2008

Intercept System to Edit, Control, and Analyze Packets (ISECAP)

Grant Rogers Brinkmeyer

Iowa State University

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

Part of the Computer Sciences Commons

Recommended Citation

Brinkmeyer, Grant Rogers, "Intercept System to Edit, Control, and Analyze Packets (ISECAP)" (2008). Retrospective Theses and Dissertations. 15415.

https://lib.dr.iastate.edu/rtd/15415

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.
Intercept System to Edit, Control, and Analyze Packets (ISECAP)

by

Grant Rogers Brinkmeyer

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Doug W Jacobson, Co-major Professor
Sree Nilakanta, Co-major Professor
Thomas Earl Daniels

Iowa State University
Ames, Iowa
2008

Copyright © Grant Rogers Brinkmeyer, 2008. All rights reserved.
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
# TABLE OF CONTENTS

LIST OF FIGURES.................................................................................................................................................. iv

ABSTRACT .......................................................................................................................................................... v

ABSTRACT .......................................................................................................................................................... v

CHAPTER 1. INTRODUCTION ........................................................................................................................... 1
  1.1 Internet Data Packets............................................................................................................................... 2
  1.2 ISEAGE Purpose and Architecture ......................................................................................................... 6
  1.3 FreeBSD and IPFW..................................................................................................................................... 8

CHAPTER 2. RELATED TECHNOLOGIES ......................................................................................................... 9
  2.1 Netfilter .................................................................................................................................................... 9
  2.2 Ettercap ................................................................................................................................................... 10
  2.3 APOC ....................................................................................................................................................... 11

CHAPTER 3. DESIGN AND IMPLEMENTATION .............................................................................................. 13
  3.1 Machine Setup......................................................................................................................................... 14
  3.2 IPFW to ISECAP ..................................................................................................................................... 16
  3.3 Pattern Matching and Event Handling .................................................................................................. 18
  3.4 Packet Changing..................................................................................................................................... 19
  3.5 Connection Tracking ............................................................................................................................... 20
  3.6 Packet Retransmission............................................................................................................................. 22

CHAPTER 4. PRACTICAL USES ......................................................................................................................... 24
  4.1 Research .................................................................................................................................................. 24
  4.2 Attack Testing ......................................................................................................................................... 24
  4.3 Cyber Defense Competitions .................................................................................................................. 26

Chapter 5. LIMITATIONS ................................................................................................................................. 27
  5.1 Network Position...................................................................................................................................... 27
  5.2 Layer Limitations .................................................................................................................................... 28
  5.3 Other Limitations...................................................................................................................................... 28

CHAPTER 6. TESTING ......................................................................................................................................... 30
  6.1 Ping Test................................................................................................................................................... 30
  6.2 Iperf Test ................................................................................................................................................ 31
  6.3 Nmap Test .............................................................................................................................................. 32
  6.4 Future Tests ........................................................................................................................................... 33

CHAPTER 7. CONCLUSIONS ............................................................................................................................. 34
  7.1 Future Work ............................................................................................................................................ 34
  7.2 Summary ............................................................................................................................................... 36
LIST OF FIGURES

Figure 1. OSI Model.................................................................................................................. 2
Figure 2. Ethernet header [4].................................................................................................. 3
Figure 3. IP header [4].......................................................................................................... 4
Figure 4. ICMP header [4] ................................................................................................... 4
Figure 5. TCP header [4]...................................................................................................... 5
Figure 6. UDP header [4] ..................................................................................................... 5
Figure 7. ISEAGE architecture [6]....................................................................................... 7
Figure 8. Low-layer packet flow ........................................................................................ 15
Figure 9. ISECAP module system...................................................................................... 17
Figure 10. Connection Tracking Process ........................................................................... 20
ABSTRACT

ISECAP is an advanced packet filtering framework designed for use with the Internet Scale Event and Attack Generation Environment (ISEAGE) at Iowa State University. ISEAGE is a network security test bed where cyber attacks can be carried out for security research. ISECAP functions as a FreeBSD-based general purpose packet control system designed to alter ISEAGE network data as it is being transmitted. Scripts are interpreted by ISECAP to modify network data in real-time, assisting in the simulation and obfuscation of various network attacks tested at the ISEAGE facility. The ISECAP system is developed as a flexible tool with a forward focus on ease of extensibility to meet the future needs of ISEAGE.
CHAPTER 1. INTRODUCTION

As the world becomes more automated with computers performing more tasks, everyday life will be managed over networks such as the Internet. Businesses and individuals alike will control every aspect of their finances with computer systems. Even today, the world economy is dependent on computer networks to monitor stocks, currencies, and every type of business transaction. With the prosperity of the modern world relying on computers, it is more critical than ever to ensure these systems are secure. In recent years, laws such as the Health Insurance Portability and Accountability Act (HIPAA), have been passed to require the safeguard of private health insurance information and include sections related to information and computer system security [1]. In 2002, the Federal Information Security Management Act (FISMA) was created as a security guideline for businesses which interact with government agencies [2]. This rule provides extensive detail into the recommended security operations of federal contractors.

Because of the importance of computer system security, tools to facilitate rigorous testing are required. ISECAP is one such tool. ISECAP is a FreeBSD-based general purpose packet control system designed to alter network data as it is being transmitted. It was created to operate as part of the Internet Scale Event Attack Generation Environment (ISEAGE) at Iowa State University. ISECAP was designed with four goals in mind. First, the system should allow for flexible script-based packet filtering. Second, ISECAP should be capable of simulating man-in-the-middle (MitM) and distributed scanning network attacks. Third, the system should execute data modifications on network traffic with a high level of performance. Fourth, ISECAP should be modularly designed with an
emphasis on future extensibility by ISEAGE researchers. This paper will explain and demonstrate how ISECAP achieves these goals.

1.1 Internet Data Packets

To have an understanding of ISECAP one must have a basic understanding of how data moves across the Internet. Data is sent between computers in a unit known as a packet. Each packet contains a quantity of data and instructions for delivering that data. A packet has one or more headers which contain information about the packet itself and assist in delivering it across the Internet and to the correct process on the destination computer. Packet headers are most often represented by Open Systems Interconnection (OSI) Model [3]. As seen in Figure 1, the OSI Model is composed of seven distinct layers that define different aspects of moving data across a network using data encapsulation. ISECAP is primarily focused on the lower layers of the model.

