Information Hiding in Networks
Covert Channels

Rubén Ríos del Pozo
Information Hiding in Networks

This final year project has been submitted by Rubén Ríos del Pozo to the University of Skövde, as a dissertation towards the degree of Bachelor of Science (B.Sc.) in the School of Humanities and Informatics.

The project has been supervised by Jesper Holgersson.

May 2007

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature: ____________________________________________
Information Hiding in Networks

Rubén Ríos del Pozo

Abstract

Covert Channels have existed for more than twenty years now. Although they did not receive a special attention in their early years, they are being more and more studied nowadays. This work focuses on network covert channels and it attempts to give an overview on their basics to later analyse several existing implementations which may compromise the security perimeter of a corporate network. The features under study are the bandwidth provided by the channel and the ease of detection. The studied tools have turned out to be in most cases unreliable and easy to detect with current detection techniques and the bandwidth provided is usually moderate but they might pose a threat if not taken into consideration.

Keyword: Network Security, Covert Channels, Bandwidth, Detection, Covert TCP, ICMP Ping Tunnel, HTTP Firepass, Ozyman DNS Tunnel.
Acknowledgements

This work would not have been possible without the invaluable help of my supervisor, Jesper Holgersson. Thank you for your comments, your suggestions, your guidance and for putting up with me all this time. I would also like to thank my examiner, Björn Olsson, for his wise advices. Finally, but not for that reason less important, I want to thank my family and friends for their support. To those of you who are in Spain and especially to all the new and good friends I made in Skövde. I will never forget you.
# Contents

1 Introduction 1

2 Background 3

2.1 Security Need 3

2.2 Information Security 4

2.3 Network Security 6

2.3.1 Cryptography 7

2.3.2 Information Hiding 10

2.3.3 Steganography 11

2.3.4 Covert Channels 12

2.3.5 Network-based Covert Channels 13

2.3.6 Smeets and Koot’s Research Report 16

3 Problem definition 18

3.1 Introduction 18

3.2 The problem 19

3.3 The aim and objectives 19

4 Research Approach 21

4.1 Introduction 21
4.2 The Work Process ............................................. 21

5 Results ......................................................... 24

5.1 Network Configuration ........................................ 24
5.2 Craig Rowland’s Covert_TCP ................................... 26
  5.2.1 Implementation Study .................................... 26
  5.2.2 Literature Study .......................................... 28
  5.2.3 Testing period ............................................. 29
5.3 Daniel Stødle’s Ping Tunnel .................................... 30
  5.3.1 Implementation Study .................................... 30
  5.3.2 Literature Study .......................................... 32
  5.3.3 Testing period ............................................. 33
5.4 Alex Dyatlov’s Firepass ......................................... 34
  5.4.1 Implementation Study .................................... 34
  5.4.2 Literature Study .......................................... 35
  5.4.3 Testing period ............................................. 36
5.5 Dan Kaminsky’s Ozyman ........................................ 37
  5.5.1 Implementation Study .................................... 37
  5.5.2 Literature Study .......................................... 39
  5.5.3 Testing Period ............................................. 40

6 Conclusions and Future work .................................... 42

  6.1 Analysis and Conclusions .................................... 42
  6.2 Future work .................................................. 44

Appendices .......................................................... 50
A Covert_TCP Measurements

A.1 Network throughput ........................................... 51

A.2 Testing Covert_TCP ........................................... 53
  A.2.1 IP Identification field .................................... 54
  A.2.2 TCP Sequence Number field ............................... 56
  A.2.3 TCP Acknowledgement Through a Bounce Server ........... 57

B Ping Tunnel Measurements ................................ 59

B.1 Tunnel throughput ............................................. 59

B.2 Ping Tunnel Tests ............................................. 61

C Firepass Measurements ..................................... 63

C.1 Configuring Firepass ......................................... 63

C.2 Firepass throughput .......................................... 64

C.3 Firepass Tests ................................................ 65

D Ozyman Measurements and DNS Tips ....................... 69

D.1 Introduction to DNS ......................................... 69

D.2 Testing Ozyman .............................................. 71
List of Tables

2.1 OSI Layers. From Pfleeger and Pfleeger [2003], Page 373. .......................... 14

5.1 Test Schedule followed. ............................................................................. 25
5.2 Summary of Covert TCP Results. .............................................................. 30
5.3 Summary of Ping Tunnel Results. .............................................................. 33
5.4 Summary of Firepass Results. ................................................................. 37
5.5 Summary of Ozyman Results. ................................................................. 41

6.1 Results Comparison. .................................................................................. 43

A.1 IP Identification Field Results. ................................................................. 55
A.2 TCP Sequence Number Field Results. .................................................... 57
A.3 TCP Acknowledgement Field Results. ..................................................... 58

B.1 Ping Tunnel Results. .................................................................................. 62

C.1 Firepass Results for 0.5 seconds delay. ...................................................... 66
C.2 Firepass Results for 1 second delay. .......................................................... 67
C.3 Firepass Results for 5 seconds delay. ........................................................ 67

D.1 Ozyman’s Results with 0.01 seconds delay. ............................................. 76
D.2 Ozyman’s Results with 0.05 seconds delay. ............................................. 76
D.3 Ozyman’s Results with 0.1 seconds delay. ............................................. 76
D.4  Ozyman’s Results with 0.4 and 0.5 seconds delay.  . . . . . . . . . . .  77
List of Figures

2.1 Information security diagram. From Ålfeldt [2006]. ................. 5
2.2 Secret key cryptography. Based on Rosenberg and Remy [2004]. ....... 8
2.3 Public key cryptography. Based on Rosenberg and Remy [2004]. ....... 9
2.4 Digital signature and verification. Based on Rosenberg and Remy [2004]. .... 9
2.5 Information hiding classification. Based on Pfitzmann [1996]. ........ 11

4.1 The Work Process. .............................................. 22

5.1 Network Configuration. .......................................... 25
5.2 TCP and IP Headers. ............................................. 26
5.3 TCP Acknowledgement with bounce server. .......................... 28
5.4 Ping Tunnel Communication ...................................... 31
5.5 ICMP Packet Layout .............................................. 32
5.6 Windows XP vs. Linux ICMP traces. ................................ 33
5.7 Firepass Configuration. ........................................... 34
5.8 Firepass Forged Header. .......................................... 36
5.9 DNS Tunnel Communication. ...................................... 38
5.10 DNS Tunnel Work Flows. ...................................... 39

D.1 DNS Hierarchical Tree. .......................................... 70
D.2 DNS Tunnel Trace. .............................................. 75
Chapter 1

Introduction

Information hiding techniques received in the last years much less attention from
the research and non-research community than other fields, such as cryptography. This
situation is now changing, and the growth of interest in this field has been exponential
since, in 1996, the First International Conference on Information Hiding took place in
Cambridge (United Kingdom).

Covert Channels are a kind of information hiding. They are defined as a way of
transmitting information in such a way that violates the system’s security policy. However,
the main topic in this work will be network-based covert channels, which are intended to
be a way of concealing information while it passes over a network.

There are many implementations of network covert channels on the Internet, and these
can be used not only for hiding communications to the eavesdropper but also to make
tunnels over other trusted protocols within a firewalled network, to extract information
from a compromised host, watermark packets (i.e. introduce unseen marks) in order to
keep track of them, etcetera.

This work will focus on the analysis of several network covert channel tools that may be
used to obtain information from trusted networks (i.e. which utilizes security mechanisms)
and tools used to create tunnels over authorized protocols in such networks. This kind of
infrastructures is common in corporate or educational environments.

The study is based on two main characteristics of covert channels, the bandwidth
provided and the “detectability” of the channel (i.e the probability of passing unseen).
These features will give insight in the possibilities of introducing a covert channel in a
network under surveillance, as well as, give hints on how to prevent the use of covert channels whose intentions are to break the security policies. For example by leaking confidential information out of the network.

According to the results obtained from our research, most of the implementations chosen are easily detectable with current solutions. The point is that these are often theoretical solutions which have not been implemented yet but in other cases there are practical solutions, such as Web Tap used for the detection of HTTP covert traffic.

On the other hand, the maximum bandwidth provided is, in general, moderate and there are usually errors during the transmission when the implementation is working at full pace. Furthermore, the higher throughput of the implementation, the higher risk it may involve since the amount of data that might be leaked out of the compromised network is bigger.
Chapter 2

Background

The following chapter will introduce the reader in the main concepts of the key topic in this project, covert channels. The chapter will start by giving an overview of the field of activity of covert channels, computer networks. As networks are in constant risk they need to be secured. In this point, a brief introduction to Information Security is done and, subsequently, a mention of the three basic pillars in security (i.e. Confidentiality, Integrity and Availability), the threats to which these are exposed to and how to keep them safe.

Next, we focus on network security and how cryptography has been and is still being the main mean of defence to the pillars of security. Furthermore, both symmetric-key and asymmetric-key cryptography are briefly explained.

Finally, Information Hiding is presented together with two out of its four subcategories, Steganography and Covert Channels. A brief history and classification of steganography is given and, to conclude, covert channels and network-based covert channels are presented.

2.1 Security Need

Computer networks have evolved so rapidly within the last few years that nowadays it is unusual to think of a computer without at least one network interface or Internet connection, whether by means of Ethernet cable (IEEE 802.3) or wireless interfaces as Wi-fi (IEEE 802.11). In fact, having an Internet connection is a ‘mandatory’ issue in most of nowadays PCs, mainly, thanks to the drop of the price and the increase of the
bandwidth offered by the telecommunications companies.

Most of the computers in the world are interconnected by means of either LANs (Local Area Networks) or Internet. However, this is not only an issue of the corporative world, it is becoming more and more common to have LANs in personal areas to share information between several computers in a house or building. Nevertheless, these computers are not only connected to each other but, in most of the cases, also to the Internet. Considering this, they are not exempt from the problems that openness implies. Openness refers to the possibility of accessing information which is located at any place in the world, at any time. For this reason, from the early years of computer networks, network security has been an important topic to work in.

2.2 Information Security

Information security is a broader term which covers both technical and administrative security (see Figure 2.1). The latter concerns the management of the necessary policies to achieve security, whereas technical security concentrates on the measures to utilize in order to achieve it. Technical security also refers to physical and information-technology security. The former is related to physical access to the information and IT security refers to security in technological systems. Finally, IT security is subdivided on computer and communication security [Åhlfeldt, 2006]. Computer security refers to the protection of the very computer and its content, whereas communication security refers to the protection of the assets utilized to interconnect computers or other devices.

A definition of information security can be found in [Longley and Shain, 1989, Page 169]: “The protection against the unauthorized disclosure, manipulation, destruction or alteration of information”. In fact, it refers to the protection of all types of information while, later, in the same page it is possible to read a narrower definition, now for information systems security: “the protection afforded to information systems in order to preserve the availability, integrity and confidentiality of the systems, and information contained within the systems”. Therefore, these are the basic pillars on which information security is based on and they are known as the CIA triad [Russell and Gangemi, 2006, Chap. 1.2]:

i. Confidentiality (or Secrecy in [Tanenbaum, 2003]): “Data is confidential if it stays
obscure to all but those authorized to use it”. Which means that data should be only available to those who have the proper rights to access it.

ii. Integrity: “Data has integrity as long as it remains identical to its state when the last authorized user finished with it”. Meaning that any alteration of the data by an unauthorized user or process would deprive it of its integrity.

iii. Availability: “Data is available when it is accessible by authorized users in a convenient format and within a reasonable time”. That’s to say, a legitimate user should not be hindered when trying to access the data.

These features are constantly under the threat of many menaces. Stallings [1997] divided the threats into four main types:

- Interruption: one or more resources become unreachable or unusable, leading to an attack to the availability.

- Interception: an unauthorized party gains access to an asset without interrupting it. Therefore, the availability is not compromised but the confidentiality is.

- Modification: an unauthorized party gains access but also modifies the asset. In this case, integrity is put at risk.

- Fabrication: an unauthorized party introduces forged elements. This is a threat to integrity and also to confidentiality if the attack is committed in order to get information from the other party.
In order to enforce the security requirements proposed on the CIA triad, information systems build a first line barrier. This preventive barrier is known as access control, which is mainly implemented by user authentication mechanisms. Authentication determines whether a user is actually who he/she claims to be. Hence, a legitimate user will be able to enter the system, run certain processes, etcetera, whenever the authentication mechanism does not fail. In order to provide authentication, most of the systems resort to [Tanenbaum, 2001]:

- the use of passwords, which is the most commonly used,
- the possession of tokens (e.g. smartcards),
- the measurement of biometric features in the user (e.g. fingerprints), or
- even hybrid methods, consisting of the use of several methods at the same time, such as both a token and a password.

Despite the access control, information systems are prone to suffer from security attacks which are successful in many occasions. In such circumstances, it is necessary to protect the system with secondary barriers, such as encryption of data, anti-viruses, etcetera.

### 2.3 Network Security

Network security is also an issue of real interest and it is currently gaining a huge importance due to the fact that suffering attacks is becoming more common due to the aforementioned openness. The use of Internet, not only for academic use but also for business and entertainment, has lead to a massive arising of new services, such as e-commerce, e-banking, e-voting, etcetera. These security-critical services would not be possible without network security.

