

Application Layer Covert Channel Analysis and Detection

CO42019 Honour Project

UNDERGRADUATE PROJECT DISSERTATION

**Submitted in partial fulfilment of the requirements of
Napier University for the degree of
Bachelor of Science with Honours in Networked Computing**

Zbigniew Kwecka
Matric No. 03008457
BSc (HONS) Networked Computing

Supervisor:
Prof. William Buchanan

Second Marker:
Dr. Neil Urquhart



NAPIER UNIVERSITY
EDINBURGH

Authorship declaration

I, Zbigniew Kwecka, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed.
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work.
3. I have acknowledged all main sources of help.
4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself.
5. I have read and understand the penalties associated with plagiarism.

Signed

Name: **Zbigniew Kwecka**

Matric No.: **03008457**

Abstract

The specification of Internet protocol stack was developed to be as universal as possible, providing various optional features to the network programmers. Consequently the existent implementations of this specification use different methods to implement the same functionality. This created situation where optional fields and variables are often transmitted only to be ignored or discarded at the receiving end. It is considered that transmission of these fields significantly reduces the bandwidth available to data transfers, however the redesign of the network protocols from various reasons is considered impossible at the present time, and this downfall of Internet protocol stack is silently accepted. Since the optional fields discussed are of no real value anymore, they are often left unmonitored. This in turn allows for implementation of covert channels.

Techniques of information hiding in covert channels have been known some time now. By definition it involves hiding information in the medium, which is not usually used for any form of information transfer. For an instance the purpose of the envelope in the standard mail communication is to enclose the message and provide space for addressing. However, even if the messages were under strict surveillance, information hidden under the stamp on the envelope could go unnoticed to the examiner. This is how covert channels operate. They use resources often perceived as safe, and unable to carry data, to hide covert payload.

This dissertation investigated Internet protocol stack and identified Application Layer as the level most vulnerable to covert channel operations. Out of the commonly used protocols, SMTP, DNS and HTTP have been recognized as those, which may carry hidden payload in and out secure perimeters. Thus, HTTP, the protocol which is often wrongly perceived as text based information transfer protocol, due to its innocently sounding name was further investigated. Since there is no tool available on the market for HTTP monitoring, a set of test tools have been developed in this project using C# programming language, which is starting to become a new networking industry standard for application deployment. The analysis of the current trends in covert channel detection and the statistic collected on the current implementations of the protocol lead to design and implementation of suitable HTTP covert channel detection system. The system is capable of detecting most of the covert channel implementations, which do not mimic the operation of HTTP browser driven by a user. However, the experiments also proved that for a successful system to operate it must fully understand HTTP protocol, recognise signatures of different HTTP implementations and be capable of anomaly analysis.

Contents

Authorship declaration	2
Abstract	3
Contents	4
List of figures and tables	6
Acknowledgments	7
1 Introduction	8
1.1 Project Overview	8
1.2 Background	8
1.3 Covert Channels Terminology	9
1.4 Project Aims and Objectives	10
1.5 Thesis Structure	11
2 Theory	12
2.1 Introduction	12
2.2 Covert Channels Definition	12
2.3 Potential Users of Covert Channels	13
2.4 TCP/IP Model	14
2.4.1 Network Layer	14
2.4.2 Internet Layer	14
2.4.3 Transport Layer	14
2.4.4 Application Layer	15
2.5 Examples of Application Layer Covert Channels	15
2.5.1 HTTP	15
2.5.2 DNS	16
2.6 Conclusions	17
3 Literature Review	18
3.1 Introduction	18
3.2 Covert Channel Classification	18
3.2.1 Storage and Timing Channels	18
3.2.2 Noisy and Noiseless Channels	19
3.2.3 Aggregated and Not-aggregated Channels	19
3.3 Application Layer Covert Scenarios	19
3.3.1 Reordering	21
3.3.2 Case Modification	21
3.3.3 Use of Optional Fields and Flags	21
3.3.4 Adding a New Field	22
3.3.5 Using Linear White Spacing Characters	22
3.3.6 Modifying Server Object	22
3.4 HTTP Protocol	23
3.4.1 HTTP Syntax and Covert Channels	23
3.4.2 General syntax	24
3.5 Detection	27
3.6 Conclusions	28
4 Design	29
4.1 Introduction	29
4.2 Evaluation Environment	29
4.3 Covert Channels Detection System	29
4.4 Experiment Design	31
4.4.1 Experiment 1 – Implementation Specific Data Gathering	31
4.4.2 Experiment 2 – Request Information Filtering	31
4.4.3 Experiment 3 – Headers Modification	32
4.4.4 Experiment 4 – Browser Signature Recognition	33
4.4.5 Experiment 5 – Covert Channel Detection	33
4.4.6 Experiment 6 – Analysing Prototype's Load on Test Network	34
4.4.7 Experiment 7 – Code Mobility Check	34
4.5 Conclusion	34

5	Implementation	36
5.1	Introduction	36
5.2	Testing Network	36
5.3	Foundation Software	37
5.3.1	HTTP Analysers	37
5.3.2	HTTP Proxies	39
5.3.3	Web browsers	40
5.4	Experimental Applications	40
5.4.1	HTTP Dumper	40
5.4.2	OffLine HTTP Analyser	41
5.4.3	Filtering Proxy	41
5.4.4	Data Hiding Proxy	42
5.4.5	Browser Caller	43
5.4.6	Browser Timer	45
5.5	Covert Channel Detection System Prototype	45
5.6	Conclusions	46
6	Experiment Data Analysis	47
6.1	Introduction	47
6.2	Experiments	47
6.2.1	Experiment 1 – Implementation Specific Data Gathering	47
6.2.2	Experiment 2 – Request Information Filtering	49
6.2.3	Experiment 3 – Headers Modification	50
6.2.4	Experiment 4 – Browser Signature Recognition	51
6.2.5	Experiment 5 – Covert Channel Detection	52
6.2.6	Experiment 6 – Analysing Prototype's Load on the Test Network	53
6.2.7	Experiment 7 – Code Mobility Check	54
6.3	Conclusions	55
7	Discussion, Conclusions and Further Work	56
7.1	Introduction	56
7.2	Discussion & Prototype Evaluation	56
7.3	Test Inadequacies	58
7.4	Conclusions	58
7.5	Further Work	60
8	References	61
9	Appendices	63
	Appendix 1 - Experiment 2 Results	63
	Appendix 2 – Experiment 3 Results	65
	Appendix 3 - HTTP Protocol	67
	Appendix 3 – Project Presentation	74
	Appendix 4 - Inline Filtering Agent - Code Listing	79
	Appendix 5 - HTTP Analyser Foundation - Code Listing	93
	Appendix 6 – Browser Timer - Code Listing	104
	Appendix 7 – Browser Caller - Code Listing	108
	Appendix 8 – Data Hiding Proxy - Code Listing	114
	Appendix 10 - HTTP Dumper - Code Listing	121
	Appendix 11 – Experiment 1 - Code Listing	130
	Appendix 12 – Experiment 2 & 3 - Code Listing	137

List of figures and tables

Figure 1-1 Communicating parties A and B, with eavesdropper E	8
Figure 2-1 Anatomy of misuse (Summers, 1996)	11
Figure 2-2 TCP/IP to OSI Model Mapping	12
Figure 3-1 HTTP Header's Reordering (Kwecka, 2006)	19
Figure 3-2 HTTP Header Name Case Modification (Kwecka, 2006)	19
Figure 3-3 Use of Optional Header Values (Kwecka, 2006)	20
Figure 3-4 Use of Unrecognised Header (Kwecka, 2006)	20
Figure 4-1 Firewall Protected Intranet	26
Figure 4-2 Detection and Filtering Network Setup	27
Figure 5-1 Test Network Topology	32
Figure 5-2 HTTP Analyser Foundation	34
Figure 5-3 HTTP Proxy Foundation	35
Figure 5-4 Web browser Foundation	36
Figure 5-5 HTTP Dumper GUI	36
Figure 5-6 OffLine HTTP Analyser	37
Figure 5-7 Filtering Proxy	37
Figure 5-8 Data Hiding Proxy	38
Figure 5-9 Data Hiding Scenario	38
Figure 5-10 Browser Caller	39
Figure 5-11 Browser Timer	40
Figure 6-1 HTTP Headers Usage Statistics	41
Figure 6-2 Browsers' Signatures	42
Figure 6-3 Responses to Requests with 'Host' Header Filtered Out	43
Figure 6-4 Threat Detection	46
Figure 6-5 Median Time Taken to Perform a Full Download	47

Acknowledgments

I would like to thank Professor William Buchanan, for his guidance and support throughout this project. In addition, I would like to thank Dr Neil Urquhart for being part of the marking process.

1 Introduction

1.1 Project Overview

The foundation of the current most popular data channel, the Internet, is made of protocols, which allow a considerable amount of “freedom” to their designers. Each of those protocols was defined based on a number of vendor specific implementations, in order to provide common procedures for cross vendor communication. Thus, every millisecond there is thousands of bits of optional and redundant information being exchanged between computers from around the World. Those bits may be employed by intruders, criminals, and possibly even terrorists in various types of malicious activity, since they are usually treated as irrelevant and ignored by the security systems. There are very high chances of those bits being used by perpetrators to implement *covert channels*, which are a form of secret communication medium. This poses a large risk to public and private information confinement.

The overall aim of this Honours project is to investigate data hiding in the Application Layer of Internet protocol suite and the main focus is on the detection of covert communication. Therefore, research into technology and knowledge required to build a successful covert channel detector and limiter were conducted. This included literature review of recent research publications dealing with covert channels, white papers and RFCs of specific technologies. In addition a prototype of Covert Channel Detection System (CCDS) was designed and implemented in order to evaluate data gathered. The intentions were to establish whether detection and elimination of the covert channels is possible, and where the further work should be conducted in order to achieve those goals.

1.2 Background

The foundations of Internet were built in accordance to 7-layer Open System Interconnection (OSI) model, suggested by the International Standard Organisation (ISO). Each of the layers provides well-defined services to the layer directly above and exchanges data or control information with corresponding layer on remote machine. It is also capable of employing services from the layer directly below. For well-defined services to operate, protocol stacks were designed to be as universal as possible, and are defined in a way which is called “open”. Most implementations have an open-ended list of protocols that they are capable of providing services to. For example, the Layer 3 IP protocol carries an 8-bit protocol type field, thus allowing it to transfer 255 different Layer 4 protocols, of which only 138 are defined, 115 are free for further development, and 2 are left for testing and experimental purposes.

IP is the most widely employed protocol for computer networking and we can clearly state that its versatility greatly contributed to this. However, the flexibility of Internet communication protocols, which allowed for the dream of simple data sharing, has a trade off, which is security. Optional protocols’ fields and variables which are transmitted only to be ignored or discarded at the receiving end, pose a large threat for information confinement. Thus, organizations which permit any form of communication of their employees, or computer systems, with the outside world, consequently consent for an arbitrary data leakage from their networks. Of course the

companies try to protect themselves from Internet threats in various ways, such as with firewalls, proxies, and so on, but most of them focus only on direct information transfers, and neglecting the possibility of transferring data by other means. A statefull firewall, often considered as sufficient protection by network administrators, acts like a locked gateway, allowing packets on selected Layer 4 ports to leave and the return packets to enter the secure perimeter. Thus only connections which originated from within the network can proceed. Unfortunately, not all of these devices are perfect and methods for their deception exist, but what is even more important is that most of the time a message originating from within a network is allowed onto the Internet. Thus, this kind of protection is not enough to ensure confidentiality of corporate and private data. Some organisations may say that they trust all the users of their network, and there is no need to filter data leaving the network. But most of them neglect the fact that the users of the network are not only human operators, but also automated software and hardware. Furthermore not all of these automated systems are legitimate, and some of them are malicious agent installed by hackers, or accidentally downloaded by legitimate system users.

The numbers of existing implementations of malicious agents are large and grow every day, so to protect against data leakage and dangerous software many organizations implement Proxy servers and intrusion detection systems (IDS). Proxy servers are protocol specific tools that act almost as transceivers¹, where they check and, if required, modify data transfers between two environments, where the IDSs constantly monitor the network traffic, examining data content, statistic and any other information useful in detection of malicious operations. These precautions give higher level of protection, however, most of them examine only legitimate data channels, that in the data payload and the transaction headers, while the legitimate data channels are not the only way that information may leave secure perimeter. B. W. Lampson in his document “A Note on the Confinement Problem” (Lampson, 1973) was first to formally suggest possibility of creating computer communication channels by employing methods originally not intended for any form of communication. He called them *covert channels* and considered them to be one of the greatest threats to confinement of information.

1.3 Covert Channels Terminology

The term *covert channel* describes a secret communication technique employed by two or more parties allowed to exchange information, while they assume the data channel in use is under surveillance. Thus, they modify the content of the genuine low security message or the envelope used to carry this message, so that eavesdropper cannot read from the secret channel.

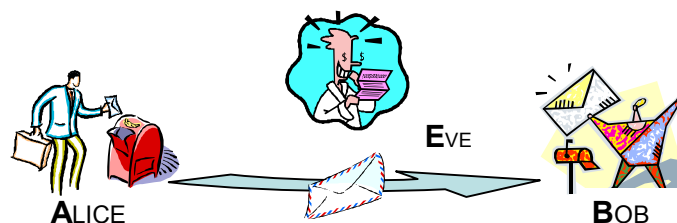


Figure 1-1 Communicating parties A and B, with eavesdropper E

¹ device capable of both transmitting and receiving, often used to allow connections between two different technologies, devices or transmission mediums;

Traditionally all the documents relating to *data hiding* and *cryptography*, since Ron Rivest's article presenting RSA cryptosystem in 1978, sender is not being called *person A* but is named *Alice*, and the receiver is called *Bob* instead of *person B* (Rivest, 1978). This scheme is used in this report, to make the discussions easier to follow, for the convenience of the reader. Thus, eavesdropper system in this scheme is also called *Eve* (Figure 1-1).

In covert channel scenarios *Alice* is often considered to be an inmate of a high security prison. It is assumed that she knows an escape plan from a prison where *Bob* is spending his sentence. *Alice* is trying to send the escape plan to *Bob*, however *Eve*, the governor checks their communication very precisely, thus they employ covert channel know to them to sent the secret messages. Figure 1-2 illustrates basic method of information hiding employing plain text message as a *carrier* (first letters of the words in the message combined together form: *LetTheMissionBegin*).

Let everyone tango.
This has Edward's
mind in some simple inquiry of nothing,
before everyone gets into Nirvana.

Figure 1-2 *Cover Channel* (Buchanan, 2006)

1.4 Project Aims and Objectives

This project began with intend of analysing possible counter measures to Application Layer *covert channels* technologies. Therefore following aims were specified:

- (a) Research possible carriers of covert information at Application Layer of Internet protocol suite.
- (b) Prototype suitable CCDS.
- (c) Evaluate the prototype and suggest a framework for identifying detection systems' sensitivity, to various types of covert traffic, as well as capability of introducing noise into the suspected channels.

In order to accomplish these aims, following objectives were also defined:

- (a) Investigate current research of covert channels and technologies available for their detection and limitation.
- (b) Choose a programming language suitable for prototyping of CCDS.
- (c) Define testing environment for the prototype's evaluation.
- (d) Propose further research areas.

1.5 Thesis Structure

- Chapter 1 **Introduction.** This chapter provides the overview of the work performed for the needs of this Honours Project Dissertation
- Chapter 2 **Theory.** Some of the underlying theory of Internet protocol stack and precise definitions of the term covert channel are given in this chapter.
- Chapter 3 **Literature Review.** This chapter provides the background of the current research conducted in the field of covert channels. Also suggestion of the suitable methods of covert chapter detection and prototype evaluation are provided.
- Chapter 4 **Design.** This chapter provides high level overview of proposed prototype.
- Chapter 5 **Implementation.** The detail behind the implementation of experiments and prototyped system are unrelieved in this chapter.
- Chapter 6 **Experiment Data Analysis.** The results form the experiments performed are presented in this chapter, together with early evaluation of the findings.
- Chapter 7 **Discussion, Conclusions and Further Work.** This chapter summarises the work performed for this dissertation, presents the findings and suggest further work required in this field.
- Chapter 8 **References**
- Chapter 9 **Appendixes**

2 Theory

2.1 Introduction

This chapter will provide some background to the technologies discussed in the report. Thus, a precise, up-to-date definition of *covert channels* will be provided here, however, in depth discussion of their classification and implementation will follow in Chapter 3. This section will also analyse the *anatomy of misuse*, in order to identify possible usage and users of the data hiding techniques.

The TCP/IP model will be discussed as a framework on which Internet protocol stack is actually based. This should explain why this project has focused on Application Layer protocols, at the same time, neglecting that implementation of *covert channels* is possible in any of the four layers of the TCP/IP model. Finally, two examples of how Application Layer *data hiding* techniques may be implemented will given.

2.2 Covert Channels Definition

Since Lampson's first discussion on computer-based covert channels (Lampson, 1973), technology has moved forward, constantly tightening security, and thus forcing data hiding techniques to transform. Hence, current sources vary in their precise definitions of the term *covert channel*. Some simply describe it as communication channel used to transmit information employing a method not originally intended for this kind of transmission (Lampson, 1973; Wikipedia, 2005), where others classify it under a transfer of information which violates security policy of the transfer system (Tsai, 1990; Gligor, 1993; PCMAG.COM, 2005). Furthermore, the term *steganography* describes hiding data in a different ways and is often incorrectly applied in regard to covert channels. Thus, to clarify the focus of this document, two most comprehensive definitions, will be combined together.

“Communication channel that can be exploited ... to transfer information in a manner that violates the system's security policy”
(NCSC, 1985)

Given a nondiscretionary (e.g., mandatory) security policy model M and its interpretation $I(M)$ in an operating system, any potential communication between two subjects $I(S_h)$ and $I(S_i)$ of $I(M)$ is covert if and only if any communication between the corresponding subjects S_h and S_i of the model M is illegal in M .

(Tsai, 1990)

The National Computing Security Center's (NCSC) definition, although very similar to Tsai's does not emphasize the fact that the *security policy* must be implemented and *nondiscretionary* for the covert channels to exist. Sadly Tsai's definition while very precise and complete is highly complex. Thus, for purpose of this document, the following definition, is used when considering covert channels:

Communication channel that can be exploited ... to transfer information in a manner that violates the system's **nondiscretionary** (e.g., **mandatory**) security policy.

2.3 Potential Users of Covert Channels

To understand who, would be interested in utilizing covert channel technologies, and why they would like to use them, the definition of this term (Section 2.2) must be interpreted. Thus, it should be noted that according to this, the main aim in exploitation of covert channels is violation of a specific *security policy*. Therefore we can classify and threat these technologies as *misuse* tools and according to *anatomy of misuse* there is a suitably motivated perpetrator behind any kind of threat (Figure 2-1).

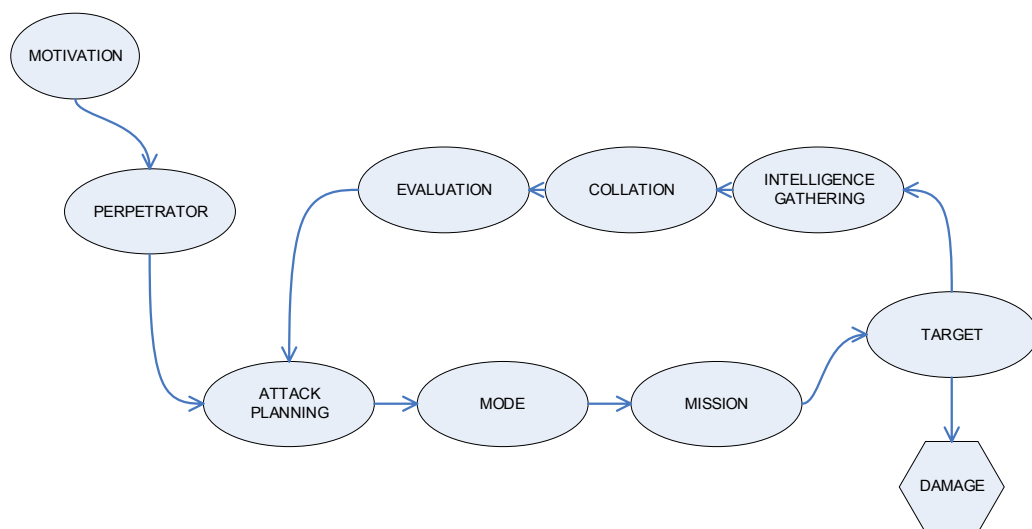


Figure 2-1 Anatomy of misuse (Summers, 1996)

For a misuse to occur there must be a perpetrator with some motivation.

Perpetrators can be classified depending on their motives or location in respect to the *secure system*. Thus categorising, based on motives, there are three groups (Summers, 1996):

- (a) Individuals driven by personal feelings, as disaffection or revenge.
- (b) Hackers motivated by curiosity, politics and culture or simply by greed.
- (c) Spies working for intelligence or commercial market.