![Figure 1. OSI Model](image-url)
The lowest layer in the OSI Model is the physical layer. This layer involves the requirements for data to be transmitted over the physical medium of the network. The physical layer allows for the network topology, data rates, and transmission modes to be established by the transmitting and receiving devices. Data on the physical layer is in its most basic form: bits represented by ones and zeros.

The second layer of the OSI Model is the data link layer. The data link layer uses the physical layer to move information between adjacent nodes on the network. This layer can include functions such as error management which allows upper layers to proceed as if the physical layer transmitted data without error. The data link layer also handles physical device addressing and both access and flow control over the medium. The type of data link layer technology used in ISECAP is the IEEE 802.3 standard, known as Ethernet. As shown in Figure 2, Ethernet packet headers are the very first part of an Ethernet data packet. The Ethernet header contains both the source and destination media access control (MAC) addresses for the packet. All other packet data occurs later in the packet. It is this header that provides the fundamental instruction for data packets to traverse the Internet.

![Figure 2. Ethernet header](image)

The third layer of the OSI Model is the network layer. This layer is responsible for guaranteeing packets are sent from the original host all the way to their final destination. In Internet traffic the Internet Protocol (IP) is used to address and detail packets at the network layer. ISECAP uses packet IP headers, seen in Figure 3, for a wide range of its...
functionality. IP manages data as it traverses multiple link-layer networks and provides fragmentation of packets to a size that can be accepted by the physical medium. A second network layer protocol known as Internet Control Message Protocol (ICMP) operates in conjunction with IP for internet traffic. ICMP is used primarily to send error messages across networks indicating that various network services do not exist. ICMP can also be used to determine if network nodes are functional, most notably with the ping command. The ICMP header is displayed in Figure 4 and contains fields designed to communicate specific messages to the receiving computer.

The fourth and final layer of primary concern to ISECAP is the transport layer. This layer is tasked with the delivery of data to the individual applications running on communicating computers. The transport layer handles segmentation and reassembly of packets and makes sure the data arrives without duplication, damage, or loss. The transport layer also handles application connection control. The most popular Internet transport layer protocols are Transmission Control Protocol (TCP) and User Datagram
Protocol (UDP). As seen in Figure 5 and Figure 6 respectively, both these protocols manage application packet distribution with specified port numbers in the packet headers.

TCP is a connection-oriented protocol that establishes a stateful and persistent connection between two network applications. This connection is established and maintained with TCP packet header fields such as flags and sequence numbers. TCP is used by a wide range of application layer protocols such as File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), and Post Office Protocol version 3 (POP3). UDP is an inherently connectionless protocol that also does not guarantee reliable packet transfer. It is most often used in areas where TCP overhead is excessive such as audio/video streaming and connectionless broadcast information.

The fifth, sixth, and seventh layers of the OSI Model are the session layer, the presentation layer, and the application layer. These layers are of less immediate concern to ISECAP than the lower four because they deal with specific program functionality. The
application layer is used by many Internet applications to control information transmission between specified corresponding network applications. For instance HTTP has its own application header information to ensure data functionality. Each application layer protocol has unique header information and can be filtered by ISECAP with a specific application layer protocol module.

1.2 ISEAGE Purpose and Architecture

A basic understanding of ISEAGE is required to understand the context in which ISECAP will operate. ISEAGE is a multi-purpose network security research test bed at Iowa State University. The project was designed to create a “virtual Internet” for security researchers to test real network attacks against real equipment with the goal of improving computing safety [5]. ISEAGE is designed to be a logically malleable networking environment capable of simulating any provided network configuration. In this setting, real world security solutions can be developed and tested by professors, students, and industry professionals. Government and telecommunications officials can use ISEAGE to simulate loss of major network infrastructure, and industry professionals can be trained in network security standards and procedures. Finally, ISEAGE is used several times each year to run Cyber Defense Competitions which simulate a real world security crisis. In these competitions, teams of students attempt to design and secure a network while other students, professors, and industry experts try to attack the networks.

The core of ISEAGE is a 64 computer cluster spread among four racks known as “icles” [6]. Each of these computers is live-booted with FreeBSD and operates based on loaded configuration files. Different boards function as different parts of the simulated
network, performing tasks such as routing and packet control. Various pieces of custom software work with the icicles to perform a wide range of simulation tasks. As seen in Figure 7, the core of ISEAGE branches out into other networks where end-users can perform simulations and experiments. ISECAP will most likely function as a process running on one or more icle boards. From here traffic can be routed through the board, and thus ISECAP, where modifications will occur. Outside observers will see only ISEAGE background routing and will have little reason to suspect data modification.

Figure 7. ISEAGE architecture [6]
1.3 FreeBSD and IPFW

FreeBSD is a UNIX-based operating system freely distributed to anyone. It is designed to be a stable and secure operating system specifically developed for use in education, research, and Internet services [7]. FreeBSD comes with a variety of software development tools and is descended from AT&T UNIX. It differs from other versions of BSD, such as NetBSD, in that it has a stronger focus on system performance and ease of use. FreeBSD offers a full operating system in one package including the kernel, device drivers, and userspace applications. It also has strong compatibility with Linux applications. FreeBSD is known for its effective performance as a network services operating system because of its exceptional stability and full range of development tools.

The core of ISEAGE, the icles, runs version 5 of FreeBSD. These will soon be updated to version seven of the operating system. Because it needs to be able to be fully integrated with ISEAGE icles, ISECAP was developed to run on FreeBSD with the tools available on the operating system itself.

One such tool is the FreeBSD Internet Protocol Firewall (IPFW). IPFW is a packet filtering and redirection program originally created for FreeBSD. It comes standard with all FreeBSD installs as a loadable kernel module and also can be compiled directly into the kernel. IPFW has been ported to, and is available on, several other operating systems including Mac OS X, Linux, and Windows. IPFW allows for various complex filtering capabilities making it a powerfully scalable firewall for personal and professional use. IPFW is one of the key components of ISECAP.
CHAPTER 2. RELATED TECHNOLOGIES

Packet filtering has been a desirable goal since the inception of computer networks. Several tools related to ISECAP exist. Tools such as Netfilter, Ettercap, and APOC have been designed with similar, though not identical, goals. Each of these tools has had varying degrees of influence on the development of ISECAP. This chapter will give a brief overview of their technology.

