Iowa State University
Digital Repository

Retrospective Theses and Dissertations | Iowa State University Capstones, Theses and Dissertations

2007

Internet scale endpoint masquerading

Thad Michael Gillispie
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Electrical and Electronics Commons

Recommended Citation
Gillispie, Thad Michael, "Internet scale endpoint masquerading" (2007). *Retrospective Theses and Dissertations*. 14656.
https://lib.dr.iastate.edu/rtd/14656

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

Internet scale endpoint masquerading

by

Thad Michael Gillispie

A thesis submitted to the graduate faculty

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-Majors: Computer Engineering; Information Assurance

Program of Study Committee:
Douglas Jacobson, Major Professor
Thomas Daniels
Clifford Bergman

Iowa State University

Ames, Iowa

2007

Copyright © Thad Michael Gillispie, 2007. All rights reserved.

المنارة للاستشارات

www.manaraa.com

UMI Number: 1447508

UMI

UMI Microform 1447508
Copyright 2008 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

المنارة للاستشارات

www.manaraa.com

Table of Contents

List of Figures iv

Abstract v

1 Introduction 1

1.1 Testbeds 1

1.2 Motivation 3

2 Related Work 5

2.1 ISEAGE overview 5

2.2 ISEAGE projects 7

2.2.1 DeepFreeze 7

2.2.2 Traffic Mapper 8

2.2.3 Packet Mangling 9

2.2.4 Traffic Generation 9

2.3 Related Technologies 10

2.3.1 Routing 11

2.3.2 Network Address Translation 11

2.3.3 Domain Name System 13

3 Design Overview 15

3.1 Goals 15

3.2 ISEAGE Integration 17

3.3 Assumptions 22

4 Architecture 24

4.1 Programming Framework 24

4.1.1 Libpcap 24

4.1.2 Libnet 25

للاستشارات المنارة

www.manaraa.com

4.1.3 Standard Template Library 26
4.1.4 Boost 27
4.2 General Considerations 28
4.2.1 Network Device Type 29
4.2.2 Threads and Packet Capturing 30
4.3 Alternate Approaches 32
4.3.1 Gateway 33
4.3.2 DNS 34

5 Implementation **36**

5.1 Virtual Endpoint Gateway 37
5.2 Virtual DNS Endpoint 39
5.3 Additional Features 41

6 Future Work **43**

7 Conclusion **45**

Appendix A: Data Class Header File **46**

Appendix B: ISEMasq Example Config File **47**

References **48**

Acknowledgements **50**

المنارة للاستشارات
www.manaraa.com

List of Figures

Figure 3.1: Small Toplogy 18

Figure 3.2: Large Toplogy 19

Figure 3.3: Small Topology Simplified 20

Figure 3.4: Large Toplogy Simplified 21

Figure 4.1: Overall architecture and data flow of a typical software approach. 31

Figure 5.1: Implementation Overview 36

Figure 5.2: Gateway Flow 38

Figure 5.3: DNS Flow 40

المنارة للاستشارات

www.manaraa.com

Abstract

To keep up with the security needs being exerted by the ever-increasing complexity of technology, new ideas and approaches are needed. Once such attempt to address this is the Internet Scale Event and Attack Generation Environment (ISEAGE) at Iowa State University (ISU). ISEAGE is a next generation Internet testbed that hopes to provide researchers with the tools and resources necessary to address the every vexing security issues in today's world. Among the many challenges involved with creating an Internet scale testbed is how to realistically virtualize the thousands of servers that make up the various destinations or endpoints on the Internet. To specifically address this problem, the Internet Scale Endpoint Masquerading tool (ISEMasq) was developed. ISEMasq is an integral part of ISEAGE that enables a small set of servers with off-the-shelf software to pose or masquerade as any number of actual Internet destinations. To accomplish this, ISEMasq leverages various API functionality from the current and upcoming release of the ISO C++ standard, as well as other widely used and accepted third-party C and C++ networking APIs. The result is a highly reliable, performance driven, packet level network tool that brings ISEAGE one step closer to completion.

المنارة للاستشارات

www.manaraa.com

1 Introduction

Modeling the vastness of the Internet with its millions of destinations is a complex and seemingly impossible task, though it is necessary in order to create a realistic testbed environment. However, what if it were possible to masquerade a small set of endpoints as countless thousands of actual endpoints? The development of the Internet Scale Endpoint Masquerading tool (ISEMasq, pronounced: [ice-mask]) makes this illusion possible. Before discussing the details of how this is done, it is first necessary to provide groundwork and background information further detailing testbeds and the project's motivation.

1.1 Testbeds

As technology grows and advances so too grows our dependency on it; ever affecting and integrating with day-to-day life. A prime example of the growth of this complex relationship is the extensive use of the Internet for communication and sharing of products and ideas. As the interdependency between technology and daily life increases, it becomes even more critical to ensure the reliably and security of this infrastructure. However, the increased complexity of modern technology as well as our dependency is also increasing the difficultly of developing and effectively testing new security technologies. Undertaking this enormous task will require new approaches and ideas as well as the creation of versatile testbed environments that allow modeling and testing of large or complex networks. According to a report published in 2005 by the President's Information Technology Advisory Committee, one of the top ten areas needing increased research and development is "modeling and testbeds for new technologies" [1]. Such testbeds would provide a host of opportunities for furthering

المنارة للاستشارات
www.manaraa.com

security education as well as security product development [2]. To that end, the committee feels that:

> "One of the barriers to the rapid development of new cyber security products is the paucity of realistic models and testbeds available for exercising the latest technologies in a real-world environment [1]."

In order to facilitate and ensure the future of a society that places such a high emphasis on technology, it is necessary to development new testbed environments that better simulate the real world. The creation of such a virtual Internet represents a new paradigm in the area of security research and cyber forensics, and it will enable new and innovative research needed to solve the current security problems facing the world today. These environments will be essential in both industry and education. Being able to better model a real-world setting will make it easier to educate new IT security students without endangering real networks. Additionally it will allow industry to more thoroughly test new security products to help ensure that they operate effectively before testing them in a real environment. This will help alleviate the risk of accidental damage to equipment or information during the training of students or during the testing and development of new security products and devices [2].

That said, the Internet modeling problem is challenging, mostly because of the Internet's scale and complexity. In other words, how does one model or simulate the vastness of the Internet with only a limited amount of resources? The Internet is vast, made up of millions of pieces of computer hardware (switches, routers, PCs and servers). Though if one takes the point of view from each device's perspective, there isn't much in the way of hardware, mostly just the signals that come to/from it from other devices

المنارة للاستشارات
www.manaraa.com

somewhere out there in cyber-space. With this perspective in mind, it is possible to lay out the various signal characteristics as just a few functional layers.

The communication of any two remote devices across cyberspace can be broken down into three simple layers. Layer 1 represents a sort of traffic control and basic infrastructure; it is comprised of a mass of interconnected routers that form the backbone of the Internet. Layer 2 contains more infrastructural features, specifically the address translation—DNS—and destinations. Layer 3 is then all the traffic; it is the client and network applications that utilize Layers 1 and 2 to accomplish their tasks. The combination of these three layers forms a fairly realistic yet manageable representation of the Internet in that it readily illustrates how ISEMasq is integrated into an Internet testbed. ISEMasq is specifically concerned with the mechanisms of Layer 2, so further discussion will be focused on primarily this layer.

1.2 Motivation

To specifically address the issues with representing Layer 2, it should first be broken down into two pieces as alluded to above. The first piece is DNS. This allows for an application to determine the address of the URL it wishes to contact. With address in hand, the second piece is needed: the destination. DNS tells a signal where to go and the destination gives it somewhere to actually go to as it traverses Layer 1. Both pieces of Layer 2 are an important part of an effective testbed environment. In order to accurately test a security solution, one needs traffic of all sorts, and traffic needs somewhere to go. Specifically, somewhere that can actively respond, otherwise it would be as if the phone rings forever with no one answering.

المنارة للاستشارات
www.manaraa.com

Such a fully reactive destination is the very essence of ISEMasq. ISEMasq provides large-scale endpoint objects without needing hundreds of actual machines by emulating multiple machines and/or networks including the name resolutions of those endpoints. ISEMasq is designed specially for integration with the Internet testbed research project at Iowa State University. The project is an ambitious attempt to meet the needs and challenges of an Internet scale testbed.

المنارة للاستشارات

www.manaraa.com

2 Related Work

2.1 ISEAGE overview

The Internet Scale Event and Attack Generation Environment (ISEAGE) is a first of its kind facility in a public university dedicated to creating a virtual Internet [3]. To fully understand the implication and importance of the work discussed in this thesis, it is essential to first have a clear and complete understanding of the vision of ISEAGE, its design, and the fundamental underpinnings needed to realize that design.

As a next-generation testbed, ISEAGE breaks away from traditional computer-based simulations and allows for researchers to perform realistic attack scenarios against different configurations of real equipment in a controlled environment [3]. ISEAGE is meant to simulate the proverbial Internet bubble on most topology diagrams, with all the eccentric traffic that comes with it. In essence, ISEAGE will provide many new tools for testing cyber defense mechanisms and even helping to solve cyber crime. As such, part of the design goal ISEAGE is to be a highly configurable environment than can model any aspect of the Internet [3].

The core of ISEAGE is a 64-node rack cluster that can be configured to support a variety of virtual networks. It is to be designed with multiple copies of each component such that configuration is scalable and can be changed quickly. Basically, each of the 64 nodes can run at least one of the tools/models and if needed, be divided up in order to run multiple experiments on separate virtual networks. This will also allow for multiple research experiments to be carried out at the same time in separate Virtual Subnets or for a single very large experiment to be carried out in a single or multiple Virtual Subnets.