According to Stallings [1997] it is possible to differentiate between two different types of attacks to the network:

- Passive attacks: called this way because the data which passes over the network is only intercepted, that’s to say, the attacker merely observes the information,
therefore, they are unlikely to be detectable. This type of attacks is a threat to confidentiality. Eavesdropping and traffic analysis are examples of passive attacks.

- Active attacks: in this type, the attacker not only observes the communication but might also interrupt, modify or fabricate data. Active attacks are easier to detect. Examples of active attacks are:
  
  - Denial of Service (DoS), which consists of the interruption of the normal use of the communications. It has a negative influence on availability.
  
  - Masquerade or spoofing, in which an entity impersonates another entity. This causes an authentication problem.
  
  - Modification of data between two parties, which has a negative effect on integrity.

In order to have a secure communication through the network, in addition to authentication and the features proposed on the CIA triad, it is necessary to provide non-repudiation. Non-repudiation is defined as “The property of a receiver being able to prove that the sender of some data did in fact send the data even though the sender might later desire to deny ever having sent that data” [Atkinson, 1995, Page 2]. In simpler words, non-repudiation permits to each end to certifiably prove that the other was involved in the communication. All these features, necessary to secure a communication, are commonly obtained by the use of a technique called cryptography. How it is done, depends on the type of cryptography utilized.

### 2.3.1 Cryptography

The word cryptography derives from the ancient Greek (kryptos and graphein) which means hidden writing [Tanembaum, 2003]. Therefore, cryptography is the science that is used to encrypt and decrypt data, that’s to say, conceal and retrieve information given a secret key. Although a Greek word is used to designate this science, it was already used in a town of Egypt nearly 4000 years ago, as it is stated in Kahn [1996]. However, these were not based on mathematical formulae as nowadays, but in the use of hieroglyphic symbol substitutions.

It is possible to distinguish two main types of cryptography [Network Associates and
• **Secret-key** or **symmetric-key** cryptography:
The secret key is both utilized to encrypt and to decrypt (see Figure 2.2), that is the reason to call it symmetric. Therefore, anyone who knows the key is able to encrypt or decrypt and consequently the key should only be known by the two parties who are to communicate (secret). As the key must be secret, this provides both confidentiality and non-repudiation. Moreover, in case of modifying the ciphertext, the resulting plaintext after decrypting will have no sense, which provides integrity. The most well-known examples are DES (*Data Encryption Standard*) and AES (*Advanced Encryption Standard*).

![Figure 2.2: Secret key cryptography. Based on Rosenberg and Remy [2004].](image)

• **Public-key** or **asymmetric-key** cryptography:
This concept was introduced by Diffie and Hellman [1976]. In this type of algorithms there are two different keys, one private (kept in secret) and another public, published to the world. Non-repudiation is possible since both keys are necessarily related but it is computationally impossible to obtain the private one from the public. Usually, the public key is used to encrypt data that will only be readable by the owner of the private key (confidentiality) (see Figure 2.3). It is also possible to use the private key to cypher and anyone with a copy of the public key will be able to obtain the data. These are the bases for the digital signature, which provides authentication (see Figure 2.4). These types of algorithms are based on mathematical problems which required an exponential time to be completed (*Non-Polynomial* problems), such as the factorization of big integers or the discrete logarithm problem. Precisely, the most famous algorithm for asymmetric-key cryptography is based on these. Its name is obtained from its authors’ initials: RSA (*Rivest, Shamir, and Adleman*) [Rivest et al., 1978].
As in secret-key cryptography, integrity is obtained since any modification in the cipher text will result in an unreadable text when decryption is done.

![Diagram of public key cryptography](image1)

Figure 2.3: Public key cryptography. Based on Rosenberg and Remy [2004].

![Diagram of digital signature and verification](image2)

Figure 2.4: Digital signature and verification. Based on Rosenberg and Remy [2004].

Symmetric-key cryptosystems present two main problems due to the fact that these utilize the same key both for encryption and decryption. The first problem is that if the key must be secret in order to ensure a confidential communication, this key should be unique for every two parties. That’s to say, if a user A wants to communicate with another B, they must share a secret key, let’s say $K_{AB}$. However, if A would like to get in contact with a new user C, they should utilize a different key to $K_{AB}$, let’s say $K_{AC}$. The reason is that if $K_{AB} = K_{AC}$, either B or C could intercept a communication between A and the other party and retrieve its content. The second problem refers to the interchange of keys between the two ends. This is the *chicken-and-egg* problem for the authentication of the parties. Two users need a secret shared key to be able to communicate safely, but in order to achieve this there are two possibilities, either agree it in person or through Internet, which is an unsafe environment by definition. In the second case, it is necessary to protect the key from the eavesdropper by utilizing a secure channel, but to obtain it, a new secret key is needed, and so forth. In fact, several protocols were developed [Needham and Schroeder, 1978], [SPORE, 2002], [Otway and Rees, 1987] to solve this
mutual authentication problem, but it was necessary to use TTP ([Third Trusted Party](#)). Anyway, all these protocols were studied and, finally, a successful compromise was found.

The concept of public-key cryptography solved the *chicken-and-egg* problem thanks to the fact that the public key (the one used to encrypt messages) was freely distributed, while the private key is kept in a safe place. On the other hand, asymmetric cryptography is not the final solution, it has its own drawbacks, which are mainly:

- **Computational cost**: Asymmetric cryptography utilizes slower encryption/decryption methods than those used by symmetric-key cryptosystems.

- **Authentication problem**: Normally, public keys are uploaded to a public repository ([Verisign, 2007](#)), but how can a user know if a certain key really belongs to the person it indicates. There is a need to use *digital certificates* signed by a Certification Authority (*CA*) which states that the certificate is authentic, that’s to say, that a certain public key corresponds to a certain user. CAs are hierarchically organized into trust chains where some CAs are certified by other CAs of higher level in order to build what is called a PKI ([Public-Key Infrastructure](#)). The main problem is solved with the use of certificates, but these can not last forever because a user might lose the private key or it might be compromised. Whenever a key is no longer valid it is added to a CRL ([Certificate Revocation List](#)) in order to warn other users that the certificate in question is not valid, so these CRLs should be constantly updated. For this reason CRLs become the biggest bottleneck of the whole system.

### 2.3.2 Information Hiding

Network security is mostly associated with cryptography, however, an alternative to cryptography is called *Information Hiding*. Information hiding is divided on several sub-disciplines as it is possible to observe in Figure 2.5. Since there is limited amount of time for the completion of this project this work will focus on two of them, steganography and covert channels, paying more attention to the latter.
2.3.3 Steganography

The word steganography comes from the Greek and literally means “covered writing”. Steganography is more than protecting the content of a message, it is also about concealing its very existence [Petitcolas et al., 1999]. Petitcolas et al. [1999] continue by saying that this is normally done by hiding the information within other information. This is the main difference between steganography and cryptography. The presence of a ciphertext may alert an attacker that some information of relevance is protected, but steganography would make the information pass unnoticed to the eyes of the attacker, reducing the risk of a possible attempt to recover the information.

According to Pfitzmann [1996] (see Figure 2.5) it is possible to distinguish between two types of steganography depending on where the hidden message is obscured:

i. Linguistic steganography: the carrier of the message is text. There are many ways of linguistic steganography, an example cited on Katzenbeisser and Petitcolas [2000] refers to the ancient China where paper templates with holes were used to hide messages. These holes were to identify the position of the covert words in the entire text. As both the sender and the receiver shared the same template, the receiver just had to place it over the text to easily retrieve the message. Another interesting example is Spammimic.com, a web-based tool which generates spam messages covering the actual information. The underlying idea is that most of the people just ignore spam.
ii. Technical steganography: the carrier is not text, but any other physical medium. Some examples of technical steganography are narrated in a book of Herodotus [1992] where he tells about a tattooed message on the shaved head of a slave which disappeared after the hair had regrown. Other examples are the use of microdots, the use of invisible inks, etcetera [Katzenbeisser and Petitcolas, 2000].

Nowadays, there are several ways of invisible communications. Katzenbeisser and Petitcolas [2000] refers to some of them, such as those taking advantage of the noise component of both digital images and digital sound, the weakness of digital signature algorithms (i.e. ElGamal), hiding data in executable files or sending sensitive information between processes from different security-level areas.

### 2.3.4 Covert Channels

The concept of covert channel was introduced by Lampson [1973]. Lampson defines a covert channel as a communication channel that is neither designed nor intended to transfer information at all. Later, the U.S. Department of Defense defined covert channel as “any communication channel that can be exploited by a process to transfer information in a manner that violates the system’s security policy” in [NCSC, 1985, Section 8].

Covert Channels are best understood by considering the prisoners’ problem, first formulated by Simmons [1983]. In this scenario, Alice and Bob are in prison and they are attempting to escape. They are allowed to communicate, however, Walter (the warden) has access to all the notes they interchange but he cannot modify them. If Walter notices they are developing a plan to escape, he will allow no more communication. There might be two types of wardens, passive and active. The former, as aforementioned cannot modify the messages whereas active wardens can.

It is possible to distinguish two main types of covert channels according to the NCSC [1985]:

- Storage covert channels: are those which allow the writing of a storage location by one process and the retrieval of that information by another. An example given in [Pfleeger and Pfleeger, 2003]: two processes communicate by the creation and deletion of files in a certain directory, in such a way that if the file exists the sender would be transmitting to the receiver a value of 1, otherwise 0.

12
• Timing covert channels: are those in which a process changes its way of using the system resources in order to signal information to another process. For example, Pfleeger and Pfleeger [2003] propose a multiprogrammed system with only the two communicating processes. The first process will use its CPU quantum of time in order to signal a 1 or rejects it to signal a 0.

Although Gligor [1993] stated that there was no fundamental distinction between storage and timing channels, other authors (e.g. Cabuk et al. [2004]) have kept on using it. Even this classification has been extended in other works by identifying also:

• Hybrid channels: they combine both methods mentioned above to conceal the information.

• Counting channels: these channels were proposed by Gray and III [1999]. The peculiarity of these channels is that the information is codified as the number of repetitions of a single event.

As well, other new taxonomies have been proposed. For example, Meadows and Moskowitz [1996] suggested high-to-low service, low-to-high service, shared service and incomparable service covert channels. This classification is based on the levels of the services provided by the different processes involved in the communication.

There are several apparently contradictory properties that an effective covert channel might have [Giffin et al., 2002]. Plausibility is a property for which a warden must believe that the user of the channel does not utilize it to send covert data. Therefore, its usage must not influence the normal use of the carrier protocol. The Undetectability states that the covert data must follow the same rules as the data in the normal channel, otherwise the channel might be detectable with the use of statistical analyses. Indispensability means that the carrier of the channel must be something useful in the system so that a warden cannot decide to interrupt that asset or service.

2.3.5 Network-based Covert Channels

The previous definition was mainly based on multilevel computer system security but a definition of network covert channels may be found in the first page of Sbrusch [2006].
Network covert channels are defined as the “manipulation of a communication protocol to transfer information in a way outside the protocol’s specification”. This is normally achieved by the use of undefined or optional fields in the header of the network protocols.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Name</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
<td>User-level data.</td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
<td>Standardized data appearance, blocking, text compression.</td>
</tr>
<tr>
<td>5</td>
<td>Session</td>
<td>Sessions or logical connections between parts of an application; message, sequencing, recovery.</td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
<td>Flow control, end-to-end error detection and correction, priority service.</td>
</tr>
<tr>
<td>3</td>
<td>Network</td>
<td>Routing, message blocking into uniformly sized packets.</td>
</tr>
<tr>
<td>2</td>
<td>Data Link</td>
<td>Reliable data delivery over physical medium; transmission error recovery, separating packets into uniformly sized frames.</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
<td>Actual communication across physical medium; individual bit transmission.</td>
</tr>
</tbody>
</table>

Table 2.1: OSI Layers. From Pfleeger and Pfleeger [2003], Page 373.

Therefore, it would be possible to find covert channels in basically any of the different layers from the OSI (Open Systems Interconnection) model (see Table 2.1), it just depends on the “weaknesses” of the communication protocols at the different levels. So far, the current research has supplied the scientific community with the following [Sbrusch, 2006]:

- **Data link layer**: the main problem that covert channels within this layer presents is the applicability. The reason is that whenever a packet passes through a router the second-level headers are replaced, so they must be used within the same LAN. Another problem is the necessity of managing with low-level layers of the OSI model. HICCUPS is a theoretical model developed by Szczypiorski [2003] which takes advantage of the interferences and noise inherent in wireless communications to conceal messages. It is a theoretical model because, according to Sbrusch [2006], Szczypiorski could not find any wireless card which allowed the manipulation of the checksums of the packets, necessary for the model to work.

- **Network layer**: the problems presented by the previous layer are reduced at this level. The packets can travel through WANs (*Wide Area Networks*) and it is easier
to deal with the programming. Several protocols have been used to create covert channels. Mainly, IP (Internet Protocol), in both IPv4 (e.g. Rowland [1997] and [Ahsan, 2002]) and IPv6 (e.g. Graf [2003]); but also others such as ICMP (Internet Control Message Protocol) [Daemon9 and Alhambra, 1996] [Daemon9, 1997], IPSec (IP security) and IGMP (Internet Group Management Protocol) both by Ahsan [2002].