2.1 Netfilter

Netfilter is an architecture for the Linux operating system designed to provide packet control functionality. It provides a framework for packet filtering through a set of hooks built into the network stack of the kernel [8]. The uses of Netfilter vary from simple stateless packet filtering to complex stateful packet mangling. The Netfilter architecture can be used to provide network address translation (NAT), firewall development, and packet prioritization. Its most established use is in the Linux iptables firewall. Iptables is a command line firewall program designed to filter packets based on preset rules.

Netfilter performs its functions by providing developers with hooks into the Linux network stack. Each layer of the stack has a set of hooks specific to the control of packets within the layer. With these hooks, developers have extremely detailed control over all packets as they traverse the stack. This control provides an unprecedented level of functionality when creating network filtering tools. In addition to packet control, Netfilter offers a series of tools designed to control Linux connection tracking functions. These tools are known as conntrack and give users and developers a window into the kernel state table used for connection tracking. Userspace programs can utilize conntrack to manage
attributes of connection state in firewalls and packet filtering software. This is a highly useful feature, as any good packet filtering system designed to alter connection-based traffic, such as TCP, requires some form of connection tracking. Netfilter’s conntrack system was influential in the development of the ISECAP connection tracking modules.

Unfortunately, Netfilter relies on hooks within the Linux kernel itself. Therefore no version of Netfilter exists for FreeBSD. As ISEAGE is almost exclusively based on FreeBSD, the Netfilter architecture and corresponding packet filtering programs are of no applicable use. ISECAP has been designed from inception with the goal of full FreeBSD functionality.

### 2.2 Ettercap

Ettercap is a set of packet filter and manipulation tools specifically designed to execute man-in-the-middle attacks. It supports packet mangling of many protocols with a filter based system [9]. Ettercap is capable of injecting itself between network nodes undetected through both ARP poisoning and DNS stealing, thus establishing a position of control over the network connection.

Upon establishing this position, Ettercap uses its filter system to modify packets passing through it. Ettercap employs an extremely complex filter system known as Etterfilter. Etterfilter scripts are created with the use of a language designed specifically for Etterfilter. When a script has been completely written, it is compiled and run as part of Ettercap. The packet filter system is based on the popular libpcap and libnet packet control frameworks. Ettercap supports plug-ins for executing MitM attacks on numerous application layer protocols such as HTTP, FTP, and even SSH.
The Ettercap software has been ported to a wide range of operating systems including Linux, Windows, Solaris, and several versions of BSD. While highly functional, Ettercap does not fully meet the needs of ISEAGE. Ettercap was written with a very narrow focus: to execute a man-in-the-middle attack. While it offers some extensibility within this small scope, it does not perform many of the functions ISEAGE requires and will require in the future. The Etterfilter system is also highly complex with a steep learning curve. Before one can even write a script one must learn the Etterfilter language. Creating new program functionality is even more challenging. ISECAP has been designed specifically with the needs of ISEAGE in mind. ISECAP has employed an extensibility-minded module design approach that ensures future users will be able to quickly and easily expand its uses.

2.3 APOC

The Advanced Packet Obfuscation and Control (APOC) program was a previous ISEAGE-related packet filtering system designed in 2006 [10]. Like ISECAP, APOC was developed as a packet filtering mechanism. APOC is a Linux application built upon the libnet and pcap libraries, using pcap to acquire network traffic and libnet to write modified traffic. Much of APOC’s functionality is dependent on technology from both Netfilter and Ettercap. APOC uses the Etterfilter system and language to create and compile scripts. Netfilter forms the basis of APOC’s packet control, packet mangling, and connection tracking. APOC uses the hooks provided by Netfilter to alter packets at various levels of the network stack as specified in the Etterfilter scripts. APOC can manage data traffic at all layers on a network interface specific basis. The APOC system has two interfaces, one
for reading traffic and one for writing traffic, depending on the direction of traffic flow. The interfaces operate in a full-duplex model and require a unique script for each.

APOC encountered several functional problems. The first was packet reiteration. This problem dealt with the fact that the system inherently did not know if a packet on the stack had yet been altered. This caused a potentially infinite loop and required that APOC implement a system to remember recently written packets. The largest issues with APOC involve its network performance. ISEAGE packet filter tools must be able to process large amounts of traffic quickly in order to be useful. APOC failed in this area. First, a data throughput connection test on the APOC system showed a decrease from a data speed of 55.5 megabits per second (Mbps) when APOC was not running to a speed of 3.75 Mbps. Similarly, a NMap test revealed an enormous increase in latency to perform a scan using random source IP addresses. The scan took an average of 10.885 seconds without APOC running. With APOC running, the scan length increased to 24.554 seconds – more than double the original time required. Between these performance issues and the fact that it requires Linux to run, APOC makes a poor tool for implementation in the ISEAGE environment. ISECAP is being developed with the express goal of high performance and will run natively in ISEAGE’s FreeBSD environment.
CHAPTER 3. DESIGN AND IMPLEMENTATION

The ISEAGE environment is in need of an efficient system for packet modification, now and in the future. This need drove the design of ISECAP and its eventual outcome as a functional product. The design principles of ISECAP were based on a set of four goals:

1. Flexible script-based packet filtering,
2. Simulation of man-in-the-middle and distributed scanning attacks,
3. High performance packet modification, and
4. Modular design with future extensibility.

While the logic and syntax of the ISECAP code determine the first three goals, the fourth goal can only be accomplished through careful preliminary design. A problem that has long plagued ISEAGE is the abundance of poorly understood code. This problem is a direct result of students graduating and leaving behind their projects. Unfortunately many of these projects are highly complex and, with no one to maintain them, sit either under-used or completely unused.