المنارة للاستشارات
www.manaraa.com

Because ISEAGE is designed specifically for use in security research it offers many advantages over a conventional network testbed. The primary advantage is that the set of tools used with ISEAGE, such as ISEMasq, will be designed for the ISEAGE testbed. ISEAGE will contain a vast warehouse of attack tools that will be able to simulate point-to-point and distributed attacks. It can also be used to recreate critical components of an infrastructure on an ongoing basis. The goal is to develop algorithms and methods to harden the networks and computers used to support the critical infrastructure. These live models of the infrastructure can be subjected to constant attacks in order to probe for weaknesses. Additionally the models will provide a means of replaying attacks, recreating a cyber crime scene, and evaluating new attacks before they become a threat to actual infrastructure [3].

To better illustrate the development of ISEAGE, one can apply the layered Internet model discussed previously with a few additions. It is important to note that although this layered model does a good job representing ISEAGE, it does in some ways oversimplify some of ISEAGE's features. Layers 0 & 4 as seen below, are ISEAGE specific layers that effectively wrap around the Internet layer model and give a more practical testbed view. The resulting implementation is divided into five distinct layers where each one relies on the previous layers in order to function.

Layer 0: Command and Control

Layer 1: Routing

Layer 2: DNS & Traffic Endpoints

Layer 3: Traffic Generation & Attack Tools

Layer 4: Security Projects & Systems Under Test

المنارة للاستشارات

www.manaraa.com

From this layered illustration one can see that in order to simulate the Internet, ISEAGE needs to provide for the core functionality of the Internet. Layers 1 and 2 above represent these fundamental components and are essential for all the other layers to be effective and successful. In short, the completion of these components is essential to facilitating the ultimate goals of ISEAGE. Additionally, this core functionality needs to be able to handle the demand of the tools and test projects that will be run on it; because unlike computer-based simulations, ISEAGE will allow for real attacks will be played out continuously against different configurations of real equipment. While this work fulfills the Layer 2 needs as will be discussed in detail in later sections, an overview of the solutions for the other layers follows in the next section.

2.2 ISEAGE projects

There are a number of projects that have been developed to facilitate the completion of the core layers of ISEAGE. ISEMasq relates and/or interacts directly with each of the following tools.

2.2.1 DeepFreeze

Developed by Nate Karstens, DeepFreeze is the proverbial Layer 0 of ISEAGE. It was developed to facilitate a graphical command and control interface for the ISEAGE project. In addition to assisting the user in monitoring and controlling ISEAGE applications, it provides ISEAGE developers with an extensive application program interface (API) for authoring and controlling applications executing within the ISEAGE environment. DeepFreeze also provides a degree of fault tolerance to ISEAGE

المنارة للاستشارات
www.manaraa.com

applications and facilitates communication between the applications and their controlling interface [4].

In the simplest terms, a DeepFreeze-Daemon (DFD) runs on each of the nodes within ISEAGE, allowing the central DeepFreeze control console to configure and launch applications on each node as needed for a given experiment. The only caveat to this elegant solution is that in order for each tool or application to take full advantage of DeepFreeze, a graphical front-end must be written and then the tool needs to leverage the DeepFreeze API so that this front-end can communicate with and control the tool. Though this is a little more difficult for some of the older ISEAGE tools because they need to be reworked to use DeepFreeze, it should be fairly straightforward for future projects, and in the end will greatly improve many aspects of running experiments on ISEAGE.

2.2.2 Traffic Mapper

Currently, the Traffic Mapper, written by Dr Douglas Jacobson, handles the Layer 1 responsibilities for ISEAGE. The Traffic Mapper is in many ways the heart of ISEAGE, as it is responsible for mimicking the route topology. It is essentially a piece of software that can model the behavior and topology of routers allowing the creation of the Virtual Subnets. The Traffic Mapper can act as a set of virtual routers so that traffic appears to have routed through the Internet. It can simulate approximately 50 routers/routes per board that are then connected via the board backplane to create very large network topologies.

The Traffic Mapper will most likely be run on a majority of the ISEAGE nodes, as it is key to representing the topology. Then additional tools/models would be run on

المنارة للاستشارات
www.manaraa.com

the remaining nodes for that subnet, and the systems under test would then be connected to this mini Internet. While a graphical front end is planned for the Traffic Mapper, the functionality is complete and it is in limited use, awaiting the completion of more supporting tools.

2.2.3 Packet Mangling

The Advanced Packet Obfuscation and Control Program (APOC), also known to ISEAGE as the packet changer/responder, was developed in 2005 by Adam Hahn [5]. It can be used to modify packets in real-time as they flow through the network allowing for a wide range of possibilities, from performing man-in-middle attacks to simply testing protocol resilience by introducing random errors. Its use was at one time considered as a way to fill the need of endpoint masquerading since it is capable of many types of packet-level manipulation. However APOC was designed to facilitate much more generic and reconfigurable packet modification. This generic solution, while effective for the purposes of APOC, also suffers from many performance issues that limit its use as a large-scale endpoint masquerading tool. While APOC is designed to only tap select points in the ISEAGE network, for a variety of purposes, the design needs of ISEMasq are for a specific purpose and much more demanding in the area of performance as is discussed in Chapter 3. This distinction, though, in no way should reflect badly on APOC as it is a highly versatile tool that provides many unique capabilities to ISEAGE.

2.2.4 Traffic Generation

Traffic generation for ISEAGE is a multipart project that consists of a statistical model or generator and various mechanisms for feeding the model. At its center is the Markov Traffic Generator (MTG), which is an application-level traffic generation tool

المنارة للاستشارات
www.manaraa.com

that scripts network applications based on usage statistics of a specific or average set of users. Unlike many alternate approaches that operate at the packet level, it follows a unique approach of generating background traffic at the session level by leveraging various user applications [6]. As such, the MTG is application dependent and is able to generate various types of TCP traffic, representative of typical network traffic, in effect providing statistically governed user drones.

Despite these important features of the MTG, there are a couple key factors missing that limit its current usefulness to ISEAGE. First is how to get the statistics to feed to the MTG. While it is possible, and in some cases desirable, to hand feed the model with specific traffic statistics, it is also necessary to have realistic traffic patterns. For this reason, an additional tool currently in development called the traffic collector is needed to capture traffic patterns from the actual Internet at particular locations so they can be replayed within ISEAGE using the MTG. This tool will often be used to gather information about traffic patterns at a client's network, which are then used as the statistical parameters for the MTG so that ISEAGE can recreate those patterns.

The second limitation, and more important to this work, is the need for actual endpoints in order for it to fully function. Since the MTG works above the transport layer it needs actual destination servers to interact with in order to fully represent user behavior and resulting traffic.

2.3 Related Technologies

The following networking technologies and terminology are important to understand before discussing the design of ISEMasq, as it relies heavily on various aspects of each.

المنارة للاستشارات
www.manaraa.com

2.3.1 Routing

Routers represent the lowest functional layer of the Internet as discussed in Chapter 1 and are responsible for the source to destination delivery of packets across one or multiple networks [7]. Routing is often confused with bridging, which performs a similar function. The principal difference between the two is that bridging occurs at a lower level and is therefore more of a hardware function, whereas routing occurs at a higher level allowing for more complex analysis to be performed in order to determine the optimal path for the packet.

Routing is a key feature of the Internet because it enables messages to pass from one network device to another through the use of routers to eventually reach the target machine. Each intermediary device performs routing by determining the proper path for data to travel between different networks, and then forwarding data packets to the next device along this path. To determine this next hop, a router compares the packet's destination IP address to an internal lookup table and then sends it on its way; the IP addresses in the packet are never modified, as that is a feature more associated with Network Address Translation (NAT) [7]. As an aside, it is common today to find devices that can both route and perform NAT; conceptually though they are separate tasks. NAT is described in more detail in the next section.

2.3.2 Network Address Translation

Network Address Translation (NAT), sometimes referred to as IP masquerading, is a very important and relevant concept to the development of ISEMasq. NAT is described in RFC 1631, and extended by RFC 3022 [8]. RFC 3022 extends address translation introduced in RFC 1631 and includes a new type of network address and

المنارة للاستشارات
www.manaraa.com

TCP/UDP port translation. Most systems using NAT do so in order to enable multiple hosts on a private network to access the Internet using a single public IP address [9].

The term 'NAT' is commonly used to refer to a network device that performs the translation. The operation of a basic NAT capable device is deceptively easy to describe in general terms. It is an active unit placed in the data path, usually as a functional component of a border router or site gateway. In the simplest terms, a NAT rewrites the source and/or destination addresses of IP packets as they pass through. It intercepts all IP packets and may forward the packet onward with or without alteration to the contents of the packet, or it may elect to discard the packet.

The essential difference here from a conventional router or a firewall is the discretional ability of the NAT to alter the IP packet before forwarding it on. NATs are similar to firewalls, and different from routers, in that they are topologically sensitive. They have an "inside" or local area network (LAN) and an "outside" or wide area network (WAN), and they undertake different operations on intercepted packets depending on whether the packet is going from inside to outside or vice versa.

When traffic is processed from inside to outside it is referred to as an outbound NAT, and when its processed from outside to inside its an inbound NAT. A traditional NAT performs outbound NATing first and inbound NATing on the return traffic whereas a reverse NAT performs inbound first, and then outbound. This is an important distinction, as the direction that is NATed first sets up the address records for maintaining that connection.

In a traditional NAT, when a packet is being passed in the direction from the inside to the outside, a NAT rewrites the source address in the packet header to a different value, and alters the IP and TCP header checksums in the packet at the same

المنارة للاستشارات
www.manaraa.com

time to reflect the change of the address field [9]. When a packet is received from the outside destined for the inside, the destination address is rewritten to a different value, and again the IP and TCP header checksums are recalculated.