Transport layer: the main protocols at this level are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). According to Sbrusch [2006] most research focuses on TCP because of the number of fields, twelve, unlike UDP, with only four. One of the most known implementations is Covert_TCP from Rowland [1997]. In this layer, it utilizes the TCP sequence number field from the first packet in the 3-way-handshake (when a connection is to be created between two entities), since it contains a random initial sequence number (ISN) in order to avoid the confusion between two possible packets in the network [Postel, 1981].

Application layer: it is a high layer and for that reason, it “presents limitless opportunities for delivery of covert data” [Sbrusch, 2006, Page 7]. The reason for this is that the covert data may not only be in the headers [Carrillo et al., 2003], but also in the payload, as occurs in [Castro and Gray-World-Team, 2006] where HTTP cookies are used to create a stealth channel. The Hypertext Transfer Protocol (HTTP) has not been the only mean to conceal information, there has also been some implementations on DNS (Domain Name System). According to Smeets and Koot [2006], one of the more popular tools is Ozyman, designed by Dan Kaminsky.

In addition to the properties proposed in Giffin et al. [2002], Bauer [2003] presents two other desirable properties when covert channels are used for anonymous communications. “Unlinkability”, which means that even if an observer determines that messages are being delivered, does not learn anything from this and is not able to identify who communicates with whom. The other property is “Unobservability”, which is a stronger property, meaning that an observer is not able to even detect if messages are being delivered.

Furthermore, covert channels may be characterized depending on several issues [Smeets and Koot, 2006], such as the behaviour of the channel (i.e. Passive or Active), the way of communicating between the sender and the receiver (i.e. Direct, Indirect or Spread), the efficiency of the channel (i.e. amount of data sent) in bits/bytes per packet or bits/bytes
per second, whether the channels are synchronized or not, etcetera. At least, the two first characterizations are specific for network covert channels.

A covert channel is *passive* if it makes use of the traffic generated by other applications or processes to convey its own data, while an *active* channel generates its own traffic. The choice between a passive or an active channel depends on which feature is more desirable, either the bandwidth or the likeliness of passing unnoticed.

Additionally, depending on the way the channels transmit the information, it is possible to identify *direct* channels, in which the sender communicates directly with the recipient without the need of a middleman; *indirect* in which intermediate servers (also known as bounce servers) are used to send the information to the recipient; and, finally, *spread* channels utilize not only one bounce server but multiple servers. Avoiding a direct communication provides a stealthier channel (i.e. more difficult to detect). Furthermore, if several intermediate servers it is even more difficult to detect the communication.

In short, network covert channels are becoming more and more popular and new studies and implementations are constantly arising. This great interest in concealing information has its advantages, if used in the right way, but can also be utilized in a manner that may suppose a threat if security policies do not take them seriously.

### 2.3.6 Smeets and Koot’s Research Report

Smeets and Koot [2006] is a research report which was accomplished as part of the Master of Science study in the field of Systems and Network Engineering at the University of Amsterdam. This document tries to give an overview of the state of research in covert channels.

This report introduces basic characteristics from covert channels, such as a taxonomy, attributes, etcetera. Furthermore, it pinpoints the possibility of using covert channels in order to compromise a corporate infrastructure giving details on the network protocols that are being used to convey covert data. As well, it describes how this can be done and points out some general techniques that could be used to prevent covert channels from being successful.

Finally, it presents an appendix with the results from the tests developed on some
network covert channel tools: a modified version of *covert-tcp*, *firepass*, *ptunnel* and *Ozyman*. They encountered problems during the execution of some tests. None of the tests developed with *firepass* resulted in a complete file transfer, and surprisingly, the number of missing bytes was always a power of two. Furthermore, when using *Ozyman* they also encountered that there were problems when transferring files, so that they suppose this application is not as reliable as its author claim it to be.
Chapter 3

Problem definition

The current chapter presents two main problems that network covert channels may imply when used in the wrong way, especially within corporate networks. Subsequently, an introduction to the problem to deal with is given. Finally, we present the aim of this work and the objectives that shall be accomplished in order to achieve this aim.

3.1 Introduction

Secrets in corporate environments are extremely valuable. Therefore, the revelation of those secrets (e.g. confidential information from a product or a commercial strategy) may result in a great loss, not only in terms of money but also in terms of prestige. Corporate espionage might be an extremely damaging threat for companies, and having knowledge on steganography in general and covert channels in particular might help them to avoid being spied on.

Competitors may offer incentives to certain employees with high privileges in the company to convey confidential information. How can this confidential information be conveyed? A possible answer is with the use of covert channels. A process may leak information from secret files to another process in a machine out of the security perimeter of the company. This information would be concealed within the header field in several network packets.

Another possible concern for a company might be the use of forbidden services with the help of tunnels over, at first glance, trusted protocols. Network covert channels may
also be used with that purpose. In fact, there are several already existing implementations available that utilize such important protocols as DNS or ICMP (see Section 2.3.5).

3.2 The problem

There are multiple implementations of covert channels for network environments (e.g. Rowland [1997], Daemon9 [1997], Castro and Gray-World-Team [2006]). These implementations were developed for different situations (e.g. tunnel IP over DNS), purposes (e.g. obtain free Internet access) or even different network environments (i.e. LANs or Internet).

Among all these implementations there are possibilities to set up covert channels within corporate environments in order to evade security policies, gain access to forbidden services over unblocked ports (e.g. HTTP) or to leak out confidential information from the company.

Some work has previously been carried out in this field. Smeets and Koot [2006] present the current state of the research in network covert channels, giving an overview of the most common channels utilized at the moment (i.e IP, TCP, ICMP, HTTP and DNS). Finally they perform tests for several tools in order to obtain information about their efficiency and performance. There is some research done on countermeasures to identify the use of covert channels, but this information is given in general and not for the specific applications tested. Furthermore, they encountered some problems during the execution of some of the implementations such as incomplete file transfers.

3.3 The aim and objectives

The aim of this work is to evaluate TCP/IP-based implementations of covert channels with respect to the bandwidth and probability of detection. This work will give insight in the possibilities and limitations of utilizing covert channels within the corporate world, helping system/network administrators to deal with this threat. Since, from the point of view of a potential attacker, covert channels offer the possibility to leak information out of the network. Therefore, this project will extend the work presented in Smeets and Koot [2006] by investigating specific means of detection for the studied implementations and a
revision of the bandwidth will be done since Smeets and Koot encountered problems.

We will focus on TCP/IP implementations since they comprise about 96% of the traffic in computer networks [Singh et al., 2004]. The choice of the stealthiness of the channel and the amount of bits transmitted per packet or session (i.e. bandwidth) as the features to study is due to the fact that these are noteworthy features in covert channels. Furthermore, these features are basic since the higher bandwidth an implementation presents, the bigger the amount of information leaked might be; and the ease of detection may let the covert channel pass unnoticed or, on the contrary, be revealed and consequently eliminated.

In order to achieve the aforementioned aim it will be necessary to obtain the following objectives:

- **Objective 1:**
  Undertake a survey on the implementations studied in Smeets and Koot [2006] to determine how these utilize network protocols to create covert channels. A special attention will be paid to the information given by the author(s) of each implementation in terms of bandwidth and probability of detection, in case this information is provided.

- **Objective 2:**
  Study which types of techniques have been developed to avoid the use of covert channels over TCP/IP networks. This study will determine whether the selected applications are easy to detect with the current solutions or, on the contrary, they may remain stealthy.

- **Objective 3:**
  Develop tests from the different implementations in order to identify the maximum bandwidth that each of them can provide, since Smeets and Koot [2006] encountered problems while performing these tests.
Chapter 4

Research Approach

This chapter will present the methodology that will be followed in order to undertake each of the objectives previously proposed. The accomplishment of these objectives will turn into the obtaining of the pursued aim.

4.1 Introduction

It is necessary to develop a working strategy before starting the real work. The reason for this is that having a pre-established line of work will give the work a solid base. As well, it will be better understood how the different activities, which must be accomplished in order to achieve the different objectives, are related and how these might be tackled.

4.2 The Work Process

In order to obtain each of the objectives necessary to achieve the aim we will develop a dynamic work flow, that’s to say, the different activities might be carried out in parallel. The reason to work this way is to be efficient, as the amount of time is quite limited.

The work flow will be established in such a manner that each of the objectives proposed in the previous chapter will be obtained from a well-defined method. The working process is clearly presented in Figure 4.1.

The Study of existing implementations task corresponds to objective 1 in section 3.3.
Within this period, the explanations given by the author of each implementation will be reviewed in order to obtain information about the techniques utilized to conceal the data within the network packets. As well, we expect to get some information on the estimated bandwidth provided by the implementation and possibly some information about existing means of detection that may compromise its stealthiness.

Another approach could have been to perform a study of the code from each of the implementations, however, this approach was discarded due to the fact that it is always a difficult task to understand a code that a third party has written and it would take much more time to accomplish the task. On the other hand, we consider that this approach might give a better understanding of the different applications.

The second objective is captured into the task named Literature Study. The purpose of this task is to find out which of the arisen techniques could be utilized in order to reveal the use of the implementations under scrutiny within a corporate network environment. The information will be obtained from technical papers related to the detection of covert channels.

This objective could have been tackled by the utilization of already existing detection
tools. However, this approach has been discarded due to time limitations.

The last task corresponds to objective number 3. Practical tests will be developed to obtain empirical results concerning the bandwidth. These tests will be carried out under quasi-ideal conditions, in a non-congested local area network. The reason is that we expect to obtain maximum transmission data rates. The election of a local area network for our tests is because we have no access to any other environment.

Finally, from the three identified objectives we will obtain the results.
Chapter 5

Results

Firstly, the working scenario where the tests will be carried out is presented. Subsequently, the chapter will be divided according to the work flow presented in Figure 4.1, which was introduced in the previous chapter. Thus, there will be a section for each of the implementations under study and under each of them it will be possible to find three subsections, one for each of the tasks. As well, for each of the tasks a brief introduction will be given.

According to the detection techniques, we want to point out that in this work we will not take into consideration active wardens (see Section 2.3.4) since they are used to hinder the communication but not to detect their presence.

5.1 Network Configuration

All the tests that are to be performed in order to gain insight in the different implementations will be developed over the network configuration that we present next.

The network is divided in two groups of two computers each. The computers in each group are interconnected through a 10/100 Fast Ethernet D-Link switch. In addition, both groups are linked together by means of an Ethernet cable that connects both switches (see Figure 5.1).

Furthermore, all computers in this network have Internet access through a gateway connected to one of the switches. Keeping the computers connected to the Internet makes
it seem more like an ordinary corporate network. Anyway, all computers are statically configured within the network so less packets (e.g. DHCP packets) are passing through the network while the tests are performed.

The tests will consist of using the covert implementations to send files with different sizes, varying from 1 KByte to 10 MBytes. However, not all applications will be tested with all different files, we will be loyal to the Smeets and Koot [2006] test schedule (see Table 5.1).

<table>
<thead>
<tr>
<th>Application</th>
<th>File Size</th>
<th>Number of tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covert-TCP</td>
<td>1 KB</td>
<td>78</td>
</tr>
<tr>
<td>Ping Tunnel</td>
<td>1 MB</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10 MB</td>
<td>3</td>
</tr>
<tr>
<td>Firepass</td>
<td>1 MB</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10 MB</td>
<td>9</td>
</tr>
<tr>
<td>OzymanDNS</td>
<td>1 KB</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>10 KB</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>100 KB</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5.1: Test Schedule followed.

Finally, it is interesting to point out that all the computers were running Ubuntu 6.10 with 2.6.17 kernel since in Smeets and Koot [2006] they use Slackware 10.1 with a different kernel and MacOS X 10.4.4 because this could pose a difference between their results and ours.
5.2 Craig Rowland’s Covert_TCP

This tool was presented in June 1997, exactly one year after the First International Workshop on Information Hiding. The main intention of the author was to unveil weaknesses in the TCP/IP Protocol Suite, which an attacker, in the worst case, could take advantage of in order to pass data in a way that could pass unnoticed to the inexperienced observer.

5.2.1 Implementation Study

The author, Craig Rowland, presents three possible vulnerabilities and the consequent methods to take advantage of them in order to pass information surreptitiously between two hosts. These vulnerabilities are related to three fields within the TCP and IP headers (see bold fields in Figure 5.2). Next these are studied more in depth.

![TCP and IP Headers](image)

(a) TCP Header from RFC 793. (B) IP Header from RFC 791.

- Method 1: Manipulation of the IP Identification Field

The IP Identification field (IP ID) is normally used to re-assemble packets that might be fragmented by intermediate nodes on the network. The client party replaces this field with the numerical representation, in ASCII format, of the character to be transmitted. Therefore, on the other side, the receiver retrieves the character by just dividing the incoming value by 256.

\[
\text{IP ID} = 18432 \quad \xrightarrow{+256} \quad 72 \quad \xrightarrow{\text{ASCII}} \quad H
\]
Rowland warns that this method is susceptible to packet filtering and Network Access Translation (NAT), where the header might be overwritten.