ISECAP’s modular design is a direct result of this problem. ISECAP divides its tasks into a number of specific, largely self-explanatory, modules through which packets flow. Each of these modules should be simple for future ISEAGE graduate students to understand, modify, and expand. Updating ISECAP to meet a new demand of ISEAGE should be a very minor task, requiring only the addition of a few functions to the ISECAP code. Appendix C details a step-by-step setup of an ISECAP machine, which will greatly assist future students. This level of extensibility should keep ISECAP in use and current
within the ever-changing infrastructure of ISEAGE. This chapter will focus on how this
goal and the three previous goals are met by the design of ISECAP.

3.1 Machine Setup

As ISECAP is designed to run fully within ISEAGE, the ISECAP development and
test machine is built like one of the 64 icle boards. The motherboard, processor, memory,
and network interface cards are identical to those used in the ISEAGE icles. ISECAP is
running a Intel Celeron processor at 2.4 GHz with 1 GB of memory on a Supermicro
P4SCE motherboard. These motherboards include dual gigabit ethernet ports, but
ISECAP, like the icle boards, has an additional gigabit PCI network card. The only
difference between ISECAP and an icle is that the ISECAP machine uses a hard disk drive
where icles are diskless and boot from a remote server. This small difference is meant to
be more convenient and should not affect system performance in any significant way.
ISECAP was coded using the standard C programming language. This language was
selected because of its strong low level control, the availability of documentation, and the
ease with which changes can be made to network data.

ISECAP is currently running the 6.2-Release version of FreeBSD. When
integrated into ISEAGE, ISECAP will be running on version 7 of FreeBSD. Though this
has yet to be tested, no major changes were implemented to any of the systems ISECAP
uses. Therefore ISECAP should be compatible with whichever release of current FreeBSD
ISEAGE chooses to run in the future. The ISECAP machine is running a slightly modified
kernel with the following options added to the default kernel installation:
These options load both IPFirewall and its divert socket module into the kernel. Though both can be loaded as user processes, it was simpler for development and testing to have them operational at all times in the kernel.

ISECAP is configured as a network gateway for the outside traffic generating computers. This setup allows ISECAP to filter traffic at the network layer and higher. Traffic is sent through ISECAP with the use of two external test machines referred to as “test 1” and “test 2”. Each test machine has one of ISECAP’s ethernet cards configured as its gateway. Each computer can then send traffic to the other through ISECAP. With this configuration, testing of the majority of ISECAP’s functionality is simple to perform. The low-layer traffic flow through ISECAP is shown in Figure 8.
3.2 IPFW to ISECAP

A model of the ISECAP system is displayed in Figure 9. It can be best visualized as a packet progressing through the various modules. The packet first encounters part of the ISECAP system at the network layer when it is passed by the kernel against a rule in the IPFW rule table. A special type of rule known as a divert socket is inserted into the IPFW table by the ISECAP user. Divert socket is an option in an IPFW rule statement that forwards an unprocessed packet to a network port specified within the statement [11]. For example:

```
# ipfw add 100 divert 7000 ip from 10.10.1.1 to 10.10.2.1
```

This shell command would add a rule to the IPFW rule table assigned rule number 100. The rule tells ipfw to send all IP packets originating at the IP address 10.10.1.1 and destined for the IP address 10.10.2.1 to port 7000. A program, such as ISECAP, can then receive the unprocessed packet from the port. Appendix B contains additional examples of IPFW divert rule statements. Diverted packets can be read as either incoming or outgoing, but will be read exclusively as incoming for use in ISECAP. All packet diversion, in and out, takes place at the network layer. When packet fragments are diverted, they are fully reassembled by the kernel before they are delivered to the specified port. Divert IPFW rules are currently entered manually by the user while executing ISECAP. The rule redirects specified packets to the ISECAP program where processing begins.
Figure 9. ISECAP module system

ISECAP begins its work by taking the input script file and ordering it in a script structure. This structure maintains a record of the diverted packet criteria and the user desired changes to the packets that meet those criteria. A few key aspects of this data structure are its options and expandability. The options field contains special instructions. The first is an automatic pass option, which sends packets on unmodified. The second is an automatic drop function, which will block all diverted packets from continuing past ISECAP. The packet changes sub-structure already includes a wide range of header modifications to various layers of the packet, and is easily expandable if the user wants to make additional modifications to selected packets. The main script structure is one of the most important aspects of ISECAP.

After acquiring the user script, ISECAP initiates the connection to the divert socket rule. When running ISECAP, the user supplies a port number that corresponds to the divert port in the IPFW rule. ISECAP binds itself to this port and is prepared to receive incoming packets. The program next initiates the main functional loop which runs until the user forces an exit. This loop is the heart of ISECAP, where packet handling occurs.
Packets are received from IPFW and loaded into a memory buffer. From here, ISECAP sends the packet to the rest of the program by way of the pattern matcher.

3.3 Pattern Matching and Event Handling

The pattern matching module begins by checking to see if the user selected either the pass or drop options in the script. If this selection was made, no packet processing is needed. Otherwise the pattern matcher examines the IP header of the packet and determines the next layer protocol. If the protocol is one supported by ISECAP, the pattern matcher moves the packet to the correct event handler. If the protocol is not supported by ISECAP, the default action is to pass the packet back to the IP stack unprocessed. If the specific type of IP protocol is not supported, IP header changes are still performed before returning the packet. The pattern matcher is meant as a hierarchical control function which can be quickly appended to support actions for new protocols.

After being processed by the pattern matcher, the event handler corresponding to the identified protocol is invoked. Each of the three current event handlers (ICMP, TCP, and UDP) operate somewhat differently. The ICMP event handler is currently quite straight-forward. It receives the packet from the pattern matcher and structures both the IP and ICMP headers so the packet can be analyzed. After analysis, this module calls various packet changing functions to make the necessary alterations. Additional ICMP actions can be easily added.

The UDP event handler operates in much the same as the ICMP handler. The packet is delivered and parsed for header information. Upon analysis the appropriate packet change functions are called. The UDP handler also is written for easy expansion.
Options such as special event handling based on attributes of the UDP packet would expand ISECAP’s functionality. Finally, the UDP header checksum is recalculated so packets are not rejected by the receiving node. This recalculation involves information from the IP header, the UDP header, and the total size of the packet.