Furthermore, according to Washington Post (Coll, 2005) now over four years after the September 11th, al. Qaeda has migrated from space to cyberspace and they will certainly use any technology they can to succeed in their battle. Therefore we think terrorist should be considered as another group with specific intention of misuse, especially since implementations of cover channels offer great secret communication tool.

Individual intruders can be either insiders or outsiders depending on their relation to NCAS. So that employees, contractors, customers or their families would be considered as insiders, and anybody who does not fit in one of those groups is treated as an outsider.

2.4 TCP/IP Model

As described in Section 2.1 ISO recommends the 7-layer OSI model for development of transmission protocols. Each of these layers provides well-defined services to the layer directly above and exchanges data or control information with corresponding layer on remote machine utilizing services of the layer directly below. In practice only few protocols were developed strictly adhering to OSI's model, while the existing Internet protocols can be easily mapped onto less complex 4-layer TCP/IP model. Figure 2-2 illustrates mapping between those two different multi-layered approaches to computer networking. It is possible to hide data in any layer of the Internet protocol stack, however, each layer provides different characteristics for possible covert channels. Remaining part of this Section will briefly describe operations and discuss possible information hiding scenarios for each layer in TCP/IP model.

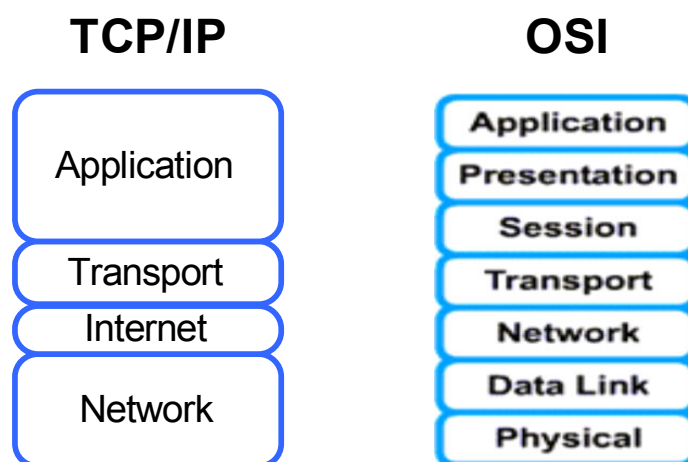


Figure 2-2 TCP/IP to OSI Model Mapping (Kwecka, 2006)

2.4.1 Network Layer

The Network Layer in TCP/IP model deals with functions described in Physical and Data Link Layers of the OSI model. Thus it deals with “getting data across one particular link or medium” including “physical characteristics of transmission” equipment (Odom, 2001, pp. 77). Consequently the major task of the protocols operating on this layer is to transfer packets between any two subsequent internet layer capable devices on the end-to-end connection.

2.4.2 Internet Layer

Internet layer is responsible for end-to-end packets delivery and it is equivalent to OSI network layer. Thus it defines logical network addressing and deals with all the operations required for packets to reach the remote host. These operations include but are not limited to routing, fragmenting and queuing of the PDUs - protocol data units (Odom, 2001).

2.4.3 Transport Layer

Transport layer provides for end-to-end transparent transfer of application layer data. Due to this feature it is often perceived to be a basic form of middleware for distributed applications. Thus if error recovery, flow control and similar functionality is required, transport layer will provide for it.

2.4.4 Application Layer

The top level of TCP/IP model is called application layer, since it is usually provided by specific software application rather than an operating system. Consequently it is the equivalent to three subsequent layers in OSI structure: session, presentation and application. Thus it is responsible for maintaining end-to-end communication sessions, representation of data being transmitted as well as application specific user interface (Odom, 2001).

Any part of data exchange between user and transport layer is considered to be performed at application layer. Thus, since messages produced on this level employ transport layer technologies for the purpose of end-to-end data delivery, and high-level data streams are usually not being altered by security applications, covert channels hidden in these messages have the widest range possible. Therefore a covert data send by *Alice* is almost guaranteed to reach *Bob* unaltered. For networks which use Proxy or SOCKS servers to protect themselves the syntax of messages transferred may get changed (Fielding, et al, 1999), however application layer control fields or data should not be altered in any significant way, thus allowing for *noiseless* covert channels. However, the list of advantages of application layer data hiding doesn't end here, since the lower layers of TCP/IP model are provided by operating system rather than user application it is virtually impossible or extremely hard to modify PDU of these layers by an unprivileged user. Therefore some easy steps, such as disallowing user access to kernel and employing host authentication for local networks, may significantly reduce risks to information confinement created by these layers. These threats have been known for some time now and most of the network access control systems (NACS) are prepared to defeat them (Dyatlov, et al, 2003). On the other hand the Application Layer is different, in most cases users are allowed by the operating system security policy to execute their own applications and to create outgoing connections through NACS. While some NACS use Proxy or SOCKS servers, any application which generates messages of protocols allowed through these security devices and is executed by (or on behalf of) an authenticated and authorized user would be able to transmit data outside secure perimeter.

2.5 Examples of Application Layer Covert Channels

Section 2.4 identified Application Layer as the only layer in the TCP/IP model where *covert channels* may be allowed to pass through well built NACS. Therefore, two examples how such hiding techniques could be implemented in this Layer will follow, to illustrate the problem.

2.5.1 HTTP

HTTP is a request-response protocol employing MIME-like syntax for control information, and more precise description will follow in the Literature Review Section. However in depth knowledge of this protocol is not required to appreciate following example. Thus, a typical request for a website on the root directory (/) of the host with `www.napier.ac.uk` DNS name, produced by a Mozilla based web browser on demand of a user, would be sent across TCP connection encoded as an ASCII string, with following syntax:

```
GET / HTTP/1.1
Host: www.napier.ac.uk
Connection: close
User-Agent: Mozilla/5.0 (WinNT)
```

HTTP transactions are usually transparent to the human operators, and under normal circumstances they would be unable to see above request generated on their demand by a web browser, however, good covert channel implementation may stay unexposed even after a visual examination. In this particular data hiding example, the technique exploit the fact that HTTP treats any amount of consequent linear white space characters (optional line feed, spaces and tabs) in the same way as a single space character (Fielding, et al, 1999). Therefore, it is possible to encode information using these nonprintable characters, since if received by the HTTP eavesdropper they would be normally discarded. The following text exposes spaces [SP], tabs [HT] and line feeds [CRLF] in the request given above:

```
GET[SP] / [SP]HTTP/1.1 [CRLF]
Host:[SP]www.napier.ac.uk[SP] [HT] [SP] [SP] [HT] [SP] [SP] [SP] [CRLF]
Connection:[SP]close[SP] [HT] [HT] [SP] [HT] [SP] [SP] [HT] [CRLF]
User-Agent:[SP]Mozilla/5.0 [SP] (WinNT) [CRLF]
```

This message would be treated by a standard web server as valid and error free, however paying attention to the abnormal amount of white space at the end of lines 2 and 3 a suspicion may be raised as to the purpose of those nonprintable characters in this particular request. In this particular scenario Alice is using tabs [HT] to represent binary ones and spaces [SP] to hide binary zeros in a covert message sent to Bob. They both know, that 8-bit encoded into the second and third lines of the request represent different ASCII characters and together they are part of the hidden message (covert payload). Thus the first character would have hex value of 0x48 (01001000) and the value of the second one would be 0x69 (01101001), together in human readable form they spell word “Hi”.

In this example HTTP transaction data (later called HTTP envelope) would be considered as a carrier and the characters encoded into white spacing a payload. This is one of the ways, in which hiding information in HTTP may be achieved.

2.5.2 DNS

This example use of Application Layer covert channel employs UDP as a Transport Layer protocol. Once again the full understanding, how the Domain Name System (DNS) works is not necessary to appreciate the syntax of the covert channel implementation proposed. The basic operation of DNS is to provide name to IP address resolution in a similar manner that phone directory provides telephone number of given institutions or individuals. Most of DNS servers keep permanently only a small number of name-to-address mappings, but are capable of providing any IP address via recursive-lookup on demand from the client. The mappings obtained in this way are usually kept in the cache of the server, for some time before they time-out. Nonrecursive-lookups are also possible, and these forbid the server from relaying the request, thus, a mapping will only be returned if the server is the originator of the mapping or has it in its cache.

Let assume that Alice wants to send ASCII encoded information to Bob. They have previously agreed eight names of very unpopular websites or services²:

0 – www.boring-web.com

² the names of the websites chosen at random, let assume they all have IP mappings in the global DNS

- 1 – www.e-zb.com
- 2 – www.unpopular.co.uk
- 3 – ...
- 4 – ...
- 5 – ...
- 6 – ...
- 7 – www.notfunnyjokes.net

They have also decided upon the storage of their covert messages, a particular DNS server which allows and caches recursive-lookups (most servers do). Thus, making a recursive-lookup to this server against an existing DNS name will result in server responding with an IP address and storing the mapping in the server's cache. However, if a nonrecursive-lookup is made server will only provide the mapping if a specific entry exists in its local database or cache.

Thus, Alice is capable of mapping ASCII characters, one at the time, onto the carrier, the cache of the DNS server chosen. Consequently for any bit of the ASCII character set to one, Alice will make an appropriate recursive-lookup to the server, for instance a lookup of www.e-zb.com if the second highest bit is one in the particular ASCII character being sent. Then Bob can perform nonrecursive-lookups of all eight addresses and by means of determining which mappings are in the cache of the server, determine the ASCII character sent by Alice. To send another character they need to wait till the entries will time-out in the server's cache (Kaminsky, 2004).

2.6 Conclusions

This section has provided the definition of the term covert channel in the scope of this report as a communication channel that can be exploited ... to transfer information in a manner that violates the system's nondiscretionary (e.g., mandatory) security policy. This helped in identification of potential users of this data hiding technology, as suitably motivated perpetrators. Thus, individuals, hackers, spies and terrorist reasoning on their own motives or working on behalf of somebody else, may want to reach for covert channels to achieve certain goals. It is, therefore, of most importance that suitable detection and prevention measures are developed.

Application Layer of the TCP/IP model was identified, as a protocol most likely to be employed as a carrier of the current covert channels implementations. This was largely due to the fact that implementations in lower layers would most likely be stopped by NACS's Proxy servers before the payload reaches the target. However, another reason the Application Layer covert channels were chosen as the subject of investigation in this project is that their implementation is relatively easy, and does not require kernel level access to the operation system.

3 Literature Review

3.1 Introduction

The aim of this project is to analyse possibilities of employing application layer data hiding and to suggest possible ways of securing networks from this kind of activity. Prototype prevention and detection systems will be designed and implemented to evaluate the findings of the research. Thus, it was decided that the range of the research should be limited to the protocols most commonly allowed to pass through firewalls of large organizations. The literature review that follows tries to answer these research questions while focusing mainly on the information contained in recent research publications dealing with covert channels, white papers and RFCs of specific technologies (HTTP, SMTP, etc). The usual use for covert channels, however, is to violate system security policy. Therefore the biggest, but unofficial research in this area is conducted underground, by hackers around the world. Their ideas should not be underestimated, as they are usually the first to employ new covert channel solutions. Fortunately for the society some of them do it only to prove that such a violation is possible and share the knowledge with the World. Thus, some of the BlackHat³ documents were considered to be valuable sources of information and will also be discussed in this section.

3.2 Covert Channel Classification

There are various ways of the covert channels' classification, however, most do not apply to the theory of the data hiding, but to its specific implementations (Gligor, 1993). In this document we will consider three different classification schemes described below.

3.2.1 Storage and Timing Channels

The NCSC suggested following distinction between *storage* and *timing* channels:

Covert Storage Channel - A covert channel that involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process. Covert storage channels typically involve a finite resource (e.g., sectors on a disk) that is shared by two subjects at different security levels.

Covert Timing Channel - A covert channel in which one process signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process.

(NCSC, 1985)

We consider that in above definitions, and similar descriptions found, the distinction between storage and timing channels depends largely on the interpretation of the terms *storage location* and *system resources*, since any storage location is usually a part of certain system's resources. Thus, in one of the covert channels implementations investigated, the inventor suggests engaging DNS servers in

³organisation dealing with Internet security, which gathers leading corporate professionals, government experts, and members of the underground hacking community.

exchange of hidden information (Kaminsky, 2004). In this scenario a local cache of DNS server known to both sender and receiver would be frequently modified by the sender and queried by the receiver using standard DNS requests. Thus we could classify this scenario as either storage or timing, based on our view as to the local DNS cache, namely: is it a storage location, or a system resource. Therefore above definitions although valid were on behalf of NCSC further clarified by Gligor:

A channel is a storage channel when the synchronization or data transfers between senders and receivers use storage variables, whereas a channel is a timing channel when the synchronization or data transfers between senders and receivers include the use of a common time reference (e.g., a clock).

(Gligor, 1993)

Furthermore Gligor notices that above classification is more design and implementation specific, rather than based on data hiding theory employed. Thus any timing channel may be transformed into a storage channel by keeping timing information in a locally stored variable, and, vice-versa, a storage channel may be altered to form a timing channel by starting to observe relative timing of events.

3.2.2 Noisy and Noiseless Channels

Covert channels scenarios can be also divided into *noisy* and *noiseless*. The noiseless channels employ a resource which is exclusive to sender and receiver. Thus allowing for their uninterrupted communication, with no error correction or detection required. Whilst noisy contains these channels which utilise a resource shared among many different processes, where distinguishing between data from a sender and operations of other users (the noise) would be required. Consequently, the receiver needs to implement some form of error detection, with an error correction or sender notification system. (Gligor, 1993) Therefore HTTP based covert channel example specified in Section 2.5.1 is noiseless, since it stores hidden payload inside an object originating from a sender and transmitted directly to the receiver, this assuming a direct HTTP client-server connection. Whilst DNS example from Section 2.5.2 is noisy, because the DNS cache can be accessed and modified by virtually any system connected to the Internet.

3.2.3 Aggregated and Not-aggregated Channels

In any kind of communication the capability of a channel to transmit certain amount of data over given time (the bandwidth) is important. Thus, covert channel's theory defines *capacity* (Lampson, 1973) which is the amount of payload that may be transmitted per unit of *carrier*. Consideration of this leads information hiding designers to use multiple carrier objects to increase the bandwidth of covert channels, of small carrying capacity. Thus, covert channels may utilize several storage locations, or system resources, at any point in time to transfer data, either in serial or parallel manner. Those are considered to be *aggregated*, and the ones which use only a single carrier object are called *non-aggregated*.

3.3 Application Layer Covert Scenarios

Things as simple as *last accessed* timestamp on a networked file (Loepere, 1989), or the pure fact that a DNS server has, or has not, got an entry in its local cache

(Kaminsky, 2004) can become engaged in secret exchange of information. The examples of Application Layer covert channels were provided in Section 2.5, thus following discussion on the subject will concentrate on underlying theory of data hiding at this Layer.

A key focus was to select possible carries for Application Layer covert channels. There are theories that a covert channel, which can transmit only a small payload per amount of carrier or unit of time, can do no harm. However, Lampson demented these theories with an example of expert agent secretly monitoring a military communication system. Such a system could employ a covert channel capable of transmitting one bit per day to alert the enemy about specific circumstances, like planned invasion (Lampson, 1973). The cold war is over now, however, still there are applications where even a very slow covert channel may be of considerable risk. Thus, if an expert system is capable of monitoring a network and detecting conditions which render this network's IDS useless even for a short period of time, it could inform the potential perpetrator, that now is the time to perform secret attack.

DNS is a classic example of a good carrier for covert channels and various malware, since because of its small carrying capabilities it hardly ever gets any focus in security planning. Thus, when this Honours project was starting, DNS was commonly perceived as harmless address enquiry protocol, however, after recent DDoS attacks using the DNS servers as reflectors, this perception is slowly beginning to change (Kawamoto, 2006). Moreover, the small PDUs of the DNS, has not stopped members of BlackHat community using global DNS to tunnel live voice data and the innovator, Kaminsky, suggested that still much greater bandwidths can be reached by splitting the load over thousands of DNS servers (Kaminsky, 2004).

Among all the different Application Layer protocols, only three were chosen capable to carry hidden information in and out of zones commonly perceived as secured:

- (a) DNS (Kaminsky, 2004; Forte, 2005)
- (b) HTTP (Borders, 2004)
- (c) SMTP (de Vivo, 1998)

This is largely due to the fact that modern institutions cannot function without access to HTTP and SMTP, for business purposes, while, these protocols would not work sufficiently without the DNS providing name to address translation. All free protocols are ones of the oldest around, thus, include many functions which are rarely used currently. Thus, studying the appropriate RFCs, DNS (Mockapetris, 1987; Elz, 1997), HTTP (Fielding, 1999) and SMTP (Postel, 1982; Klensin, 2001), suggestion of de Vivo that Internet security was traded for easiness of sharing (de Vivo, 1998), proved to be correct. Above RFCs suggest syntax and wording of some basic commands and headers, but make no real attempt to limit the scope of their specifications, thus allowing for the future expansion of optional features in the protocols, and, therefore, security gaps, as well.

Generally there were five different methods of implementing covert channels within application layer header found, their description follows.

3.3.1 Reordering

Reordering is illustrated in Figure 3-1. The order of the headers in the HTTP envelope is negligible, and differs from implementation-to-implementation. Modification of the headers order could then be used to encode a covert payload (Dyatlov, 2003).

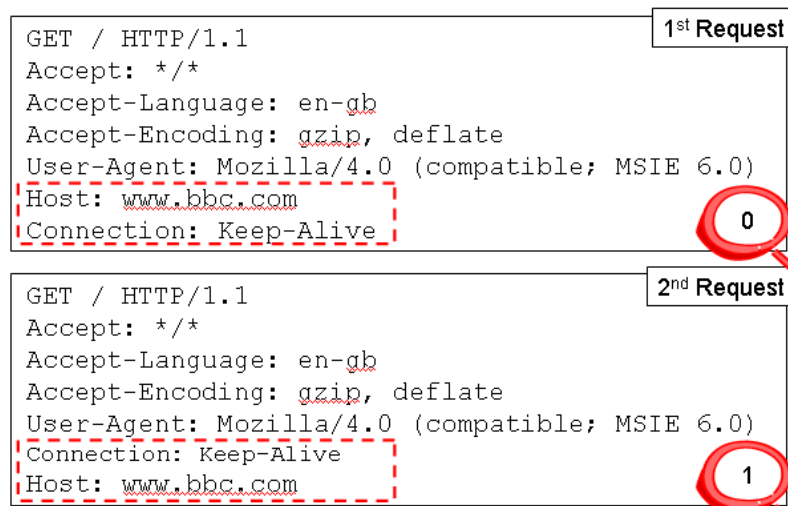


Figure 3-1 HTTP Header's Reordering (Kwecka, 2006)

3.3.2 Case Modification

For easiness of usage the protocols are often case insensitive. Thus, modification of the case of a header name, would be ignored by a standard HTTP application, or mailing agent (Dyatlov, 2003). Thus, it could be used, as suggested in Figure 3-2 to encode bits of ASCII code into lower-case (binary ones) and capital (binary zero) letters.

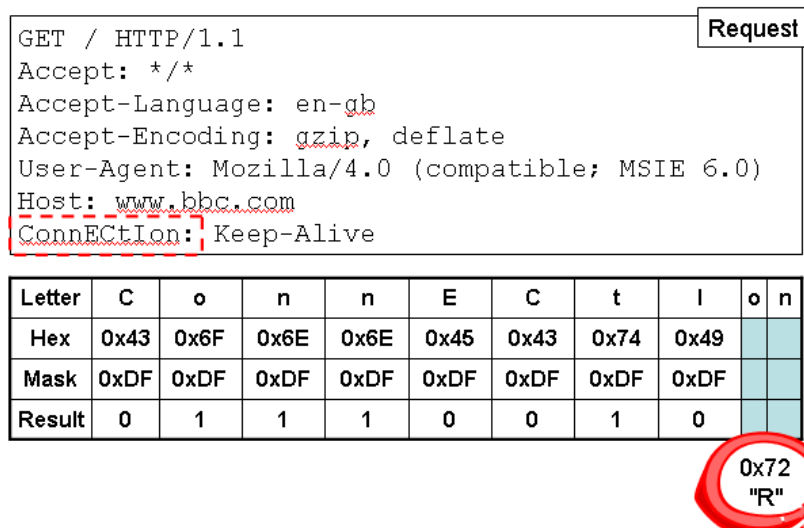


Figure 3-2 HTTP Header Name Case Modification (Kwecka, 2006)

3.3.3 Use of Optional Fields and Flags

The Internet protocols have many unused, or rarely used, fields that could be employed in the transmission of data (Dyatlov, 2003). For instance *Accept* header transmitted from a client to a web server inside HTTP envelope, may precisely define file types accepted in the response, or may provide wildcard (*/**) to show they will

allow any file type in the response. Thus, this feature may be, once again, used to encode data. The example in Figure 3-3 shows a possible covert channel implementation, where a wildcard in the value of the accept header is treated as binary zero, and a specific file type provided as binary one.