In a reverse NAT, traffic is initiated on the WAN and passed through the NAT device to an internal server. There are a couple types of reverse NAT. First is a one-to-one or DMZ host, in which all externally initiated traffic is sent to a single internal machine. The second type is more discriminate and in some cases referred to as port translation. In this approach the traffic is port forwarded to a specific internal machine and port number based on the destination port number in the originating traffic.

In both types of reverse NAT, when an incoming packet arrives on the external or WAN interface, the destination address is checked. If it is one of the NAT pool addresses, the NAT box looks up its translation table. If it finds a corresponding table entry, the destination address is mapped to the local internal address, the packet checksums are recalculated, and the packet is forwarded. If there is no current mapping entry for the destination address, the packet is discarded.

2.3.3 Domain Name System

The primary use of the Domain Name System (DNS) is to translate hostnames to IP addresses, making it the proverbial phonebook for the entire Internet. For example, in order to find the Internet address of www.iseage.org, DNS can be used to determine that it is 129.186.207.10.

The domain name space consists of a tree of domain names. Each node or leaf in the tree has one or more resource records, which hold information associated with the domain name. The tree sub-divides into zones. A zone consists of a collection of

المنارة للاستشارات
www.manaraa.com

connected nodes managed by an authoritative DNS server. DNS distributes the responsibility for assigning domain names and mapping them to IP networks by allowing an authoritative server for each domain to keep track of its own changes, avoiding the need for a central registrar to be continually consulted and updated [10].

For example, .edu would be a root server node, and under it would be iastate, uni, and even uiowa servers, to name a few. Then at each university there could be engineering, business, or agricultural sub domain servers. These in turn could be broken down even further as needed. The end result is a URL like ece.eng.iastate.edu. For someone at another university to look this up, (and assuming all the cache is cleared) their computer would contact its DNS server, which would eventually contact the .edu root, which would ask the iastate node and on down to the lowest level DNS server that actually has the IP address needed to allow communication with the Electrical and Computer Engineering department's web server. The main thing to take away from this is that once the initial request is made to the user's immediate DNS server, the rest happens behind the scenes, with the user only being aware of the time it ends up taking to get a response.

This however is only a small example; the entire DNS architecture is even more complex, and most of the related details are beyond the scope of this work. What is important, however, is the functionality that is provided to the user and not so much how it is provided. To this end, one of the goals of ISEMasq is to mimic the functionality of DNS at the user level without the complicated architecture.

المنارة للاستشارات

www.manaraa.com

3 Design Overview

3.1 Goals

The main goals of ISEMasq can be briefly summarized into functionality, performance, and flexibility. From a general perspective, ISEMasq needs to incorporate the functionality of routing, inbound NAT, and DNS. In order to best accomplish this, ISEMasq will be divided into two components: a virtual DNS endpoint and a virtual endpoint gateway. The functional requirements of these two components need to be as follows:

1) The virtual DNS endpoint should be a scalable, demand-driven custom DNS server for ISEMasq. While it needs to perform common DNS functionality in the form of user queries, the backend will be anything but a traditional DNS server. The name records will not exist as usual nor will they have the same temporal requirements.

2) The virtual endpoint gateway functionality should masquerade a small set of endpoints as many. As such, ISEMasq's design will be specialized for this task and will be implemented at the IP layer so as to have no more impact on the network throughput than either a router or NAT. Also, unlike NAT, it will do port forwarding ubiquitously instead of based on the destination IP.

The virtual endpoint gateway component of ISEMasq will be very similar to a reverse NAT but will accept all traffic regardless of the destination IP address as long as the destination MAC is correct, much like a router. As such, it should have two

المنارة للاستشارات
www.manaraa.com

interfaces, a WAN and a LAN, much like a NAT, with the WAN interface being the side connected to ISEAGE.

In addition to being able to handle both UDP and TCP traffic types, the traffic going to and from the WAN interface of ISEMasq needs to remain realistic from the transport layer on down. If one were to capture the traffic at the WAN interface it should appear to be heading to numerous different destinations, even though it is condensed to only a few actual endpoints on the LAN side.

Since ISEMasq will be directly connected to the Traffic Mapper as the next hop in the traffic route (as illustrated in the next section), it needs to handle at least as much traffic as a single Traffic Mapper is capable of handling and do so in near real-time. Specifically, ISEMasq is designed to masquerade as a border router and all subsequent endpoints within; therefore it needs to be able to handle the volume of traffic characteristic of all the endpoints it is simulating. For this reason, it's easy to see how performance is not only crucial, but in fact critical to ISEMasq.

While the DNS and gateway functionality of ISEMasq are in many ways separate components, they will be combined into a single tool. By doing so and providing three running modes, it actually increases the flexibility and effectiveness of the tool. Having a single application will allow for simpler command and control while maintaining the intended functionality. The three run modes are as follows:

1) DNS Endpoint Only
2) Endpoint Gateway Only
3) Both 1) & 2)

المنارة للاستشارات
www.manaraa.com

In other words, when running a large-scale experiment on ISEAGE, the researcher will not have to have to configure separate tools, but one. Since the endpoints and DNS are network dependent on one another, one can't practically exist without the other anyway. For example, first consider ISEMasq is running in Mode 2. If URL-based communication with the endpoint(s) is desired, then somewhere on the ISEAGE network ISEMasq will have to be running in either Mode 1 or 3. Conversely, if ISEMasq is running in Mode 1 and replies to IP traffic are desired, then an endpoint is needed somewhere on the network. Additional topology flexibility from offering two modes of operation can be seen in the next section.

3.2 ISEAGE Integration

Starting with the network topology layer that is provided by the Traffic Mapper, it is now worthwhile to discuss how ISEMasq works with it to provide other tools or test systems plugged into ISEAGE the destinations they need to connect to, as discussed in the Introduction. Like the other tools of ISEAGE, ISEMasq will run on one of the many ISEAGE nodes and process all IP traffic that is sent to it, regardless of the destination IP. To this end, ISEMasq will rely on the Mapper and its associated topology to route traffic to it. For example, ISEMasq could be connected to the default route of the Traffic Mapper(s), or any other subnet endpoint that exists at the edges of the Mapper topology.

From the Mapper's perspective ISEMasq will simply be seen as another router, though once the traffic reaches ISEMasq it is processed in one of two unique ways depending on the run mode. First, if the traffic is DNS and the DNS mode is enabled, then a lookup is performed and a response is generated. Otherwise, the traffic is processed very similarly to an inbound NAT and forwarded to a designated server

المنارة للاستشارات
www.manaraa.com

connected behind ISEMasq. In both cases, the traffic is essentially condensed to a single set of servers and services behind ISEMasq without the ISEAGE network at large being aware of it. To illustrate this setup as well as other possible ways to integrate ISEMasq into the ISEAGE network, consider the following simple topology given in Figure 3.1.

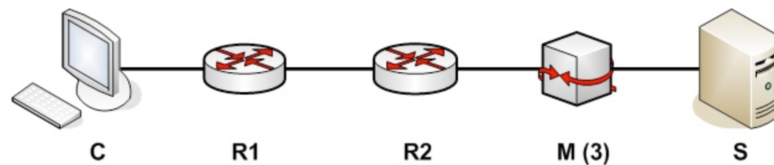

Figure 3.1: Small Toplogy

Here R1 and R2 are routers, C and S represent a Client and Server pair and M is ISEMasq running in Mode 3. When placed between R2 and S, ISEMasq simply needs to be treated as a router would, such that the next hop for both R2 and S point to M instead of each other. With this simple topology example in mind, now consider what an actual Internet topology might look like in Figure 3.2 below.

Notice that in Figure 3.2 that there are two instances of ISEMasq, one as just a gateway (Mode 2) and one as just DNS (Mode 1). While it is possible to combine them, as will be discussed later in this section, this helps illustrate the variety of integration options. Also note that ISEMasq could be connected to any router in Figure 3.2 with the exception of the red router, marked R0. This is due to redundant route paths around R0, so that if ISEMasq were hooked to it, other connections would be left disconnected and in need of rerouting.