The TCP event handler is somewhat more complicated. Like the other event handlers, when it receives a packet, the TCP handler structures the IP and TCP headers for analysis. After analysis, the connection tracking module is invoked for the TCP packet. This module maintains state information about connection-based TCP streams and will call the packet modification functions at the appropriate times. As with UDP, TCP packets require a TCP checksum recalculation. For TCP, this calculation requires information from the IP header, the TCP header, total packet size, and the TCP options.

3.4 Packet Changing

The ISECAP packet changing module is a set of functions designed to alter packet information based on protocol type. A packet changing function receives the appropriate header information and user script from an event handler. It begins by parsing the change script structure. This structure contains all packet changing information initially entered by the user. The packet changing module checks to see which fields in the packet need to be changed and performs the necessary data type changes. When this is complete, it alters the packet information in memory.

A special packet changing case occurs if the user instructed a packet field to be random. The packet changing module checks for this random request value and then calls
functions designed to return a randomized version of the appropriate field. ISECAP currently supports random IP addresses and random application ports.

3.5 Connection Tracking

Figure 10 shows the connection tracking process in ISECAP. The purpose of ISECAP’s connection tracking is to maintain a state table monitoring active TCP connections of interest to the ISECAP system. Because the connection tracking needs of ISECAP are similar to many other packet filtering programs, both Netfilter and APOC strongly influenced the design of ISECAP’s connection tracking.

The first aspect of the connection tracking to understand is the data structures involved. The basic data structure is called a conn_end. This structure is a simple 4-tuple of the source IP address, source port, destination IP address, and destination port. The second data structure of importance is the conntrack structure. This structure is a grouping of four conn_end structures and a variable marking the timeout value for the connection.
the connection hash table. The hash table lookup can result in one of two possibilities: either the connection is new or the connection is part of a previously established stream.

If the table lookup fails to find a record corresponding to the current packet, it is assumed to be a new connection. The TCP event handler immediately modifies the current packet based on the user change script. After this modification, the second conn_end value is created in the conntrack structure for this connection. The third conn_end value is created by reversing the source and destination information of the original packet. Finally, the fourth conn_end value is added to the conntrack structure by reversing the source and destination information of the modified version of the current packet.

At this point, a total of four complete conn_end sub-structures exist within conntrack. The first value represents how the packet will be seen initially on one of the outside interfaces. The second value represents how the packet will appear as it is being re-injected, based on the modifications by the ISECAP user script. The third value indicates what packets received from the other side of this connection should look like before data is re-injected and transmitted to the original host. The fourth value represents what ISECAP will receive from the other side of this connection before any changes are made to the packets. These values are the key to ISECAP’s connection tracking capabilities. The first and forth conn_end structures are installed in the hash table. This dual installation occurs because these are the two values that will enter the ISECAP computer and result in table lookups. Finally, the new connection packet is sent back to the network stack.

If the original packet’s hash table lookup resulted in a value, ISECAP determines this packet is part of a previously established connection. The hash table lookup returns
the memory address of the *conntrack* structure corresponding to this connection. ISECAP then renews the timeout value, thus extending it. The TCP handler continues by modifying the packet based on the related value in the *conntrack* structure. This modification ensures the packet maintains its place within the TCP stream. After altering the packet appropriately, the packet is retransmitted.

The hash function used in ISECAP is the Jenkins Hash. This hash function has been proven to have efficient and fast performance for non-cryptographic implementations [12]. Both Netfilter and APOC use the Jenkins Hash, indicating it is effective for connection tracking in packet filtering applications. Hash table functionality is provided by storing data with a seeded hash value. The table index value is a combination of information about the hash table itself, the *conn_end* structure data, and an incremented integer.

### 3.6 Packet Retransmission

When all requested and required packet functions have been completed, ISECAP begins the reinjection process. The first step for this process is to recompute the IP header length, which might have changed during packet alteration. This value must be accurate for retransmission to occur without error. Divert sockets use the *sendto* function to retransmit packets. Upon reinjection of the packet into the network stack, the IP checksum is automatically recalculated to account for any changes.

When *sendto* is called, diverted packets are automatically flagged by the divert functions as outgoing packets, seamlessly avoiding packet reiteration problems. This immediately reduces the processing required by ISECAP to ensure packets are not diverted
multiple times. Because it is handled by the kernel, ISECAP can reduce processing overhead dramatically. When the `sendto` function has been successfully called, the packet will continue to its destination outside the ISECAP machine.
CHAPTER 4. PRACTICAL USES

The uses of ISECAP are widely varied. As a general purpose packet filter solution, ISECAP can be used to solve problems ranging from advanced firewalling to Internet traffic shaping. Though it could be implemented in any number of environments, this chapter will focus on the uses of ISECAP within the ISEAGE environment.

4.1 Research

As an advanced security test bed, ISEAGE has the need for a wide range of evolving network applications. ISEAGE continues to employ the talents of computer and network security graduate students to maintain and expand its functionality. Many of these students will be developing security research tools based on the manipulation of packets. ISECAP provides a perfect framework for this type of research. As ISECAP is fully integrated with the ISEAGE FreeBSD environment and already functions as an advanced packet filtering system, the basic code is in place to create an enormous range of network devices. Future ISEAGE students can use modules from ISECAP intact or modify them to any degree to meet their needs. Many upcoming projects will have a much shorter development life-cycle because much of the groundwork can be derived from ISECAP. ISECAP will allow for the functionality of ISEAGE to expand more rapidly than ever.

4.2 Attack Testing

Though ISEAGE is highly effective at simulating a vast number of network attacks, ISECAP will enhance this functionality. This capability is especially important now that the ISEAGE facility is fully functional. Testing on marketplace security devices is
scheduled to begin within a few months. Being able to run a diverse field of security
attacks against these devices is crucial to ISEAGE’s success as a security testing facility.
ISECAP will be particularly useful in expanding two types of popular network attacks:
port scanning and denial of service.

