, 1] [3221291264, 3221291519] [0, 65534] [0, 4294967295] [80, 80] ACCEPT

Table 5.4: Results of the Misconfiguration Removal Algorithm

Table 5.5 presents the results of the test set generation. Weights are computed as described in Chapter 4. The network information file used is in Appendix C. The number of packets generated is 241 packets with a *testLength* value of 250. The generated packets are different from the test length due to the fact that redundant values are removed from the set of produced packets. The next step is to select a number of packets per rule using the random approach described in the previous chapter.

Rule Number	Matches	Weight	Generated Packets
1	0	1.09	4
2	0	1.09	4
3	0	1.09	4
4	3	4.35	17
5	0	1.51	6
6	0	1.14	4
7	0	1.40	5
8	0	2.33	9
9	0	1.25	5
10	0	1.51	6
11	1	2.27	9
12	0	1.40	5
13	0	2.33	9
14	0	1.25	5
15	0	3.30	13
16	0	3.30	13
17	0	3.29	13
18	18	20.7	84
19	5	6.53	26

Table 5.5: Generated Packets per Rule

To perform the test, the setup described earlier is used. Packet headers, from the test generation, are crafted into hping packets. A necessary configuration is needed to run and get the results of the test. In the following, two configuration steps to perform the test are described.

First, the Linux host, where the test framework is running, is configured with root access. Root access is mandatory to have access to hping crafting capabilities.

As a regular operation, iptables will log the *dropped packets* into */log/var/messages*. The problem is that this file contains log messages from all processes running in the machine. In order to make it easier to parse the log file results, as a second step, messages are logged into a separate file called *log/var/fwlog*. To do so, the *syslog.conf* file has to be configured. The *syslog.conf* file, which specifies rules for logging, is the main configuration file used by UNIX systems to log system messages. Therefore, the following line needs to be added in the file: kern.warn /var/log/fwlog.txt.

After this addition, the system is told to store all messages that have a warn alert in fwlog.txt. Since logging in iptables is by default of level 4, i.e. level warn, all iptables messages are logged in that file.

After the log file has been parsed and the dropped packets stored in a Vector container, the next step is the analysis of the results by a system administrator.