- Method 2: Initial Sequence Number Field

In this method, the TCP sequence number is utilized. This 32-bits field was defined to ensure that packets are decoded in the right order. The first of them is called Initial Sequence Number (ISN) and it is used to negotiate the establishment of a new TCP connection. The ISN must be chosen in such a way that it is not possible to encounter two packets from the same connection with the same sequence number. Craig Rowland generates the sequence number in the SYN packet (i.e. synchronization packets used to request for a new TCP connection) by multiplying the original ASCII value of the character by 16777216 (i.e. 65536 * 256 representation). This enables a more “realistic” sequence number since, as mentioned before, it is a 32-bits field. The destination host will simply divide by 16777216 to retrieve the actual sent value.

- Method 3: The TCP Acknowledge Sequence Number Field “Bounce”

The current method makes use of bounce servers (see Section 2.3.5) to act as intermediates in the communication. This is achieved by IP spoofing (i.e. forgery of the IP address). The sender creates a SYN packet with a forged source address and TCP Port which states that the source of the packet is actually its recipient. This way the bounce server will respond to this packet as if it were actually from the fake source, which was the intention of both the sender and the recipient of the covert message. In order to clarify this concept it is recommended to see Figure 5.3.

The benefit of this method is the concealment of the source of the packet since the communication is not directly established. Furthermore the sender could decide to bounce packets off not only one but hundreds of Internet hosts.

This technique can also be useful when the recipient is within a firewalled network which only allows inbound packets from certain trusted sites. Therefore, the sender could choose one or more of these sites to act as bounce servers.

Rowland reminds that a network with a correctly configured router or firewall may not allow outbound traffic with a network address which does not belong to its domains.
In all cases, the author states that the type of codification utilized will produce the same values for the same encoded letters. This is not good since the proposed header fields must not clash. Therefore, his suggestion is to incorporate methods for random number generation, like XOR-ing or using previously encrypted data. Encryption will also provide further protection of the transmitted data.

5.2.2 Literature Study

This section was going to be divided into three different subsections since this application utilizes three different methods to convey covert data. Nonetheless, as they share common characteristics which may compromise their stealthiness they will be treated simultaneously.

All the three methods conceal the data in the different fields by directly substituting the usual content of the field in question by the ASCII value of the character to be sent. Therefore, if no randomizing methods are utilized it would be easy to detect the presence of repeated values in these fields, which are calculated with algorithms such that the number of repetitions are minimized, since their intention is to make the packet unique.

However, randomizing the data contained in the fields is not enough to avoid being
detected since each Operating System exhibits well-defined characteristics in generated TCP/IP fields. This fact is pointed out in Murdoch and Lewis [2005], where 14 tests are proposed to identify covert channels within TCP/IP. From these tests, the first 4 are based on IP ID characteristics depending on the O.S., from 5 to 11 are related to TCP ISN and the last 3 are not of interest for the application we are testing. In fact, Covert_TCP was tested and did not pass unnoticed in any of the tests proposed.

Previously, Sohn et al. [2003b] proposed the use of a SVM (Support Vector Machine) to detect the use of Covert_TCP when using the IP ID and TCP ISN methods. SVMs are a set of supervised learning methods used for pattern classification. Murdoch and Lewis [2005] criticizes the use of SVMs for this purposes since this type of techniques are suitable for identifying features that are not well understood, however, there are algorithms for generating the IP ID and TCP ISN values. Anyway, the results obtained from their tests over Covert_TCP were promising, achieving in most cases a high detection rate.

Finally, another issue that may alert of the use of Covert_TCP is that even if there are multiple attempts to establish a TCP connection (i.e. many SYN packets sent) none of them is finally set since the data is concealed in the ISN and the connection is reset after the server has received that packet [Owens, 2002].

5.2.3 Testing period

The tests have not been developed directly over Craig Rowland’s original version of covert_TCP. Instead, the modified version proposed by Smeets and Koot [2006] has been used. This version is basically the same but it gives the opportunity of settling the inter-packet transmission time in terms of nanoseconds. This way it is also possible to use 1 second delay times as it is statically established in the original code.

Each of the three proposed methods were tested with different timings, starting from a delay of 1 second until using no delay between two transmitted packets. The results from these tests are in disagreement with those presented in Smeets and Koot [2006] since their results yielded a maximum transmission speed of 1.1 seconds for 1 KByte files (i.e. ≈ 930 Bytes/s) although it resulted in packet losses, however, the maximum speed in our tests was not under 4 seconds in any case (i.e. ≈ 256 Bytes/s). Furthermore, due to this latency on the transmission we achieved many satisfactory transmissions. We summarize
the results from the tests performed in Table 5.2. For a detailed information please refer to Appendix A.

<table>
<thead>
<tr>
<th>Method</th>
<th>Fastest Execution</th>
<th>Max Throughput</th>
<th>Total Errors – Correct bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP ID</td>
<td>4.128 seconds</td>
<td>248.06 Bytes/sec</td>
<td>5/26 – 99.99%</td>
</tr>
<tr>
<td>TCP SEQ</td>
<td>4.128</td>
<td>248.06 Bytes/sec</td>
<td>0/26 – 100%</td>
</tr>
<tr>
<td>TCP ACK</td>
<td>4.126 seconds</td>
<td>248.18 Bytes/sec</td>
<td>0/26 – 100%</td>
</tr>
</tbody>
</table>

Table 5.2: Summary of Covert TCP Results.

According to the results presented in Table 5.2 it may seem that only the IP ID method have problems with wrong transmissions. However, we believe that the IP ID method is not especially unreliable compared with the other two methods. These two were also tested in another occasion resulting in errors. Therefore, all these methods are vulnerable to packet loss since no reliability mechanisms are provided in the application.

5.3 Daniel Stødle’s Ping Tunnel

Ping tunnel, also known as Ptunnel, was proposed in Stødle [2005]. This application is useful when a host within a network needs to create a connection outside the network but the service is blocked by either a firewall or a proxy server.

5.3.1 Implementation Study

Daniel Stødle presents its tool as an application “For those times when everything else is blocked” [Stødle, 2005, Page 1]. It is actually a way of tunnelling TCP connections to a remote host using ICMP echo request and reply packets. These packets are normally used by the ping command in order to detect whether a host is “alive”, that is, accessible through the network.

According to the author, Ptunnel’s connections are reliable. This is handled by resending packets that seem to be lost, that are those packets which have not been acknowledged after 1.5 seconds. However, no congestion control mechanisms are provided.

Another information of great interest for our purposes given by Stødle is related to
the bandwidth. The measured maxima for one tunnel is about 150 Kb/s downstream and 50 Kb/s upstream. However, this can be improved by increasing the size of the send and receive window in the header file and thus recompiling.

In order to start describing how Ptunnel works, we will first introduce the three elements necessary in the communication. The client, which is the computer trying to get access to the service; the destination that is the provider of the service; and the proxy that is the intermediary between the two other parties.

![Ping Tunnel Communication Diagram](image)

**Figure 5.4: Ping Tunnel Communication**

The proxy is the main element in this diagram. As can be seen in figure 5.4, the ICMP packets carry the covert data in the ICMP payload. This covert data has a specific format and is divided in several fields. The magic field contains a special number which is used to differentiate this type of packets from usual ping ones. The IP and port are used to indicate where the client wants the packets to be forwarded. The state code is mainly used to differentiate among the five different types of packets. The ack and seq have the normal usage. The length indicates the size of the data field in case it is a data packet, otherwise it is 0. Finally, the rsv is a two bytes field which is actually only used for padding.

The protocol starts with a request of connection from the client to the proxy. Then, the proxy opens a TCP connection to the destination given by the IP and port fields in the packet. The client sends data packets (i.e. ICMP echo request) and the proxy conveys them through the TCP connection. The proxy also gets packets from the destination and covert them to ICMP echo reply packets to send them to the client.
5.3.2 Literature Study

Ping Tunnel takes advantage of the data field in the ICMP echo message to conceal its own payload. The usual data field in echo messages is intended to record route information or timing record in order to calculate RTTs (Round Trip Time).

However, Ping Tunnel protocol adds a lot of extra data compared to average ICMP packets (see Figure 5.5). This increase in the size of the packets is a possible way of detecting the use of this application. On the other hand, there are also legitimate uses of large ICMP packets. For example, large ICMP echo packets are used to check if the network is able to carry big packets [Singh et al., 2003]. Nevertheless, this is not the most common use.

In addition, all Operating Systems have default values to be sent in ICMP messages. For example, in Windows XP, the data field is 32 bytes long and comprised of the letters of the alphabet (see Figure 5.6 (a)) whereas in Ubuntu Linux, the size of this field is 56 bytes and it contains both special characters and numbers (see Figure 5.6 (b)). Therefore, intrusion detection systems could store the values of the most common Operating Systems or at least of those running within the network under surveillance and simply warn the administrator or deny those packets with a suspicious payload [Sbrusch, 2006].

In addition, the use of SVMs (Support Vector Machine) was proposed also for the detection of ICMP corrupted payloads [Sohn et al., 2003a]. The idea was to train the SVM with two learning data sets of normal (i.e. O.S. based) and abnormal (i.e. covert
channel based) ICMP packets. Achieving a detection rate of nearly 99% and a false detection rate below 1% in most cases. This might raise false alarms. However, although these tests were developed over another implementation (i.e. Loki2 [Daemon9, 1997]) these might be translated into the tool we are studying since the SVM method is based on the study of the payload on ICMP packets.

Furthermore, ICMP traffic is not very common during long periods of time. Therefore, coming across with multiple ping requests within a small time interval may indicate the existence of a covert channel in the ICMP protocol [Chauhan, 2005].

Finally, another mean of detection is the analysis of the payload in search of the magic field since this value is statically fixed within the header file (i.e. ptunnel.h) to the value 0xD5200880. However, as this value might be changed by “expert” users, previous methods are more advisable.

### 5.3.3 Testing period

The tests consist of the transmission of both 1MB files and 10MB files by using netcat tunnelled over Ping Tunnel. We performed several tests in order to determine if the information provided by Stødle regarding the bandwidth of the application was in concordance with the actual results we obtained. The results are simplified in Table 5.3.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Average Time</th>
<th>Average KB/s</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MByte</td>
<td>10.99 sec</td>
<td>93.09</td>
<td>0/3</td>
</tr>
<tr>
<td>10 MBytes</td>
<td>153.94 sec</td>
<td>66.51</td>
<td>0/3</td>
</tr>
</tbody>
</table>

Table 5.3: Summary of Ping Tunnel Results.

Figure 5.6: Windows XP vs. Linux ICMP traces.
In fact, the information provided seems not to be in agreement with the results obtained, at least for the transfer of “small” (i.e. 1 MB) files, where our tests run at \( \approx 93 \) KB/sec. However, for 10 MB files the bandwidth is reduced to \( \approx 66 \) KB/s. Our hypothesis for this throughput reduction is that the packet loss handling mechanism included in the application might be reducing it. Since the files are bigger, there might be more packets lost. For further information the reader is encouraged to visit Appendix B.

5.4 Alex Dyatlov’s Firepass

Firepass is a covert channel implementation proposed in 2003 which allows the creation of tunnelled connections in order to break through firewalled networks making use of an usually unblocked service as HTTP.

5.4.1 Implementation Study

This covert channel implementation was brought out by Dyatlov [2003], a member of the Gray-World.net team. It is defined as a tunnelling tool which allows bypassing firewall restrictions by encapsulating TCP/UDP connections within HTTP POST request/response packets. It consists of two perl scripts, one of them is a CGI (Common Gateway Interface) script, placed on a Web Server, that attends the requests from the other script, placed on the firepass client, the provider of the tunnel (see Figure 5.7).

![Figure 5.7: Firepass Configuration.](image)

The firepass client is listening for incoming connections on a TCP/UDP port. Whenever it receives a new request it creates an HTTP connection to the firepass server. The data originating from the client is transmitted as HTTP post requests. In the first request of each session the firepass client attaches additional lines in the HTTP header containing
the redirect rule for the rest of the packets (i.e. where is the service provider placed). On the Web server, the firepass cgi script creates a “Connection Manager” which is in charge of tunnelling the connection between the client and the service provider. The firepass client receives data from the server within HTTP response messages which it extracts and sends to its client. In case there is no incoming/outgoing data the firepass client/server just sends empty HTTP messages.

Finally, it only remains to be said that no information is given on detectability or the maximum bandwidth the application may provide but there is a warning about the current state of development of the tool whose version is alpha and there might be some bugs.

5.4.2 Literature Study

Firstly, it is important to highlight that the inspection of HTTP packets in the search of covert channels requires costly software and may hinder the performance of web applications. Thus, applying strict inspections is not always feasible [Sbrusch, 2006]. Maybe that is the reason why it has been rather difficult to find information on HTTP covert channel detection.

Every Internet browser has a unique header signature (i.e. pattern) and makes use of a certain set of header fields. However, firepass uses custom header fields in the first connection of each client to the server. There are three forged elements added to the header in order to tell the server where to forward the packets received (see Figure 5.8). This code can be found in the `fpclient.pl` file:

```perl
$header = join("\r\n", $header, "X-Host: $host", "X-Port: $port", "X-Proto: $proto");
```

Web Tap [Border and Prakash, 2004] parses each header and raises an alert when it identifies indicatives of unknown browser requests. Firepass was tested and immediately detected due to the aforementioned fields added.