A port scan is a popular attack used to gain information about target machines. In
these attacks a variety of TCP and UDP packets are sent to a range of ports on the target
system. The response to this barrage of packets indicates which network services the
attacker can use to launch an attack. Port scans are simple to detect due to the spike in
traffic from a single host. When a port scan is detected, the IP address of the offending
machine is usually filtered or even blocked. The robust randomization function in
ISECAP provides the capabilities needed to prevent accurate port scanning detection. By
randomizing source IP addresses for all port scanning packets, an attacker can successfully
mask the identity of the originating computer. If a defender cannot determine from where
a port scan is coming, blocking the port scan and subsequent attacks becomes exceedingly
difficult.

A denial of service attack is an attempt by the attacker to either limit or end the
availability of the target computer. Types of denial of service attacks vary by traffic type
and individual packet purpose, but each attack is meant to cripple the target system with a
volume of traffic that makes the computer unable to respond to normal network requests.
The problem with standard denial of service attacks is that it is easy to trace the computer
performing it and filter all the traffic with a simple firewall rule. ISECAP overcomes this
problem by using its random functions to randomize source IP addresses and port numbers.
If the defending machine is unable to isolate a specific source of the attack, it is
exceptionally difficult to stop the effects. A well-performed, random source, denial of service attack has a high chance of success and is extremely desirable to replicate in a security testing situation.

### 4.3 Cyber Defense Competitions

Cyber Defense Competitions (CDCs) are one of the most popular uses for ISEAGE. ISECAP holds great potential as a tool for the CDC Red Team. In these competitions the Red Team is given a limited range of IP addresses which, experience has shown, are blocked frequently by the competing teams. ISECAP could easily randomize individual Red Team member source IP addresses for all scanning and attack traffic. This would not only prevent the defending teams from blocking the attacker but also likely be of great amusement to the Red Team as the defenders attempt to isolate the source of the scans bombarding their networks.

If implemented on an ISEAGE board or on a machine connected to one of the external team hubs, ISECAP could be used as a launching point for numerous attacks. After a successful ARP poisoning or DNS stealing attack, ISECAP would function as a man-in-the-middle attack device to eavesdrop on team communications and acquire login information and passwords from the defenders. With the man-in-the-middle position, teams could be redirected to load various Red Team software onto their computers or fooled to believe they are not under attack at all. With proper enhancements, the Red Team could use ISECAP to obtain even encrypted information from normally safe protocols.
Chapter 5. LIMITATIONS

Though ISECAP provides great functionality as a security testing tool, it does have some limitations in its ability to perform certain specific tasks. Most of these limitations are a result of the chosen method of development for the FreeBSD platform and systems therein. This section will document many of the limitations to the ISECAP packet filtering system and will suggest methods of overcoming these limitations.

5.1 Network Position

ISECAP functions well within specific environments. Unfortunately, its ability to position itself within a network, especially undetected, is limited. The ideal functioning mode for an ISECAP machine would be to run as a transparent network bridge. Network bridges operate as simple packet forwarding mechanisms. They are not considered a hop on network and therefore can be virtually undetectable. They pass on traffic at the data link layer without performing any sort of MAC or IP address routing. This position would be ideal because other network devices would be incapable of detecting ISECAP and therefore unable to detect changes to packets. Within FreeBSD, however, IPFirewall is limited to only two options when a computer is used as a bridge connection: pass and drop. Without divert sockets functionality, no method exists to pass packets to the ISECAP program.

To solve this limitation, the user must position ISECAP on the network in a similar undetected position by executing an ARP poisoning or DNS stealing attack. ARP poisoning would be ideal as it would cause machines on the network to forward all IP
packets to ISECAP, while unaware of its presence. ISECAP could then forward the modified traffic to the standard network gateway.

5.2 Layer Limitations

Though ISECAP has been shown to perform its function with success at the network layer using the IP protocol, it currently is not able to handle packets at lower layers with non-IP protocol headers. The divert socket functions are limited to IP packets, leaving successful ethernet header or ARP packet diversion and modification outside the scope of this project. This limitation is due to divert sockets originally being designed to work exclusively as a kernel mechanism for network address translation on FreeBSD. Thus far divert sockets has not been updated to include these additional abilities, but doing so would be useful to many FreeBSD developers using divert sockets for non-network address translation tasks.

The simplest solution to this problem would be to use another packet filtering program to handle non-IP protocols. A heavily reduced version of Ettercap could be implemented to handle ARP packets and ethernet headers. In fact, with some modification, the Ettercap engine could be redesigned to simply forward these packets to ISECAP where they could be processed by special modules. This addition would enhance ISECAP’s functionality by providing support for another protocol and another OSI layer.

5.3 Other Limitations

The current implementation of ISECAP’s code contains some issues with transport layer checksum recalculations. Due to the FreeBSD provided TCP header structure not
supporting direct control over the TCP options field, checksum recalculations are occasionally incorrect when certain options field values are set in the packet. The options field is an integral part of the checksum input data, and without an easy method of obtaining these values, complications in the code have caused some checksum calculational errors. This issue likely can be solved with a detailed code review.

Finally, a few limitations exist in the connection tracking system. An efficient connection tracking system must be able to timeout connections after they have expired and cease to track them. ISECAP does not currently possess the functionality to remove entries in a timely matter from the connection tracking hash table. The connection tracking system in general has yet to be tested for full functionality and efficiency.
CHAPTER 6. TESTING

While the design aspects of ISECAP are sound, no system is complete until it has been met with rigorous examination in real world situations. Though ISECAP was developed for use in a network simulator, the performance demands of ISEAGE will be as great, if not greater, than real world situations. This section will detail the testing of ISECAP and suggest future tests to be conducted.

6.1 Ping Test

The first test to which ISECAP was subjected was a basic ICMP distributed ping test. Details of this test can be found in Appendix A. This test was to confirm three areas of functionality. First, the test needed to allow for the proper handling of ICMP traffic. Second, the randomization function needed to function correctly and ISECAP needed to return all randomized traffic to the original host. Third, ISECAP needed to perform all these functions efficiently.

ISECAP was initiated in two instances for this test. One instance was charged with changing ping traffic to random source addresses. The other instance was charged with translating all incoming ICMP traffic to the destination address of the original pinging host. Traffic was captured with tcpdump on both ISECAP test interfaces to ensure functionality.