المنارة للاستشارات
www.manaraa.com

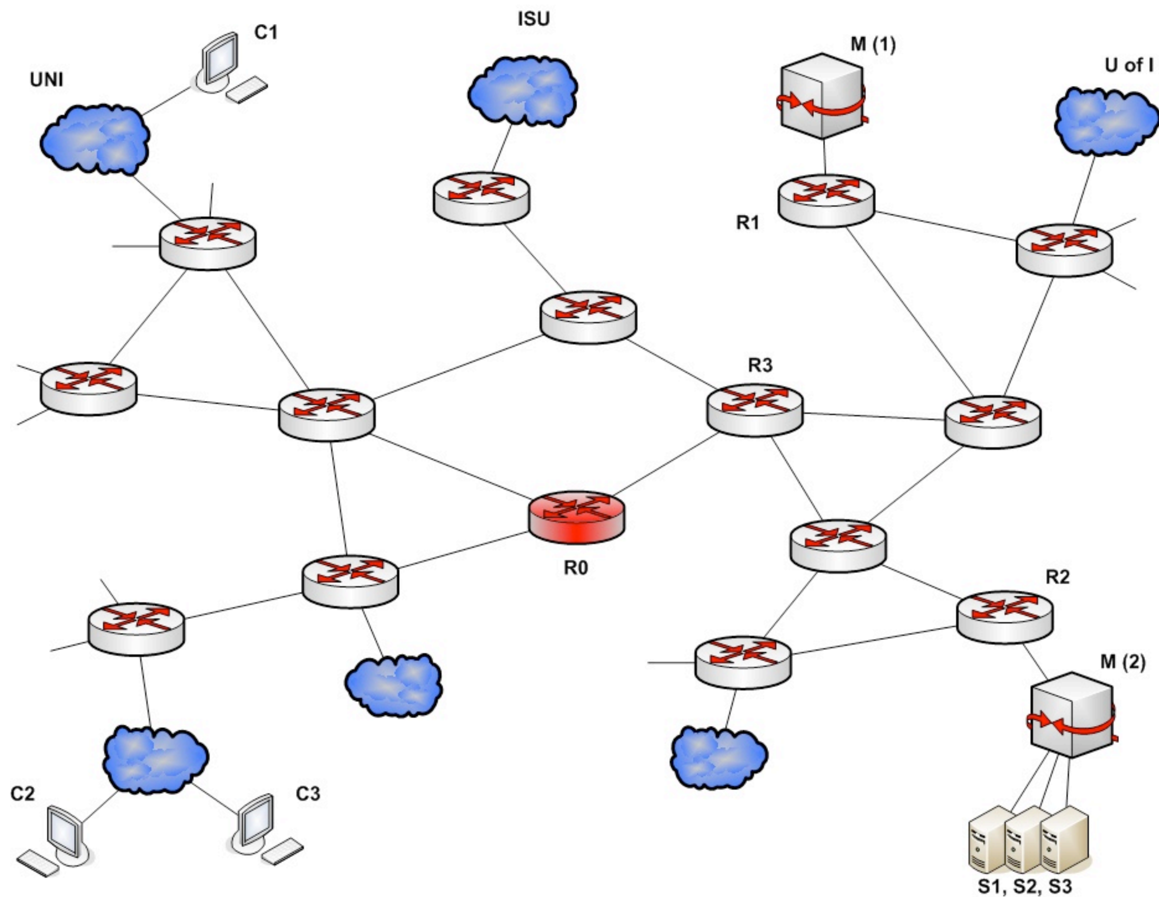

Figure 3.2: Large Toplogy

In both of the above examples, ISEMasq is essentially acting as the border router for the destinations that it is masquerading for. When inserted into a topology to act as said destinations, one of two setup approaches can be taken. First, one could simply modify the next hop of the first upstream router (i.e. R1 in both examples) to point to the IP address of ISEMasq instead of the address of the original border router. Or, one could just use the original router's IP address on the WAN interface of ISEMasq, thus eliminating any modifications to R1.

Since ISEMasq was specifically designed to create the illusion of many servers and routers existing on a network it helps make it possible to simulation of large network

المنارة للاستشارات
www.manaraa.com

topologies without requiring the footprint of a large topology. In other words, not only can ISEMasq reduce the number of actual endpoints needed for a simulation, it can also reduce the number of route points leading to a set of destinations. Going back to Figure 3.1, assuming for the moment that topology is not as important (as it might be in some experiments), it becomes possible to reduce the setup to its simplest form as seen in Figure 3.3. When this same reduction is done to the more realistic Internet topology (Figure 3.2), the resulting topology, although functionally the same, is now much simpler to model as seen below in Figure 3.4. Notice that the two instances of ISEMasq were condensed while maintaining their functionality by running a single instance in Mode 3.

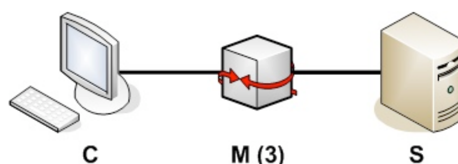

Figure 3.3: Small Topology Simplified

Regardless of the location in the route path, ISEMasq will always have the WAN interface toward the client and the LAN toward the server. The WAN is labeled as such since all traffic off that interface is as it would be on the actual Internet, i.e. any and all IP addresses are expected to be seen there. The LAN interface, on the other hand, only expects to see the IP addresses of the servers, which will most likely be on the same subnet. While it is possible to put multiple servers on multiple subnets and add an alias IP for each on the LAN interface, it is not necessary to the functional performance of the ISEMasq, and in fact just over-complicates the setup.

المنارة للاستشارات
www.manaraa.com

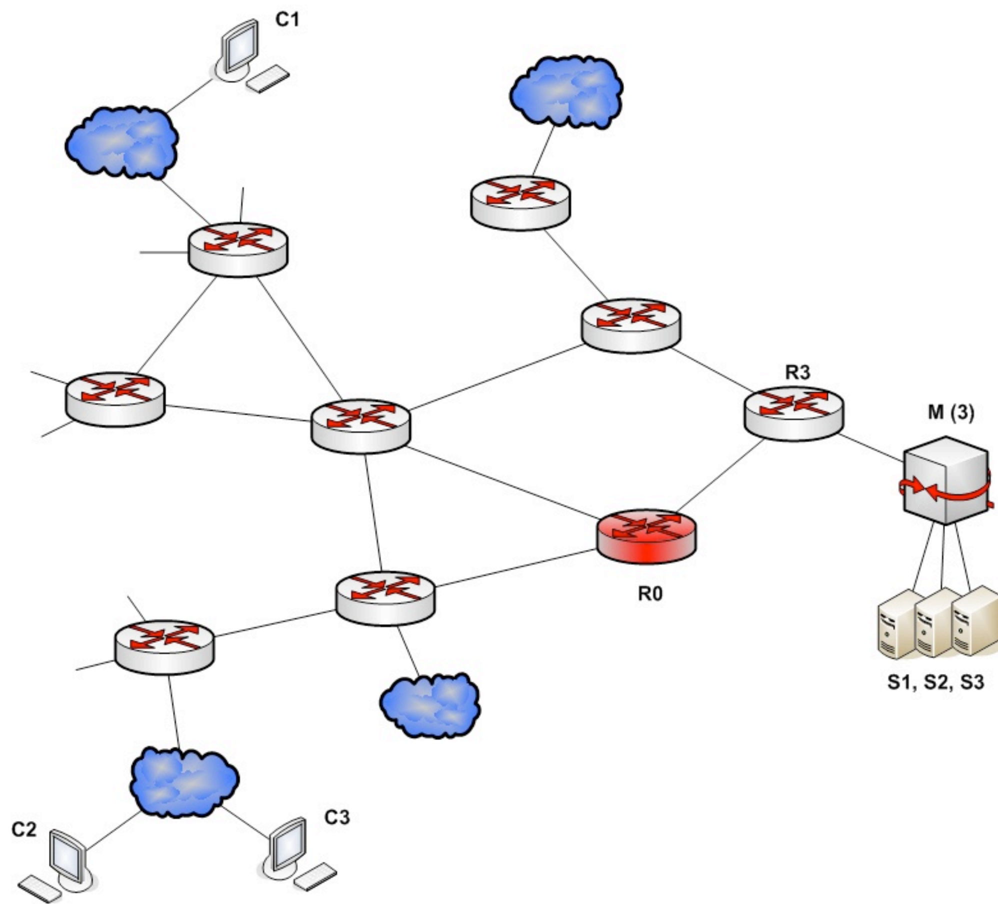

Figure 3.4: Large Topology Simplified

Finally, ISEMasq was designed so that the server(s) behind it could be any off-the-shelf solutions. One could even run ISEMasq in Mode 2, and then add an actual DNS server to the Server pool, maybe even with caching or forwarding to the real Internet. It would even be possible to run some sort of web-proxy service, intrusion detection system (IDS), or any other specific type of server platform for development and testing; the possibilities are really quite vast. See Section 5.3 for even more ideas.

المنارة للاستشارات
www.manaraa.com

3.3 Assumptions

During the design phase it became clear that much of the functionality and performance of ISEMasq would depend on the backend data structure needed for tracking unique connections. Since this connection mapping of the traffic is the most critical and resource-intensive part of the design, many of the assumptions largely revolve around the underlying data structure.

The first of these is the assumption that all client OSs that initiate connections though the gateway will behave in a typical manner, such that they will use a new socket and source port pair for each outgoing connection to a specific destination. Failure of a client to do so will result in the database losing track of where the connection originated. Furthermore, and more importantly, it is assumed that the client does not reuse the same socket to connect to a different destination until the database has flagged the old connection as stale and purged it.

Secondly is the assumption that there will be no need for long-standing connections without data flow. In this sense it means that the state of the connection information in the database has limited state, and will quickly timeout when idle. This is due to the fact that the virtual endpoints are to mainly serve as hosts that the application-level packet generation model (e.g., the MTG) or attack tools can bounce traffic off of.

Finally, the worst-case scenario is considered: a very large SYN scan. To the software, this could have the effect of a denial-of-service attack because it would increase the size of the connection database quite rapidly. While there is a timeout built into the database, it can't be very aggressive since it would destroy many normal connections. To address this, a database limit was added. Basically, in addition to checking the database for stale entries as laid out in the next section, the program should additionally check that

المنارة للاستشارات
www.manaraa.com

the database hasn't exceeded its maximum size. If it has, then and only then are entries that have not reached their maximum lifetime overwritten.

المنارة للاستشارات
www.manaraa.com

4 Architecture

In general, ISEMasq consists of various API functionality from the current and upcoming release of the ISO C++ standard, as well as other widely used and accepted third-party C and C++ networking APIs. The resulting architecture not only has programming framework considerations, but also other logistical considerations relating to functionality and how it compares to alternative approaches.

4.1 Programming Framework

ISEMasq was designed and written in C++ with a few C-based APIs. The use of the C family of computer languages allowed for the low-level control and speed needed as well as providing access to high-quality libraries. These libraries helped facilitate development, resulting in fewer bugs, reduced reinvention-of-the-wheel, and lower long-term maintenance costs. This section discusses the main libraries and APIs used in the design and development of ISEMasq. These APIs were instrumental in the development and include libpcap, libnet, the Standard Template Library and the Boost library.