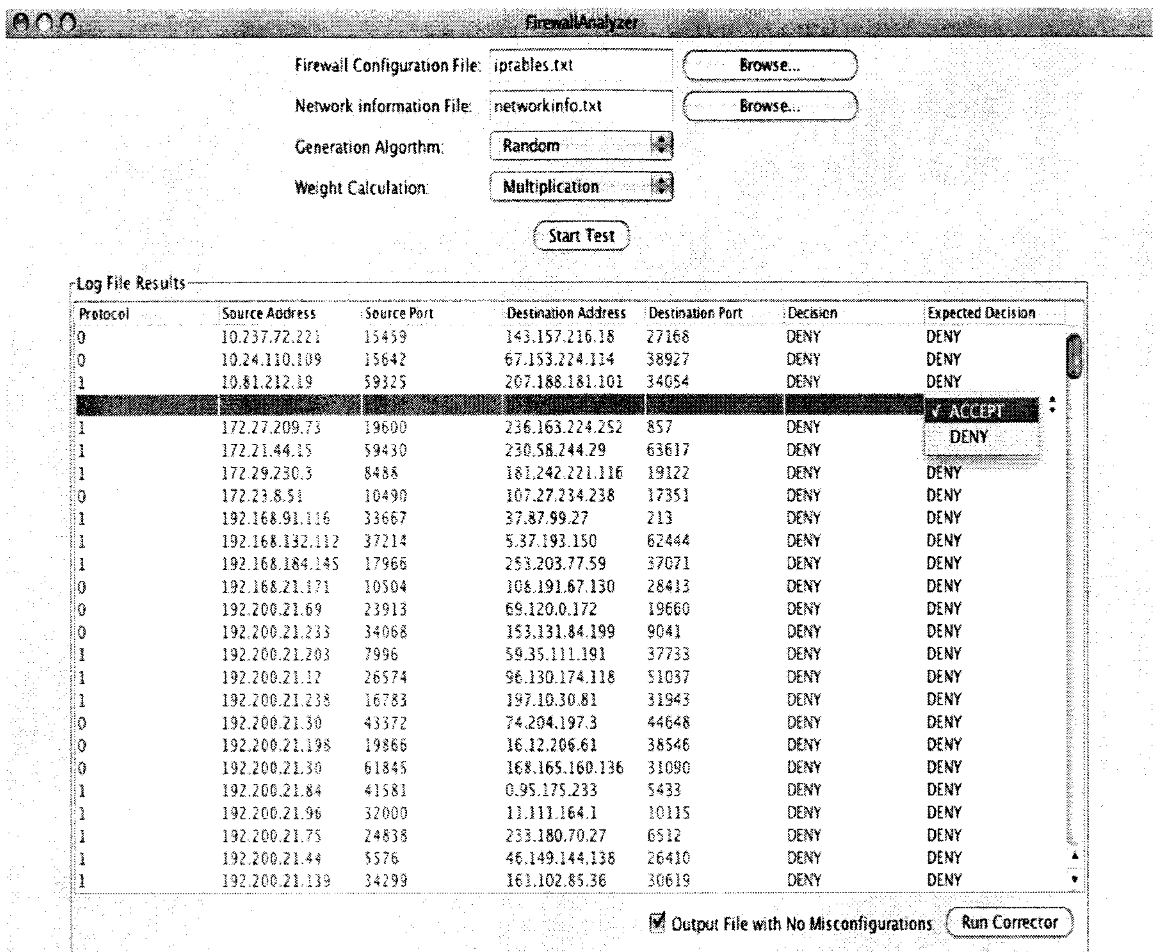


Figure 5.2: Our GUI interface for Firewall testing

The GUI interface, in Figure 5.2, makes it easy to change the results by simply changing the decision for ACCEPT to DROP or the reverse. After analyzing the results, two faulty packets were found shown in Table 5.6.

Faulty packets
Faulty packet 0: [1, 1] [184065102, 184065102] [23934, 23934] [2620904916, 2620904916] [6253, 6253] ACCEPT
Faulty packet 1: [2, 2] [24655340, 24655340] [28, 28] [3234338205, 3234338205] [52740, 52740] DROP

Table 5.6: Faulty packets with their decision

Rule 0: [1, 1] [184065102, 184065102] [23934, 23934] [2620904916, 2620904916] [6253, 6253] ACCEPT
Rule 1: [2, 2] [24655340, 24655340] [28, 28] [3234338205, 3234338205] [52740, 52740] DROP
Rule 2: [0, 0] [167772160, 184549375] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 3 : [2, 2] [167772160, 184549375] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 4 : [1, 1] [167772160, 184065101] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 5 : [1, 1] [184065103, 184549375] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 6 : [1, 1] [184065102, 184065102] [0, 23933] [0, 4294967295] [0, 65534] DROP
Rule 7 : [1, 1] [184065102, 184065102] [23935, 65534] [0, 4294967295] [0, 65534] DROP
Rule 8 : [1, 1] [184065102, 184065102] [23934, 23934] [0, 2620904915] [0, 65534] DROP
Rule 9 : [1, 1] [184065102, 184065102] [23934, 23934] [2620904917, 4294967295] [0, 65534] DROP
Rule 10 : [1, 1] [184065102, 184065102] [23934, 23934] [2620904916, 2620904916] [0, 6252] DROP
Rule 11 : [1, 1] [184065102, 184065102] [23934, 23934] [2620904916, 2620904916] [6254, 65534] DROP
Rule 12: [0, 2] [2886729728, 2887778303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 13: [0, 2] [3232235520, 3232301055] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 14: [0, 2] [3234338048, 3234338303] [0, 65534] [0, 4294967295] [0, 65534] DROP
Rule 15: [2, 2] [0, 24655339] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 16 : [2, 2] [24655341, 167772159] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 17 : [2, 2] [24655340, 24655340] [20, 27] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 18 : [2, 2] [24655340, 24655340] [29, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 19 : [2, 2] [24655340, 24655340] [28, 28] [3234338048, 3234338204] [0, 65534] ACCEPT
Rule 20: [2, 2] [24655340, 24655340] [28, 28] [3234338206, 3234338303] [0, 65534] ACCEPT
Rule 21: [2, 2] [24655340, 24655340] [28, 28] [3234338205, 3234338205] [0, 52739] ACCEPT
Rule 22 : [2, 2] [24655340, 24655340] [28, 28] [3234338205, 3234338205] [52741, 65534] ACCEPT
Rule 23: [2, 2] [184549376, 2886729727] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 24: [2, 2] [2887778304, 3232235519] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 25: [2, 2] [3232301056, 3234338047] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 26: [2, 2] [3234338304, 4294967295] [20, 50] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 27: [2, 2] [0, 167772159] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 28: [2, 2] [184549376, 2886729727] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 29: [2, 2] [2887778304, 3232235519] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 30: [2, 2] [3232301056, 3234338047] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 31: [2, 2] [3234338304, 4294967295] [60, 70] [3234338048, 3234338303] [0, 65534] ACCEPT
Rule 32: [2, 2] [3234332928, 3234333183] [1, 19] [3234338048, 3234338303] [0, 65534] DROP
Rule 33: [2, 2] [3234332928, 3234333183] [51, 59] [3234338048, 3234338303] [0, 65534] DROP
Rule 34: [2, 2] [3234332928, 3234333183] [71, 1024] [3234338048, 3234338303] [0, 65534] DROP
Rule 35: [1, 1] [2852192256, 2852257791] [80, 80] [0, 4294967295] [55, 55] ACCEPT
Rule 36: [1, 1] [3221291264, 3221291519] [0, 65534] [0, 4294967295] [80, 80] ACCEPT