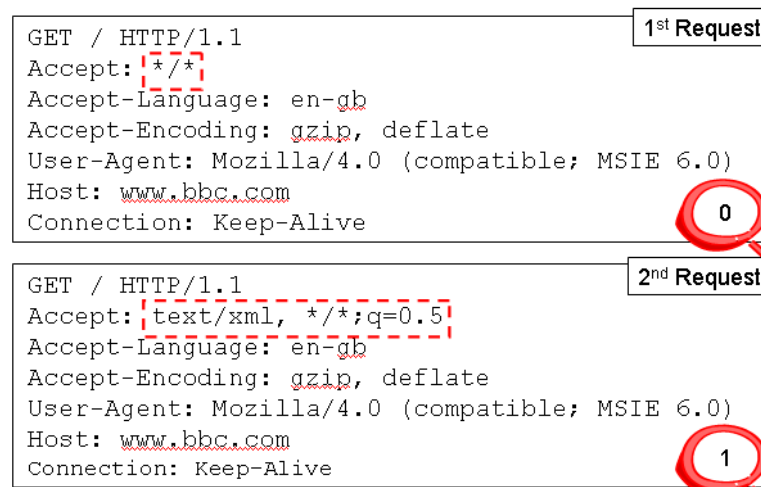


Figure 3-3 Use of Optional Header Values (Kwecka, 2006)

3.3.4 Adding a New Field

As it was stated before, there is no real limitation to the specifications and the new tags could be added to Application Layer envelope (Dyatlov, 2003). Additionally some applications are configured to ignore any unrecognised headers and treat request and responses in a way they would be treated if the problematic header was not there (Fielding, et al, 1999). Thus, scenario from Figure 3-4 could be implemented, where a covert payload is exchanged in a plain text inside HTTP envelope. However, this would be undetectable to *Eve* if standard HTTP software was used to eavesdrop.

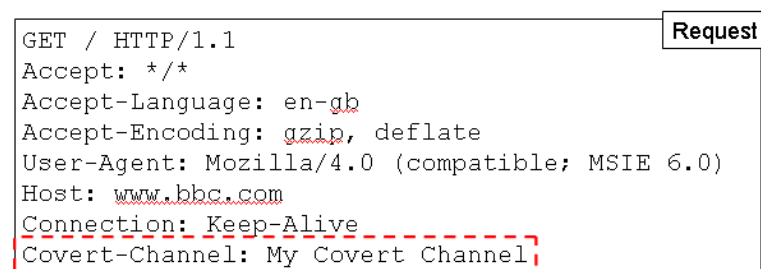


Figure 3-4 Use of Unrecognised Header (Kwecka, 2006)

3.3.5 Using Linear White Spacing Characters

For a web browser there is no difference if there is one or more spaces between HTTP header values, similarly linear white spacing is ignored in SMTP (Dyatlov, 2003). Please refer to Section 2.5.1 for example of linear white spacing header modification.

3.3.6 Modifying Server Object

In this scenario *Alice* and *Bob* could use a previously agreed server object to exchange information. Thus using more objects or altering the frequency of probing could increase the bandwidth (Kaminsky, 2004).

3.4 HTTP Protocol

Following description of HTTP protocol is a part of larger document, written by the author of this dissertation, which highlights the possibilities of implementing covert channels in this Application Layer protocol. The full content of this document is attached to this dissertation in Appendix 4.

The application layer protocol called HTTP is often perceived as very basic protocol for distribution of World Wide Web pages. We could say that even its name Hypertext Transfer Protocol is very suggestive and implies that the purpose of this protocol is to transfer hypertext, where hypertext is defined as textual data “linked” across many documents or locations. It makes no wonder then, that some network administrators do not consider HTTP as a threat or think that as long as only outgoing established connections are permitted and every machine in the network uses some kind of firewall and antivirus software, the network is secure. However the true face of the protocol is different. The most recent specification of HTTP is RFC 2616 and the purpose of the protocol is described as follows:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. (Fielding, et al, 1999, pp. 7)

HTTP is now well established protocol and the current version is 1.1, however the idea of the protocol stayed the same. Through employing a simple human readable (MIME-like) syntax and allowing transfer of virtually any kind of data, HTTP became a preferred protocol in development of “on-line” applications. Furthermore the fact that a large group of network administrators allowed almost any outgoing connections of HTTP either directly or through proxies contributed strongly to this trend. Nowadays almost any software application, which requires communication over the Internet, employs HTTP or has a built-in functionality allowing its application layer protocol to be tunnelled in HTTP. Example of the first kind could be antivirus software that uses HTTP for downloading signatures of the newest threats from the central server, or an update agent for an application like internet messenger. The implementations of the remote method invocation or remote procedure call are, thus, common examples of the second kind of the applications.

HTTP was identified as one of three protocols, which can be employed to create covert channels for sending data in and out of networks commonly considered to be secure. Thus the following section will identify, where RFC 2616 as the document which defines the current version of HTTP in use, gives hackers an open field for hiding data.

3.4.1 HTTP Syntax and Covert Channels

RFC 2616 was created to clear up some hard to understand statements from the previous documentations of HTTP (namely 1.0 and 0.9) and to introduce few optional

features of HTTP/1.0 as standard in the new protocol HTTP/1.1. In the time when this specification was written the biggest concern of the creators was interoperability between all the applications using the new standard and a backward compatibility to the already existing implementations, which conformed to previous RFCs. Some security concerns were raised, regarding leakage of personal information, attacks based on path names and DNS spoofing, however threats of covert channels' implementation was overlooked. Following interpretation of a RFC 2616, describes operation of HTTP and highlights areas where transfer of data in a covert manner is possible.

3.4.2 General syntax

The basic units of HTTP communication are messages, made up of a structured sequence of octets and transmitted via a transport layer virtual circuit (connection). There are two types of messages allowed: requests and responses. Both use generic message format. This consists of a start-line, zero or more header-fields (headers), an empty line and optional message body:

```
generic-message =   start-line           ; a Request-line or a Status-line
                   *(message-header CRLF) ; one or more header
                   CRLF                 ; compulsory empty line
                   [ message-body ]     ; optional application layer data
```

Start-line is made of either a Request-line in a request message or a Status-line in a response message. CRLF is the only end-of-line marker allowed for all HTTP protocol elements except the entity-body. It stands for carriage return (CR) and line feed (LF). This is a good practice, there is only one standard allowed, which makes protocol implementation easier, and prevents information hiding. There are few different types of message-headers: general-header, request-header, response-header and entity-header. All of them are built using the same syntax. Each header consists of a case-insensitive field name followed by a colon and an optional field value.

```
message-header = field-name ":" [ field-value ]
```

Case-insensitivity.

Field-names are case-insensitive, thus they are ideal carrier for hiding bits (payload). Both clients and the servers will interpret a field-name in the same way no matter the case of the letters. The coincidence is that capital letters in the ASCII code differ from the lower case letters only by a value of the 3rd bit. Thus binary form of letter *R* is 01010010 and the one of letter *r* is 01110010. Then a mask of 0xDF can be employed to extract or encode a payload in ASCII characters. Lower case letters would decode as 1's, where capital letters would decode as 0's. An example follows to illustrate how covert payload maybe hidden in a typical HTTP header given below:

```
Connection: keep-alive
```

This header can be modified to carry the bit pattern of 0111001011 and would look as follows:

```
ConnECtIon: keep-alive
```

Consequently above encoding method can be employed to transfer as many bits of payload as the total number of letters in field-names in any particular message. A

visual inspection of the HTTP envelope would reveal this covert channel, however as it is unusual for anybody to examine HTTP message header and HTTP clients and servers would ignore casing, this kind of covert channel could be successfully deployed.

Linear white spacing.

Another reason for concern is the fact that according to RFC 2616 the field-content (the data part of field-value) can be preceded and followed by an optional linear white spacing (LWS). LWS can be made up from a non-compulsory CRLF and one or more space (SP) or horizontal tab (HT) character.

$$\text{LWS} = [\text{CRLF}] \ 1^* (\text{SP}|\text{HT})$$

Header values may be folded onto multiple lines using CRLF as long as the new line starts with a space or horizontal tab. Thus, all linear white space in a header may be replaced with a single SP before processing or forwarding the message downstream. This allows for a text decoration in the HTTP messages and has no real meaning in processing of the requests and responses. However it creates room for bidirectional covert channels. In a case where there is no HTTP Proxy between communicating sites, or linear white spaces are left unaltered by a Proxy, it is possible to encode information using SP and HT characters. An example illustrates how linear white space characters may be employed to transfer covert payload in the following header:

`"Connection: keep-alive"`

Header name *Connection* followed by a colon ":", a space SP and the value *keep-alive*:

`"Connection" ":" SP "keep-alive"`

Let assume that 1's are encoded as HTs and 0's as SPs. Now if the single SP would be replaced with a combination of HTs and SPs, the meaning of the HTTP header would not change, but a binary stream could be hidden in the header. Thus a byte of information which in binary form is 01011100 could be encoded in one of the following ways:

`"Connection" ":" SP HT SP HT HT HT SP SP "keep-alive"`

`"Connection" ":" SP "keep-alive" SP HT SP HT HT HT SP SP`

`"Connection" ":" SP "keep-alive" CRLF
SP HT SP HT HT HT SP SP`

There is virtually no limit to a number of bits per message that can be sent using this method, apart of the size limits set for different kinds of requests and responses. If bits encoded in this way are placed in front of a header value a visual examination of the HTTP message would be enough to reveal the disguise, however if they follow the field-value contents or a CRLF only examination of the message bytes would expose the covert channel.

Order of headers.

Above technique is not the last reason why HTTP messages are such a good carrier for covert channels. The generic-message syntax does not specify the order in which

the headers should occur in the messages. Although it suggests that it is *good practice* to include general-header fields first, followed by request or response specific headers and incorporate entity-header fields as last ones, the order in which header-fields (of differing names) are received is insignificant. Thus if both sides wishing to use covert channel agree that specific order of header-fields is significant they would be capable of transmitting 1bit per two headers of any message. This in turn could be hard to detect, since in the previous scenarios the RFC allowed for creation of the hidden channel, but most of the current HTTP applications used standard semantics (i.e. only one SP before a field-value, ended by a CRLF and all the field-names using title casing) it was possible to spot a potential covert communication quite easy. However here the headers' order vary from implementation to implementation. Thus, for example in a basic covert channel groups of two consequent headers ordered alphabetically could stand for 1's and reverse ordered pairs could decode as 0's. Example:

```
Connection: keep-alive           ;would decode as 1
Host: www.napier.ac.uk
```

```
Host: www.napier.ac.uk          ;would decode as 0
Connection: keep-alive
```

Uniform Resource Identifiers.

All HTTP request and some response (i.e. for relocation purposes) messages consists of uniform resource identifiers (URIs), used to identify a resource on the network. There are two different forms allowed, absolute and relative. Thus a presence of one or the other can be an arbitrary 0 or 1, and if absolute URI is used it should follow syntax:

```
Absolute URI = "http:" //" host [ ":" port ] [ absolute_path [ "?" query ] ]
```

Where *http:* is the scheme name, *host* is a DNS name of a node hosting the resource, optionally followed by a port number and/or absolute path to the resource. RFC 2616 suggest that clients and servers should:

- interpret an empty or not given port as a default port 80
- treat host name and scheme name in a case-insensitive manner
- interpret an empty absolute path as a path of "/" (document root)
- most characters can be represented in their "%" HEX HEX ("%" + hexadecimal value of the ASCII code) encoding

The first statement implies that *http://abc.com/*, *http://abc.com:/* and *http://abc.com:80* have the same meaning in HTTP. Therefore as previously shown optional form of data which is interpreted in the same way by client and server software, can be used to hide covert payload. For instance if a port number is present in a message this can decode as one and when it is omitted it could decode as zero. Allowing for case-insensitive parts of URIs creates similar possibilities as it did in field-names. Furthermore statements with empty absolute paths are treated in a same way as they would request document root ("/"), and again could be used to cipher data. Example:

```
http://abc.com           ;could decode as 0
http://abc.com/          ;could decode as 1, both mean the same to HTTP applications
```

Also any URI ASCII characters which can be send in alternative formats, as a ASCII code, or a ““%” HEX HEX “, could be employed in creating a covert channel. Thus, the following URIs all point to the same resource:

```
http://abc.com/~smith  
http://abc.com/%7Esmith  
http://abc.com/%7esmith
```

Following a question mark (“?”) a query can be add to the URI. This is a common way to transmit data from html forms to the servers. In many cases additional information not required by the server is ignored and individuals can be tempted to use it as a cover channel (Dyatlov, et al. 2003). Although development of an automated system to uncover this type of activity can prove to be a complex task, the channel may be identified by simple visual examination of an address bar in a browser.

3.5 Detection

Typical systems of detection can be divided into three different categories (Castro, 2003):

- (a) *Protocol-based Detection* checks the network transactions for the compliance to the appropriate specification of communication protocol being used. Thus, is capable of detection of abnormalities and variations to standards. Figure 3-5 illustrates this method of detection.

```
HTTP/1.1 200 OK  
Date: Thu, 30 Mar 2006 19:46:22 GMT  
Server: Apache/2.0.54 (Unix)  
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT  
Transfer-Coding: chunked  
Content-Length: 233  
Accept-Ranges: bytes  
Keep-Alive: timeout=5, max=300  
Connection: Keep-Alive  
Content-Type: text/html
```

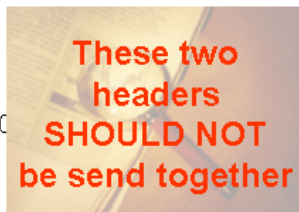


Figure 3-5 Protocol-based Detection (Kwecka, 2006)

- (b) *Signature-based Detection* is using database of signatures against packet contents. There are two different types of signatures used in this method, one type focuses on detection of signatures of known covert channels scenarios, where the second one checks that the packet was originated by genuine software using a particular protocol implementation and that the packet was not modified since. Figure 306 illustrates this concept.

```
HTTP/1.1 200 OK
Date: Thu, 30 Mar 2006 19:46:22 GMT
Server: Apache/2.0.54 (Unix)
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT
ETag: "e9-bb533400"
Accept-Ranges: bytes
Content-Length: 233
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html
```

Apache always
uses "title case"
to generate
message
headers

Figure 3-6 Protocol-based Detection (Kwecka, 2006)

- (c) *Behaviour-based Detection* employs methods of creating profiles of the entire network user. Thus, if any user suddenly changes its usual *habits*, or behaviour of the user becomes suspicious, an alarm is raised.

These various ways of detection of covert channels may be characterised based on the sensitivity and cost (processing) required for their execution. Thus, protocol-based detection systems are usually very simple to implement, and they do not require high processing power, however they can detect only a badly written implementations of covert channels. Signature-based detection is slightly more sensitive and will raise an alarm whenever signature of the specific protocol implementation does not match an entry in a base of allowed message originators. Consequently, the level of processing is usually higher than this of protocol-based detection, but still moderate. The behaviour-based approach to the detection of malicious packets seems to be the most sensitive and very efficient, however, still not 100% precise and prone to *false positives*. Additionally it is characterised by a high processing requirements.

3.6 Conclusions

This chapter has identified three Application Layer protocols: *DNS*, *HTTP* and *SMTP* as the ones, which may be employed to carry a covert payload. Also methods of creating covert channels were noted. Thus, an important conclusion about the requirements for covert channel implementation may be drawn, since all forms of covert channels need some optional fields, values or behaviours to operate. It can be assumed then, that by limiting a number of optional functions of a given protocol, possibility of implementing a covert channel in this protocol is also reduced.

Finally, the chapter discussed ways of detecting covert channel implementation and identified three different methods, each of different characteristic. Protocol, signature and behaviour based detection is possible, however the first one will detect only the badly constructed threats and the last one is capable of detection of virtually any covert channel. The greater the sensitivity of the detection method, the greater is the processing requirement. Thus, the design of the Covert Channel Detection System prototype described in Chapter 4, focused on implementing all three detection methods, as various levels of protection in a configuration, which should not affect the monitored network.

4 Design

4.1 Introduction

This chapter describes how the findings of the Literature Review (Chapter 3) were used to design the prototype of the Covert Channel Detection System. Thus, the typical environment for operation of such a system was defined as well as ways to evaluate the prototype.

4.2 Evaluation Environment

The project aims and literature review have helped us to define the most likely scenarios of usage for Application Layer Covert Channel Detections and Filtering system. We perceive covert channels together with protocol tunneling as a large threat to information security. Therefore, in this project we decided to focus on detecting and filtering possible covert channels traffic outgoing from secure perimeters (i.e. intranets of various organizations), as to protect against arbitrary information leakage. The literature review has identified that most networks are protected by state-full rule based firewalls, with Proxy servers being implemented sporadically.

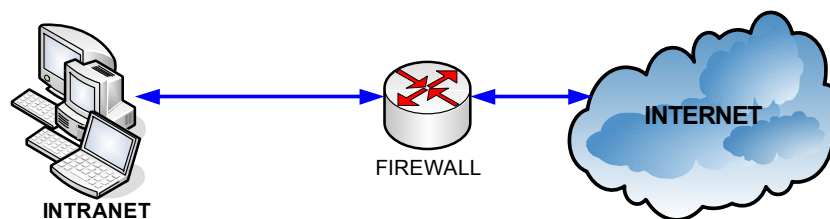


Figure 4-1 Firewall Protected Intranet

The literature review has also shown that a state-full firewall is incapable of providing a reasonable level of protection from threats where connections originate inside the secure perimeter. More precisely, most configurations of state-full firewalls permit any connections that originated from the protected intranet as long as the protocols used are permitted by the Security Policy. Thus, there is no option to provide fine grain filtering. Certainly some organizations use Proxy servers, or even services provided by specialist content filtering companies, such as Bloxx⁴. However their focus once again is concentrated on protection from outside threats. Thus, if the perpetrator was an “insider”, or a hacker, who managed to trick somebody inside secure perimeter to run malicious software, the company would not be able to block the breach in security taking place, if it would exploit policy permitted Internet protocols. Thus our test network will reassemble basic model of institutional network (intranet) connected to the Internet via state-full firewall (Figure 4.1 illustrates this setup). Consequently the next sections of the design process will focus on the design of detection and filtering tools that if located in similar network could help to prevent information leakage exploiting protocol tunnelling and covert channels.

4.3 Covert Channels Detection System

The literature review has identified three different ways of covert channels detection:

⁴ www.bloxx.com

- protocol-based
- signature-based
- behavior-based

The protocol-based detection is the least costly of those three, unfortunately it is able to detect only very poorly crafted threats. At the same time behavior-based methods have proven track of being hard to mislead, but they processing overhead is the greatest. Thus, it is assumed that all free methods must be applied to traffic in various ways, as to produce optimal covert channel detection system. One of the literature review conclusions suggested that well build Proxy server, would be capable of limiting basic, and therefore most common examples of covert channels operating in TCP/IP layer 4 and all of the lower layer implementations as a standard function. Thus, if an organization was to protect itself from covert channels, deployment of filtering Proxy servers would be the first and most important step in this process.

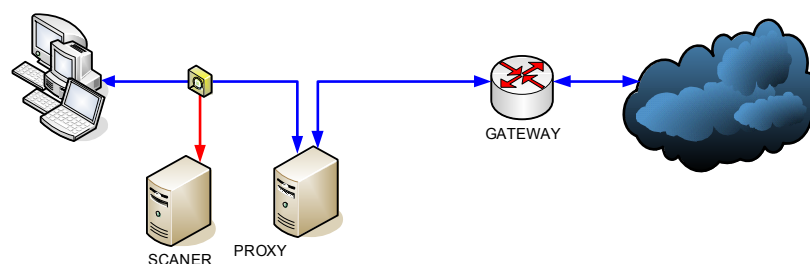


Figure 4-2 Detection and Filtering Network Setup

The way Proxy servers operate, they need to be placed inline with the connections. Thus, they must perform very well as not to create bottle necks in the communication infrastructure. Therefore, level of processing performed by Proxy servers is limited and much lower than that required by behavior-based detection. Consequently we have decided to implement behavior-based detection not as a part of the filtering Proxy, but as a standalone network traffic scanner (Sniffer Detection Agent). Thus the Proxy based agent (Inline Filtering Agent) will be required to perform the protocol and signature based detection.