4.1.1 Libpcap

The libpcap API provides a framework for capturing packet traffic from the network. Many different open source tools use this library in their development including TCPdump and Snort [11]. Using libpcap to capture traffic is commonly referred to as sniffing. Often times a sniffing application may only be interested in specific traffic, such as telnet, FTP, or DNS. Whatever the case, it is rarely useful to just blindly sniff all network traffic. Fortunately libpcap provides the developer a unified, easy-to-use interface for filtering the traffic it sees. And, because libpcap's filters interact behind the

المنارة للاستشارات
www.manaraa.com

scenes directly with system specific filtering mechanisms, it is far more efficient than filtering traffic manually, which involves numerous steps. Because of this performance and its solid API, libpcap is a perfect fit for the incoming communications needs of ISEMasq, and when combined with libnet, enables the bidirectional flow of traffic that ISEMasq must provide.

4.1.2 Libnet

Libnet is a high-level API that allows the application programmer to construct and inject network packets. It provides a portable and simplified interface for low-level network packet shaping, handling and injection [12]. As such it was a perfect fit for the development of ISEMasq especially when combined with libpcap.

The libnet library has a number of injection modes that it can run in depending on what level of injection control over the network stack is needed. The two options for the Gateway component of this project were the advanced modes of the standard LIBNET_RAW and LIBNET_LINK modes, or LIBNET_RAW_ADV and LIBNET_LINK_ADV, respectively. The advance modes are necessary since libnet is not being used to build the packet, but merely to write out the ones modified from the libpcap capture. By initializing libnet in advanced mode it is then possible to use the libnet_write_raw_adv() function call is used instead of libnet_write(). This variant of the libnet_raw_adv() injection call is only available in version 1.1.3+.

The DNS component on the other hand used LIBNET_RAW instead, since the reply packets had to be constructed mostly from scratch, and the regular mode provides functionality that nicely assists the packet creation process at each network layer. If either the standard or advanced version of the LINK mode were used for either component, then

المنارة للاستشارات
www.manaraa.com

ISEMasq would also need to have code to handle address resolution protocol (ARP) in order to fill in the Ethernet header. It is therefore preferable to use LIBNET_RAW_ADV for sending IP packets, as it sends the data through the kernel, which uses the system's routing function to determine proper link-layer encapsulating headers.

4.1.3 Standard Template Library

The Standard Template Library (STL) is a software library that was designed and published by SGI and is partially included in the C++ Standard Library. Both libraries include some features not found in the other [13]. Though since ISEMasq uses ISO C++ as it base language, the following discussion will only be concerned with its implementation of the STL, which is a subset of the original SGI STL.

The design of ISEMasq hinges on the ability to quickly and reliably store and retrieve connection information. Now, while it would have been possible to write custom data structures from scratch, they would have to be extensively tested to ensure performance, reliability, and data consistency. The STL library not only satisfies these concerns, but it is highly flexible due to its template-based design and its additional library of STL algorithms.

Of the many available STL containers, ISEMasq relies extensively on the map class, which can be conceptually thought of as an "associative array" in which key values are associated with corresponding values. Maps are based on the red-black tree data structure, and guarantee O(log n) insertion and lookup time [13]. Additional details on the use of this and the other structures used can be found in the Chapter 5.

المنارة للاستشارات
www.manaraa.com

In addition to being designed with STL data structures, the back-end database is encapsulated in a separate C++ class; see Appendix A for the class header file. This code modularity will aid making further improvements to the back-end in two ways:

1) As is standard for object-oriented design, the implementation of the class can be easily changed without affecting the interface and hence any of the Main code.

2) Since the implementation relies on the STL, changing the underlying data structures would require very little code rewrite. Ideas for what to change are discussed in more depth in Chapter 6.

4.1.4 Boost

Boost was started by members of the C++ Standards Committee Library Working Group and has since expanded to include thousands of programmers from the C++ community at large with a license that encourages both commercial and non-commercial use [14]. Adobe, Real Networks, and McAfee are a few of the big name organizations that use the Boost libraries [15]. The Boost Group emphasizes libraries that work well with the C++ Standard Library. In fact, ten of the Boost libraries are included in the C++ Standard Library's Technical Report 1 (TR1), and so are slated for later full standardization. TR1 is a draft document specifying additions to the C++ Standard Library, and though it is not yet standardized, will likely, as it stands now, be part of the next official standard due out by 2009. More Boost libraries are in the pipeline for TR2. For these reasons, using Boost libraries can give ISEMasq a head start in adopting new technologies.

المنارة للاستشارات
www.manaraa.com

While the Boost library offers dozens of various libraries, two in particular were of use in ISEMasq: Boost program options and Boost threads. The Boost program options library provides ISEMasq with an object oriented API for creating and parsing program parameters from either the command line or a configuration file [16]. The Boost threads on the other hand were used to provide the concurrent traffic processing needed for ISEMasq to handle full duplex communications.

There are several advantages to using Boost libraries instead of existing C variants, especially since C requires careful use in C++. For instance the Boost thread interface was designed from the ground up and is not just a simple wrapper around any specific C threading API. Many features of C++, such as the existence of constructors/destructors, function objects, and templates, were fully utilized to make the interface more flexible. The resulting implementation currently works for POSIX, Win32, and Macintosh Carbon platforms [17]. In short, the use of Boost libraries adds interface flexibility to ISEMasq where it otherwise wouldn't exist and facilitates developmental consistency by keeping as much of the project based in C++ as possible.

4.2 General Considerations

With the programming framework set up, it is necessary to turn the focus towards other architectural considerations that were encountered during the development of ISEMasq. These include what type of network device ISEMasq will mimic and the ability of multiple threads to simultaneously access the same network traffic.

المنارة للاستشارات
www.manaraa.com

4.2.1 Network Device Type

There are two main types of network devices: active and passive. Most network devices act as active nodes on a network, meaning that they often have their own IP address and are visible to the other devices on the network. In this case a device's NIC (network interface card) operates normally by checking the destination MAC address of the Ethernet frame against its own and if it matches, saves the frame for further processing by the operating system, or else it discards the frame. Passive devices on the other hand, such as switches and network taps, are transparent, meaning that they are essentially invisible to the other devices on the network. The main purpose of a transparent device is to monitor or control traffic without other devices needing to know of its existence.

Since ISEMasq is only concerned with traffic that is sent specifically to its MAC address, like most routers, it is better suited as an active device. However, even though ISEMasq need not be aware of every packet that a passive device sees, it does need to use the same mechanism for listening to the wire. The reason for this is that most active devices use system calls based on sockets and get their access to the network through various mechanisms in the kernel. These mechanisms allow the network application to read and write traffic only to or from the IP address that the kernel has associated with the network interface. Given that ISEMasq is required to handle all IP traffic regardless of IP address, socket-based design just won't work. A potential caveat to this are divert sockets in the FreeBSD system which according to the man page may provide a socket-based program the ability to deal with unassociated IP addresses [18]. Instead, ISEMasq uses libpcap and libnet as previously mentioned to read and write packets to the network.

المنارة للاستشارات
www.manaraa.com

While ISEMasq uses the same libpcap listening mechanism that some passive devices use, it does not do packet capturing in promiscuous mode. This has an added benefit because there is no need for additional filtering code, thus overhead is reduced and the rate of data that the box can handle is increased. Instead of having to keep track of the packets that are written out in order to filter them from looping back in, all that is needed is a simple Ethernet destination filter applied to libpcap. So, if the destination host is equal to the ISEMasq interface that captured it then it is processed, otherwise the packet is ignored, letting the kernel handle it if so desired.

4.2.2 Threads and Packet Capturing

From a data flow perspective, the gateway function of ISEMasq effectively bridges two network interfaces together such that all traffic from one is forwarded to the other. In the software world this bridging can be done using C libraries such as libpcap and libnet. To speed up such an implementation, threads can be used. In Figure 4.1, Thread 1 is constantly capturing (via libpcap) from the WAN interface (eth0) and writing (via libnet) to the LAN interface (eth1), as Thread 2 simultaneously reads from eth1, and writes to eth0.

المنارة للاستشارات

www.manaraa.com

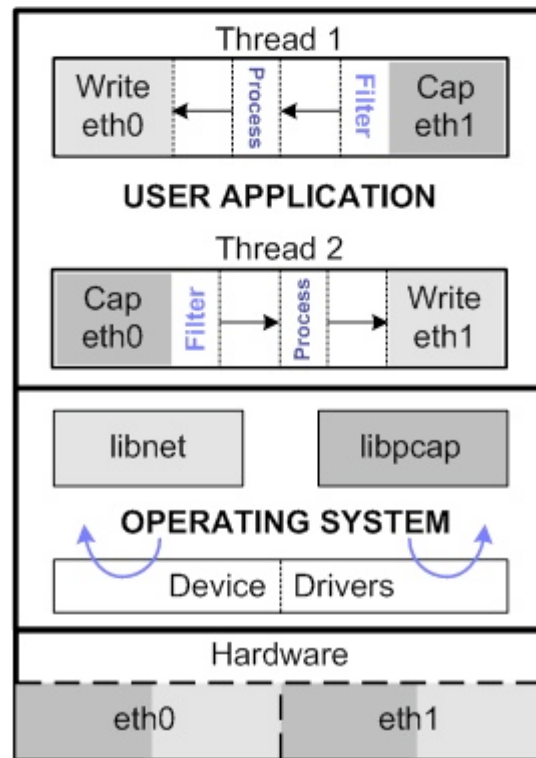

Figure 4.1: Overall architecture and data flow of a typical software approach.