The results of the ping test prove ISECAP is relatively efficient at handling ICMP traffic and can perform random IP address translation with little delay. The average ping latency for an unfiltered connection through the ISECAP machine gateway was 399 ms.
With ISECAP filtering, the average ping latency was 526 ms. This increase of approximately 32 percent indicates ISECAP’s modification functions are sufficiently fast.

### 6.2 Iperf Test

The second test performed on ISECAP was with a free tool called Iperf. Iperf is designed to test network throughput. It performs this function by transmitting 16 KB packets for 10 seconds on a TCP connection. To run Iperf, one of the external test computers runs an Iperf server while the other runs a client. This test was performed multiple times in two ways. First, Iperf was tested through the ISECAP machine without ISECAP running. Second, the Iperf program was executed with ISECAP running.

As seen in Appendix A, the unfiltered test resulted in a throughput of 95.5 Mb/s. Given that the data rate is limited to a theoretical maximum of 100 Mb/s by the switch through which the connections run, this data rate is quite high and demonstrates the efficiency of FreeBSD gateway operations. The second test, with ISECAP running, resulted in a data rate of 72.2 Mb/s. While not as fast as the direct connection, the results of the Iperf test are encouraging. ISECAP results in a drop of approximately 24 percent in network throughput.

While these results are not ideal, when compared to other packet filtering programs, such as APOC, ISECAP’s performance is exemplary. The same test was performed on APOC, resulting in significantly worse performance [10]. When a straight connection was run on the APOC computer the data rate was 55.5 Mb/s, while the connection with APOC running allowed for only 3.75 Mb/s. This is a performance drop of over 93 percent and is one of the major limiting factors of the APOC system. ISECAP’s performance penalty of
24 percent still allows for a high performance throughput of traffic and meets the goals set forth in this project.

6.3 Nmap Test

The third major test performed on ISECAP was an Nmap port scan test. Nmap is a popular port scanning tool that sends a high number of packets to a wide range of TCP ports on the target machine. Because one of the primary goals of ISECAP is to simulate distributed port scanning attacks, performance on this test is crucial. As seen in Appendix A, Nmap scans were performed with the standard nmap command across the ISECAP computer network. The first test was conducted without ISECAP running. The second test involved two instances of ISECAP. The first instance randomized source IP addresses, thus performing a distributed port scan. The second instance translated the return packet destination addresses to the scanning machine.

Both tests completed successfully and found ports 22 and 80 to be open on the target machine. The straight connection scan finished in 13.276 seconds. The ISECAP filtered connection scan finished in 13.502 seconds. This small increase represents an additional delay of less than 1.7 percent and is virtually insignificant.

Again, a similar Nmap test was performed by APOC using its functionality to randomize source IP addresses. The straight Nmap scan took only 10.885 seconds while the filtered APOC scan required 24.554 seconds, an increase of almost 130 percent. Once again, performance is a constant challenge for packet filtering software, and is one of the driving principles in ISECAP’s design. ISECAP completes this important test with
excellent speed and efficiency, indicating its usefulness as a high performance packet filtering solution.

6.4 Future Tests

Though ISECAP performed well on numerous performance tests, the system has yet to see a live and varied network environment. All tests presented in this chapter were performed as a connection between only two computers. Future testing should take place directly on the ISEAGE environment. With multiple hosts on each side of the system, ISECAP will have much more challenging tasks to perform. One of these important tasks will be to connection track efficiently among many hosts.

Integration into the ISEAGE environment will also provide more varied network traffic. Testing thus far has been limited to a small subset of the possible traffic ISECAP could encounter. New types of traffic testing could involve ISECAP’s ability to handle malicious traffic. By implementing ISECAP at future Cyber Defense Competitions, much of the future testing could be performed quickly within a highly realistic network setting.
CHAPTER 7. CONCLUSIONS

This paper has demonstrated the effectiveness of the ISECAP filtering system. This chapter will conclude the paper with suggestions of future work and a summary of ISECAP.

7.1 Future Work

ISECAP has been specifically designed to be expanded for future uses in the ISEAGE environment. As such, there is extensive room for future additions and improvements. One potential problem previously mentioned was the need for ISECAP to be able to position itself on a network in an undetectable location for packet filtering. If FreeBSD bridging divert sockets support is not added to the operating system in the future, it would be prudent to write a set of modules for ISECAP to execute when initiated. These modules would launch an automated ARP poisoning attack on the current network based on the gateway IP address specified in the initial user script.

ISECAP protocol support could be expanded into the link layer and into non-IP packets by adding modules to pass traffic into ISECAP from other methods. By borrowing some functionality from other packet filters such as Ettercap or the standard Berkeley Packet Filter, functions could run in parallel with ISECAP to capture this data for processing. Such capabilities could extended into other popular protocols such as the IEEE 802.11 standard for wireless medium communication. Also, as IP version 6 becomes more popular, support for its unique header standards should be added.

Though ISECAP provides excellent network and transport layer change capabilities, no specific modules have yet been written to handle popular application layer
data. When this need arises, additions should require only a few functions appended to the current ISECAP module system. Support for popular application layer protocol such as HTTP, DNS, FTP, and POP3 would give a finer control over specific uses of Internet networks.

The final area in need of work is that of ISECAP’s usability. At this point, most of the ISECAP process initiation is done manually. IPFW divert rules are entered at the command line by the user. Change data is loaded into the change data structure by altering the structure loading source file before compilation, and each instance of ISECAP needs to be manually started.

A full script parsing program would make ISECAP much more user friendly. This program would take a basic and easy-to-write script and extract the necessary information for the ISECAP program. ISECAP would then write the conditions into a bash script which would execute from within the program. This bash script would automatically add all requested IPFW divert rules. ISECAP should also gain the functionality to spawn additional instances of itself. This ability would help with everything from more efficient connection tracking to simpler program initiation.