Table 5.7: Updated Rules

Now, the last step will consist of writing a new configuration file, where no more errors are to be found. At first, error detection based on the three faulty packets is performed as described in Chapter 4. The information that has been held in the parser module is then used to reconstruct the new configuration file. The faulty firewall is not over written and is kept for backup and log. At last, a comparison with the most advanced techniques for firewall testing is presented as follows. Table 5.7 shows the list of updated rules. In bold are the rules that have been added or modified after the corrector module

has been applied. We parse back the rule set and rewrite them as iptables rules. The output configuration file is shown in Table 5.8.

```
$IPT --flush
$IPT -t nat --flush
$IPT -t mangle --flush

$IPT -A INPUT -i lo -j ACCEPT
$IPT -A OUPUT -o lo -j ACCEPT

$IPT --policy INPUT DROP
$IPT --policy OUTPUT DROP
$IPT --policy FORWARD DROP

$IPT -A INPUT -i eth0  -p tcp -s 10.3.170.30 --sport 2020 -d 144.202.125.243 --dport 63513 -j ACCEPT
$IPT -A INPUT -i eth0  -p icmp -s 192.200.21.109 --sport 54461 -d 173.115.81.0 --dport 18670 -j ACCEPT
$IPT -A INPUT -i eth0 –p icmp -s 10.0.0.0/8 -j DROP
$IPT -A INPUT -i eth0  -p udp -s 10.0.0.0/8 -j DROP
$IPT -A INPUT -i eth0  -p tcp --src-range 10.0.0.0-10.3.170.29 -j DROP
$IPT -A INPUT -i eth0  -p tcp --src-range 10.3.170.31-10.255.255.255 -j DROP
$IPT -A INPUT -i eth0  -p tcp -m multiport -s 10.3.170.30 --sport 0:2019 -j DROP
$IPT -A INPUT -i eth0  -p tcp -m multiport -s 10.3.170.30 --sport 2021:65534 -j DROP
$IPT -A INPUT -i eth0  -p tcp -s 10.3.170.30 --sport 2020 --dst-range 0.0.0.0-144.202.125.242 -j DROP
$IPT -A INPUT -i eth0  -p tcp -s 10.3.170.30 --sport 2020 --dst-range 144.202.125.244-255.255.255.255 -j DROP
$IPT -A INPUT -i eth0  -p tcp -m multiport -s 10.3.170.30 --sport 2020 -d 144.202.125.243 --dport 0:63512 -j DROP
$IPT -A INPUT -i eth0  -p tcp -m multiport –s 10.3.170.30 --sport 2020 -d 144.202.125.243 --dport 63514:65534 -j DROP
$IPT -A INPUT -i eth0  --src-range 172.16.0.0-172.31.255.255 -j DROP
$IPT -A INPUT -i eth0  -s 192.168.0.0/16 -j DROP
$IPT -A INPUT -i eth0  -p tcp  -s 192.200.21.0/24 -j DROP
$IPT -A INPUT -i eth0  -p udp -s 192.200.21.0/24 -j DROP
$IPT -A INPUT -i eth0  -p icmp --src-range 192.200.21.0-192.200.21.108 -j DROP
$IPT -A INPUT -i eth0  -p icmp --src-range 192.200.21.110-192.200.21.255 -j DROP
$IPT -A INPUT -i eth0  -p icmp -m multiport -s 192.200.21.109--sport 0:54460 -j DROP
$IPT -A INPUT -i eth0  -p icmp -m multiport -s 192.200.21.109 --sport 54462:65534 -j DROP
$IPT -A INPUT -i eth0  -p icmp -s 192.200.21.109 --sport 54461 --dst-range 0.0.0.0-173.115.80.255 -j DROP
$IPT -A INPUT -i eth0  -p icmp -s 192.200.21.109 --sport 54461 --dst-range 173.115.81.1-255.255.255.255 -j DROP
$IPT -A INPUT -i eth0  -p icmp -m multiport -s 192.200.21.109 --sport 54461 -d 173.115.81.0 --dport  0:18669 -j DROP
$IPT -A INPUT -i eth0  -p icmp -m multiport -s 192.200.21.109 --sport 54461 -d 173.115.81.0 --dport 18671:65534 -j DROP
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 0.0.0.0-9.255.255.255 --sport 20:50 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 11.0.0.0-172.15.255.255 --sport 20:50 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 172.32.0.0-192.167.255.255 --sport 20:50 -d 192.200.21.0/24  -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 192.169.0.0-192.200.20.255 --sport 20:50 -d 192.200.21.0/24  -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 192.200.22.0-255.255.255.255 --sport 20:50 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 0.0.0.0-9.255.255.255 --sport 60:70 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 11.0.0.0-172.15.255.255 --sport 60:70 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 172.32.0.0-192.167.255.255 --sport 60:70 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 192.169.0.0-192.200.20.255 --sport 60:70 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport --src-range 192.200.22.0-255.255.255.255 --sport 60:70 -d 192.200.21.0/24 -j ACCEPT
$IPT -A INPUT -i eth0  -p udp -m multiport -s 192.200.1.0/24 --sport 1:19 -d 192.200.21.0/24 -j DROP
$IPT -A INPUT -i eth0  -p udp -m multiport -s 192.200.1.0/24 --sport 51:59 -d 192.200.21.0/24 -j DROP
$IPT -A INPUT -i eth0  -p udp -m multiport -s 192.200.1.0/24 --sport 71:1024 -d 192.200.21.0/24 -j DROP
$IPT -A INPUT -i eth0  -p tcp -s 170.1.0.0/16 --sport 80 --dport 55 -j ACCEPT
$IPT -A INPUT -i eth0  -p tcp -s 192.1.1.0/24 --dport 80 -j ACCEPT
```