While this approach is effective, a problem arises when the design is expanded to run multiple threads that need information from the same interface. Specifically, when ISEMasq is run in Mode 3, both DNS and gateway need to listen for incoming traffic on the WAN interface (i.e. eth0). In this scenario, the design needs to provide for a way to allow two separate software components to simultaneously process data from the same interface efficiently. It was decided to have each of the two components run a separate libpcap capture on the WAN interface. The alternative would have required a manager and worker thread design, which would have required more user-land processing as well as additional shared data structures and corresponding mutexes. Instead, the simultaneous capture approach allows for the main part of the program to simply spawn off a thread (or set of threads) for each mode that's activated. Not only does this simplify the code

www.manaraa.com

organization, but it also speeds up packet processing as can be seen by looking at the libpcap internals.

The libpcap packet capture library is optimized for use with the Berkeley Packet Filter (BPF) on UNIX-based platforms. The BPF not only provides a very effective mechanism for capturing traffic it also provides built-in filtering capabilities that happen in kernel space. So, in addition to leveraging the BPF for performing the actual packet capture, libnet provides a filter compiler for the BPF pseudo-machine code. On most systems supporting it, a kernel-resident BPF implementation processes the filter code and applies the resulting pattern matching instructions to received frames. Those frames matching the patterns are received through the BPF machinery; those not matching the pattern are otherwise unaffected [19].

When a packet filter instance of BPF is created, it is bound to an actual network interface such as eth0 or eth1 and shows up as a special device such as bpf0, bpf1, etc. On a FreeBSD system, a given interface can be shared by multiple BPF instances, and the filter underlying each descriptor will see an identical packet stream [20]. Whenever a packet is received by an interface, all BPF descriptors listening on that interface apply their filter and each descriptor that accepts the packet receives its own copy. This allows ISEMasq to simply create multiple BPF instances on the WAN interface, with each BPF instance seeing and filtering the incoming data in the kernel before it is passed to the respective thread within ISEMasq for further processing.

4.3 Alternate Approaches

During the design process, various alternative software approaches were considered for the DNS and gateway components of ISEMasq. As the following

المنارة للاستشارات
www.manaraa.com

discussion shows, they are not as well suited for the design goals as the programming architecture and framework included in ISEMasq and already discussed.

4.3.1 Gateway

The default firewall software that comes with FreeBSD is known as PF or PacketFilter [21] and is similar to IPF/IPNAT, IPFW/NATd, and IPTables. In conducting initial research during early development of ISEMasq, it was discovered that PF could provide functionality very similar to that of the gateway portion of ISEMasq. All that is needed is to enable PF and the gateway options in FreeBSD's system configuration, then add one or more rules in the PF configuration file [21]. A very simple example of such a file would look like this:

```
rdr pass on eth0 proto tcp from any to any port 1234 -> $Dest_Server_IP
```

Considering that PF is designed and written to run in kernel-space, this seems like a quick and effective solution, and maybe in some environments it would be fine, but in the ISEAGE world this is far from sufficient. The most obvious drawbacks compared to ISEMasq are that it does not have the DNS capabilities as well as it would be very difficult to integrate with DeepFreeze since the DFD is designed to interact with user executables. Therefore, in order to integrate a PF-type solution with ISEAGE, a custom application or tool will still need to be written in order to allow the DeepFreeze console to control and configure it. The tool would run on one of the nodes just like ISEMasq and would have to integrate with the DFD so that it could then configure and restart PF based on some configuration settings. Also as mentioned, since PF requires a separate rule to be

www.manaraa.com

generated for each connection mapping desired, configuring it would be more involved than the simple configuration file that ISEMasq is able to provide

Additional drawbacks of PF include fault tolerance, scalability, and customization. Even though PF runs in kernel-space, if it crashes it could potentially crash the system. Also, since its functionality is based on a firewall, it could interfere with other applications, especially network-dependent ones such as the DFD. Regarding scaling, a separate rule would need to be added for each destination port to be handled, where as ISEMasq was designed to use a single set of logic to map ports regardless of the number of different mappings.

Finally, even though PF is capable of performing many types of packet filtering and redirecting, what is required for this task is but a small and uncommon use of it. ISEMasq, on the other hand, has been designed specifically for its task, enabling it to provide additional features that PF cannot (see Chapter 5). Because of this specialization, initial testing using the worst case of an nmap scan showed that ISEMasq is able to perform as well as PF, even though much of it runs in user-space.

4.3.2 DNS

Developed by the Internet Systems Consortium (ISC), BIND (Berkeley Internet Name Domain) is used on the majority of name serving machines on the Internet [22]. Attempting to provide the DNS functionality of ISEMasq with software such as BIND presents significant hurdles. First, one would need to make sure that all the DNS traffic makes it to the BIND service regardless of the actual destination IP address. This could be accomplished by either aliasing all possible IP address on the interface or by using PF to redirect all incoming traffic to a specific location such as the localhost. Neither of these

المنارة للاستشارات
www.manaraa.com

is very feasible for two reasons: 1) it is impossible to alias all IPs, and 2) any PF solution would be very difficult to integrate into ISEAGE as discussed previously in Section 4.3.1.

Now, even if the hurdles of using BIND can be overcome to this point, there is still the issue of providing responses to the lookups. Since BIND would be running without the DNS infrastructure at large to facilitate name resolutions, an all-inclusive lookup cache would have to be created. Such a cache would be incredibly large and nearly impossible to manage properly.

By developing a custom solution such as ISEMasq, all the issues mentioned above are avoided. First of all, ISEMasq uses low-level network APIs such as libpcap to make sure that it has access to all the traffic coming into the interface allowing it to make the decision if the traffic is relevant. Second, the DNS records are not predefined, but rather created on the fly and then simply timed out if not re-queried after some time. To ensure that the records that have been requested are not held on to past this timeout, the TTL option in the DNS message packets is set to a value that is less then the timeout. If a specific record is re-queried before the timeout has expired, then the timeout value is simply updated and counting starts again.

المنارة للاستشارات

www.manaraa.com

5 Implementation

Based on all the design goals and architecture decisions described previously, the final implementation form of ISEMasq consists of a main program, a DNS endpoint thread, two virtual endpoint gateway threads, and a back-end database class. These components interact as seen in Figure 5.1.

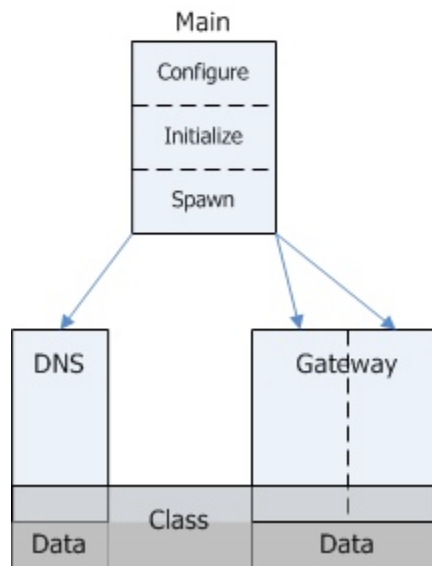

Figure 5.1: Implementation Overview

The main modestly parses the user-input configuration options, both from the command line and configuration file (see Appendix B), then initializes the custom data class, initializes libnet and libpcap on the appropriate interfaces to include necessary traffic filters, and spawns the corresponding threads to handle the packet flow. The libpcap filter setup and thread spawning is dictated by which run mode is selected by the user and is very important here as it determines what traffic the gateway the DNS components see and process. The implementation of each major component of ISEMasq is examined in more detail in the following sections.

المنارة للاستشارات

www.manaraa.com

5.1 Virtual Endpoint Gateway

The virtual endpoint gateway component consists of two threads and three back-end data structures. Looking at the data structures first, they consist of two STL maps and one STL queue. The maps have the following < key , value > pairs:

Internal Destination < port number, IP address >
Current Connections < IP-tuple, custom struct >

And the queue has a single < value > layout:

Stale Connections < STL iterator >

The value of the key in the Current Connections data structure is the connection ID and is derived from the source IP, source port, and destination port, also known in networking as the IP-tuple, resulting in a 64-bit value that is unique to the connection. While the elements of the map are stored based on this key value, the queue elements are stored in order of the creation time of the corresponding map elements that they point.

Instead of having a single monolithic data structure for tracking connections, it was decided to use two smaller dynamic data structures that can be rapidly changed: one for the data entries and one for timestamps to identify stale entries that need deletion. Having two separate data structures for connection tracking effectively separates the database insertion operation from the purge operation. This in turn provides better performance as each structure can be chosen to optimize for its specific data access needs. The map provides O(log n) insertions based on the connection ID while the queue provides O(1) additions and deletions based on the connection timestamp.

The two threads comprising the virtual endpoint gateway are responsible for processing traffic from the WAN interface to the LAN interface (inbound traffic), and

المنارة للاستشارات
www.manaraa.com

from the LAN to the WAN (outbound traffic) as can be seen in Figure 5.2. It is important to note that the inbound thread is the only one modifying or writing to the data structures, as the outbound thread merely reads data. Carefully controlling access to the database as such helps reduce complexity and possible data corruption, resulting in better performance and reliability.