The IFA will implement following classes/objects:

- *Proxy.cs* – the GUI of the agent
- *Listener.cs* – object listening for the connections from the clients
- *Client.cs* – handler of the requests made by the user
- *Buffer.cs* – class used as an interface between GUI and asynchronous functional classes

SDA should consist of following classes/objects:

- *WinPcap .NET wrapper* – class interfacing between .NET common interface language and C++ code of winpcap.dll
- *HTTPAnalyser.cs* – the GUI of the application, implementing the logic operations as well
- *PacketCollection.cs* – object providing storage for packets within separate TCP connections
- *ConnectionCollection.cs* – object keeping track of various connections heard and ordering them into *PacketCollections*

4.4 Experiment Design

Experiments are the crucial part of this project. As it was mentioned previously the HTTP protocol specification, which we are going to investigate, allows for large variation between the actual implementations. Thus, we will try to identify the differences of those implementations. Hopefully the findings will allow us to create base of Web browsers' signatures, i.e. kind of fingerprint that could be used to precisely identify the User-Agent originating the request. Later we shall experiment with reducing a number of information sent during the requests, so to provide data to the analysis of which fields seems to be disused now. After the analysis of the first set of experimental results an implementation of filtering Proxy and covert channel scanner, will be proposed. Eventually a prototype of covert channel detection and filtering software should be implemented. Thus, in order to test this software some basic covert channel scenarios will be designed and later implemented.

4.4.1 Experiment 1 – Implementation Specific Data Gathering

In this experiment our objective is to collect HTTP requests generated by various Web browsers. We have identified four different Web browsers as the most popular at the present time:

- Internet Explorer
- Firefox
- Opera
- Netscape

These are the browsers web design companies consider when developing websites⁵, since they represent “99.9% of the Web browsers” currently in use. Thus we will need to install all these browsers on a single machine connected to the Internet and use them to access the same set of websites, while creating HTTP traffic dumps on this machine. Ideally the test procedure should be automated and exclude the human factor, to ensure the test conditions for each browser are the same. To do that a piece of software which would allow for timed process execution and termination should be developed. Also another piece of software should be available to collect traffic dumps. After the data gathering phase the packet dumps, which are in binary form, will need to be analyzed, thus third piece of software will be required for this experiment. This software will iterate through packet dumps and extract necessary information.

The software required for this experiment:

- **Browser Caller.** An application triggering Web browsers to request websites from predefined list.
- **HTTP Dumper.** Piece of software employing WinPCap to collect binary dumps of packets from HTTP conversations. Ideally only the packets containing the HTTP protocol envelope should be saved.
- **OffLine HTTP Analyser.** The purpose of this application would be data mining from the binary packet dumps in order to collect experiment results.

4.4.2 Experiment 2 – Request Information Filtering

At this stage we assume that Experiment 1 will result in defining a typical set of HTTP headers used by the most common implementations of HTTP client software, i.e. popular Web browsers. The objective of this experiment is to analyse the World

⁵ information sourced from the directors of Edinburgh based Efero company (www.efero.com)

Wide Web browsers' and servers' implementations regard their usage of HTTP protocol, since literature review findings suggested that a great number of information sent within HTTP conversations is not actually used by either of sides. Thus, once again a piece of software will need to trigger various Web browsers (same set as in Experiment 1) to request websites from predefined list. However, in this experiment we should locate a Proxy server inline with the requests and filter parts of HTTP envelope so that WWW servers receive reduced set of information from the request. Then the servers' responses should be analysed and compared to the responses received after sending complete client requests. Thus, similarly to Experiment 1 the HTTP conversations will be saved in binary dump format and then analysed offline by another piece of software. Ideally the process of collecting the data from modified requests should be simultaneous to the traffic dumps from the unmodified requests, so that the outside factors should not compromise the data collected and the final results.

The software required for this experiment:

- **Browser Caller.** An application triggering Web browsers to request websites from predefined list.
- **Filtering Proxy.** Forward Proxy server, which placed inline with the request, would be able to modify the client-server HTTP flow.
- **HTTP Dumper.** Piece of software employing WinPCap to collect binary dumps of packets from HTTP conversations. Ideally only the packets containing the HTTP protocol envelope should be saved.
- **OffLine HTTP Analyser.** The purpose of this application would be data mining from the binary packet dumps in order to collect experiment results.

4.4.3 Experiment 3 – Headers Modification

Previously, based on findings of Dyatlov and Kaminsky, as well as our analysis of HTTP specification (RFC 2616), we have identified few different techniques, which could be employed to create covert channels in HTTP. They, generally, fall into following categories:

- headers' reordering
- headers' and values' case changing
- usage of optional headers, values or flags
- injection of an undefined header
- usage of various linear spacing characters
- server object modification

Since there is a number of different client and server HTTP software implementations currently used and all of them differ from each other, the purpose of this experiment will be to analyse the practical use of the techniques suggested. The techniques will be analysed in terms of the level of the processing required, noisiness of the channel, and level of discreteness (i.e. if any given techniques generates errors in either client or server software it should not be considered a very good base for covert channel implementation). To get a similar cross section of the implementations available, as in the previous experiments, the BrowserCaller application with the same set of targets will be used to generate genuine requests. Then a specially modified forward Proxy server will be used to modify the request with covert channel simulation. Once again HTTPMessageDump and OffLineHTTPAnalyser will be used to collect the results.

Summarising, the software required for this experiment consist of:

- **Browser Caller.** An application triggering Web browsers to request websites from predefined list.
- **Data Hiding Proxy.** Forward Proxy server, which placed inline with the request, would be able to modify the client-server HTTP flow by applying suggested covert channel techniques.
- **HTTP Dumper.** Piece of software employing WinPCap to collect binary dumps of packets from HTTP conversations. Ideally only the packets containing the HTTP protocol envelope should be saved.
- **OffLine HTTP Analyser.** The purpose of this application would be data mining from the binary packet dumps in order to collect experiment results.

Findings of the Experiments 1 - 3 together with the literature review suggestions will be used to prototype Sniffer Detection and Inline Filtering Agents. Thus, in order to evaluate the project, following experiments will help to collect data required.

4.4.4 Experiment 4 – Browser Signature Recognition

Results from Experiment 1 are expected to provide signatures of the various WWW browsers. In this experiment we will test the capabilities of the Inline Filtering Agent (IFA) to recognise those signatures, i.e. identify client software. However, clients that conform to HTTP specification (Fielding, et al, 1999) should provide a form of identification in a value of User-Agent header, thus, during this experiment we shall not use this value for the recognition purposes. Thus, to prove that application identification is possible even when the User-Agent header is obfuscated, or when malicious software is trying to hide its identity by providing header value associated with genuine software. Therefore, in this experiment user will use various browsers, while they connect to the internet through IFA. The IFA task will be to recognise the signature of the client software and for the purpose of data gathering the Proxy will generate a text file where the outcome of signature matching against User-Agent header value will be stored.

The extra software required for this experiment will be the **Inline Filtering Agent** of the prototyped system.

4.4.5 Experiment 5 – Covert Channel Detection

In this experiment Sniffer Detection and Inline Filtering Agents prototypes will be employed to detect covert channels in the traffic generated by Browser Caller and Data Hiding Proxy. Thus, the results will illustrate the system's success of detection. All HTTP data hiding techniques previously identified will be tested, one at a time as well as few combined together to form aggregated covert channel scenario.

The software required for this experiment:

- **Sniffer Detection Agent**
- **Inline Filtering Agent**
- **Browser Caller**
- **Data Hiding Proxy**

4.4.6 Experiment 6 – Analysing Prototype's Load on Test Network

Another important parameter of the proposed solution to covert channels' detection is its load on the system it is going to be implemented in. Thus, this experiment will measure the time difference in accessing a predefined set of websites when the prototype Proxy filtering agent is inline with the traffic and when the websites are accessed directly. For the purpose of this experiment there is a need of designing an application, which could measure the time taken for a full page download. Thus, ideally, this application would communicate with the HTTP client software, or incorporate HTTP client software itself.

The software required for this experiment:

- **Inline Filtering Agent**
- **Browser Timer** – An application capable of either generating HTTP requests itself or triggering requests using standard WWW browsers, which could measure the time taken for a full website download.

4.4.7 Experiment 7 – Code Mobility Check

The C# .NET was chosen for the prototype's development language and one of the reasons behind this choice was the mobility of the code. Thus, applications developed should be capable of optimal operation on any Windows based platform, with WinPCap installed. Hence, in this experiment components, of the prototype will be tested on variety of hosts running different operating systems. This should help to evaluate the programming language chosen. Consequently the experiment will require heterogenous test network.

4.5 Conclusion

This chapter gave a high level view of the components necessary for development of the Covert Channel Detection System prototype. We have suggested that the system should consist of unless two different types of software agents:

- Inline Filtering Agent (IFA)
- Sniffer Detection Agent (SDA)

The reason behind the suggestion that different types of detection applications are necessary is the load on the system. We consider behaviour-based detection systems as very resource consuming and therefore as unsuitable to be employed in the same machine as real time covert channel filtering agent. Also this chapter have suggested how the prototype evaluation may be performed in practice, by designing an overview of various experiments. Thus, following applications will be required to test the final system and produce results:

- Browser Caller
- HTTP Dumper
- OffLine HTTP Analyser
- Filtering Proxy
- Data Hiding Proxy
- Browser Timer

Looking at the list of the software development required for this project, we can distinguish 3 different families of applications, i.e. HTTP Analysers, HTTP Proxies and HTTP Traffic Generators. Thus, at we hope that implementation of generic foundations for those applications will be possible, so that particular implementations

would require only a limited amount of work. However, these considerations will be tested for feasibility during the implementation phase described in the next chapter. The next chapter will also provide detailed specification of the test network and software used to develop and test the prototype.

5 Implementation

5.1 Introduction

Development of networking application is closely bond to the programming language chosen to implement the product. After a quick research of the tools available on the market, a decision was made to use Microsoft .NET and C#. This platform, designed by Microsoft, speeds up development by freeing the programmer from low-level issues (memory management and etc.) and provides standards Windows controls that can be used in .NET applications. Microsoft .NET becomes more and more popular in networking professionals' community.

5.2 Testing Network

The test network was implemented in a way to reassemble a basic scenario of institutional intranet. Thus, a Cisco router is employed as a border gateway. This router provides network address translation (NAT), dynamic host allocation (DHCP) and state-full firewall services to the intranet. Consequently host machines and servers connect to a Cisco Catalyst 2950 switch attached to the intranet port of the router (Fa0/1). The router's other FastEthernet interface (Fa0/0), is connected to Internet Service Provider (ISP). Since the design phase proposed experiments requiring code mobility testing, the test network is not homogenous. Thus, variety of hosts running different Windows based operating systems is connected to the test network (Figure 5-1).

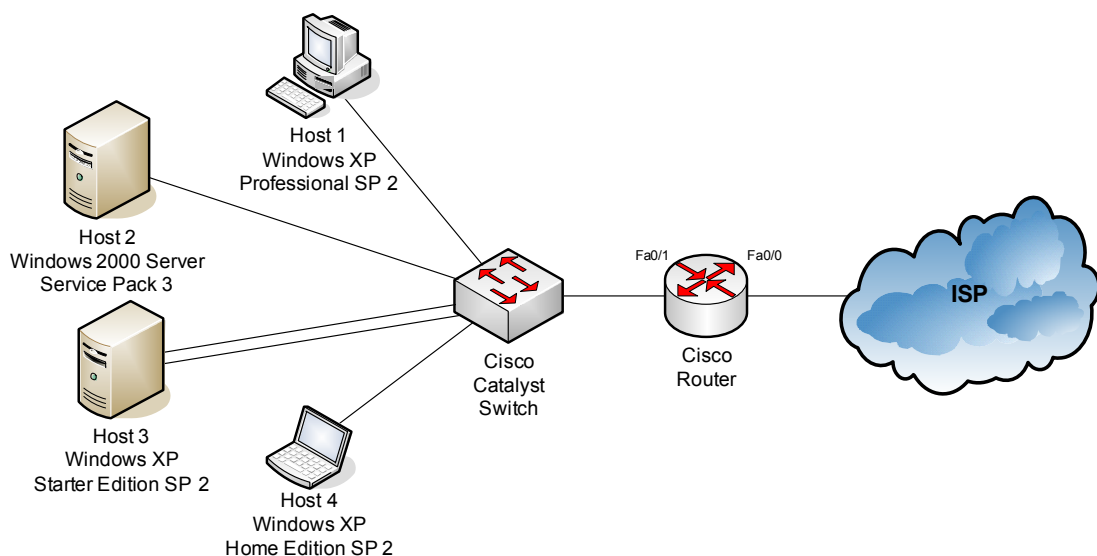


Figure 5-1 Test Network Topology

To provide for the requirements of experiments based on comparison of HTTP responses to modified and unmodified requests, Host 3 is connected to the Cisco switch via two identical links. Thus, two separate Proxy servers can run on this machine at the same time, each with its own listening interface (more details will follow with experiments implementation description). The specification of the test network components can be found in Table 5-1.

Name	Specification
Host 1	OS: Windows XP Professional SP 2 IP Address/Mask: 192.168.1.2/24 CPU: AMD Athlon XP 1800 RAM: 480MB
Host 2	OS: Windows 2000 Server SP 3 IP Address/Mask: 192.168.1.20/24 CPU: Intel Pentium II RAM: 256MB
Host 3	OS: Windows XP Starter Edition SP 2 IP Address/Mask: 192.168.1.7/24 192.168.1.8/24 CPU: Intel Pentium MMX RAM: 64MB
Host 4	OS: Windows XP Home Edition SP 2 IP Address/Mask: 192.168.1.3/24 CPU: AMD Athlon XP 2800 RAM: 768MB

Table 5-1 Component Specification

5.3 Foundation Software

In the design section we have identified that the software required for this project (prototype and experiments) falls into three categories:

- HTTP Analysers,
- HTTP Proxies,
- HTTP Traffic Generators.

Thus, this implementation began with producing of foundation software, base applications easily adaptable to particular tasks, in order to speed up the development.

5.3.1 HTTP Analysers

The Sniffer Detection Agent of the prototype is the major piece of software, which falls into this category together with application, required for the experiments, such as HTTP Dumper and OffLine HTTP Analyser. They all have one thing in common, as they must understand raw network traffic, since they may be required to perform promiscuous mode network traffic monitoring or production of traffic dumps. Promiscuous mode operations on the various kinds of networking adapters, can be performed using standard functions of Unix based operating systems, however in Microsoft Windows environment this functionality is not provided by default. Basic Windows functionality allows for accessing the network only via genuine protocol stack. Therefore, Windows Packet Capture Library (WinPcap) currently treated as industry standard in low-level operations on networking adapters was used during HTTP Analysers base implementation. This gave us capability to (WinPcap Team, 2005):

- capture raw packets, both the ones destined to the machine where it's running and the ones exchanged by other hosts (on shared media)
- filter the packets according to user-specified rules before dispatching them to the application
- produce traffic dumps in libpcap format

- transmit raw packets to the network
- gather statistical information on the network traffic

However, our software employs only the first three functions listed above. Since the prototype was build using C# programming language and WinPcap should be driven by C++, and interface between those two was required. We have tried two different wrappers allowing simplified usage of WinPcap in .NET framework:

- PacketX, commercial ActiveX control⁶
- SharpPcap, freely available network traffic capture library⁷

WinPcap and SharpPcap are an open source packages and their licenses permit redistribution and usage free of charge, however PacketX is a commercial product. Thus, the copyright owners, BeeSync Technologies, were contacted and kindly granted the licence permitting use of PacketX free of charge for duration of this project. After building simple test applications using both wrappers, they both performed to similar level. However, taking into consideration the usability we have decided that SharpPcap designed by Tamir Gel was better for the project. While PacketX done exactly that what we expected, allowed link level reading from a network interface, it produced a downfall in mobility of the code, since it requires installation. On the other hand SharpPcap provides functionality as long as the application has an access to the code library. Additionally, the later one provided high-level information on the data captured, while PacketX produced only raw bytes. Thus code that was necessary to calculate the value of acknowledgment field, when using PacketX:

```
long ack;
int flags_byte = 27 + 4*(Convert.ToInt16(oPacket.DataArray.GetValue(14))& 0x0F);
ack = Convert.ToInt16(oPacket.DataArray.GetValue(flags_byte-5));
ack = ack*256 + Convert.ToInt16(oPacket.DataArray.GetValue(flags_byte-4));
ack = ack*256 + Convert.ToInt16(oPacket.DataArray.GetValue(flags_byte-3));
ack = ack*256 + Convert.ToInt16(oPacket.DataArray.GetValue(flags_byte-2));
```

Could be implemented using SharpPcap in the following way:

```
long ack = oPacket.AcknowledgmentNumber;
```

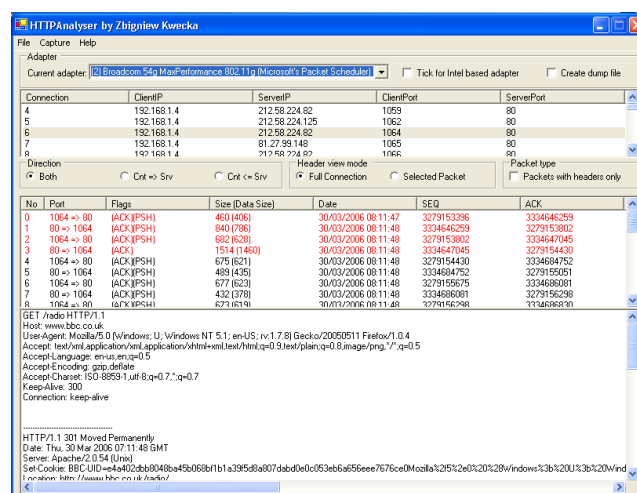


Figure 5-2 HTTP Analyser Foundation

⁶ Autor: BeeSync Technologies; Website: <http://www.beesync.com/packetx/index.html>

⁷ Autor: Tamir Gal; Website: <http://www.tamirgal.com/home/dev.aspx?Item=SharpPcap>

The final solution developed is capable of capturing HTTP traffic, as well as writing and reading tcpdump format. To allow for HTTP connection monitoring the HTTP Analyser Foundation (illustrated in Figure 5-2), splits the traffic based on combination of client IP, client port, server IP and server port. This at the beginning proved to be problematic, since nested ArrayList were chosen to store raw packets. Microsoft guarantees that ArrayLists are thread-safe for read and write operations, however only way to search an ArrayList is to iterate through it, and here was the problem. The program produced errors, when there was a write to an ArrayList under iteration. This problem was solved by using synchronised wrappers for ArrayLists used. Then, for the purpose of HTTP conversation monitoring the application was programmed to “understand” basic TCP and HTTP parameters. Thus, logic of the implementation treats following packets as interesting:

- First packet from client to server after 3XX or zero in length response from the server.
- Packet with TCP sequence number equal to the acknowledgment number from the last client’s request.

Additionally some extra logic was added into the application, following testing. Those improvements included adding a tick box which allows work in promiscuous mode with some Intel based adapters, which proved to inverse logic to that used in most of the adapters. Also some extra filtering capabilities were added, to allow directional visualisation.

5.3.2 HTTP Proxies

In the design section we have identified that development of HTTP Proxy foundation could create a base for implementation of the prototype’s Inline Filtering Agent, as well as applications required for experiments (Filtering Proxy, Data Hiding Proxy). Thus, we have performed an investigation with an objective to find genuine open source Proxy server that could be adjusted to our needs. This resulted in identification of a Mentalis.org Proxy⁸, open source software, with a licence permitting redistribution and modification. While, the original software was a console application, which could perform functions of HTTP, FTP and SOCKS Proxy servers, we have employed only the classes responsible for HTTP operation, i.e. client, HTTPclient, listener and HTTPlistener, out of the original design. Consequently graphic user interface was produced, to control the application. The final HTTP Proxy Foundation (illustrated in Figure 5-3) is using asynchronous system calls. Therefore, an interfacing is needed between the GUI and the functional classes. At first it was troublesome, since the GUI instantiated other classes, and therefore, had a full control over them, but the instances of the functional classes could not communicate back to it. This was solved by the implementation of ConsoleBuffer class, with a number of static variables. Static variables are the same among different instances of the same class, thus different components of HTTP Proxy Foundation implementation use it to communicate between each other.

⁸ Autor: KPD-Team; Website: <http://www.mentalis.org/soft/projects/Proxy/>

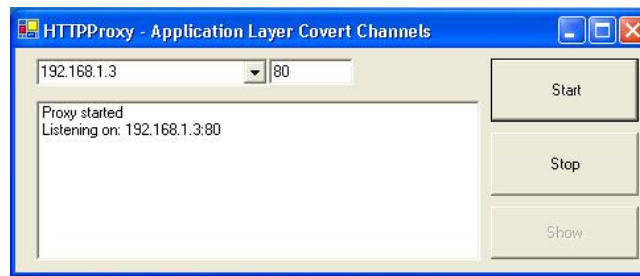


Figure 5-3 HTTP Proxy Foundation

5.3.3 Web browsers

For the purpose of gathering data necessary to produce the prototype and its evaluation, we were required to develop automatic traffic generation software, i.e. Browser Caller and Browser Timer, as identified in the design. Both of these applications have one thing in common, they require to read predefined list of websites and trigger an action associated with the addresses.



Figure 5-4 Web browser Foundation

Therefore, only a timer, a stream reader and a couple of buttons were required to produce this base class. The stream reader is implemented in a way it requires “sites.txt” file in the working directory of the application. This file contains a list of website addresses, one per line. We have manually specified 10 first addresses in this list, where the rest comes from the 2002 Webaward winners list provided by Web Marketing Association⁹. The full list consists of 900 addresses and can be found on the CD attached to this report.