Table 5.8: Output configuration file

Testing Approach	Complexity
Random	O(n)
Misconfiguration Removal	$O(n^2)$
Policy Segmentation	$O(2^n)$
Our approach	$O(n^2)$

Table 5.9: Comparison with other testing approaches

Finally, we conduct a comparison of our method with the most common techniques with regard to complexity. Table 5.9 presents the results of the comparison. Our method's complexity is $O(n^2)$. The complexity of the random approach is inferior, but it also the less accurate detection and testing approach. Policy segmentation's complexity is exponential and thus cannot be applied to large and interrelated test sets. Our approach is as fast as the misconfiguration removal algorithm while being more accurate as it detects the mistyping and policy errors in addition to misconfiguration errors. The conclusion that we can draw from this table is that our method can carry large test sets while producing the most accurate results.

5.3 Summary

This chapter contains the practical part of our work, namely the implementation as well as the proof of concept of our tool. It started by describing the tools and software used. Then, a high level picture of the implementation by an UML diagram was presented. Next, the case study shows an example of an iptables firewall configuration file that contains common misconfigurations and policy errors. The framework for testing

shows how the conjugation of misconfiguration removal and network knowledge information can help us in removing and getting an error free firewall file. The resulting firewall is written into another file so the system administrator can compare the modified firewall with the original one. At last, a comparison with other methods was presented showing the performance of our approach over other methods. The next chapter is a summary of the whole thesis. It presents the achievements of this research as well as further work that can be done in this field.

Chapter 6: Conclusions and Summary

Due to the necessity of the global connectivity as well as the nature of the TCP/IP protocol, firewalls form a central element in protecting organizations. In that prospect, the problematic of testing firewalls and ensuring their functionality are becoming ever more vital. Moreover, the increasing number of complex security policies increases the risk of errors in firewall configurations [1].