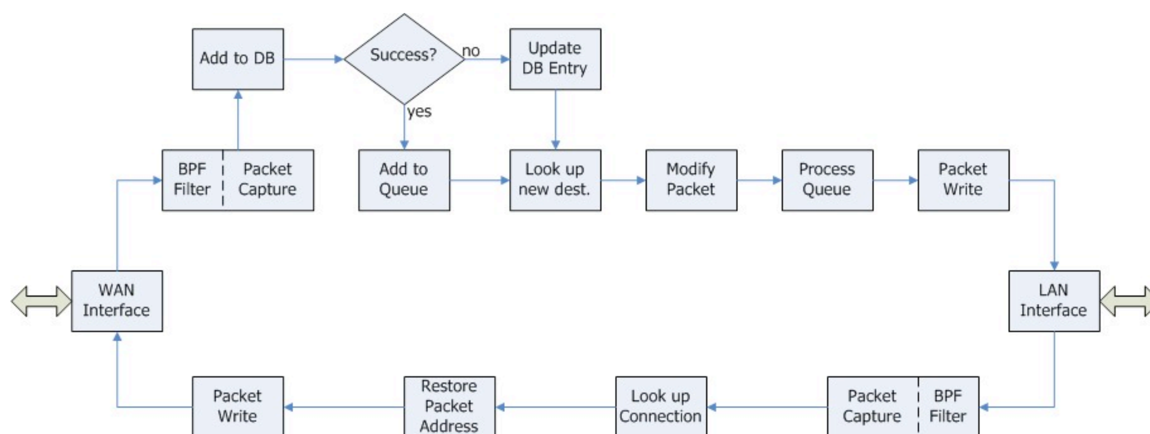

Figure 5.2: Gateway Flow

With this flow in mind, it is worthwhile to examine the endpoint masquerading process in a little more detail by taking a look at the processing of each thread individually starting with the incoming traffic. Once traffic has been captured and filtered on the WAN interface it is processed as follows:

- Get the current time
- Generate the connection ID
- Attempt to add the ID to the database
- If the Add succeeded, add a corresponding entry to the purge queue
- Else if the Add failed, update the time stamp of the existing connection

المنارة للاستشارات
www.manaraa.com

- Use the destination port to lookup the new destination IP
- Modify the packets destination IP and TTL, then fix checksums
- Check next queue entry for expiration
- If stale, then remove
- Else add to back of queue
- Write packet out to LAN interface

Since much of the work needed for the masquerading process is handled by the inbound thread, completing the process on the return or outbound traffic as it is captured and filtered on the LAN interface is simply as follows:

- Generate the connection ID
- Use ID to lookup the original destination IP
- Restore the packets original addresses
- Write packet out to WAN interface

With the packet processing of the gateway component clearly outlined, it's time to move on to the DNS component, which ensures that Internet applications are able to look up ways to get to the gateway.

5.2 Virtual DNS Endpoint

The virtual DNS endpoint serves the purpose of providing a more realistic gateway in ISEMasq by allowing the use of URLs instead of IP addresses. This component also has three back-end data structures and is very similar in data and network flow to the virtual gateway endpoint component except that it only consists of a single

المنارة للاستشارات
www.manaraa.com

thread. Again considering the data structures first, the two STL maps used have the following < key , value > pairs:

Standard Lookups < URL, IP Address >

Reverse Lookups < IP Address, STL iterator >

And the queue has this < value > layout:

Stale Records < STL iterator >

All three structures will always be the same in size, since the reverse lookup map and stale record queue are cross-linked against the standard lookup map via STL iterators. This linking, combined with the custom structure that contains data, such as the timestamps, makes up the standard lookup map the main data structure for this component, much like the current connections map in the gateway component. With these structures in place, it is time to discuss the process of handling incoming DNS requests. While the code needed to actually parse and construct DNS message packets is a little tricky, the general process overview of this component can be seen here in Figure 5.3.

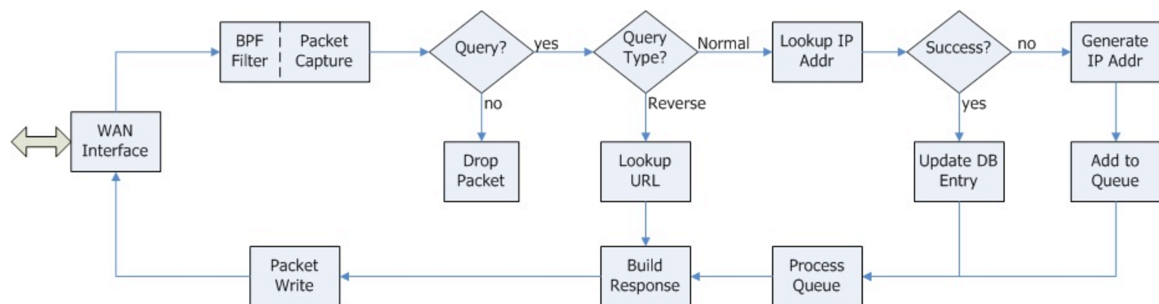

Figure 5.3: DNS Flow

المنارة للاستشارات
www.manaraa.com

With this flow in mind, it is worthwhile to examine the DNS masquerading process in a little more detail by taking a look at the processing of individual DNS requests. Once traffic has been captured and filtered on the WAN interface, the thread checks the type of DNS message. If the message is a query, it attempts to look up the appropriate reply based on the type of query and then builds and transmits a response message. If the lookup is successful then the timestamp for the corresponding entry is updated. In the case that the lookup does not find an existing entry, then one is dynamically generated and added to the standard lookup map. Following this, a corresponding entry is also added to the back of the stale entry queue. These on-demand record generations can have various constraints; the most common being the allowed range of values from which an IP address can be picked, as will be discussed in the following section. Though this description of the DNS component does not discuss the many nuances of the DNS message format and its parameter specifics, the overall concept is fairly straightforward and effective.

5.3 Additional Features

There are several additional features built into ISEMasq that perform functions ranging from providing feedback to controlling program flow. At present there is an additional thread that runs every few seconds for the sole purpose of displaying connection and database statistics. In the future this should be integrated with the proposed GTK+ front end as discussed in more detail in Chapter 6. Also, ISEMasq has the ability to control which subnet(s) the DNS endpoint is allowed to host lookups for. The range can be all routable IP's or a specific class A, B, or C subnets, depending on the constraints of the simulation.

المنارة للاستشارات

www.manaraa.com

Another feature in ISEMasq is the destination flow control, which consists of two parts. First is the default destination option, which if set sends all traffic from ports that aren't specifically mapped to the default server, much like a DMZ NAT mapping. If left unset, the software will randomly cycle through the existing destination mappings each time an unknown destination port is detected.

The second part of the destination flow control allows for multiple destinations (servers) to operate on the same port, i.e. provide the same service. In this case it would, for example, be possible to have multiple web servers behind the gateway. If enabled, the software then spawns off a thread that will periodically (e.g. every minute or so) change the mapping for that service, in this case port 80. In addition to load balancing, this also allows for session-level content to vary dynamically over time.

Finally, though not designed with this in mind, ISEMasq can be hooked up in reverse to allow for some interesting traffic control and honey potting. If ISEMasq were to be hooked up to a wireless access point (AP) as its default route, then it is possible to control the actual destinations of all traffic from anyone connected to the AP. In this case, ISEMasq would be used to condense specific traffic to either a pool of custom servers or to specific servers elsewhere on the network or even Internet.

المنارة للاستشارات
www.manaraa.com

6 Future Work

While ISEMasq has reached a solid milestone in its development with the major functionality and design goals having been met, there is always room for improvement. Future scalability with any large network application is always a concern, especially considering that ISEAGE will inevitably continue to grow. For this reason the following changes and add-ons to the ISEMasq design are suggested so that performance and ease of use keep pace with the further development of ISEAGE. First it may be beneficial to try using an unsorted-map data structures instead of the current STL map structure, especially for the larger internal databases. Slated for release as part of TR2, the unsorted map will finally provide the C++ STL with a hash table based structure, thus allowing for constant time lookups and insertions.

Next, in order to integrate with DeepFreeze, two tasks need to be done. First, a GTK+ front-end needs to be designed and written to work with the DeepFreeze console. GTK+ is a developer toolkit for creating graphical user interfaces [23]. This task will require creating a single GTK+ window that implements the function stubs defined by the DeepFreeze API, and provides full configuration control for the ISEMasq tool. The traditional way of creating this interface would be to code it entirely in C/C++. However, the latest version of Glade (Glade-3) [24], the most used User Interface Designer for GTK+, has deprecated the generation of C code in the interface creation process in favor of the libglade approach [25]. This new approach only requires the base framework to be written in C/C++ while the actual interface is done in XML and loaded dynamically at run-time, thus and increases flexibility and reusability while greatly reducing development time. The resulting interface then needs to be compiled as a loadable module that the DeepFreeze Console will read in and add as a tab to the main control

www.manaraa.com

interface. Additionally, the DeepFreeze Console configuration file will need to be appended with the specific ISEAGE nodes that the new tool is to run on.

Finally, in order for the loadable console to control the remote tool once DeepFreeze has launched it, the command line version of ISEMasq (as it exists now), needs to be modified to read and write data over the DeepFreeze communication channel. DeepFreeze conveniently sets these channels up to simply be stdin and stdout, though the programmer of the tool needs to come up with his/her own payload format for the data. To this end, the configuration of the tool will no longer be done from the command line of each ISEAGE node running the tool, but rather pushed down from the front-end control console. Once completed the ISEMasq Console plug-in for DeepFreeze will be able to control all instances of ISEMasq from the loaded interface module.

المنارة للاستشارات

www.manaraa.com

7 Conclusion

While there is some additional work that can be done to further improve ISEMasq, it is mostly cosmetic, such as the DeepFreeze interface or common code polishing to eliminate any minor bugs. As it stands, the command line version of ISEMasq has quite successfully met the current design goals. To compliment this, the up-and-coming upgrade to ISEAGE, a.k.a. ISEAGE 2.0, will provide new hardware, including multi-core processors on each of the ISEAGE nodes. Not only will this upgrade benefit all the ISEAGE tools in general, it will specifically benefit ISEMasq due to its multithreaded design, which can take full advantage of the multi-core processors.