5.4 Experimental Applications

The design section has identified six different pieces of software required to produce the prototype and to test it. Thus, the description of their implementation follows

5.4.1 HTTP Dumper

The HTTP Dumper was implemented to produce tcpdumps of the packets containing only the interesting traffic. Here by the term interesting traffic we mean packets with HTTP envelope. Thus, this application is based on HTTP Analyser Foundation, its GUI is slightly modified (see Figure 5-5 for illustration), but generally the new logic implemented is compulsory filtering out of the data only packets.

⁹ Webaward's Official Website: <http://www.webaward.org>

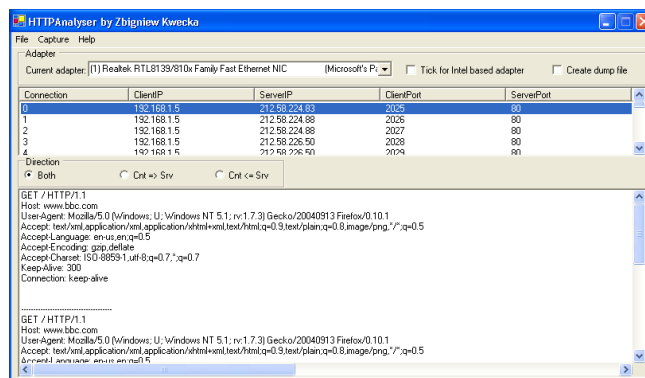


Figure 5-5 HTTP Dumper GUI

5.4.2 OffLine HTTP Analyser

This software was implemented in order to handle tcpdump files and produce output in a text format (this will be analysed in next section of this report). Once again the HTTP Analyser Foundation is employed. This time, however, the live capture function is disabled, and the only input possible is from tcpdumps produced with HTTP Dumper. Thus, the logic of the application was designed to process each and every packet read. Since, the output is written to text files, rather than a database, depending on the particular usage the format of the file will vary. Please see Figure 5-6 for illustration of OffLine HTTP Analyser GUI.

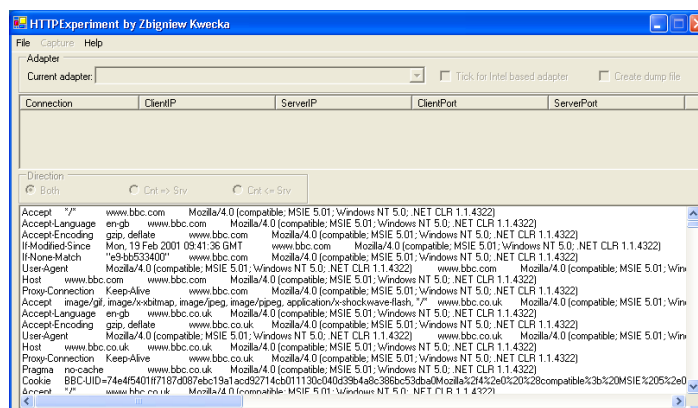


Figure 5-6 OffLine HTTP Analyser

5.4.3 Filtering Proxy

The Filtering Proxy uses the foundation provided by the HTTP Proxy. This enables it to perform functions of a standard HTTP forward Proxy server. For the purpose of the experiments is incorporates logic to filter out and append HTTP request headers. This logic is controlled by a simple GUI as illustrated in Figure 5-7, where the control information is passed to the asynchronous HTTPClient classes using the ConsoleBuffer object with a number of additional static variables.

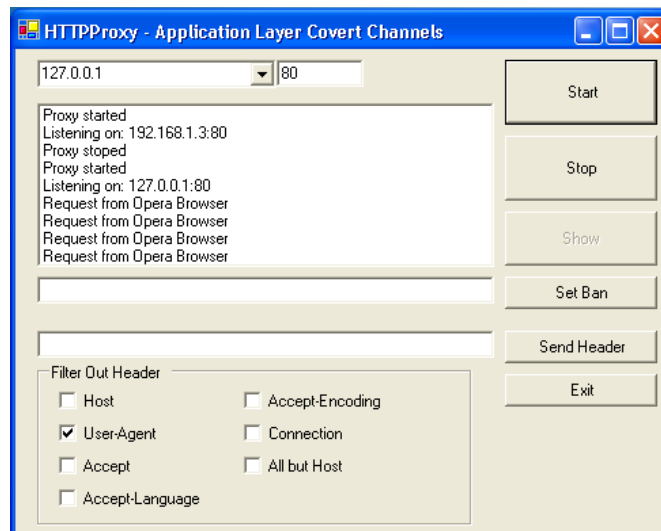


Figure 5-7 Filtering Proxy

5.4.4 Data Hiding Proxy

The Data Hiding Agent was designed so it may perform functions of writing and reading from the covert channel. Since we are experimenting with six different data hiding methods, basic implementation of all of them is present. Thus, in order to create traffic and write to covert channel a standard Web browser is used, with Proxy settings pointing to the local loop back address (127.0.0.1) of the host. On the local loop back address there is a modified implementation of HTTP Proxy, i.e. Data Hiding Proxy (Figure 5-8), preset with the destination IP address and in the ‘Sender’ mode.

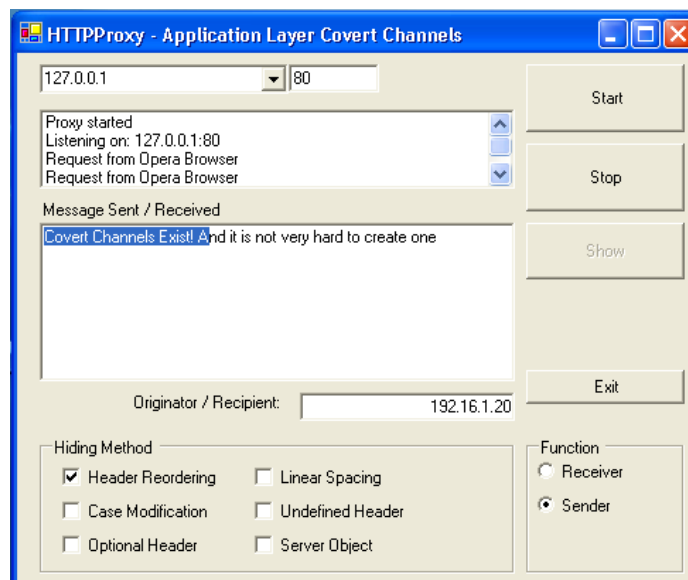


Figure 5-8 Data Hiding Proxy

Therefore, any request made from the browser will be modified with covert channel information and forwarded to the recipient. Then, the recipient, a Data Hiding Proxy set in ‘Receiver’ mode, will read the data from the covert channel (only when the request came from the preset originator), and forward the request to the target. Thus,

the response will follow the reverse path of the request and inbound covert channel may also be implemented. Figure 5-9 illustrates the data flow in this operation.

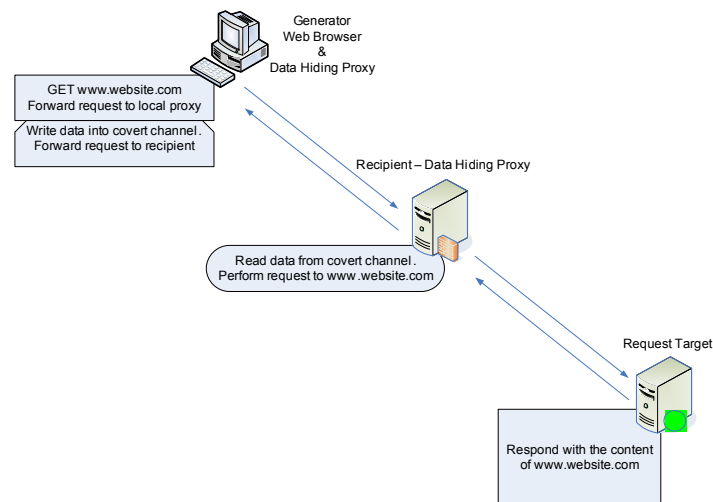


Figure 5-9 Data Hiding Scenario

5.4.5 Browser Caller

This application is based on Web browser Foundation. Its purpose is to generate HTTP traffic, depending on experiment, being of well defined conditions or random. Thus, it is able to start various Web browsers (Internet Explorer, Firefox, Opera, Netscape) and cause them to navigate to a website either sequentially or randomly chosen from the 'sites.txt' described earlier. There is 900 website addresses in this file, therefore, as not to overload the system, the application also needed to close browsers opened, before opening a new one. This proved to be troublesome, for Netscape and Opera browsers. While Internet Explore and Firefox disposed themselves gracefully after kill command, Netscape and Opera, have got quality agents build-in, and notified the user on the next start. This disabled downloading of the requested pages, without user interference and stopped our automated Browser Caller from operation. The problem have been solved with 'CloseMainWindow' signal being sent to the browsers, instead of 'kill' and a 'wait signal' delay awaiting browsers to close.



Figure 5-10 Browser Caller

Also, to accommodate for generating traffic through two different paths, as required by the experiments, which test server responses to modified and unmodified requests, the Browser Caller requests every page twice, modifying 'hosts' file¹⁰ with an address of either forward Proxy or filtering Proxy between the requests. Therefore, the browsers are configured to use Proxy server specified by a domain name ('www.filteringproxy.com') rather than IP address. Then, the line of the *hosts* file defining the 'www.filteringproxy.com' is modified with IP address of the Proxy to be used.

Code required to modify a host file in the operating system and cause a web browser to navigate to a given site follows:

```
try{
    StreamWriter hostFile = null;
    if(checkBox1.Checked == true)
    {

        if(File.Exists("C:\\WINDOWS\\system32\\drivers\\etc\\hosts"))
        {
            hostFile = new StreamWriter("C:\\WINDOWS\\system32\\drivers\\etc\\hosts", false);
        }
        else
        {
            hostFile = new StreamWriter("C:\\WINNT\\system32\\drivers\\etc\\hosts", false);
        }
    }
    if(current < read) //&& sites[current] != Environment.NewLine)
    {
        target = sites[current];
        if(secondExecution == false && checkBox1.Checked == true)
        {
            hostFile.WriteLine("127.0.0.1\\tlocalhost");
            hostFile.WriteLine("192.168.1.7\\twww.filteringproxy.com");
            hostFile.Close();
            secondExecution = true;
        }
        else if(secondExecution == true && checkBox1.Checked == true)
        {
            hostFile.WriteLine("127.0.0.1\\tlocalhost");
            hostFile.WriteLine("192.168.1.8\\twww.filteringproxy.com");
            hostFile.Close();
            current++;
            secondExecution = false;
        }
        else
        {
            current++;
        }
    }
    ProcessStartInfo startInfo;
    if(rbIExplorer.Checked == true)
    {
        System.Diagnostics.Process[] p = System.Diagnostics.Process.GetProcesses();
        for(int i=0 ;i<p.Length;i++)
        {
            if (p[i].ProcessName.ToLower()=="iexplore")
            {
                p[i].CloseMainWindow();
                p[i].WaitForExit(60000);
                if(!p[i].HasExited)
                p[i].Kill();
            }
        }
    }
}
```

¹⁰ %SystemRoot%\system32\drivers\etc\hosts

5.4.6 Browser Timer

The design phase has identified that this piece of software will need to request various websites and to check the time their full download takes in the environment when Inline Filtering Agent is used to interface the requests and when the requests are direct. Browser Timer (Figure 5-11) uses the Web browser Foundation as a base for sourcing the website addresses and performing timed operation. However, in this application we needed to know when the full download finished. Here, .NET predefined components came in handy. The 'AxSHDocVw.AxWebBrowser' is a component providing a framework for creating WWW browsers. Thus, we were able, in short time, to modify the Web browser Foundation to have full capabilities of WWW browser. Therefore, the Browser Timer is a time driven WWW browser, which reads addresses from the predefined text file ('sites.txt') at performs downloads. The average time taken to download a set of pages is taken at the end of the process. Since AxSHDocVw.AxWebBrowser' component, inherits Proxy settings from Internet Explorer settings, the only way to modify them is through windows registry or Internet Explorer GUI.

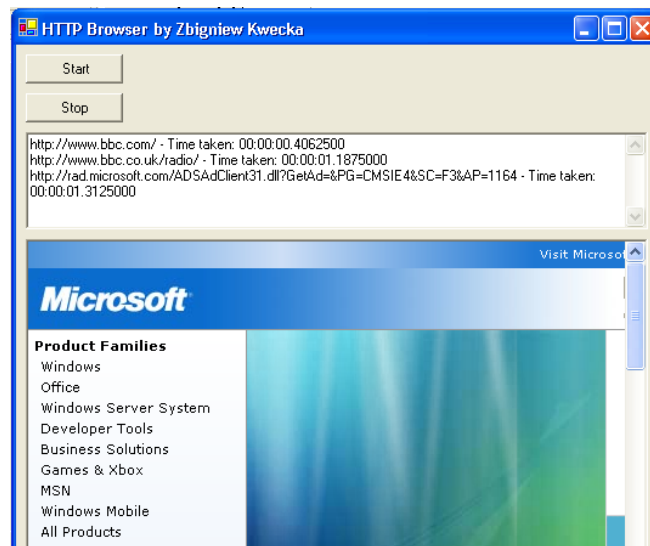


Figure 5-11 Browser Timer

5.5 Covert Channel Detection System Prototype

Inline Filtering Agent

The implementation of the Inline Filtering Agent, a forward Proxy server capable of detection of basic HTTP covert channel implementations has been developed based on Filtering Proxy. Previously we were planning to use HTTP Proxy Foundation, to prepare this part of the prototype, however, the implementation of the Filtering Proxy added some vital improvements to this framework. Thus, the GUI of this application is very similar to the one of Filtering Proxy. The Inline Filtering Agent has got basic signatures of four previously specified browsers (Internet Explorer, Firefox, Opera and Netscape) coded in. This way it is capable of recognizing the request produced by those applications. In case a request will come from a different application, or they will have any marks of being tempered with, the IFA will raise an alarm. In this version of the prototype, functionality of protocol-based detection will not be implemented, since it would require the application to be more versatile than an advanced HTTP

server. Here if the requests do not follow the major rules set by HTTP specification, they will be ignored. However, this is a vital part of the detection process, which could a large number of covert-channels' implementations with minimal processing required.

5.6 Conclusions

Due to the time restrictions of this project only the Inline Filtering Agent has been implemented out of the original design of the prototype. However, implementation of a filtering Proxy server was previously (Chapter 4) identified as the first and most important step in protection against threats to information confinement posed by covert channels. Additionally set of test software tools was successfully implemented, and the experiments could proceed to collect data for analyse of the validity of the proposed solution to problem.

6 Experiment Data Analysis

6.1 Introduction

The experiments performed in the due course of this project are unique, since the literature review (Chapter 3) did not identify similar experiments being conducted by the various teams working on information confinement problem. This chapter will discuss how the results were collected from the test network and identify the key findings.

6.2 Experiments

This section describes the results of the seven different experiments, which were conducted for the needs of this dissertation.

6.2.1 Experiment 1 – Implementation Specific Data Gathering

The aim of this experiment was to learn signatures of different HTTP clients. We have limited a number of clients to four most common Web browsers (Internet Explorer, Firefox, Opera and Netscape). In the experiment all four host machines from the implemented test network were used. Thus, each host has had following browsers installed:

- Internet Explorer version 6.0 SP 2
- Mozilla Firefox version 1.5.0.3
- Netscape version 8.1
- Opera version 8.53

No Proxy server was used during this experiment, since we were trying to establish standard behavior of the browsers. Thus, Browser Caller was used to generate traffic, and HTTP Dumper was used locally to save the traffic into tcpdump files. As the result we have established syntax employed by the browsers to produce HTTP requests. See Figure 6-1 for an illustration of percentage usage of various headers by different browsers.

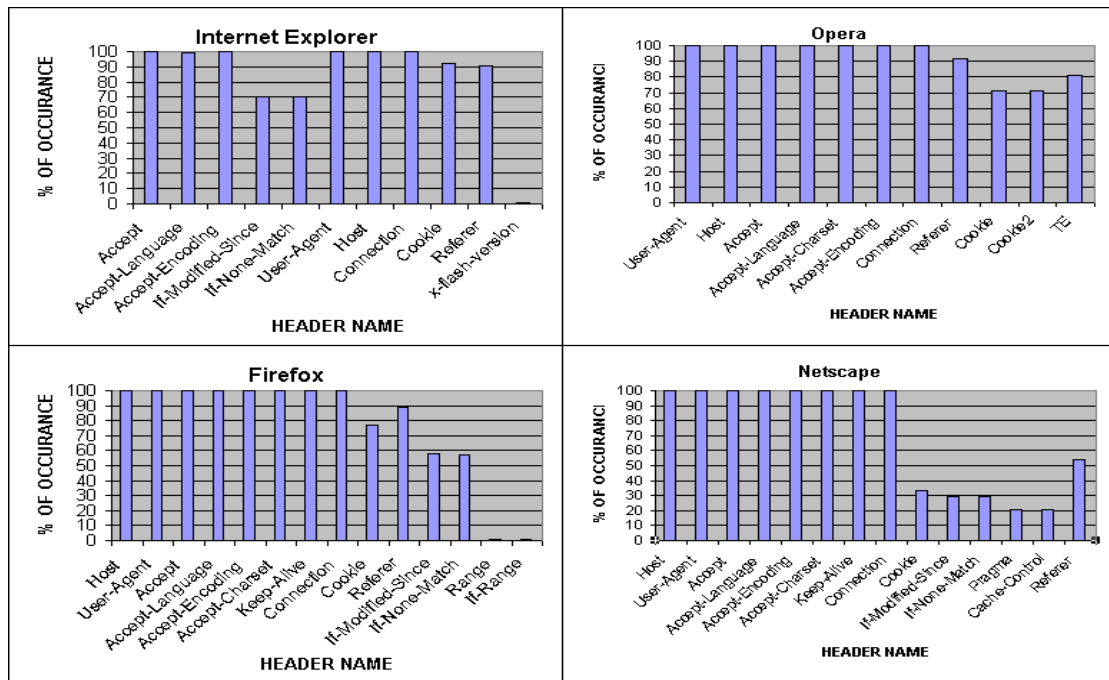


Figure 6-1 HTTP Headers Usage Statistics

The headers in the Figure 6-1 are in the order they would normally appear in a request made by a given browser. Thus, the first dissimilarity is that Explorer always uses 'Accept' as the first header, where Opera puts 'User-Agent' and both Firefox and Netscape use 'Host'. Furthermore, Firefox and Netscape use exactly the same order of the headers, where Internet Explorer and Opera differs greatly. So then in order to create signatures for each browser, we have analysed typical request produced (Figure 6-2). Once again Netscape and Firefox proved to be indistinguishable, however this time we had an answer for it. Although, in the Netscape's marketing website, there is no notice about it being based on Mozilla (the engine behind the Firefox) as we could suspect, the 'About' box provides greyed-out information it is actually based on Firefox. Thus, there is no wonder the signature of those two browsers is the same.

```

Internet Explorer:
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
Host: www.bbc.com
Connection: Keep-Alive

Firefox and Netscape:
Host: www.bbc.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;
Accept: text/xml,application/xml,application/xhtml+xml,
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive

Opera:
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
Accept: text/html,application/xml;q=0.9, image/gif, image
Accept-Language: en
Accept-Encoding: deflate, gzip, x-gzip, identity, *,q=0
Connection: Keep-Alive

```

Figure 6-2 Browsers' Signatures

Thus, to distinguish those three types of Web browsers (since Netscape is actually an implementation of Firefox), we can use three key factors:

- first header in the request
- linear spacing around comma separators
- casing used in values of case-insensitive headers

6.2.2 Experiment 2 – Request Information Filtering

From the data collected in Experiment 1 we have identified HTTP headers used by typical web browsers. The objective of this experiment was to identify, which of those headers are not used anymore. Thus, we have set Browser Caller to use two different proxies, one forwarding the headers without modification, and one capable of filtering headers out of the requests. Both proxies were located on the same machine, Host 3 (see Figure 5-1), but listening on different network adapters/logical addresses. This way, we were able to compare the fault rates of the requests with filtered out headers, to a baseline produced in parallel. Thus, lowered the risk of faulty network connections or overloaded servers affecting the results.

We have found that filtering out ‘User-Agent’ header produces a large number of server side processing errors (code 500). Additionally, we have noticed that some pages (especially Microsoft build websites) look different in agents other than Internet Explorer and servers return different CSS sheets, when ‘User-Agent’ header value differs from the one provided by Microsoft Explorer. Thus, it is advisable for this header to be allowed to pass through Inline Filtering Agent, however the string contained in the value should always be checked against database of allowed client software. Another concern was raised when filtering out “Host” header. Although most of the servers responded with no errors to this request, 5% of the responses were of code 400 (Figure 6-3). This indicated “Bad Request” response from the server. This usually happened for smaller websites, where the server software must differentiate between different websites it hosts using “Host” header. Thus, we consider this header, as one which should be under surveillance, but must be allowed to pass through the filtering software. Apart of “User-Agent” and “Host” headers, we have tried filtering out “Accept”, “Accept-Encoding” and “Accept-Language” headers, however the results returned showed that, these headers seems to be used more sporadically. Since the response codes from the requests with these headers filtered out match the ones where forward Proxy was used (see Appendix 1 for full set of results from this experiment).