6.1 Achievements

In this thesis, a new method for testing a particular firewall configuration has been presented thanks to an approach that takes into account both the internal functioning of the firewall as well as network topology information. The intelligent framework for firewall testing that has been developed includes the following modules. First, we parse the firewall file into a formal language. The rules in the formal language are then passed through an algorithm for misconfiguration error detection and removal. The test set is then generated based on the internal configuration and the information about the topology of the network. After the test packets are applied, an analysis is conducted that will check for mistyping errors and correct them. Using this approach, the problematic of firewall testing and its main problems were confronted: test coverage and accuracy by using an intelligent method for packet selection, and completeness by using an error detection and correction algorithm.

6.2 Future Work

As future work, we can extend our work to a multiple firewall environment. Usually, large organizations contain more than one firewall and the interaction between different firewalls leads to a different set of final decisions and errors. The work would focus on adapting the current misconfiguration removal technique and test generation process. Also, more refinement should be brought to the packet selection process, and different selection methods should be tested and compared. Another enhancement would be a module that contains tests for known and upcoming vulnerabilities can be added. This module will be updated on a regular basis just like an antivirus. This procedure is a complementary addition to our white box approach. In the future, more refinement should be placed on the packet selection process, and different selection methods should be tested and compared. On another hand, a formal test coverage quantification method should be developed in order to accurately compare our testing framework with existing methods.

References

[1] A. Wool, A Quantitative Study of Firewall Configuration Errors, *In IEEE Computer*, 2004, 62-67.

[2] G Vigna, A Topological Characterization of TCP/IP Security, *In Proc. of the international Symposium of Formal Methods Europe*, 2003, 914-939

[3] W. Cheswick, and S. Bellovin, How Computer Security Works: Firewalls, *Scientific American*, 1998, pp. 106-107.

[4] RFC 1918 Specification, http://www.rfc.net/rfc1918.html.

[5] US CERT, Computer Emergency Readiness Team, http://www.us-cert.gov.

[6] CIAC, Computer Incident Advisory Capability, http://ciac.llnl.gov/ciac.

[7] K. Al-Tawil, and I. Al-Kaltham, Evaluation and Testing of Internet Firewall, *In International Journal of Network Management*, 1999, 135-149.

[8] Info About SATAN, http://www.cerias.purdue.edu/about/history/coast/satan.php

[9] The Firewall Toolkit, http://www.fwtk.org.

[10] Dante a SOCKS Implementation, http://www.inet.no/dante.

[11] J. Jürjens, and G. Wimmel, Specification Testing of Firewalls, *In the 4t International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2001, 308-316.

[12] K. Yoo, and D. Hoffman, Blowtorch: a Framework for Firewall Test Automation, *In Proc. of the 20th IEEE/ACM international Conferencne on automated software engineering*, 2005, 96-103.

[13] Y. Du, and D. Hoffman, PBit: A Pattern-Based Testing Framework for iptables, *In Proc. of the Second Annual Conference on Communication Networks and Services Research*, 2004, 107-112.

[14] G. Vigna, A Formal Model for Firewall Testing.

[15] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia-Alfaro, Detection and Removal of Firewall Misconfiguration, *In International Conference on Communication, Network and Information Security*, 2005, 154-161.

[16] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer, Policy Segmentation for Intelligent Firewall Testing, *In 1st IEEE ICNP Workshop on Secure Network Protocols,* 2005, 67-72.

[17] A. Mayer, A. Wool, and E. Ziskind, Fang: a Firewall Analysis Engine, *In Proc. Of IEEE Symposium on Security and Privacy*, 2000, 177-187.

[18] W. Geng, S. Flinn, and J. DeDourek, Usable Firewall Configuration, *In the Third Annual Conference on Privacy, Security and Trust*, 2005

[19] Checkpoint SmartMap, http://www.checkpoint.com.

[20] Y. Bartal, A.Mayer, K. Nissim and A. Wool, Firmato: a Novel Firewall Management Toolkit, *In IEEE Symposium on Security and Privacy*, 1999, 17-31.

[21] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, A Formal Approach to Specify and Deploy a Network Security Policy, *In IFIP International Federation for Information Processing*, 2004, 203-218.

[22] E. Al-Shaer, and H. Hamed, Firewall Policy Advisor for Anomaly Discovery and Rule Editing, *In Proc. of IFIP/IEEE Eight International Symposium on Integrated Network Management,* 2003, 17-30.

[23] JavaCC Project Page, https://javacc.dev.java.net.

[24] Hping Packet Assembler, http://www.hping.org.

[25] iptables/Netfilter Project, http://www.netfilter.org.