Furthermore, firepass client’s configuration file (`fpclient.conf`) contains a parameter which determines the delay between HTTP requests, which is by default set to 0.5 seconds. Although this value might be changed, in Border and Prakash [2004] it is also proposed the checking of the inter-request delay in order to identify channels operating on timers.

The size of the packets is also a feature that may reveal the use of firepass (see Figure
Normal HTTP messages contain very little information but the use of a firepass may require large amounts of data in order to transmit files [Border and Prakash, 2004, Pack et al., 2002].

As mentioned in the previous section, when there is no data to be transmitted, firepass sends empty packets. Although many empty packets do also appear in normal web traffic, such appearance only lasts a short period of time [Pack et al., 2002]. However, when using firepass there might be long periods of time in which there is no data to be sent.

There are other proposed means of detection like the number of packets per session, however, these might be prone to false positives since it depends on the type of HTTP information accessed. The amount of packets transmitted by a video stream web server is much bigger than the number of packets received when accessing a text-based web server.

### 5.4.3 Testing period

The tests performed consist of the transmission of 1 MB and 10 MB files using netcat over the tunnelled connection provided by the Firepass client and server. The results are summarized on Table 5.4.

From this table it is possible to determine that Firepass can provide a reasonably high (≈ 440 KBytes/s), nevertheless, the amount of errors is also high. In fact, any of the tests resulted in a fruitful file reception. In addition, the more the delay is increased, the more the throughput is reduced and also the percentage of errors, reaching a maximum mean of 99.31% of successfully received bytes. However, these values are still higher with 10 MB files which has no apparent reason. In fact, we tried to get in touch with the author of the application in order to have a second opinion but we received no answer. For detailed
information on how these tests were performed please refer to Appendix C.

### 5.5 Dan Kaminsky’s Ozyman

The only available material for the understanding of this tool provided by the author are the slides he utilized for its presentation in The Black Hat Briefings in 2004 [Kaminsky, 2004]. It is quite difficult to retrieve accurate information when one has not attended the presentation. However, it has been possible to obtain some information about how the tunnelling is done, the elimination of this type of channels and a few details on how the application works.

#### 5.5.1 Implementation Study

In order to have a better understanding of how this tunnelling tool works it is necessary to have a basic knowledge of how DNS works. Therefore, the reader is encouraged to read section D.1 from the appendices.

Ozyman is written in perl and it is comprised of basically a client (*droute.pl*), which conceals the data into DNS requests, and a server (*nomde.pl*), which on the reception of these packets retrieves the data and forwards it to the actual recipient of that TCP stream. In between of the client and the server the presence of another name server is required, which redirects the DNS requests to the fake DNS server (see Figure 5.9). This is possible since this naive name server is unaware of the IP address on the request and,
therefore, has to ask for it.

Figure 5.9: DNS Tunnel Communication.

The upstream, as can be seen in figure 5.9, encodes the data to convey in the request sent to the name server. There are some restrictions on the way of constructing these requests. The total length must be no longer than 253 characters, furthermore there might not be more than 63 between two dots (indicating subdomains) and there are only 63 allowable characters. Therefore it is necessary to use Base32 (i.e. only characters between a-z and 0-6) to encode characters only with 5 bits.

The downstream, on the other hand, carries TXT records which have minimal restrictions. These provide a high capacity of storage and utilize Base64 (i.e. a-z, A-Z, 0-9, =, /) to encode 6 bits per character.

After the data there is a unique ID intended for reordering of the packets if some are delayed since DNS requests are built over UDP and it has no mechanisms for determining the order of packets. In addition, the keyword up or down is used to indicate whether the traffic is upstream or downstream. Next we present an example of a forged DNS request (up):

ntez345sy2qk3jsg2og3eswo2jujscb3r43as6m6h12wsxobm1h2olu4tmaq.1yazbf2e2rdynrd3f1dvy2w3
Moreover, according to Kaminsky [2004], the application is intended only for shell and IM (Instant Messaging) use since it is said to have a bandwidth lower than 1 Kilo-byte/second. The work flow of the implementation is depicted in figure 5.10.

![Diagram](image)

(a) Upstream work flow.  
(b) Downstream work flow.

Figure 5.10: DNS Tunnel Work Flows.

Finally, Kaminsky also gives some information related to the detectability of the application. There is a huge amount of data within requests and responses, which although possible, is not normal. In addition, this data has a high entropy (a measure of the uncertainty associated with a random variable) since DNS names tend to follow language distributions and deviations from these patterns could be noticed. Moreover, Kaminsky [2004] proposes multi-packet analysis to differentiate these malicious activities. The idea is to measure the number of queries per minute and domain since the tendency in usual traffic is to have a moderate amount.

### 5.5.2 Literature Study

We will start this section with a quote: “We are not aware of any work having been done on the detection of tunneling using DNS messages” [Horton and Safavi-Naini, 2006, Page 2]. We have not found any work related to the detection of DNS tunnelling either,
for this reason we have based our analysis on information mainly found on the World Wide Web.


Some features that may raise an alarm are the appearance of a high volume of DNS requests from clients in the network, since this type of traffic is usually moderate. In addition, the content and size of the messages originating from Ozyman are different to usual ones. Ozyman’s are usually longer and contain higher entropy than usual since DNS names tend to follow language patterns. Furthermore, usual requests should not exceed 312 bytes including headers and responses typically do not exceed 512 bytes. Finally, the total amount of data transmitted over port 53 (DNS) is much higher than usual.

5.5.3 Testing Period

It has been quite challenging to get this implementation to work. We had problems with the code because it is prepared to work with the default settings and to tunnel a specific application, ssh (secure shell). However, the tests we were to perform consisted of the transfer of 1 KB, 10 KB and 100 KB files using netcat.

In order to overcome these problems we had to deal with the code itself and try to figure out what was not working. All these handicaps are documented in Appendix D.

The results obtained from the tests (see Table 5.5) reveal that the application is not completely reliable, at least, for its use under the circumstances we propose, file transfers. However, as aforementioned, Kaminsky states that his tunnelling tool is only intended for light-weight communications.

<table>
<thead>
<tr>
<th>Delay seconds</th>
<th>Data Size KBytes</th>
<th>Average Time seconds</th>
<th>Average Throughput Bytes/sec</th>
<th>Total Errors – Correct bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1</td>
<td>0.53</td>
<td>1927.73</td>
<td>0/3 – 100%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>5.11</td>
<td>2003.49</td>
<td>0/3 – 100%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50.99</td>
<td>2008.45</td>
<td>3/3 – 98.85%</td>
</tr>
<tr>
<td>0.05</td>
<td>1</td>
<td>1.09</td>
<td>939.80</td>
<td>0/3 – 100%</td>
</tr>
</tbody>
</table>

... ... ...
Table 5.5: Summary of Ozyman Results.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10.35</td>
<td>989.21</td>
<td>1/3 – 99.64%</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>104.05</td>
<td>984.15</td>
<td>3/3 – 99.06%</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>2.00</td>
<td>510.43</td>
<td>0/3 – 100%</td>
</tr>
<tr>
<td>10</td>
<td>19.86</td>
<td>515.61</td>
<td>0/3 – 100%</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>197.25</td>
<td>519.13</td>
<td>2/3 – 99.85%</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>100</td>
<td>756.41</td>
<td>135.37</td>
<td>2/3 – 99.81%</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
<td>942.56</td>
<td>108.63</td>
<td>1/3 – 99.96%</td>
</tr>
</tbody>
</table>

On the other hand, even using delays that involve a bandwidth lower than 1 KByte/sec, which is the maximum supported according to Kaminsky [2004], there were still problems with packet losses although these are only present with big file transmissions. However, although the error rate is in most cases below 1%, this is enough to ruin the communication in such cases as when encrypted data is involved. With the default inter-packet delay (i.e. 0.01 seconds) the maximum throughput we obtained was over 2 KBytes/sec and we had no packet losses with 1 KB and 10 KB files.
Chapter 6

Conclusions and Future work

This last chapter will describe the main contributions of this dissertation together with an overall analysis of the results obtained from the whole research process and the final conclusions. Furthermore, we will present possible lines of work in the near future.

6.1 Analysis and Conclusions

We have successfully evaluated TCP/IP covert channel implementations with respect to the bandwidth and their capacity of passing unnoticed.

As a result of the preliminary study of the authors’ documentation we gain insight into the operation of the implementations. Not only information on how to make them work but also on which techniques are utilized to convey the data within the network packets, characteristics that may expose the channel, etcetera. This information was useful for the Literature Study task.

We have extracted many detection techniques for the different implementations from our literature study. These implementations might be revealed with current detection solutions. Moreover, there are currently some practical tools, such as Web Tap which is able to detect covert HTTP traffic and in particular Firepass. However, the analysis of all the packets generated in broad networks with big amounts of traffic in order to find covert channels is a very heavy task to be carried out. Furthermore, in this work we have only studied four covert channel solutions but there are lots of tools available and, with no doubt, many more will appear in the future. Finally, disabling the potential carriers
(i.e. protocols) of covert communications is not the solution since TCP/IP, HTTP, DNS or ICMP provide basic services such as access to the World Wide Web. However, we encourage to take covert channels into consideration and be aware of the security flaws they take advantage of in order to diminish their possible harmful effects.

On the other hand, our tests of the different applications have revealed that, up to date, the bandwidth provided by these implementations is moderate and in some cases low (see Table 6.1). However, this is always relative to which might be the reason for using the covert channel. The low throughput might be a problem only if the time to retrieve the information is pressing.

<table>
<thead>
<tr>
<th>Application</th>
<th>Maximum Throughput</th>
<th>Erroneous Tests – Correct bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covert_TCP</td>
<td>0.24</td>
<td>5/78 – 99.99%</td>
</tr>
<tr>
<td>Ping Tunnel</td>
<td>93</td>
<td>0/6 – 100%</td>
</tr>
<tr>
<td>Firepass</td>
<td>442</td>
<td>18/18 – 88.44%</td>
</tr>
<tr>
<td>OzymanDNS</td>
<td>2</td>
<td>12/36 – 99.74%</td>
</tr>
</tbody>
</table>

Table 6.1: Results Comparison.

The real problem takes shape when we take a look at the number of erroneous tests and the percentage of successfully received bytes. We have realized that a high bandwidth without the certainty of a correct reception of the packets is in most cases useless since the information transmitted might loose its meaning. However, this is dependent on the type of information that is intended to be transmitted. For example, if the intention is to transmit multimedia streaming there is no problem if there are a few bytes lost because, although there is a loss of quality of the information, it is still possible to retrieve it. However, when the information to be transmitted is ciphered, the loss of a single byte results in an unsuccessful decryption.

Although the average rate of satisfactory bytes received is, in general, high, it is possible to observe in Table 6.1 that only one of the implementations, Ping Tunnel, has turned out to be 100% reliable. Therefore, this tool could be used to convey encrypted files. On the other hand, Firepass could possibly be used for multimedia streaming transmissions.

Furthermore, it is important not to forget that all these tests have been developed under controlled circumstances, in a LAN environment, whereas they were mainly designed for their use on the wilderness of the Internet where there might be even more vulnerable.
6.2 Future work

One of the objectives of this work, the study of the probability of detection of the different tools, has been based on a survey over different specialized papers on the topic. However, a possible line of work could be based on the investigation of practical solutions for the detection of covert channels. Of further interested would be the development of tests in order to empirically determine how effective these tools might be.

As far as we are concerned, the monitorization of all the traffic from a network in search of covert channels is quite heavy. However, the are some general purpose tools like IDSs (Intrusion Detection System) that might be useful for the detection of some of these types of channels since they are based on the detection of anomalies within the normal signature (i.e. a pattern) of the protocol headers. Therefore, this might be an interesting field of action as well.

The development of new tests over the studied implementations could also be interesting, above all, if these are carried out within several real corporate environments with their already mounted defence mechanisms in order to determine how vulnerable they are. These tests could determine as well how the implementations work under real adverse network conditions.

Although covert channels have been present for more than two decades, there are still possible covert channel carriers. Not only current unexplored protocols but also new arising protocols might be studied for the concealment of data. These protocols should be studied and put in the spotlight in order to prevent covert communications which may damage the integrity of the systems or the privacy of users.

Another possible line of work could be the study of other available tools in order to determine their possibilities of being settled within corporate environments. In this work we have paid special attention to storage timing channels, however, there are also other ways of covert communication such as passive or timing channels. These two are currently providing a much lower bandwidth, however, they are stealthier and therefore more difficult to detect, becoming a good choice when the intention is to transmit small quantities of information or, on the contrary, big quantities of data but there are no restrictions with regard to the time.
Bibliography


<http://www.criptored.upm.es/guiateoria/gt_m142m.htm>.

<www.cisa.umbc.edu/courses/cmsc/444/fall05/studentprojects/sweety.pdf>.

<http://www.phrack.org/archives/51/P51-06>.

<http://www.phrack.org/archives/49/P49-06>.


<http://gray-world.net/it/pr_firepass.shtml>.


<http://citeseer.ist.psu.edu/361793.html>.

46


Appendix A

Covert_TCP Measurements

This appendix presents the information obtained from the tests performed over the modified version of Craig Rowland’s Covert_TCP written by Smeets and Koot [2006].

Firstly, a measurement of the maximum throughput of our network configuration was performed. Once this information was known, the execution of the different scheduled tests were performed.