Finally, because ISECAP is meant to run as a part of ISEAGE, it should be integrated into the ISEAGE graphical user control interface DeepFreeze. DeepFreeze is a program for ISEAGE designed to provide a unified command system across all ISEAGE components [13]. It is designed to allow easy integration for all ISEAGE projects. A DeepFreeze graphical control interface would be a strong improvement in the usability of ISECAP.
7.2 Summary

ISECAP provides a solid packet filtering system and a forward-looking framework for extensibility. Though influenced by the design of other available packet filtering systems, ISECAP is unique in its intent and purpose as a scalable system specifically designed for use with ISEAGE. ISECAP implements a modular design, supporting a range of base protocol standards while allowing for future expansion. ISECAP contains a robust pattern matching system, detailed packet altering capabilities, effective randomization functions, and a connection tracking system. These systems come together to provide useful enhancements to ISEAGE’s ability to perform tasks ranging from system research to advanced attack simulation. Though the basic FreeBSD functions used to create ISECAP still have some limitations, the extensible design of ISECAP allows for future development to cover all areas of functionality. While some room for improvement exists, tests of the fully functional ISECAP system have shown it to be an efficient utility. ISECAP is a powerful tool to expand the capabilities to the ISEAGE security test bed and is poised to become even more useful as future needs arise.
APPENDIX A: TEST DATA

Ping Test Data:

Straight connection ping:

```
Ping Test Data:

```

Ping through ISECAP with random source address translation:

```
Ping through ISECAP with random source address translation:
```
Tcpdump on em2 showing altered packets outbound:

Iperf Test Data:

Straight connection throughput:
```
root@lesli:/home/grant# iperf -c 10.10.2.1
-------------------------------------------
Client connecting to 10.10.2.1, TCP port 5001
TCP window size: 16.0 KByte (default)
[ 3] 0.0-18.1 sec 115 MBytes 95.5 Mbits/sec
root@tesli:/home/grant#
```

ISECAP connection throughput:
```
root@tesli:/home/grant# iperf -c 16.16.2.1
-------------------------------------------
Client connecting to 10.10.2.1, TCP port 5001
TCP window size: 16.0 KByte (default)
[ 3] 0.6-30.0 sec 86.2 MBytes 72.2 Mbits/sec
root@tesli:/home/grant#
```
NMap Test Data:

Straight connection port scan:

```
Starting Nmap 4.55 ( http://insecure.org ) at 2008-07-25 21:05 CDT
Interesting ports on 10.10.2.1:
Not shown: 1712 closed ports
PORT      STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Nmap done: 1 IP address (1 host up) scanned in 13.276 seconds

root@test1:/home/grant#

ISECAP port scan with random address translation:

```
root@test1:/home/grant# nmap 10.10.2.1
Starting Nmap 4.55 ( http://insecure.org ) at 2008-07-25 21:03 CDT
Interesting ports on 10.10.2.1:
Not shown: 1712 closed ports
PORT      STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Nmap done: 1 IP address (1 host up) scanned in 13.562 seconds

root@test1:/home/grant#
APPENDIX B: USER SCRIPT EXAMPLES

This appendix contains example IPFW rules and scripts.c information based on user needs. The format is a condition set followed by the example script(s) and rule(s).

Condition: Alter source address of IP packets from 10.10.1.1 to 10.10.10.2

```
myscript->script_change.ipaddr_src = "(enter IP address here)";

ipfw add 100 divert 7000 ip from 10.10.1.1 to 10.10.10.2
```

Conditions: Randomize source address and TCP port of TCP packets from 10.10.1.1
Alter destination address and TCP port of TCP packets to 10.10.1.1

```
myscript->script_change.ipaddr_src = "RANDOM";
myscript->script_change.tcpport_src = "RANDOM";

myscript->script_change.ipaddr_dst = "10.10.1.1";
myscript->script_change.tcpport_dst = "60";

ipfw add 100 divert 7000 tcp from 10.10.1.1 to any
ipfw add 101 divert 7001 tcp from any to 10.10.1.1
```

Conditions: Randomize source address and udp/tcp ports of all ip traffic from 10.10.1.1
Pass on ICMP traffic from 10.10.1.1 unmodified
Drop all return traffic to 10.10.1.1

```
myscript->script_change.ipaddr_src = "RANDOM";
myscript->script_change.tcpport_src = "RANDOM";
myscript->script_change.udpport_src = "RANDOM";

myscript->script_change.change_option = PASS;
myscript->script_change.change_option = DROP;

ipfw add 100 divert 7000 tcp from 10.10.1.1 to any
ipfw add 100 divert 7000 udp from 10.10.1.1 to any
ipfw add 101 divert 7001 icmp from 10.10.1.1 to any
ipfw add 102 divert 7002 ip from any to 10.10.1.1
```
APPENDIX C: ISECAP SETUP INSTRUCTIONS

The following are the basic instructions to setup a computer to run ISECAP:

1. Install FreeBSD 6.2 or higher on a machine with at least 3 ethernet interfaces.

2. Compile the kernel with the following options:

   options IPFIREWALL
   options IPFIREWALL_DEFAULT_TO_ACCEPT
   options DIVERT

3. Add the following lines to the /etc/rc.conf file (hostname line is optional):

   gateway_enable = "YES"
   firewall_enable = "YES"
   hostname = "isecap"

4. Configure the two ISECAP network cards to static IPs in the /etc/rc.conf file. An example is shown here:

   ifconfig_em1 = "inet 10.10.1.100 netmask 255.255.255.0"
   ifconfig_em2 = "inet 10.10.2.100 netmask 255.255.255.0"

5. Restart network services with: #/etc/rc.d/netif restart.


7. Compile ISECAP with gcc command stored in “make” file.

8. Enter appropriate IPFW divert rules with ipfw add command.

9. Configure external computers to use ISECAP IP addresses as their default gateways.

10. Run ISECAP with the port number indicated in the corresponding IPFW divert rule.
BIBLIOGRAPHY


ACKNOWLEDGEMENTS

I would like to first thank my committee members Drs. Jacobson, Nilakanta, and Daniels for their support of this project and of my graduate student career at Iowa State. I would like to offer a special thank you to Dr. Jacobson who has given me the great opportunity of working with a world-class security research facility at ISEAGE.

Most importantly, I would like to thank my family. Without their constant and unconditional love and support, throughout my university career and all of life, I would not be where I am today.