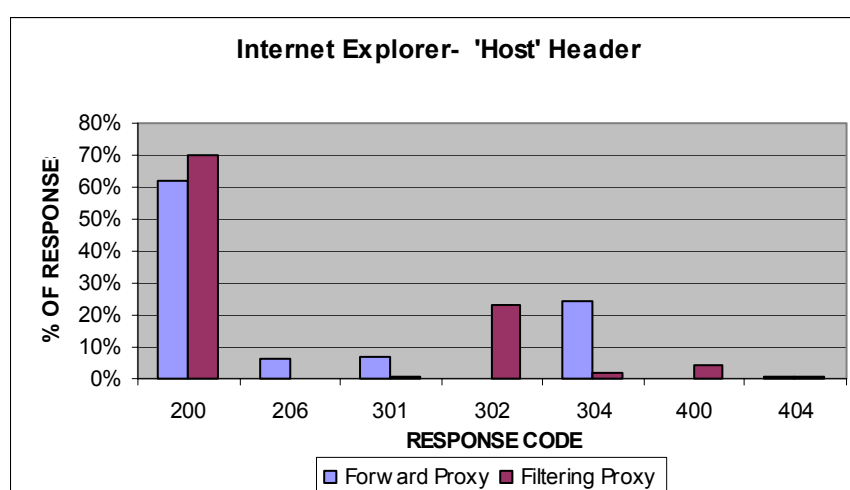


Figure 6-3 Responses to Requests with ‘Host’ Header Filtered Out

This experiment has not tested responses to requests with cache control headers filtered or modified. Since we consider that in highly secured environment cache

control should not be employed, due to a nature of information send by client's software in requests using cache control. This is due to the fact, that some of this information (sent in plain text) may give potential listener detailed data on software used by the inside host.

We think current implementations of HTTP/1.1 protocol send a number of redundant data in their messages. In this experiment only a number of request headers have been put to test, however, out of five headers, three have been identified as not being relevant anymore. Thus, we think an in depth study of data send be HTTP clients and server responses is needed, to produce a new specification to this protocol based around statistical data of current implementations.

6.2.3 Experiment 3 – Headers Modification

In previous sections of this report, we have identified six different techniques to hide data inside HTTP transaction messages. However, these methods were identified as theoretical and since Experiment 1 proved that there are many different implementations of HTTP specification, the objective of this experiment was to test the behaviour various servers, to suggested data hiding scenarios. The physical and logical setup for this experiment was very similar to that of Experiment 2. Thus, Host 2 was generating two requests for each website in 'sites' file. The requests followed two different paths, one with forward Proxy, where no modifications were performed, and one with Data Hiding Proxy on the way, so that the request could be modified accordingly to the requirements. Thus, we have tested five different data hiding scenarios:

(a) Case Modification

From HTTP specification, we know that all header names are case-insensitive. Thus, in this scenario, Data Hiding Proxy has been used to change header names' casing, from the usual title-case, to uppercase.

(b) Undefined Header

Client and server software, which conform to HTTP/1.1 standard must ignore unrecognised headers, i.e. threat the transaction, as they would if the header was not there. Therefore, to test this data hiding technique Data Hiding Proxy was configured to add an extra header ('Covert-Channel: A covert data') to every request passing through it.

(c) Linear Spacing Modification

This scenario employed the fact that HTTP software should interpret consequent linear spacing characters as a single white space. Thus combination of white spaces and linear tabulators was appended to every header in requests passing through Data Hiding Proxy.

(d) Optional Header

Optional header 'Via' with a value 'A covert data' was added to each request when testing this scenario.

(e) Headers' Reordering

In this test Data Hiding Proxy was used to change the order of two first headers in each request, since software conform to HTTP/1.1 should ignore the order of the headers of different name.

The only data hiding technique not tested in this experiment was the modification of server object. This is due, to the fact that appropriate scenario would need to employ standard unmodified HTTP requests, to access server object. Thus, since the technique doesn't involve modification of the request it cannot negatively affect HTTP server software and would produce results identical to the baseline.

The results from this experiment were used to analyse server responses to various data hiding techniques and their graphical representation is attached to the report in Appendix 2. Thus, out of five scenarios tested, the number of error response codes returned in the Data Hiding Agent path, were different only for Linear Spacing Modification scenario. The difference, however, was negligible (0.1%), so we confider badly written scripts rather that server software to be the reason behind them. Thus, the theoretical data hiding scenarios can be implemented in practice, without affecting HTTP operations and countermeasures should be developed to stop such implementations.

6.2.4 Experiment 4 – Browser Signature Recognition

Every web browser installed at test network hosts were configured with the IP address of the Inline Filtering Agent (running on Host 3) for the purpose of this experiment. Then we have tried browsing the Internet using various browsers (those previously specified) on different machines simultaneously. The IFA passed this test matching 100% of the requests to their originators. Thus, we have added an early version of Data Hiding Proxy inline with the request path, between the hosts and the Inline Filtering Agent. The results were very surprising, as the filtering agent recognized signature mismatch in every request. This was later identified as being caused by the default behaviour of the proxies based on HTTP Proxy Foundation, since it uses Dictionary Collection to store headers and rebuild requests, messages passing through Data Hiding Proxy had their headers ordered alphabetically. This behaviour was consequently modified so that Data Hiding Proxy produced exact copies of original requests, when hiding techniques are not employed (this new version was use to produce scenarios in Experiment 3). Then the test was repeated using the new version of hiding software and this time once again the filtering agent has reported 100% match of the requests' signatures to the originators specified in 'User-Agent' field. However, when the data hiding techniques were used the IFA did not report any mismatches.

In this experiment we have proven that recognition of different HTTP client software based on unique signature is possible. Thus, by analysis of the message syntax we are able to check the value supplied in the 'User-Agent' header by the originator. However the, data hiding techniques, were able to pass through filtering agent without raising alerts and caused concerns to the definition of the signatures. We think the fault was in the signatures being defined in a way do distinguish between different browsers, i.e. identified browser specific aspects, however did not evaluate any common factors in the requests. Thus, we think that in order to produce signatures, that could be employed to detect data hiding techniques usage, a full syntax of the message must be considered. For the set of browsers used following factors where identified that should produce more precise signatures:

- usage of title-casing to produce header names
- single space between colon at the end of header name and header value
- no linear spacing characters at the end of value field
- set of headers used to produce requests

These factors were introduced to signature checking process of the Inline Filtering Proxy version used in Experiment 5.

6.2.5 Experiment 5 – Covert Channel Detection

Only the Proxy part of the designed prototype has been implemented in the course of the project, thus the functionality of the system is limited to protocol and signature-based detection. In this experiment Inline Filtering Agent was located inside the test network and the hosts (previously specified Web browsers on the hosts) were configured to use it. Various requests were then generated, automatically as well as by human operators, most of them using browsers permitted to pass through the filtering Proxy. This was to produce background noise and try to simulate a load of the Filtering Agent. Later Data Hiding Proxy was used to imitate few data hiding scenarios. During the experiment we were able to detect 100% of the data hiding scenarios based on:

- HTTP header-name case modification
- linear white space injection at the beginning or the end of header-value
- modification of the headers' order
- addition of uncommon or undefined header

In all the cases the IFA has properly identified the originators of the threats, by providing remote-end socked information. This shown the advantage of using detection and filtering agents in-line with the requests. Since a Proxy must process all the traffic before forwarding it, there is no chance of overlooking any well defined signature.

During the experiment we have also detected few applications of unknown signatures, different than Data Hiding Proxy. Thus, with a help of HTTP Analyser Foundation the alerts were further investigated (Figure 6-4). Finally following automated agents of the test network, has been identified:

- Background Intelligent Transfer Service version 6.6 – Windows update agent
- MSN Messenger 7.0 – MSN configuration agent
- Gadu-Gadu Autoupdate – update agent of popular Polish communicator
- GG – advert download of the Internet communicator mentioned above
- Symantec Anti-virus Live Update agent

We were aware of the above software agents running in the test network, however, none of them were explicitly configured to use the Inline Filtering Agent. Later, it has been identified, that Internet Explorer Proxy settings are being inherited by those applications.

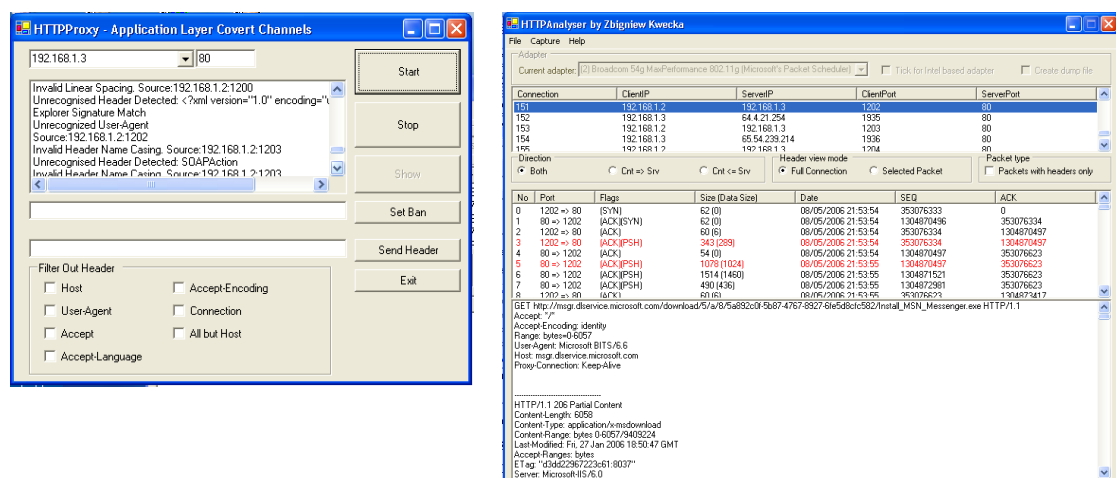


Figure 6-4 Threat Detection

We think that, considering security aspects, it is not an ideal solution, and Proxy settings set in one application should not propagate onto another. It can be considered as behaviour similar to password ‘hijacking’, and therefore should not be allowed by the operating system.

The Sniffer Detection Agent has not been implemented, thus, automated detection of more sophisticated covert channels was not possible. Therefore following methods of covert channels’ implementation did not result in raising alerts:

- modification of the server object
- changing field-values of headers different than ‘User-Agent’ or ‘Connection’
- providing an allowed header twice in the same request

The first two, would require machine with high processing power and large physical storage, capable of recording various characteristics of the traffic under observation. This could be achieved, to some degree, by the Sniffer Detection Agent designed in the project, however the third undetected scenario, falls into signature-based detection, which should be performed by the Inline Filtering Agent. Thus, there is a need to develop more precise signatures and better understanding of HTTP protocol in the filtering agent.

Another objective of this experiment was to check the filtering behaviour of the IFA, thus, during the tests HTTP Analyser Foundation was used to capture the data flow between the Proxy and the Internet. The results showed that IFA successfully obfuscated covert channels induced by header-name case changing, linear spacing (at the beginning and the end of the header-value) as well as removed unrecognised and rarely used headers. This was done with no extra cost, since the functionality of the Proxy was implemented in a way all requests are first processed and then rebuilt and it is actually faster to rebuild the request using standard message syntax, rather than syntax it arrived with.

6.2.6 Experiment 6 – Analysing Prototype’s Load on the Test Network

In this experiment we have tested how much the scanning process performed by the Inline Filtering Proxy affects the hosts on the test network. Thus we have collected statistical information on the time it takes Browser Timer testing software to download the content of various websites.

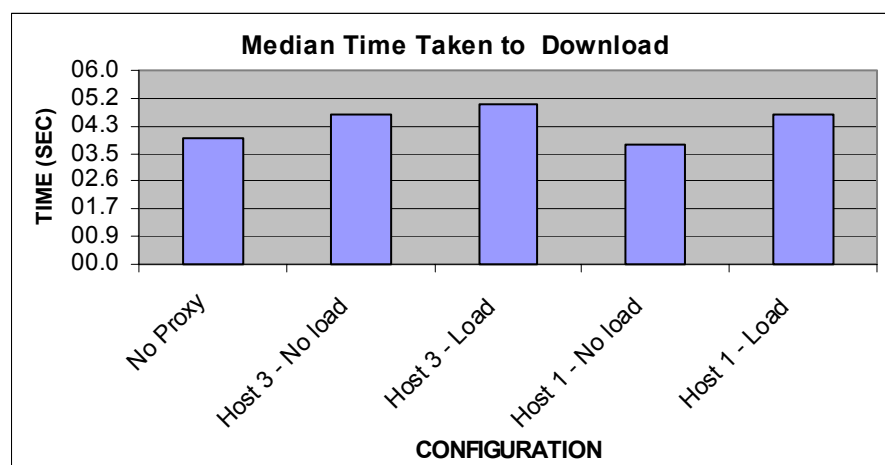


Figure 6-5 Median Time Taken to Perform a Full Download

This was performed using different Proxy configurations, with Host 2 set as originator. These configurations were:

- no Proxy
- Proxy hosted on low processing power machine (Host 3) without load
- Proxy hosted on low processing power machine (Host 3) with load
- Proxy hosted on high processing power machine (Host 1) without load
- Proxy hosted on high processing power machine (Host 1) with load

Figure 6-4 illustrates the results, i.e. median time between the request and response. We can see that the configuration where Inline Filtering Agent was running on the heavily loaded machine with the lowest processing power in the test network (Host 3) slowed down a typical operation by 1.7sec. Surprising result, however, is that of Inline Filtering Agent being executed on Host 1 (high power PC) with no load. Here a typical download operation took less time than in the scenario with no Proxy used. Since browser on Host 2 was not allowed to keep cache, this fact could only be explained, by slightly different network conditions. Thus, after short investigation we have identified that the data in 'no Proxy' configuration were collected around 21:00GMT, where the 'Host 1 – No load' setup was tested at 22:30GMT and by that time load of our ISP link to the Internet as well as load of target websites was slightly smaller. Still, the test error introduced is small and we consider that the delay added by the Inline Filtering Agent is negligible and would not affect the operation of a production network, especially considering the fact that 100% HTTP/1.1 compliant Proxy implementation would lower the number of internal and external TCP connections opened. Thus, by the means of persistent connections, such a Proxy would actually be able to speed up WWW transactions, with most popular (among the user of the intranet) websites.

6.2.7 Experiment 7 – Code Mobility Check

During this experiment we have tried to execute the prototype on every host in the test network, to check how mobile is the code produced. As described earlier each host, in the test network runs slightly different operating system. The tests shown that Inline Filtering Agent executes on all the operating systems used and it doesn't require any special libraries installed. At the same time, Sniffer Detection Agent was implemented based on WinPcap library, which was installed on each host prior to testing. Even so that the installation was successful on all four machines, during the experiment Sniffer Detection Agent would not run on the Starter Edition and Home Edition platforms. We have expected this, from Home Edition software which was not designed to perform any low level operations, however bearing in mind that Microsoft Windows Starter Edition is actually based on Windows Professional, the fact the software didn't operate on this platform was surprising. However, information found on Microsoft website confirmed that, Starter Edition restrictions are not only hardware restrictions (this platform will run only on low level computers with less than 256MB of RAM and less than 80GB of disk space) but also some elements of the Professional version were removed, to restrict platforms use in professional environment. However the core of the problem has been identified as 'npptools.dll' missing from both systems. Thus, after coping this library from Host 1 running Windows XP Professional onto Host 3 and 4 the problem has been fixed. Therefore, we consider our prototype as operational on all major releases of Windows operating systems, but the Sniffer Detection Agent prototype will require installation process to make sure WinPcap and 'npptools.dll' are present.

6.3 Conclusions

First set of experiments performed has proved that recognition of the connection originator is possible, even if the user agent field of the HTTP protocol is obfuscated. Therefore signatures of four commonly used browsers were identified for the use in the prototype. Then the set of information sent in a request for a certain web page was reduced, and from the response codes received the conclusions may be drawn that a percentage of headers in HTTP standard is sent in the request but never used by the receiving server in connection with typical requests. Thus, *Accept*, *Accept-Encoding* and *Accept-Language* have been identified as headers, which in English speaking environment, are redundant if using typical multifunction web browser.

The evaluation of the prototype has been performed and all the covert channel scenarios, that the IFA was designed to detect, has raised an alert when executed. Additionally five agents of various MS Windows based software were detected. Thus, it has been established that some application *hijack* proxy setting of the Internet Explorer.

7 Discussion, Conclusions and Further Work

7.1 Introduction

The main aim of this dissertation was an investigation of *covert channels* in Internet protocol stack. In the previous chapters the information collected shown that implementation of this data hiding technique is possible, and will most likely take place in the Application Layer of TCP/IP model. In addition a suitable prototype of the detection system was proposed and evaluated. Thus this chapter discusses the findings, provides conclusions and suggests further work that would need to be undertaken in this field, to create virtually covert channel free environment.

7.2 Discussion & Prototype Evaluation

The main aim of this dissertation was to investigate covert channel technologies in Internet protocol stack in the context of information confinement. Thus, Application Layer has been identified as the most likely level of data hiding in TCP/IP networking model. Previously there have been many successful approaches to building covert channels in lower layers of the TCP/IP model (Buchanan & Llamas, 2004), however currently their usage is limited and possible only in low security networks. The modern network access control systems (NACSs) are capable of replacing TCP/IP connection information of the traffic by the use of Proxies or suitably configured NAT (network address translation) servers, thus they can render useless any covert channel implementations, operating below Application Layer (Dyatlov & Castro, 2003). Therefore the technologies of data hiding in these lower layers may be interesting from the point of view of suspect surveillance, where a person under observation may use low security networking environment, such as internet cafe or SOHO¹¹ network. However, they may be perceived as ineffective when considering *information confinement problem* of large institutions, with secure networks.

There is a strong tendency, in the recent years, of the information hiding experts to turn their heads towards the relatively new subject of Application Layer Covert Channels. Most of the papers in this field agree that for the successful detection system to work, it should employ three different methods of detection, *signature*, *protocol* and *behaviour-based* (Borders & Prakash, 2004; Castro, 2003; Dyatlov & Castro, 2003). For this dissertation a system capable of performing this, was designed, however, due to the time restrains only the protocol and signature-based detection system was implemented and tested. The test results suggested that the system is capable of successful detection of pre-programmed threat signatures and covert channel implementations which do not comply with the HTTP protocol specification, however detection of unknown implementations or timing channels was impossible. Thus, although fast and precise (low level of false-positives) these two methods proved to have some limitations, and behaviour-based detection should be considered as a must, if the system is expected to detect new or more sophisticated threats. Thus, the findings of this dissertation agree with the results of other researchers of the field.

¹¹ Small-Office-Home-Office

Although current papers usually show the same opinion on the detection techniques required, they vary in terms of traffic being under surveillance. Some sources suggest that only the inbound traffic should be monitored, were others consider monitoring traffic in both directions as necessary (Dyatlov & Castro, 2003). However, there was only one current document found to advise observation of sole outbound traffic (Borders & Prakash, 2004). Thus, the approach used in this project was chosen by analysis of the *anatomy of misuse*, covert channels usage scenarios and the working environment for the prototype of the detection system. The conclusions suggested that in the environment where a stateful firewall is used to protect the intranet, any potential perpetrator, *insider* or an *outsider*, would require to establish an outgoing connection to either send covert data out of the secure perimeter or to perform a covert attack. In addition Internet traffic statistic show, the upload from intranets is usually considerably smaller than their download¹². Thus, the suggestion of well built system being capable of covert channel detection by monitoring only the small percentage of the total traffic (the outbound traffic) were considered to be very interesting. This also meant that the processing power of the system didn't need to be as high as in the solutions suggested by Dyatlov and Castro, thus limited the costs and increased the theoretical sensitivity of behaviour-based monitoring module. Thus, Borders and Prakash idea was confirmed by the foundations of misuse detection.