A.1 Network throughput

In order to measure the maximum capacity of the working scenario the Linux tool iperf was utilized. The decision of using this tool is due to the fact that our intention is to try to follow the same strategy as Smeets and Koot [2006].

Next a brief explanation on how this tool was used will be given. It is necessary to run iperf on two machines. The server will be waiting for connections and it can be launched with the following command:

iperf -s

The client is executed on another machine and it works by repeatedly sending an array of len bytes for time seconds. Default values are 8 KB for TCP and 1470 bytes for UDP, and 10 seconds. The command used for the client is:

iperf -c 10.0.26.2
The IP address specified after the -c option refers to the machine where the server is running.

We performed tests both for machines connected to the same switch and for machines connected to different switches (see Figure 5.1). The results for the first case are shown next:

ruben@ruben-desktop:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 39501
[ 4] 0.0-10.0 sec  111 MBytes  93.4 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 34030
[ 5] 0.0-10.0 sec  111 MBytes  93.2 Mbits/sec
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 34031
[ 4] 0.0-10.0 sec  111 MBytes  93.5 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 34032
[ 5] 0.0-10.0 sec  111 MBytes  93.4 Mbits/sec
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 34033
[ 4] 0.0-10.0 sec  111 MBytes  93.2 Mbits/sec

The following results belong to the execution between the two machines connected to different switches. Precisely, these machines are 10.0.26.2 and 10.0.26.4.

ruben@ruben-desktop:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.4 port 52187
[ 4] 0.0-10.0 sec  109 MBytes  91.7 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.4 port 52188
[ 5] 0.0-10.0 sec  110 MBytes  91.7 Mbits/sec
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.4 port 52189
[ 4] 0.0-10.0 sec  109 MBytes  91.8 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.4 port 52190
[ 5] 0.0-10.0 sec  109 MBytes  91.6 Mbits/sec
As a conclusion we can state that, obviously, using 10.0.26.2 as server and 10.0.26.3 as client results in a higher bandwidth, around 93 Mbits/s. On the other hand, when using 10.0.26.2 as server and 10.0.26.4 as client the bandwidth obtained is around 91.5 Mbits/s. This is because of the longer path, with one more switch (10/100 Fast Ethernet D-Link) in between.

A.2 Testing Covert_TCP

This section will gather the results from the three different types of tests performed over Covert_TCP. These consists of sending an ASCII file automatically generated with a simple perl script that we show next:

```perl
#!/usr/local/bin/perl
#
#Automatically generates de data

for($i=1; $i <=100000; $i=$i*10)
{
    $file = $i."KB.txt"; #Creates file name

    open(FILE, ">$file") or die "Could not open file $file: $!"; # Output file

    #16chars x 64 = 1024 chars = 1KByte
    print FILE "ABCDEFGHIJKLMNO\n"x(64*$i);

    close(FILE);
}
```

In fact, this script generates several files of different sizes from 1 KBytes to 100 MBytes. However, in this case the only used will be the 1 KByte file.

Moreover, in order to check if the covert communication has been successful another perl script has been written. This script checks if each of the received files are equal to the automatically generated 1 KB file. The script is the following:
#!usr/bin/perl -w

# Writes to a file whether the file transmission was successful

use strict;

# Variables declaration
my $name="1KB";
my @delays=(1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0);
my @nums=(1, 2, 3);

foreach my $delay (@delays)
{
    my $file = $name.$delay; # Creates file name

    if($delay >= 0.5){
        $file = $file.".txt";
        system("diff -s -q 1KB.txt $file >> check_results.txt");
    }else{ # 3 repetitions
        foreach my $num (@nums){
            my $newFile = $file."_$num.txt"; # otherwise it'd concatenate
            system("diff -s -q 1KB.txt $newFile >> check_results.txt");
        }
    }
}

Furthermore, the time has been measured with the famous Linux tool called time. Therefore, the command use to run one execution should look like the following:

$ time sudo ./covert_tcp -dest 10.0.26.2 -source 10.0.26.4 -file 1KB.txt -sec 1 -nsec 0

Finally, it only remains to be said that our intention was to fully automatized this process but it was not possible since Covert_TCP, when working on server mode, does not recognize the end-of-file character and the program has to be quit in a rough way, using the interruption sequence Ctrl+C.

A.2.1 IP Identification field

The following table summarizes all the information that has been gathered from the different test carried out for the IP ID method.
<table>
<thead>
<tr>
<th>Delay (seconds)</th>
<th>Time (Real)</th>
<th>Time (User)</th>
<th>Time (System)</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17m8.207s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.5</td>
<td>8m36.184s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.1 (1)</td>
<td>1m46.581s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.1 (2)</td>
<td>1m46.595s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.1 (3)</td>
<td>1m46.576s</td>
<td>0m0.000s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.05 (1)</td>
<td>0m56.735s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.05 (2)</td>
<td>0m56.432s</td>
<td>0m0.008s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.05 (3)</td>
<td>0m53.297s</td>
<td>0m0.008s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.01 (1)</td>
<td>0m16.539s</td>
<td>0m0.008s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.01 (2)</td>
<td>0m12.386s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.01 (3)</td>
<td>0m16.410s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.005 (1)</td>
<td>0m8.228s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.005 (2)</td>
<td>0m12.326s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.005 (3)</td>
<td>0m8.228s</td>
<td>0m0.000s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (1)</td>
<td>0m4.142s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (2)</td>
<td>0m4.137s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (3)</td>
<td>0m4.150s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>NO, 1 byte missing</td>
</tr>
<tr>
<td>0.0005 (1)</td>
<td>0m4.142s</td>
<td>0m0.008s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0005 (2)</td>
<td>0m4.141s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>NO, 1 byte missing</td>
</tr>
<tr>
<td>0.0005 (3)</td>
<td>0m4.132s</td>
<td>0m0.004s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (1)</td>
<td>0m4.151s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (2)</td>
<td>0m4.128s</td>
<td>0m0.004s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (3)</td>
<td>0m4.146s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>NO, 1 byte missing</td>
</tr>
<tr>
<td>0 (1)</td>
<td>0m4.163s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
<td>NO, 1 byte missing</td>
</tr>
<tr>
<td>0 (2)</td>
<td>0m4.138s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>Yes</td>
</tr>
<tr>
<td>0 (3)</td>
<td>0m4.131s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>NO, 2 bytes missing</td>
</tr>
</tbody>
</table>

Table A.1: IP Identification Field Results.

From this table we can conclude that, at least, for our network configuration this method of covert messaging has not been reliable. Furthermore, our results notoriously differ from those presented in Smeets and Koot [2006]. The times presented in our table are always higher than 4 seconds whereas on their work times are reduced up to 1.1
seconds with no delay between packets sent. Moreover, all their tests result in a perfect transmission until the 0.0001 second barrier is reached.

A.2.2 TCP Sequence Number field

The current section will explore the results obtained from the testing of the application in the second mode of use. As it has been done in the previous section, all the results will be included in a table that is shown next.

<table>
<thead>
<tr>
<th>Delay (seconds)</th>
<th>Time</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real</td>
<td>User</td>
</tr>
<tr>
<td>1</td>
<td>17m8.201s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.5</td>
<td>8m36.190s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.1 (1)</td>
<td>1m46.552s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.1 (2)</td>
<td>1m46.550s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.1 (3)</td>
<td>1m46.556s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.05 (1)</td>
<td>0m53.273s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.05 (2)</td>
<td>0m54.405s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.05 (3)</td>
<td>0m53.754s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.01 (1)</td>
<td>0m15.475s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.01 (2)</td>
<td>0m12.337s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.01 (3)</td>
<td>0m16.401s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.005 (1)</td>
<td>0m8.223s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.005 (2)</td>
<td>0m8.236s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.005 (3)</td>
<td>0m8.779s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.001 (1)</td>
<td>0m4.126s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.001 (2)</td>
<td>0m4.137s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.001 (3)</td>
<td>0m4.137s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.0005 (1)</td>
<td>0m4.135s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.0005 (2)</td>
<td>0m4.139s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.0005 (3)</td>
<td>0m4.135s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.0001 (1)</td>
<td>0m4.129s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.0001 (2)</td>
<td>0m4.129s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.0001 (3)</td>
<td>0m4.128s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0 (1)</td>
<td>0m4.138s</td>
<td>0m0.000s</td>
</tr>
</tbody>
</table>
Unlike the previous method, in this case all transmissions resulted in the correct reception of the file. However, the necessary time to complete each transmission time is still lower than the one provided for the same situation in Smeets and Koot [2006]. In fact, the results presented in that report show that they suffer from packet losses below 4 seconds, exactly with times of 1.1 seconds (i.e. when no delay is introduced).

### A.2.3 TCP Acknowledgement Through a Bounce Server

Finally, the outcome from the third and last method is gathered on the following table. This method makes use of an intermediate server to act as a proxy to send the messages. In this case we have utilized only one server although it would have been possible to use several.

<table>
<thead>
<tr>
<th>Delay seconds</th>
<th>Time</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real</td>
<td>User</td>
</tr>
<tr>
<td>1</td>
<td>17m8.199s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.5</td>
<td>8m36.170s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.1 (1)</td>
<td>1m46.556s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.1 (2)</td>
<td>1m46.541s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.1 (3)</td>
<td>1m46.545s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.05 (1)</td>
<td>0m56.385s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.05 (2)</td>
<td>0m56.124s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.05 (3)</td>
<td>0m55.995s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.01 (1)</td>
<td>0m12.341s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.01 (2)</td>
<td>0m16.411s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.01 (3)</td>
<td>0m12.329s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>0.005 (1)</td>
<td>0m12.354s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>0.005 (2)</td>
<td>0m8.245s</td>
<td>0m0.000s</td>
</tr>
</tbody>
</table>

Table A.2: TCP Sequence Number Field Results.
Once again all tests resulted in a correct file transmission. However, as happened with the previous tests, the minimum time reached is not below 4 seconds. This result is quite bizarre since our network configuration yielded a maximum throughput even higher than the one presented in Smeets and Koot [2006].

<table>
<thead>
<tr>
<th>Duration</th>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005 (3)</td>
<td>0m8.227s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (1)</td>
<td>0m4.883s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (2)</td>
<td>0m4.128s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.001 (3)</td>
<td>0m4.126s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0005 (1)</td>
<td>0m4.197s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0005 (2)</td>
<td>0m4.130s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0005 (3)</td>
<td>0m4.134s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (1)</td>
<td>0m4.211s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (2)</td>
<td>0m4.141s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0.0001 (3)</td>
<td>0m4.131s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0 (1)</td>
<td>0m4.348s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0 (2)</td>
<td>0m4.137s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
<tr>
<td>0 (3)</td>
<td>0m4.135s</td>
<td>0m0.000s</td>
<td>0m0.000s</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table A.3: TCP Acknowledgement Field Results.
Appendix B

Ping Tunnel Measurements

In this appendix we will describe the tests performed over Daniel Stødle’s Ping Tunnel. For the development of these tests we have followed the same way of proceeding as Smeets and Koot [2006]. Therefore we will start with a measurement of the maximum capacity of the application in question and subsequently we will continue by tunnelling a netcat connection.

B.1 Tunnel throughput

The tool iperf will be used to measure the maximum throughput of Ping Tunnel. Please refer to Appendix A.1 for more information on how this tool works.

First of all it is necessary to launch the iperf in server mode to listen to the incoming data. This server was settled in 10.0.26.4 and listening in port 5001. Next, we start ptunnel working as proxy in 10.0.26.3 and waiting for clients to get connected. It will yield the following information:

```
root@ruben-desktop:~# ptunnel
[inf]: Starting ptunnel v 0.60.
[inf]: (c) 2004-2005 Daniel Stoedle, daniels@cs.uit.no
[inf]: Forwarding incoming ping packets over TCP.
[inf]: Ping proxy is listening in privileged mode.
```

Once the proxy is running is time to start ptunnel client in 10.0.26.2. It will listen in
port 5002 for incoming data. As well, it is necessary to indicate where is the server to communicate with. In our case it is placed in 10.0.26.4 and listens in port 5001.

```
root@ruben-desktop:~# ptunnel -p 10.0.26.3 -lp 5002 -da 10.0.26.4 -dp 5001
[inf]: Starting ptunnel v 0.60.
[inf]: (c) 2004-2005 Daniel Stoedle, daniels@cs.uit.no
[inf]: Relaying packets from incoming TCP streams.
[inf]: Incoming connection.
[evt]: No running proxy thread - starting it.
[inf]: Ping proxy is listening in privileged mode.
```

Finally, iperf must be launched in client mode in order to communicate the data. Obviously, it must be running in the same machine as the ptunnel client, in 10.0.26.2. It is also necessary to indicate that iperf will connect to port 5002 that is the one in which the ptunnel client is listening.

```
iperf -c 127.0.0.1 -p 5002
```

During the connections the proxy prompted information about the status of the communication. Next, we show a piece of this information regarding to a connection.

```
[inf]: Incoming tunnel request from 10.0.26.2.
[inf]: Starting new session to 10.0.26.4:5001 with ID 19587
[inf]: Received session close from remote peer.
```

Sometimes it was also possible to observe error messages because of duplicate session identifiers. This occurs because ptunnel can handle multiple connections but it is the client who randomly generates the identifier when it starts a session. Thus, there might be clashes and there is currently no mechanism for reporting this to the client (remember it might be placed in any other place on the Internet).

```
[inf]: Incoming tunnel request from 10.0.26.2.
[inf]: Starting new session to 10.0.26.4:5001 with ID 64339
[err]: Dropping duplicate proxy session request.
```

The iperf server yielded the following information concerning to the different connections established.

```
root@ruben-desktop:~# iperf -s
```

-------------------------------------------------------------

60
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

------------------------------------------------------------
[ 5] local 10.0.26.4 port 5001 connected with 10.0.26.3 port 41268
[ 5] 0.0-14.0 sec  904 KBytes  529 Kbits/sec
[ 4] local 10.0.26.4 port 5001 connected with 10.0.26.3 port 41269
[ 5] local 10.0.26.4 port 5001 connected with 10.0.26.3 port 41270
[ 4] 0.0-13.0 sec  840 KBytes  529 Kbits/sec
[ 6] local 10.0.26.4 port 5001 connected with 10.0.26.3 port 41271
[ 5] 0.0-25.5 sec  904 KBytes  291 Kbits/sec
[ 6] 0.0-39.7 sec  968 KBytes  200 Kbits/sec
[SUM] 0.0-39.7 sec  2.65 MBytes  559 Kbits/sec
[ 4] local 10.0.26.4 port 5001 connected with 10.0.26.3 port 41272
[ 4] 0.0-15.0 sec  960 KBytes  524 Kbits/sec

As we might conclude from this results the average throughput of the tunnel is around 530 Kbits/s. However, according to Stødle [2005] the bandwidth of the application is 150 Kb/s downstream and 50Kb/s upstream. The only thing we can think about is that the author means KB/s instead of Kb/s and therefore our measurements would turn out to be around 66 KB/s.