With the goals met and the desired functionality of ISEMasq and its components firmly in place, the core framework for the ISEAGE testbed is finally near completion. This framework, i.e. Layers 1 and 2, complimented by Layer 0, should contribute significantly to the further completion of ISEAGE and will hopefully lead to a bright future of further development and testing of security products and concepts.

المنارة للاستشارات
www.manaraa.com

Appendix A: Data Class Header File

```
#ifndef ISEMasq_DS_H
#define ISEMasq_DS_H

#include <arpa/inet.h>
#include <stdint.h>
#include <ctime>
#include <cstdlib>              //rand()
#include <iostream>             //cout & such
#include <string>
#include <deque>                //double ended que from STL
#include <map>                  //one-to-one mapping from STL

struct Node {
        uint32_t ip_addr;
        time_t time_stamp;
};

class masq_ds
{
  public:
  //Constructor and Destructor, wii...=)
        masq_ds();
        ~masq_ds();

  //Member Functions for DNS mode
        uint32_t Lookup_IP(std::string url);                //dns_thread
        std::string Lookup_URL(uint32_t ip);                //dns_thread

  //Member Functions for Gtw mode
        void Set_Default_Dst(uint32_t dst_ip);              //main()
        void Add_Dest(uint16_t port, uint32_t dst_ip);      //main()

        void Update_DS(uint64_t key, uint32_t dst_ip);      //In_thread
        uint32_t Lookup_Dst(uint16_t dst_prt);              //In_thread
        void Process_Que();                                 //In_thread
        uint32_t Lookup_Src(uint64_t key);                  //Out_thread

  //Stat Functions
        int get_map_size()      { return main_map.size(); }         //Stat_thread
        int get_que_size()      { return purge_que.size(); }        //Stat_thread
        int get_dns_size()      { return dns_map.size(); }          //Stat_thread

  private:
  //member vars and data structures for DNS functionality
        uint32_t max_dns_size;
        std::map<std::string, Node> dns_map;
        std::map<uint32_t, std::map<std::string, Node>::iterator > dns_rev_map;
        std::deque< std::map<std::string, Node>::iterator > dns_que;

  // member vars and data stuctures for Gtw funcionality
        uint32_t default_dst, max_que_size;
        std::map<uint16_t, uint32_t>::iterator dst_it;
        std::map<uint64_t, Node> main_map;
        std::map<uint16_t, uint32_t> dst_map;
        std::deque< std::map<uint64_t, Node>::iterator > purge_que;
};
```

www.manaraa.com

Appendix B: ISEMasq Example Config File

```
#################################
## ISEMasq example config file ##
#################################

Wan_if = rl0
Lan_if = ed0

# 1 is DNS only, 2 is GTW only, and 3 is Both
Run_mode = 3

# If the run mode is DNS(1) or Full(3),
# then any port 53 Dst_Server entries will be ignored

# All desired mappings in form IP_addr:Port
Dst_Serv = 128.32.8.1:53
Dst_Serv = 64.32.16.8:80
Dst_Serv = 1.2.3.4:80
Dst_Serv = 5.6.7.8:22

# Any ports that are specifically mapped above are randomly sent
# to one of the Dst_Serv enties unless this is set.
# Default_Serv = 1.2.3.4

# Setting this will enable service rotation for any Dst_Serv
# entries that use the same port, such as port 80 above;
# otherwise only the first entry of port 80 is used.
# Serv_Rotation = 1
```

المنارة للاستشارات

www.manaraa.com

References

[1] President's Information Technology Advisory Committee (PITAC). (2005, February). Cyber Security: A Crisis of Prioritization. PITAC Reports to the President. [Online] Available: http://www.nitrd.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf. Retrieved: 31 Oct 2007.

[2] A. Hoenecke, T. Gillispie, B. Anderson, and T. Daniels. The role of information warfare in information assurance education: a legal and ethical perspective. presented at 2007 ASEE Annual Conference & Exposition. [Online]. Available: http://www.asee.org/acPapers/AC%202007Full2158.pdf

[3] D. Jacobson. (2003). ISEAGE Overview. Iowa State University Information Assurance Center. [Online]. Available: http://www.iac.iastate.edu/iseage/iseage_overview.pdf. Retrieved: 31 Oct 2007.

[4] N. L. Karstens, "DeepFreeze: a management interface for ISEAGE," M.S. Thesis, Info. Assurance and Comp. Engr., Iowa St. Univ., Ames, 2007.

[5] A. L. Hahn, "Advanced packet obfuscation and control program (APOC)," M.S. Thesis, Comp. Engr. and Info. Assurance, Iowa St. Univ., Ames, 2006.

[6] H. J. Qureshi, "Generating background network traffic for network security testbeds," M.S. Thesis, Comp. Engr. and Info. Assurance, Iowa St. Univ., Ames, 2006.

[7] B. A. Forouzan, "Delivery and Routing of IP Packets," in *TCP/IP Protocol Suite*, 2nd ed. Boston, MA: McGraw-Hill, 2003 ch. 6, pp. 147-168.

[8] *Traditional IP Network Address Translator (Traditional NAT)*, RFC 3022, 2001.

[9] B. A. Forouzan, "Network Address Translation (NAT)," in *TCP/IP Protocol Suite*, 2nd ed. Boston, MA: McGraw-Hill, 2003, ch. 30, sect. 3, pp. 780-783.

[10] B. A. Forouzan, "Domain Name System (DNS)," in *TCP/IP Protocol Suite*, 2nd ed. Boston, MA: McGraw-Hill, 2003, ch. 18, pp. 497-526.

[11] SourceForge, Inc. (2007). SourceForge.net: The libpcap project. [Online]. Available: http://sourceforge.net/projects/libpcap/. Retrieved 31 Oct, 2007.

[12] The Packetfactory. (2007). The Million Packet March. [Online]. Available: http://www.packetfactory.net/libnet/. Retrieved: 31 Oct 2007.

[13] The C++ Resources Network. (2007). STL Containers – C++ Reference. [Online]. Available: http://www.cplusplus.com/reference/stl/. Retrieved: 31 Oct 2007.

[14] B. Dawes, D. Abrahams, and R. Rivera. (2007). Boost C++ Libraries. [Online]. Available: http://www.boost.org/. Retrieved: 31 Oct 2007.

[15] Various Authors. (2005). Shrink Wrapped Boost. [Online]. Available: http://www.boost.org/doc/html/who_s_using_boost_/shrink.html. Retrieved: 31 Oct 2007.

[16] V. Prus. (2004). Chapter 10. Boost.Program_options. Boost C++ Libraries. [Online]. Available: http://www.boost.org/doc/html/program_options.html. Retrieved: 31 Oct 2007.

[17] W. E. Kempf. (2006). Chapter 15. Boost.Thread. Boost C++ Libraries. [Online]. Available: http://www.boost.org/doc/html/thread.html. Retrieved: 31 Oct 2007.

المنارة للاستشارات

www.manaraa.com

[18] The FreeBSD Project. (2007). divert. FreeBSD Hypertext Man Pages. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=divert&sektion=0&apropos=0&manpath=FreeBSD+6.2-RELEASE. Retrieved: 31 Oct 2007.

[19] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in *Proceedings of the Winter 1993 USENIX Conference*, San Diego, CA, pp. 259--269.

[20] The FreeBSD Project. (2007). bpf(4). FreeBSD Hypertext Man Pages. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=bpf&sektion=4&apropos=0&manpath=FreeBSD+6.2-RELEASE. Retrieved: 31 Oct 2007.

[21] The FreeBSD Project. (2007). pfctl(8). FreeBSD Hypertext Man Pages. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=pfctl&sektion=8&apropos=0&manpath=FreeBSD+6.2-RELEASE. Retrieved: 31 Oct 2007.

[22] Internet Systems Consortium, Inc. (2004). ISC BIND. [Online]. Available: http://www.isc.org/products/BIND. Retrieved: 31 Oct 2007.

[23] The GTK+ Team. (2007). GTK+ - The GIMP Toolkit. [Online]. Available: http://www.gtk.org/. Retrieved: 31 Oct 2007.

[24] V. Geddes, Sun GNOME Documentation Team, and M. Vance. (2006). Glade Interface Designer Manual. [Online]. Available: http://glade.gnome.org/manual/index-info.html. Retrieved: 31 Oct 2007.

[25] J. Henstridge. (2005). Libglade. [Online]. Available: http://www.jamesh.id.au/software/libglade/. Retrieved: 31 Oct 2007.

المنارة للاستشارات
www.manaraa.com

Acknowledgements

As I turn the page on this chapter of my life, I would like to pause for a moment to express my deepest gratitude to all those who made the successful completion of this journey possible. First off, to my major professor Dr. Jacobson for his patience, funding, and the opportunity to work with the ISEAGE project. To both Dr. Daniels and Dr. Bergman for their feedback and participation on my thesis committee. Many thanks to my graduate student colleges, who listened and put up with me, and who provided valued technical input when needed. Also, I would like to extend a special thank you to my family for their unwavering support and words of encouragement over the last couple of years.

Lastly, and most of all, I would like to thank my wife, Dr. Meagen Gillispie, for her unwavering kindness and support these past couple years. Not only did she stay engaged to me until I returned from deployment in 2004, she has embraced the changes it had on me, both good and bad. She has stood by me in all my endeavors and has provided me with a solid foundation that I have come to depend on. Without her love and encouragement I would not be were I am today. Meagen, I love you so much; thank you from the bottom of my heart.

المنارة للاستشارات
www.manaraa.com