Various ways of installing a detection system in the targeted intranet environment were considered. Currently there aren't any tools on the market, which could work as a covert channel detection system, and only a couple of prototypes have been found. Dyatlov and Castro proposed a system, where one application (Snort) is collecting tcpdumps of the interesting traffic and the other is iterating through the offline data to find the covert channels. This method has an advantage of low cost on the network resources, since it only listens and does not affect the traffic, however, its major disadvantage is working on the offline data. In such a system reaction to ongoing threat would not be possible, or late, thus the efficiency would be low. On the other hand *Web Tap*, the prototype of the monitoring system designed by Borders and Prakash, has used inline scanner (a Proxy server), and the results they achieved using this method should be considered as one of the best in the current research in the field. However, they results were based on providing proxy services to a relatively small group of users (30) and the load on the network (the delay in request-reply chain versus level of processing conducted) was already a factor. Thus, the prototype described in this dissertation, which is targeted to provide detection services to intranets, was designed with a network load in the mind. It was identified that the protocol and signature-based detection, do not require high levels of processing, thus may be located inline with the traffic, but the more sophisticated behaviour-based monitoring consumes vast amounts of resources and should not run on any machine inline with the traffic. Therefore the prototype spited the functionality of the system into two agents. The results of the Experiment 6 has proven this to be the right choice, since the web operations using even heavily loaded *Inline Filtering Agent*, with *HTTP Analyser Foundation* (simulating the load of the *Sniffer Detection Agent*) listening on the span port of the intranet switch, were delayed only by a 1.7sec (median value). Thus, the successful large scale implementation of covert channel limiter software could be based around this method.

¹² which is proven by the popularity of asymmetric connections to ISPs

7.3 Test Inadequacies

The vast majority of the experiments performed for the needs of this dissertation, were performed using automated software developed especially for this project using C# programming language of the .NET framework. Thus, due to time restraints and the considerable amount of optional functions in HTTP specification, the test environment was not fully HTTP compliant. This in turn could lead to some test inadequacies.

In Experiment 1 the objective was to analyse the usage of HTTP headers in the request messages. The HTTP Dumper software was used to produce tcpdumps of the traffic to be analysed, packets containing HTTP protocol envelope. Thus, the software was capable of recognising and storing the packets where the request and the response information should start. However, the application did not check the packet content and therefore was unable to detect HTTP messages which span across multiple TCP packets. Thus, some sporadically used headers could be overlooked in the results of this experiment. However, later tests proved the chances of HTTP envelope, of automatically generated requests, being larger than the MTU (max transfer unit) of the test environment, as lower than 1/1000. Thus, since in this experiment the focus was on the most common headers, the ones overlooked would not affect the results.

The results from Experiments 2 and 3, which were used to analyse the amount of flexibility in the current HTTP implementations, were based on the response codes from the web servers. This has proven, that even with certain information inside request messages modified, in most cases the web servers will provide the services to the client. However, the differences between the levels of these services were not considered. It has been noted, that some pages, especially those running distributed services developed by Microsoft, provided *success* response codes (1xx, 2xx, 3xx) to the clients with *User-Agent* field obfuscated or removed, but sent only basic versions of layout files (such as *css*). Thus for the purpose of complete evaluation of the HTTP implementations currently used a number of human operators would need to perform the tests themselves or supervise the automated request system.

The automated generation of the requests using *Browser Timer* and *Browser Caller* applications developed for the needs of this dissertation, was required in order to collect a large base of the test data. Thus, the amount of data collected and experiments conducted could be accomplished without the use of this software. However, it limited the validity of the results to the *GET* HTTP requests, since only this request method was used in various tests performed. The tests were based around modification of the message syntax allowed by the generic message in HTTP specification (Fielding, et al, 1999), thus they could also be performed on other request methods, but the generation of the requests, would need to be performed by human operators, or set of messages generated and recorded in advance to the tests, could be replayed.

7.4 Conclusions

This dissertation looked at the problem of *covert channels* in communication systems, from a different than usual approach. Most of the documents in the field focus on threats incoming from the Internet, where the findings provided suggest the biggest threat of covert channels usage is that of *information confinement*. Thus, data leaving

the network should be perceived as that which can cause more damage. This suggestion was first published by Borders and Prakash in their document describing the operation of the *Web Tap*, covert channel detection software of their design. Thus, the findings of this dissertation are the second to consider the above approach to covert channel analysis.

The *Hyper-Text Transfer Protocol* (HTTP), one of the most popular Internet protocols currently in use, was identified as the most likely carrier for the covert payload. This conclusion was drawn after defining Application Layer as the level of uninterrupted covert channel operation, where channels are usually noiseless *end-to-end*. Thus, *Simple Mail Transfer Protocol* (SMTP) and *Domain Name System* (DNS) have also been considered, but due to the strict e-mail monitoring and logging, and the slow movement to more secure DNS services, after the recent DDoS attacks, HTTP, the protocol which provides open-ended Internet specification for raw data transfers, was chosen the most likely choice of perpetrators.

Since, operation of HTTP is usually transparent to the end user and the fact that this protocol, due to its usability an innocently sounding name suggesting text based informational services, there are no tools on the market which would allow for observation of this protocol. Thus, a set of test tools has been designed for the needs of this dissertation, and they operation falls into three groups:

- (a) traffic generators
- (b) protocol manipulation
- (c) link observation

The above tools have been implemented using C# programming language of .NET framework. This allowed for high code mobility and interoperability. The programming environment of Visual Studio, helped in rapid code development and the tools produced proved to be of vital operability during various experiments.

The first set of experiments conducted proved the theory that recognition of unique signatures of different HTTP client software implementation is possible, by the analysis of request messages syntax and that HTTP clients are providing a lot of information in their requests, which is often ignored or discarded on the receiving end.

This knowledge has been used to produce a prototype of covert channel detection system, once again using C# programming language. The prototype implemented protocol and signature-based monitoring techniques and therefore was capable of recognition of any software agent which was not allowed to transfer information using HTTP in the system. There were no false-positives generated, since precise signatures of the messages allowed to pass the proxy server were used. However the prototype sensitivity proved to be low for the covert channels implementations mimicking request of genuine HTTP clients. It is considered, that further precision of the signatures used, could bring the sensitivity level up, however it would most likely cause a larger consumption of the network resources by the system. Thus, it is suggested that the signature base detection, which is performed inline with the requests by the IFA (*Inline Filtering Agent*) should be limited to the general syntax of the requests, and more precise signature matching, and behaviour-based detection

should be conducted in parallel to the traffic flow (by the use of live network traffic capture techniques).

The final conclusion is that implementation of covert channels in Application Layer of the Internet protocol stack is possible, and may be performed without perpetrator's access to the kernel of the compromised machine. Thus, development of suitable prevention and detection system is required. Such a system would need to employ three different types of monitoring techniques:

- (a) protocol-based
- (b) signature-based
- (c) behaviour-based

The first two may be successfully used inline with the traffic for the purpose of basic covert channel detection. In addition the knowledge of the HTTP protocol may be employed to *obfuscate* most of the possible carriers of the covert channel. This functionality would need to be located inline with the traffic flow, but the processing required would be limited to standard operations of HTTP proxy server. Thus, outgoing requests would be first phrased and then rebuild using implementation specific semantic for all optional functions defined in HTTP specification. These precautions would limit in a great level the possibilities of information leakage from NACS protected in this way. However some covert channels implementation, which mimic operation of generic HTTP software or use timing techniques, would stay intact and undetected using these methods. Thus, behaviour-based detection, which works by analysis of traffic anomaly, needs to be implemented in the covert channel detection system. In addition, due to the level of processing required this function must be performed in parallel to the traffic.

7.5 Further Work

In this dissertation, most tests were performed using *GET* requests. This request method, even that most popular, should not be considered a sole method used by the HTTP clients. In addition the responses from the servers were considered on the basis of the response code, which did not guarantee that the pages downloaded were those provided by the server, when the request obfuscation was not activated. Thus, in order to supply more precise results human testers would be required to produce the requests and observe the responses, or more sophisticated automation technique.

The suggestion should be made that the results obtained in this dissertation also show, that elimination of the covert channels for institutional intranets is possible. In order to do so development of secure HTTP client software, which does not allow input different than that from a genuine user and authenticates each and every request made to NACS, using hash signature of the request encrypted together with a random number know only to the browser and NACS devices. This, solution would eliminated in 100% automated covert channel implementations, and will place only a limited amount of trust in the human operator.

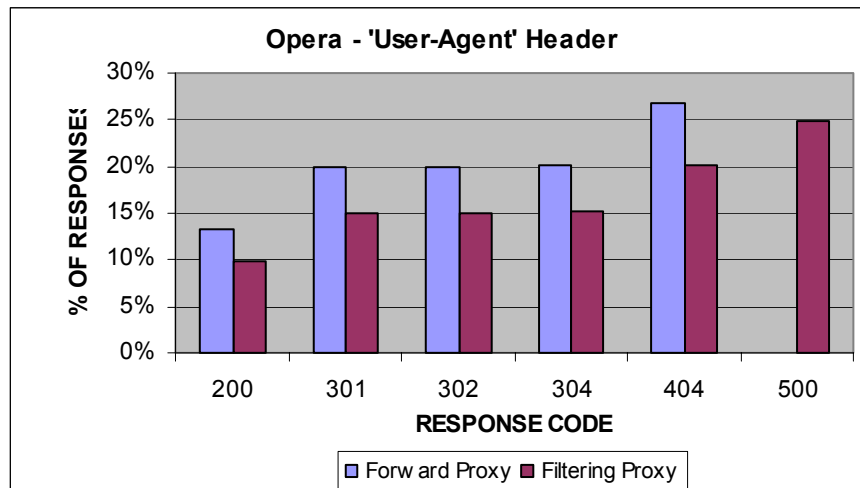
8 References

- Bauer, M. 2003. New covert channels in HTTP: adding unwitting Web browsers to anonymity sets. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*. New York: ACM Press.
- Borders, K., & Prakash, A. 2004. Web tap: detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*. New York: ACM Press.
- Buchanan, W. & Llamas, D. June 2004. Covert Channel Analysis and Detection with Reverse Proxy Servers using Microsoft Windows. *The 3rd European Conference on Information Warfare and Security*. University of London.
- Buchanan, W. 2006, Unit 9 – Data Hiding, URL: www.dcs.napier.ac.uk/~bill/asmn/unit04_forensic_computing.pdf [28 April 2006]
- Castro, S. November 2003. Covert Channel and Tunneling Over the HTTP Protocol Detection, *GW Implementation Theoretical Design*. URL: <http://www.gray-world.net/projects/papers/html/cctde.html> [21 December 2005]
- Coll, S. August 2005. Terrorists Turn to the Web as Base of Operations - Computer Crime Research Center. URL: http://www.crime-research.org/articles/Terrorists_Turn/ [20 November 2005]
- de Vivo, M., de Vivo, G. O., & Isern, G. 1998. ACM SIGOPS Operating Systems Review, *Internet security attacks at the basic levels*, **32/2**, 4-15. New York: ACM Press.
- Dyatlov, A., & Castro, S. June 2003. Exploitation of Data Streams Authorized by a Network Access Control System for Arbitrary Data Transfers, *Tunnelling and Covert Channels Over the HTTP Protocol*. URL: http://www.gray-world.net/projects/papers/covert_paper.txt [14 November 2005]
- Elz, R., & Bush, R. July 1997. RFC2181, *Clarifications to the DNS Specification*. URL: <http://www.ietf.org/rfc/rfc2181.txt> [20 November 2005]
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. June 1999. RFC2616, *Hypertext Transfer Protocol - HTTP/1.1*. URL: <http://www.w3.org/Protocols/rfc2616/rfc2616.html> [16 November 2005]
- Forte, D. V. November 2005. *SecSyslog: an Approach to Secure Logging Based on Covert Channels*. URL: <http://www.dti.unimi.it/~honeynet/honeynet.it/doc/SecSyslog.pdf> [21 December 2005]
- Gligor, V. D. November 1993, A Guide to understanding Covert Channel Analysis of Trusted Systems. *Technical Report NCSC-TG-030*. Meade, Maryland: National Computer Security Centre.
- Gligor, V. D. November 1993. A Guide to Understanding Covert Channel Analysis of Trusted Systems. URL: <http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-030.html> [12 November 2005]
- Internet World Stats, December 2005. INTERNET USAGE STATISTICS, *The Big Picture, World Internet Users and Population Stats*, URL: <http://www.internetworldstats.com/stats.htm> [28 February 2006]
- JEM (Joint Experts Meeting). September 2000. Report to the Federal Communications Commission on Surveillance of Packet-Mode Technologies URL: http://www.tiaonline.org/policy/filings/JEM_Rpt_Final_092900.pdf [20 November 2005]
- Kaminsky, D. December 2004. *Black Ops of DNS*. URL: http://www.doxpara.com/dns_bh [21 December 2005]

- Kawamoto, D. March 2006. DNS recursion leads to nastier DoS attacks. URL: <http://news.zdnet.co.uk/internet/security/0,39020375,39257938,00.htm> [14 April 2006]
- Klensin, J. April 2001. RFC 2821, *Simple Mail Transfer Protocol*. URL: <http://www.ietf.org/rfc/rfc2821.txt> [20 November 2005]
- Kwecka, Z. April 2006. Application Layer Covert Channels. *BCS Symposium on Intelligence in Security and Forensic Computing*. URL: <http://www.dcs.napier.ac.uk/~bill/bcs2006/zbig.pdf> [14 April 2006]
- Lampson, B. W. October 1973. A Note on the Confinement Problem, *Communications of the ACM*, **16**/10, 613-615. New York: ACM Press.
- Loepere, K. 1985. Resolving covert channels within a B2 class secure system. *ACM SIGOPS Operating Systems Review*. **19**/3, 9-28. New York: ACM Press.
- Loepere, K. 1989. The covert channel limiter revisited. *ACM SIGOPS Operating Systems Review*. **23**/2, 39-44. New York: ACM Press.
- Microsoft, January 2006, Microsoft Windows XP Starter Edition Facts Sheet, URL: <http://www.microsoft.com/presspass/newsroom/winxp/08-10WinXPStarterFS.mspx> [19 April 2006]
- Mockapetris, P. November 1987. RFC1034, *Domain Names - Concepts and Facilities*. URL: www.ietf.org/rfc/rfc1034.txt [20 November 2005]
- NCSC (National Computer Security Center), December 1985, *Department Of Defense Trusted Computer System Evaluation Criteria*. Meade, Maryland: National Computer Security Centre.
- Odom, W. 2001. *Cisco CCNA Exam #640-507 Certification Guide*. Indianapolis: Cisco Press
- PCMAG.COM. 12 November 2005. Definition: Covert Channel. URL: http://www.pcmag.com/encyclopedia_term/0,2542,t=covert+channel&i=40417,00.asp [12 November 2005]
- Postel, J. B. August 1982. RFC 821, *Simple Mail Transfer Protocol*. URL: <http://www.ietf.org/rfc/rfc0821.txt> [20 November 2005]
- Rivest, R. L., Shamir, A., & Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. **21**/2, 120-126. New York: ACM Press.
- Rogers, R. 2004. Understanding Covert Channels of Communication. URL: <http://www.blackhat.com/presentations/bh-asia-04/bh-jp-04-rogers.ppt> [16 December 2005]
- Slater, D. 1987. A note on the Relationship Between Covert Channels and Application Verification, *SIGSAC Rev.* **5**/1 (Jan. 1987), 22. New York: ACM Press.
- Summers, R. C. November 1996, *Secure Computing: Threats and Safeguards*, Columbus, Ohio: McGraw-Hill Companies
- Tsai, C. R., Gligor, V. D., & Chandersekaen, C. S. June 1990, Formal Method for the Identification of Covert Storage Channels in Source Code, *IEEE Transactions on Software Engineering*, **16**/6, 569-580. Los Alamitos, CA: IEEE, Inc.
- von Ahn, L., Hopper, N., & Langford, J. 2005. Covert two-party computation. *Proceedings of the Thirty-Seventh Annual ACM Symposium on theory of Computing* (May 2005), 513-522. New York: ACM Press.
- Wikipedia. March 2005. Covert channel - Wikipedia, the free encyclopedia. URL: http://en.wikipedia.org/wiki/Covert_channel [12 November 2005]
- WinPcap Team, 2005, WinPcap 3.1 Documentation, URL: <http://www.winpcap.org/docs/docs31/html/main.html> [28 February 2006]

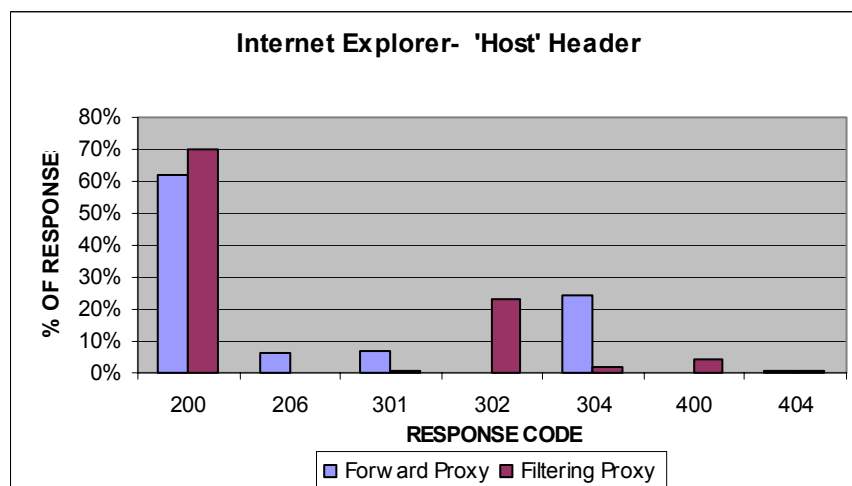
9 Appendices

Appendix 1 - Experiment 2 Results



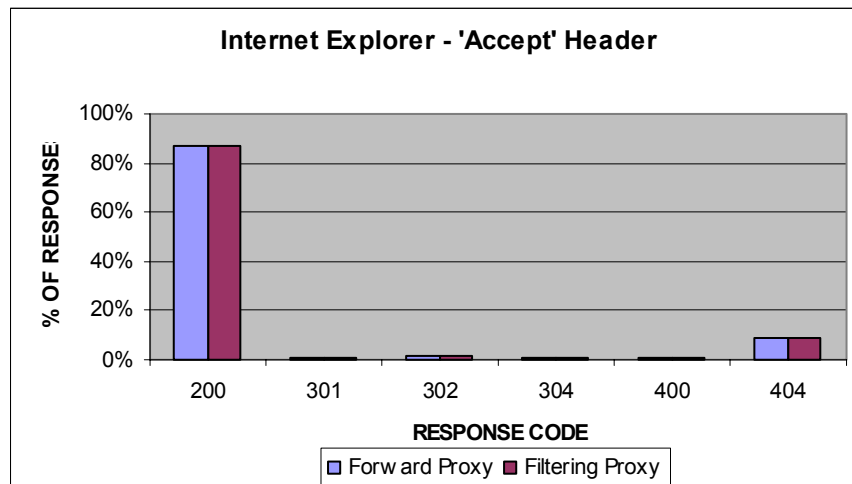
Responses to Requests with 'User-Agent' Header Filtered Out

A large number of responses with code '500' signify remote servers having problems in processing requests with 'User-Agent' header missing.



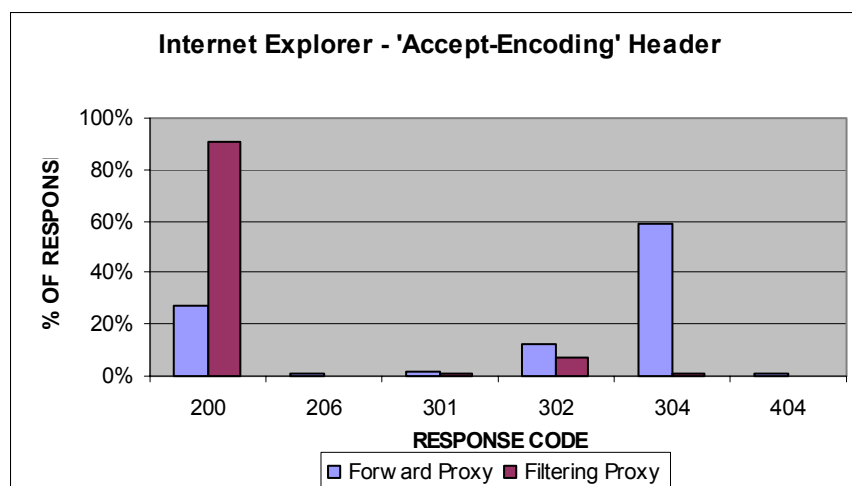
Responses to Requests with 'Host' Header Filtered Out

The graph shows that most of the servers treat requests with 'Host' headers missing as valid, however a significant percent (5%) replies with code '400', Bad Request. Further investigation found usually less significant websites responded with this error code.



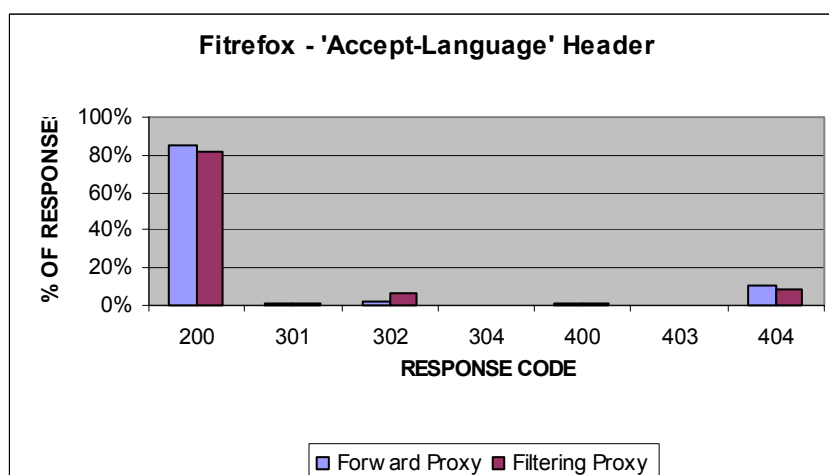
Responses to Requests with 'Accept' Header Filtered Out

Responses to modified requests are very similar, to those from unmodified requests.
This suggests that 'Accept' header is virtually unused.