B.2 Ping Tunnel Tests

The current section is dedicated to present the results obtained from the tests performed to the tunnel a file transmission using the tool netcat.

This task is quite similar to the one explained in the previous section, but instead utilizing iperf now it is necessary to configure the new tool to transmit files. To that end, we have used the following commands:

cserver@10.0.26.4:~# nc -l -p 5555 > file.txt
cclient@10.0.26.2:~# time nc 127.0.0.1 5000 -q 0 < file.txt

The results obtained from this commands have been arranged into the table we show next:
<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>Throughput</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBytes</td>
<td>Real</td>
<td>User</td>
<td>System</td>
</tr>
<tr>
<td>1 (1)</td>
<td>0m11.004s</td>
<td>0m0.000s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>1 (2)</td>
<td>0m10.983s</td>
<td>0m0.000s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>1 (3)</td>
<td>0m11.011s</td>
<td>0m0.000s</td>
<td>0m0.020s</td>
</tr>
<tr>
<td>10 (1)</td>
<td>2m34.010s</td>
<td>0m0.000s</td>
<td>0m0.080s</td>
</tr>
<tr>
<td>10 (2)</td>
<td>2m33.897s</td>
<td>0m0.004s</td>
<td>0m0.068s</td>
</tr>
<tr>
<td>10 (3)</td>
<td>2m33.929s</td>
<td>0m0.000s</td>
<td>0m0.072s</td>
</tr>
</tbody>
</table>

Table B.1: Ping Tunnel Results.

From this table it appears that the tunnel experiences a higher throughput when the data to transmit is smaller. These results are surprising because for 1MB transmissions the throughput obtained is around 93 KB/s whereas using iperf it yielded around 66 KB/s. Although, this also happened to Smeets and Koot [2006] the difference was not so significant. On the other hand, the 10 MB tests present a similar throughput to the obtained in our baseline measurement. We reckon that this may be due to the built-in packet loss handling since for bigger files it is more likely to get packets lost and therefore it would be necessary to resend them after 1.5 seconds (see Section 5.3.1).
Appendix C

Firepass Measurements

The current appendix gives information on the tests developed over the tool Firepass. It is important to notice that there are different versions available at Dyatlov [2003], however, we will use the version 1.1.2a, which is the latest. Moreover, as we have not the possibility of using an external HTTP server we have worked within the same LAN. Anyway, the tests performed in Smeets and Koot [2006] were also developed inside the same local network so this seems to be not a big issue.

We will follow quite a similar structure to the one used in previous appendixes. However, first of all a few lines about the configuration of the tool will be added.

C.1 Configuring Firepass

This section is useful to determine needed steps to be able to get Firepass working since we have encountered some problems when trying to follow the recommendations in Smeets and Koot [2006].

Server

Firstly, copy the fpserver directory contained in the .tar.gz file downloaded into /var/www/ which is the apache local directory for web pages.

Subsequently, we added the following line to the httpd.conf file (in /etc/apache2/):

`ScriptAlias /fpserver/ /var/www/fpserver/`
This line indicates that any query under the directory fpserver will be attended by /
var/www/fpserver/ that is where our fpserver.cgi script is placed.

In addition, it is necessary to create the “inout” and “log” directories under /var/temp/ where the server should be able to read and write because it is used as a data exchange directory.

Finally, run fpserver.cgi with the configure option and the path to the configuration file, but before we needed to modify the first line in the fpserver.cgi script so that the bash could find the perl interpreter which, in Ubuntu, is under /usr/bin/perl instead of /bin/perl

```bash
./fpserver.cgi configure conf/fpserver.conf
```

**Client**

No modifications were made to the fpserver configuration file conf/fpclient.allow but a line needed to be inserted into the client’s rules file (conf/fpclient.rules) to allow the use of iperf for our usual baseline measurement. The added line is the following:

5002 tcp 10.0.26.2 5001 tcp

This means that the fpclient will listen to (local) incoming connections in port 5002/tcp and will ask the Web server (where the fpserver.cgi script will be running) to redirect them to actual service provider (10.0.26.2:5001/tcp) where the server will be running (see Figure 5.7).

(OBS: In Smeets and Koot [2006] they provide that line in a wrong order. The two last elements should be the other way round: “5002 tcp 172.16.16.11 tcp 5001”).

## C.2 Firepass throughput

Once more we use iperf tool to measure the maximum throughput of the tunnel provided by firepass. The steps are the following:

i. Launch iperf server in 10.0.26.2 with the command

```bash
iperf -s
```
ii. Start the fpclient in 10.0.26.4:

./fpclient.pl conf/fpclient.conf 10.0.26.3/fpserver/fpserver.cgi

Which stands for “use the config file in conf/fpclient and execute the fpserver.cgi script located in 10.0.26.3”.

(OBS: The server address must be given without “http://” however, in Smeets and Koot [2006] the command line is given with it, resulting in a network resolution error).

iii. Connect the the iperf client in 10.0.26.4 to the local port 5002:

iperf -c 127.0.0.1 -p 5002

The results yielded are presented next:

---------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
---------------------------------------------
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 33050
[ 4] 0.0- 9.7 sec 5.05 MBytes 4.36 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 33051
[ 5] 0.0-10.0 sec 4.16 MBytes 3.50 Mbits/sec
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 33052
[ 4] 0.0- 9.9 sec 4.18 MBytes 3.54 Mbits/sec
[ 5] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 33053
[ 5] 0.0- 9.9 sec 4.12 MBytes 3.49 Mbits/sec
[ 4] local 10.0.26.2 port 5001 connected with 10.0.26.3 port 33054
[ 4] 0.0-10.2 sec 4.12 MBytes 3.38 Mbits/sec

Although the first connection was faster, the average throughput is around 3.5 Mbits/s. Smeets and Koot [2006] modified the window size and experimented a higher throughput and we did likewise obtaining also an improvement, reaching around 5.3 Mbits/s.

C.3 Firepass Tests

In order to transmit the files we use netcat combined with the command time to measure how long does it take. This is executed in 10.0.26.4 with the following command:
We added a new line in the conf/fpclient.rules file telling the it to listen in port 4444 to tcp connections and to connect to the service (nc) in port 5555/tcp.

```
4444 tcp 10.0.26.2 5555 tcp
```

All tests performed by Smeets and Koot [2006] to this tool turned out to be unsuccessful. The same amount of bytes were missing in all repetitions for each of the files transmitted. For 1MB files there were 40960 bytes left and 81920 bytes for 10 MB files. They state these losses are always in the shape of a power of two, however, it is easy to calculate that “40960 = 4096 * 10 = 2^{12} * 10 = 2^{13} * 5” and “81920 = 8192 * 10 = 2^{13} * 10 = 2^{14} * 5” but instead they are in the shape of multiples of two.

We proceed to show the results obtained in the tests we performed in order to compare them to their results.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time Real</th>
<th>Time User</th>
<th>Time System</th>
<th>KBytes/sec Ideal</th>
<th>KBytes/sec Real</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>0m2.546s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
<td>402.19</td>
<td>344.28</td>
<td>NO, 147456 bytes missing</td>
</tr>
<tr>
<td>1 (2)</td>
<td>0m2.590s</td>
<td>0m0.000s</td>
<td>0m0.012s</td>
<td>395.36</td>
<td>342.40</td>
<td>NO, 147456 bytes missing</td>
</tr>
<tr>
<td>1 (3)</td>
<td>0m2.550s</td>
<td>0m0.000s</td>
<td>0m0.012s</td>
<td>401.56</td>
<td>343.74</td>
<td>NO, 147456 bytes missing</td>
</tr>
<tr>
<td>10 (1)</td>
<td>0m23.011s</td>
<td>0m0.000s</td>
<td>0m0.152s</td>
<td>445.00</td>
<td>436.81</td>
<td>NO, 188416 bytes missing</td>
</tr>
<tr>
<td>10 (2)</td>
<td>0m23.240s</td>
<td>0m0.000s</td>
<td>0m0.148s</td>
<td>440.62</td>
<td>432.51</td>
<td>NO, 188416 bytes missing</td>
</tr>
<tr>
<td>10 (3)</td>
<td>0m23.181s</td>
<td>0m0.004s</td>
<td>0m0.148s</td>
<td>441.74</td>
<td>433.61</td>
<td>NO, 188416 bytes missing</td>
</tr>
</tbody>
</table>

Table C.1: Firepass Results for 0.5 seconds delay.

In order to check the results we will use the commands “diff” and “wc”. No script was written since there are very few files to check.

We stands for word count, but in fact there is an option (-c) which allows the printing of the number of bytes. This is the option we have used and it has yielded the following information from the original transmitted files:

```
root@ruben-desktop:~# wc -c 1000KB.txt
1024000 1000KB.txt
root@ruben-desktop:~# wc -c 10000KB.txt
10240000 10000KB.txt
```
From the results in Table C.1 we can determine that the application is not reliable at all. Furthermore, the number of missing bytes are still divisible by two. The bandwidth of the application varies from 400 KBytes/s to 445 Kbytes/s however the 1 MB tests showed a constant reliability of 85.6% and the 10 MB tests showed also constant reliability of 98.16%. This constant number of missing packets is what surprises us the most but we have not been able to figure out what causes it.

Due to this constant number of missing packets we decided to perform the same tests with different delays between HTTP requests. We chose a 1 second delay (see Table C.2) and 5 seconds delay (see Table C.3), which is the value recommended by the author (in fpclient.conf) for non critical time protocols to prevent noise in httpd and HTTP proxy logs.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>KBytes/sec</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBYtes</td>
<td>Real</td>
<td>User</td>
<td>System</td>
</tr>
<tr>
<td>1 (1)</td>
<td>0m4.551s</td>
<td>0m0.004s</td>
<td>0m0.004s</td>
</tr>
<tr>
<td>1 (2)</td>
<td>0m4.590s</td>
<td>0m0.004s</td>
<td>0m0.012s</td>
</tr>
<tr>
<td>1 (3)</td>
<td>0m4.550s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
</tr>
<tr>
<td>10 (1)</td>
<td>0m36.032s</td>
<td>0m0.012s</td>
<td>0m0.112s</td>
</tr>
<tr>
<td>10 (2)</td>
<td>0m36.240s</td>
<td>0m0.012s</td>
<td>0m0.148s</td>
</tr>
<tr>
<td>10 (3)</td>
<td>0m37.375s</td>
<td>0m0.000s</td>
<td>0m0.132s</td>
</tr>
</tbody>
</table>

Table C.2: Firepass Results for 1 second delay.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>KBytes/sec</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBYtes</td>
<td>Real</td>
<td>User</td>
<td>System</td>
</tr>
<tr>
<td>1 (1)</td>
<td>0m10.424s</td>
<td>0m0.004s</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>1 (2)</td>
<td>0m20.539s</td>
<td>0m0.000s</td>
<td>0m0.008s</td>
</tr>
<tr>
<td>1 (3)</td>
<td>0m15.488s</td>
<td>0m0.004s</td>
<td>0m0.008s</td>
</tr>
<tr>
<td>10 (1)</td>
<td>2m53.109s</td>
<td>0m0.000s</td>
<td>0m0.128s</td>
</tr>
<tr>
<td>10 (2)</td>
<td>3m18.302s</td>
<td>0m0.004s</td>
<td>0m0.152s</td>
</tr>
<tr>
<td>10 (3)</td>
<td>3m13.245s</td>
<td>0m0.000s</td>
<td>0m0.144s</td>
</tr>
</tbody>
</table>

Table C.3: Firepass Results for 5 seconds delay.
From this two new sets of experiments it is possible to observe that even increasing the inter-packet delay, the application is still unreliable. Furthermore, in some cases there have been increases on the number of bytes missing comparing it to other situations with a shorter delay. Surprisingly, the percentage of satisfactory received bytes is in all cases higher with 10 MB files.
Appendix D

Ozyman Measurements and DNS Tips

This appendix will focus on giving insight on the basics of DNS in order to be able to understand better how this implementation works. Once the introduction to DNS comes to an end the presentation of the results of the tests will presented.

D.1 Introduction to DNS

DNS (Domain Name System) is a distributed and hierarchical database which stores information related to domain names in networks such as Internet. In fact, the most common usage is the mapping between domain names and IP addresses. This is not only because the it is easier to remember the name but also because the IP address might change under many circumstances.

DNS is organized in domains which are subdivided to subdomains and so forth until the hostname. The top domain is the “root” and the domains below are called Top Level Domains (TLDs). That is the reason why the domain names are dotted names, each dot specifies a subdomain. Therefore, a name like www.his.se would correspond to the host “www” in the subdomain “his” from the top level domain “se”. In theory, this subdivision can go down to 127 levels deep, and each label can contain up to 63 characters, as long as the whole domain name does not exceed a total length of 255 characters.

There are three main components in DNS. The client or resolver, which generates
the name resolution requests; the *name servers*, which reply to the requests (by using TCP and UDP on port 53) originated from the clients with the information stored in the resource records; and the *zones of authority*, which are subdivisions of the domain name space managed by different name servers.

The information from each zone of authority is stored in a local file within the DNS server. This file might contain several types of records. The most commonly used are the following: A (address) record maps the name of a host to an IP address. CNAME (canonical name) record is an alias of one name. MX (mail exchange) record is used to define a list of mail servers for a domain. NS (name server) record defines the main DNS authoritative servers in a domain. TXT (text) record allows to insert arbitrary text into a DNS record. An important example:

```
<table>
<thead>
<tr>
<th>Host</th>
<th>Type</th>
<th>Record</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a06rubri.net</td>
<td>IN NS</td>
<td>dns1.a06rubri.net</td>
<td></td>
</tr>
<tr>
<td>dns1.a06rubri.net.</td>
<td>IN A</td>
<td>10.0.26.3</td>
<td></td>
</tr>
<tr>
<td>fakedns.a06rubri.net</td>
<td>IN NS</td>
<td>dns.fakedns.a06rubri.net</td>
<td></td>
</tr>
<tr>
<td>dns.fakedns.a06rubri.net.</td>
<td>IN A</td>
<td>10.0.26.2</td>
<td></td>
</tr>
</tbody>
</table>
```

This is making the host “dns1.a06rubri.net” with IP address “10.0.26.3” be the name server for the domain “a06rubri.net”. Moreover, “dns.fakedns.a06rubri.net” with IP address “10.0.26.4” is set to be the name server in all subdomains under “fakedns.a06rubri.net”, that is, the main name server dns1 delegates all responsibilities to the other name server.

This is basic for the preparation of the name server we mounted in order to get all packets redirected to the fake name server, in which will be running the Ozyman’s server implementation.
D.2 Testing Ozyman

It has turned out to be a difficult task to run the tests for this implementation. Many problems have appeared, not only because it was the first time we have worked with DNS but also because of bugs (i.e. errors) on the code which hindered the development of the tests.

Since Kaminsky’s DNS Tunnel implementation utilizes the standard input/output it was no possible to carry out the usual preliminary test of the network throughput with iperf.

Once the DNS server (BIND was used) was working and properly configured to redirect the traffic to the fake server as we showed the previous section we started to perform the tests.

Both the client (*droute.pl*) and the server (*nomde.pl*) had bugs. We will point them out in order to explain the problems that they provoke.

This is the output of the *diff* command with the option -u over the two files. The minus sign (-) on the left indicates that a line has been modified and the plus sign (+) indicates that a line has been added. It possible to appreciate two modifications. On the first one the error impedes the user to insert a service and the IP address and port of the machine to forward the packets of the clients. The second bug has to do with the binding of the local IP address and port. This is actually not important since by default it uses...
the local address.

On the client there was a similar problem with the retrieval of the options. The function -m, which was in charge of getting the delay in microseconds between the transmission of two consecutive packets, was also malfunctioning and it the delay was fixed to 100 microseconds (i.e. 0.01 seconds).

We will describe next how the tests were run. It is necessary to remember that the primary DNS server was configured to delegate all incoming requests for a subdomain under fakedns.a06rubri.net to 10.0.26.2, where we run the fake server:

```
root@10.0.26.2~# ./nomde.pl -i 10.0.26.2 -L netcat:10.0.26.3:5555 fakedns.a06rubri.net
```

It means that the fake server will attend in 10.0.26.2 to all requests under the domain netcat.fakedns.a06rubri.net and will forward the data obtained to the port 5555 on 10.0.26.3.

Thus, we run the recipient on 10.0.26.3 which will receive the packets with netcat with the usual command.

```
root@10.0.26.3~# nc -l -p 5555 > file.txt
```

Finally, start the client of the tunnel (droute.pl) which will receive the files to send from a pipe (|) and will ask the legitimate name server for the domain netcat.fakedns.a06rubri.net and therefore will be redirected to the forged one. This is done with the following command:

```
root@10.0.26.4~# cat file.txt | ./droute.pl netcat.fakedns.a06rubri.net
```

On the other hand, it was necessary to measure the capacity of the channel in order to compare our results the those provided by Smeets and Koot [2006]. Until now, we had been using the Linux command `time` but this was no longer possible. The reason is that as we are using netcat (acting as server) to receive the files but the client is using droute.pl to send it is no possible to close the transmission after sending the EOF character. Remember that the -q option is only valid from the client side.

Therefore, as in Smeets and Koot [2006] is not suggested the way in which they performed these measurements, our idea was to create a script to read the increments of size of the file during the transmission and with that information obtain an average of the bytes/second. This perl script is shown next:
#!/usr/bin/perl -w
#
# Calculates the Bytes/s from the size of the file during the reception
# and the final size of the file.

use strict;
use Time::HiRes qw(sleep); # Replaces normal sleep with a more fine-grain one

# Variables
my $prog='./timing.sh';
my $file= $ARGV[0];
my $seconds=$ARGV[1];
my $outfile;

# Declaration of signal 'INT' handler
$SIG{'INT'} = 'control_C_handler';

if ($#ARGV != 1){
  print "Usage: $prog <FILE> <SECONDS>\n";
  exit -1;
}

$outfile="results.".$file;
while (1){
  system("ls -al $file >> $outfile");
  sleep $seconds;
}

#---------
# SUBROUTINES 
#---------

sub control_C_handler{
  my @sizes;
  my $i=0;

  open (FILE, "$outfile");
  while (<FILE>){
    if (/(/\D+)(\d+)(\D+)(\d+)/){
      $sizes[$i] = $4;
    }
  }
}
$i++; 
}
close(FILE);

# Present the results and remove temporal file
print "Results from $file: \n";
print &bytes_per_sec(@sizes)." Bytes/sec\t";
print $sizes[#$sizes]." Bytes\n";

system("rm $outfile");
exit 1;

sub bytes_per_sec (@){
    my @array = @_;  
    my $temp = 0;  
    my $count = 0;  
    # counts number of differences != 0

    # the number of bytes/sec = mean (diff[i]/seconds)
    for (my $i = 1; $i <= $#array; $i++)
    {
        my $diff = $array[$i] - $array[$i-1];
        if ($diff){
            $temp += $diff /$seconds;
            $count++;
        }
    }

    if ($count > 0){$temp / $count;}
    else{ 0;}
}

The idea was simple, however, the results obtained where not promising. We observed
that even using small periods of time between two consecutive measurements, the size of
the file were not increased in different proportions. We first thought that the problem
might lie in accuracy of the ls function, that is to say, that it only made new estimations
of the size when there was a considerable increase. However, the increase was substantial
110 bytes. Therefore, that idea was discarded. Finally, we realize that it was because of
the amount of data received per packet which was exactly 110 bytes (see Figure D.2).
Subsequently, another perl script was created making use of a module to measure statistics from the network interfaces. This module is Sys::Statistics::Linux::NetStats. However, the results contained information about all the inbound and outbound packets of the interface so it was not useful since the only packets we are interested in are those from the tunnel.

The final decision was to measure the time “manually”, setting up Wireshark (i.e. a network protocol analyser) with a proper filter and observing when does the last TCP packet with data headed to the port 5555 arrives. The following tables have been extracted from those results.
Table D.1: Ozyman’s Results with 0.01 seconds delay.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>Throughput</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBytes</td>
<td>Seconds</td>
<td>Ideal(B/s)</td>
<td>Real(B/s)</td>
</tr>
<tr>
<td>1 (1)</td>
<td>1.12</td>
<td>914.28</td>
<td>914.28</td>
</tr>
<tr>
<td>1 (2)</td>
<td>1.07</td>
<td>957.00</td>
<td>957.00</td>
</tr>
<tr>
<td>1 (3)</td>
<td>1.08</td>
<td>948.14</td>
<td>948.14</td>
</tr>
<tr>
<td>10 (1)</td>
<td>10.17</td>
<td>1006.88</td>
<td>1006.88</td>
</tr>
<tr>
<td>10 (2)</td>
<td>10.49</td>
<td>976.16</td>
<td>976.16</td>
</tr>
<tr>
<td>10 (3)</td>
<td>10.40</td>
<td>984.61</td>
<td>974.03</td>
</tr>
<tr>
<td>100 (1)</td>
<td>104.42</td>
<td>980.65</td>
<td>975.38</td>
</tr>
<tr>
<td>100 (2)</td>
<td>103.00</td>
<td>994.17</td>
<td>984.56</td>
</tr>
<tr>
<td>100 (3)</td>
<td>104.74</td>
<td>977.65</td>
<td>965.05</td>
</tr>
</tbody>
</table>

Table D.2: Ozyman’s Results with 0.05 seconds delay.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>Throughput</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBytes</td>
<td>Seconds</td>
<td>Ideal(B/s)</td>
<td>Real(B/s)</td>
</tr>
<tr>
<td>1 (1)</td>
<td>1.97</td>
<td>519.79</td>
<td>519.79</td>
</tr>
<tr>
<td>1 (2)</td>
<td>2.00</td>
<td>512.00</td>
<td>512.00</td>
</tr>
<tr>
<td>1 (3)</td>
<td>2.05</td>
<td>499.51</td>
<td>499.51</td>
</tr>
<tr>
<td>10 (1)</td>
<td>19.53</td>
<td>524.32</td>
<td>524.32</td>
</tr>
<tr>
<td>10 (2)</td>
<td>20.21</td>
<td>506.67</td>
<td>506.67</td>
</tr>
<tr>
<td>10 (3)</td>
<td>19.85</td>
<td>515.86</td>
<td>515.86</td>
</tr>
<tr>
<td>100 (1)</td>
<td>197.30</td>
<td>519.00</td>
<td>519.00</td>
</tr>
<tr>
<td>100 (2)</td>
<td>197.05</td>
<td>519.66</td>
<td>519.10</td>
</tr>
<tr>
<td>100 (3)</td>
<td>197.40</td>
<td>518.74</td>
<td>517.07</td>
</tr>
</tbody>
</table>

Table D.3: Ozyman’s Results with 0.1 seconds delay.
The results presented in these tables show that as long as the delay is increased the throughput is obviously reduced and the number of missing bytes is also reduced. Furthermore, it is possible to realize that the amount of bytes lost is in most cases a multiple of 110, this is because the packets sent are usually comprised of that quantity of bytes as we pointed out before (figure D.2).

From all these results we can conclude that this application does not provide a reliable channel of communication. Though the amount of packets lost are not excessively important, in the case of ciphered data the lost of a minimal part of the information ruins the decryption. Moreover, it is important to remember that these tests have been developed within a local area network when this application is supposed to work in the wilderness of Internet.

Nevertheless, these problems due to packet losses are justified since the author of the application states in [Kaminsky, 2004] that the tool is appropriate for shell and IM (Instant Messaging) only.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time</th>
<th>Throughput</th>
<th>Correct Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBytes</td>
<td>Seconds</td>
<td>Ideal(B/s)</td>
<td>Real(B/s)</td>
</tr>
<tr>
<td>100 (1)</td>
<td>756.27</td>
<td>135.40</td>
<td>134.96</td>
</tr>
<tr>
<td>100 (2)</td>
<td>756.48</td>
<td>135.36</td>
<td>135.36</td>
</tr>
<tr>
<td>100 (3)</td>
<td>756.50</td>
<td>135.36</td>
<td>135.06</td>
</tr>
<tr>
<td>100 (1)</td>
<td>942.40</td>
<td>108.65</td>
<td>108.65</td>
</tr>
<tr>
<td>100 (2)</td>
<td>942.69</td>
<td>108.62</td>
<td>108.62</td>
</tr>
<tr>
<td>100 (3)</td>
<td>942.61</td>
<td>108.63</td>
<td>108.51</td>
</tr>
</tbody>
</table>

Table D.4: Ozyman’s Results with 0.4 and 0.5 seconds delay.