Responses to Requests with 'Accept-Encoding' Header Filtered Out

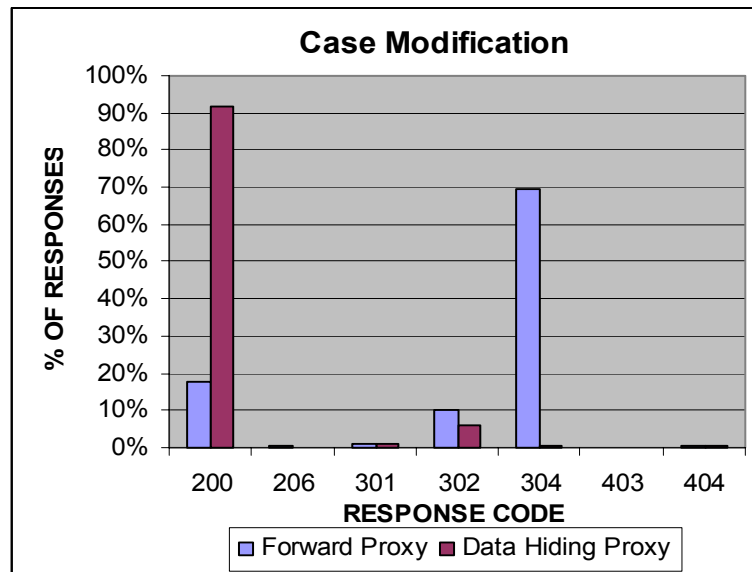
Numbers of error responses ('404') are similar for both proxies.
This suggests that 'Accept-Encoding' header is virtually unused.



Responses to Requests with 'Accept-Language' Header Filtered Out

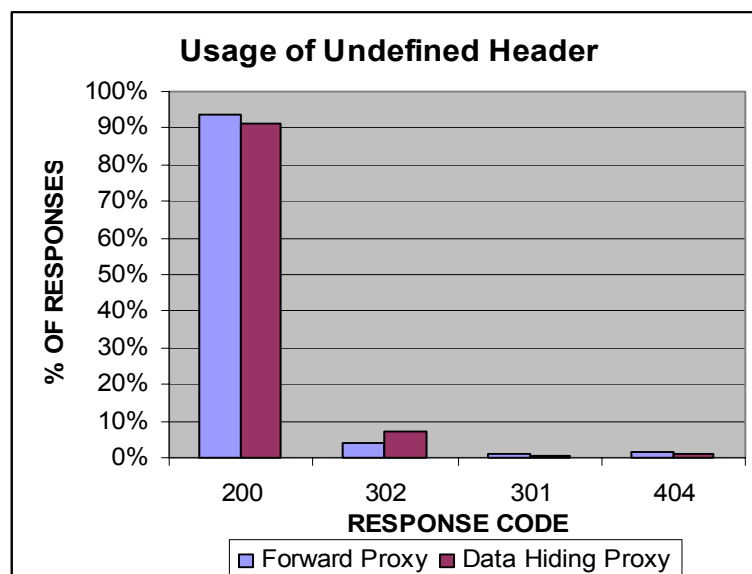
Numbers of error responses ('404') are similar for both proxies.
This suggests that 'Accept-Language' header is virtually unused.

Appendix 2 – Experiment 3 Results



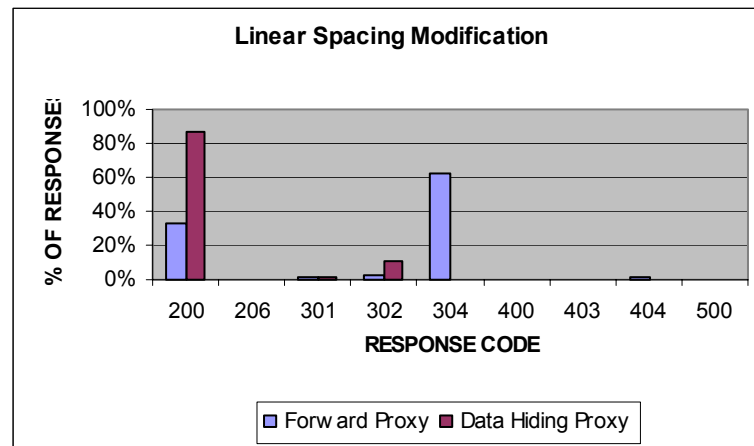
Responses to Requests with Header Names Capitalised

Numbers of error responses ('403', '404') are similar for both forward Proxy and Data Hiding Agent. This suggests that case modification may be employed to produce covert channels in HTTP.



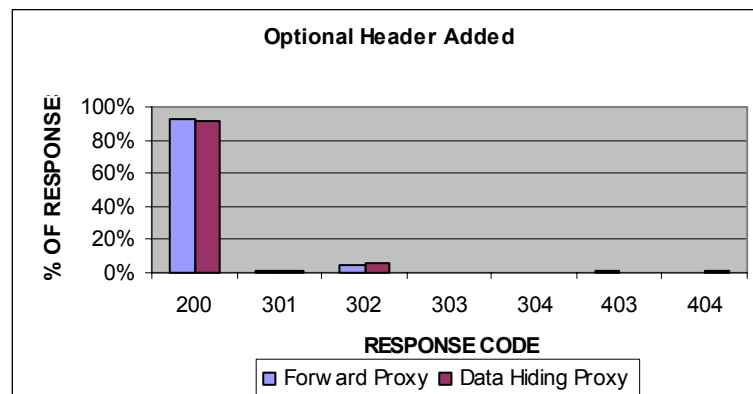
Responses to Requests with Custom (Undefined in HTTP) Header

Numbers of error responses ('404') are similar for both forward Proxy and Data Hiding Agent. This suggests that case additional headers may be added to HTTP transactions, without affecting them, thus also to produce covert channels in this application layer protocol.



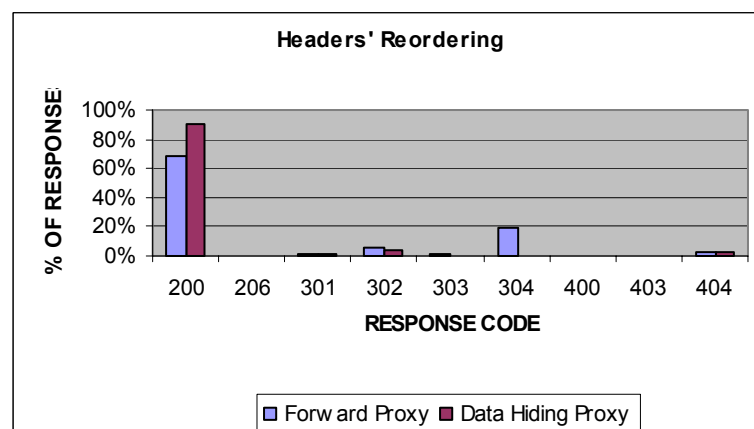
Responses to Requests with Linear Spacing Modified

Numbers of '404' error responses are similar for both forward Proxy and Data Hiding Agent. A small percentage of modified requests, however, resulted in '400', '403' and '500' type responses. Thus, although this data hiding method is allowed by RFC 2616 some server software implementations do not process them in required manner. However, covert channel implementation employing this vulnerability of RFC 2616 is still possible, since the number of error responses is negligible.



Responses to Requests with Optional Header ('Via') Added

Numbers of error responses ('403', '404') are negligible and similar for both forward Proxy and Data Hiding Agent. This suggests that HTTP servers' implementations comply with RFC 2616 specification, by ignoring any unrecognised headers.



Responses to Requests with Modified Headers' Order

Numbers of '403' and '404' error responses are similar for both forward Proxy and Data Hiding Agent. Thus, with only negligible number of code '400' responses (less than 1%), we assume header reordering may be used to produce covert channels implementations.

Appendix 3 - HTTP Protocol

Definition, purpose and usage

The application layer protocol called HTTP is often perceived as very basic protocol for distribution of World Wide Web pages. We could say that even its name Hypertext Transfer Protocol is very suggestive and implies that the purpose of this protocol is to transfer hypertext, where hypertext is defined as textual data “linked” across many documents or locations. It makes no wonder then, that some network administrators do not consider HTTP as a threat or think that as long as only outgoing established connections are permitted and every machine in the network uses some kind of firewall and antivirus software, they network is secure. However the true face of the protocol is different. The most recent specification of HTTP is RFC 2616 and the purpose of the protocol is described as follows:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. (Fielding, et al, 1999, pp. 7)

HTTP is now well established protocol and the current version is 1.1, however the idea of the protocol stayed the same. Through employing a simple human readable (MIME-like) syntax and allowing transfer of virtually any kind of data, HTTP become a preferred protocol in development of “on-line” applications. Furthermore the fact that a large group of network administrators allowed almost any outgoing connections of HTTP either directly or through proxies contributed strongly to this trend. Nowadays almost any software application, which requires communication over the Internet, employs HTTP or has a build in functionality allowing its application layer protocol to be tunnelled in HTTP. Example of the first kind could be antivirus software that uses HTTP for downloading signatures of the newest threats from the central server, or an update agent for an application like internet messenger. The implementations of the remote method invocation or remote procedure call are, thus, common examples of the second kind of the applications.

HTTP was identified as one of three protocols, which can be employed to create covert channels for sending data in and out of networks commonly considered to be secure. Thus the following section will identify, where RFC 2616 as the document which defines the current version of HTTP in use, gives hackers an open field for hiding data.

HTTP Syntax and Covert Channels

RFC 2616 was created to clear up some hard to understand statements from the previous documentations of HTTP (namely 1.0 and 0.9) and to introduce few optional features of HTTP/1.0 as standard in the new protocol HTTP/1.1. In the time when this specification was written the biggest concern of the creators was interoperability

between all the applications using the new standard and a backward compatibility to the already existing implementations, which conformed to previous RFCs. Some security concerns were raised, regarding leakage of personal information, attacks based on path names and DNS spoofing, however threats of covert channels' implementation was overlooked. Following interpretation of a RFC 2616, describes operation of HTTP and highlights areas where transfer of data in a covert manner is possible.

General syntax

The basic units of HTTP communication are messages, made up of a structured sequence of octets and transmitted via a transport layer virtual circuit (connection). There are two types of messages allowed: requests and responses. Both use generic message format. This consists of a start-line, zero or more header-fields (headers), an empty line and optional message body:

```
generic-message =  start-line           ; a Request-line or a Status-line
                   *(message-header CRLF) ; one or more header
                   CRLF                  ; compulsory empty line
                   [ message-body ]      ; optional application layer data
```

Start-line is made of either a Request-line in a request message or a Status-line in a response message. CRLF is the only end-of-line marker allowed for all HTTP protocol elements except the entity-body. It stands for carriage return (CR) and line feed (LF). This is a good practice, there is only one standard allowed, which makes protocol implementation easier, and prevents information hiding. There are few different types of message-headers: general-header, request-header, response-header and entity-header. All of them are built using the same syntax. Each header consists of a case-insensitive field name followed by a colon and an optional field value.

```
message-header = field-name ":" [ field-value ]
```

Case-insensitivity.

Field-names are case-insensitive, thus they are ideal carrier for hiding bits (payload). Both clients and the servers will interpret a field-name in the same way no matter the case of the letters. The coincidence is that capital letters in the ASCII code differ from the lower case letters only by a value of the 3rd bit. Thus binary form of letter "R" is 01010010 and the one of letter "r" is 01110010. Then a mask of 0xDF can be employed to extract or encode a payload in ASCII characters. Lower case letters would decode as 1's, where capital letters would decode as 0's. An example follows to illustrate how covert payload maybe hidden in a typical HTTP header given below:

```
Connection: keep-alive
```

This header can be modified to carry the bit pattern of 0111001011 and would look as follows:

```
ConnECtIon: keep-alive
```

Consequently above encoding method can be employed to transfer as many bits of payload as the total number of letters in field-names in any particular message. A visual inspection of the HTTP envelope would reveal this covert channel, however as it is unusual for anybody to examine HTTP message header and HTTP clients and

servers would ignore casing, this kind of covert channel could be successfully deployed.

Linear white spacing.

Another reason for concern is the fact that according to RFC 2616 the field-content (the data part of field-value) can be preceded and followed by an optional linear white spacing (LWS). LWS can be made up from a non-compulsory CRLF and one or more space (SP) or horizontal tab (HT) character.

$LWS = [CRLF] \ 1^* (SP|HT)$

Header values may be folded onto multiple lines using CRLF as long as the new line starts with a space or horizontal tab. Thus, all linear white space in a header may be replaced with a single SP before processing or forwarding the message downstream. This allows for a text decoration in the HTTP messages and has no real meaning in processing of the requests and responses. However it creates room for bidirectional covert channels. In a case where there is no HTTP Proxy between communicating sites, or linear white spaces are left unaltered by a Proxy, it is possible to encode information using SP and HT characters. An example illustrates how linear white space characters may be employed to transfer covert payload in the following header:

`"Connection: keep-alive"`

Header name "Connection" followed by a colon ":", a space SP and the value "keep-alive":

`"Connection" ":" SP "keep-alive"`

Let assume that 1's are encoded as HTs and 0's as SPs. Now if the single SP would be replaced with a combination of HTs and SPs, the meaning of the HTTP header would not change, but a binary stream could be hidden in the header. Thus a byte of information which in binary form is 01011100 could be encoded in one of the following ways:

`"Connection" ":" SP HT SP HT HT HT SP SP "keep-alive"`

`"Connection" ":" SP "keep-alive" SP HT SP HT HT HT SP SP`

`"Connection" ":" SP "keep-alive" CRLF
SP HT SP HT HT HT SP SP`

There is virtually no limit to a number of bits per message that can be sent using this method, apart of the size limits set for different kinds of requests and responses. If bits encoded in this way are placed in front of a header value a visual examination of the HTTP message would be enough to reveal the disguise, however if they follow the field-value contents or a CRLF only examination of the message bytes would expose the covert channel.

Order of headers.

Above technique is not the last reason why HTTP messages are such a good carrier for covert channels. The generic-message syntax does not specify the order in which the headers should occur in the messages. Although it suggests that it is "good practice" to include general-header fields first, followed by request or response

specific headers and incorporate entity-header fields as last ones, the order in which header-fields (of differing names) are received is insignificant. Thus if both sides wishing to use covert channel agree that specific order of header-fields is significant they would be capable of transmitting 1bit per two headers of any message. This in turn could be hard to detect, since in the previous scenarios the RFC allowed for creation of the hidden channel, but most of the current HTTP applications used standard semantics (i.e. only one SP before a field-value, ended by a CRLF and all the field-names using title casing) it was possible to spot a potential covert communication quite easily. However here the headers' order varies from implementation to implementation. Thus, for example in a basic covert channel groups of two consequent headers ordered alphabetically could stand for 1's and reverse ordered pairs could decode as 0's. Example:

```
Connection: keep-alive      ;would decode as 1
Host: www.napier.ac.uk

Host: www.napier.ac.uk     ;would decode as 0
Connection: keep-alive
```

Uniform Resource Identifiers.

All HTTP request and some response (i.e. for relocation purposes) messages consist of uniform resource identifiers (URIs), used to identify a resource on the network. There are two different forms allowed, absolute and relative. Thus a presence of one or the other can be an arbitrary 0 or 1, and if absolute URI is used it should follow syntax:

```
Absolute URI = "http:" "://" host [ ":" port ] [ absolute_path [ "?" query ] ]
```

Where 'http:' is the scheme name, 'host' is a DNS name of a node hosting the resource, optionally followed by a port number and/or absolute path to the resource. RFC 2616 suggests that clients and servers should:

- interpret an empty or not given port as a default port 80
- treat host name and scheme name in a case-insensitive manner
- interpret an empty absolute path as a path of "/" (document root)
- most characters can be represented in their "'%" HEX HEX" ("%" + hexadecimal value of the ASCII code) encoding

The first statement implies that "http://abc.com/", "http://abc.com/" and "http://abc.com:80" have the same meaning in HTTP. Therefore as previously shown optional form of data which is interpreted in the same way by client and server software, can be used to hide covert payload. For instance if a port number is present in a message this can decode as one and when it is omitted it could decode as zero. Allowing for case-insensitive parts of URIs creates similar possibilities as it did in field-names. Furthermore statements with empty absolute paths are treated in a same way as they would request document root ("/"), and again could be used to cipher data. Example:

```
http://abc.com      ;could decode as 0
http://abc.com/     ;could decode as 1, both mean the same to HTTP applications
```

Also any URI ASCII characters which can be sent in alternative formats, as a ASCII code, or a "'%" HEX HEX ", could be employed in creating a covert channel. Thus, the following URIs all point to the same resource:

```
http://abc.com/~smith  
http://abc.com/%7Esmith  
http://abc.com/%7esmith
```

Following a question mark (“?”) a query can be added to the URI. This is a common way to transmit data from HTML forms to the servers. In many cases additional information not required by the server is ignored and individuals can be tempted to use it as a cover channel (Dyatlov, et al. 2003). Although development of an automated system to uncover this type of activity can prove to be a complex task, the channel may be identified by simple visual examination of an address bar in a browser.

This is not the end of the optional arrangements of the RFC under examination. Another possibility to encode few bits per message is by alternative use of the three possible formats of the date allowed. Fortunately this time the creators of the specification stated that although on the receiving end all three formats should be treated as valid, implementations **MUST** generate only the RFC 1123 format. Thanks to this statement all HTTP/1.1 messages consisting of different date format than the one specified in the relevant document can be considered as invalid, while any packets with HTTP version different than 1.1 should be treated as highly suspicious anyway.

Request message

In HTTP communications request messages are sent from clients to servers in order to request a service. The client must specify the method (service required), identify resource and the protocol version it is willing to employ in the start-line of HTTP request message. As described earlier, the start-line is followed by message-header fields, compulsory empty line and optional message-body.

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

From the above syntax the fact that “linear white spacing” hiding technique cannot be employed in the Request-Line can be derived. Method and HTTP-version fields are case-sensitive, but the Request-URI follows requirements described in URI section of previous paragraph, so may be used as a covert channel. There are eight methods specified by RFC 2616, but the document allows for extending this list with additional custom methods as long as both sides can understand them. Following is a list of methods identified by the document:

- OPTION
- **GET**
- HEAD
- **POST**
- PUT
- DELETE
- TRACE
- CONNECT

In theory it is possible to cipher a payload by alternating request methods or by defining a new set of methods. For example any time a client sends GET request the recipient could treat it as 0 and any time a HEAD method is received it should be decoded as 1. When a new set of request methods is defined it is possible to transmit

much more data than one bit per message. Communicating parties could agree that any characters transmitted between name of the method (GET, POST or etc.) and the space character (SP) are part of the covert payload. The remote party would, however, need to remove those characters from HTTP stream before interpreting the message or transmitting it downwards.

```
Request-Line with covert channel =      Method
                                       [ ASCII encoded payload ]
                                       SP
                                       Request-URI
                                       SP
                                       HTTP-Version
                                       CRLF
```

Request-Line does not have a limit of size specified and any number of ASCII characters could be encoded into it in this manner. Still analyzing statistical data collected on request messages proves that this technique is possibly very easy to detect as the set of currently used methods is practically limited to GET and POST (they are the most common ones in use).

Another interesting fact about a start-line of a request message is the fact that host part of the URI specified in this line must be ignored if “Host” header is contained within this request. This is yet another redundancy in the RFC 2616 specification, which can cause security holes in a system allowing HTTP transfers. We can imagine following scenario. A client sends a message with URI in the request-line set to “http://COVERTCHANNEL/some_existing_document” and “Host” header with value set to “some_host”. Any Proxy or the receiving server would treat this request as valid, but a recipient of the covert payload could also easily extract ASCII encoded payload.

```
Request-message with a covert channel =
GET SP ["http://" ASCII payload] Resource_locator SP HTTP-Version CRLF
Host: SP remote_host_DNS_name CRLF
CRLF
```

Yet again the amount of the payload here would be virtually unlimited as the request-line has no size constraints.

Response message

According to the specification HTTP response-message should consist of a status-line, followed by message header, compulsory empty line and optional message body. A status-line is made up from HTTP version, numeric status code of the response and its associated textual phrase, where each element separated by SP characters.

```
Status-line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Since every element of the HTTP code have been designed in a way that both machines and people can understand its syntax without greater difficulty, the status-line consists of status code intended for machine interpretation and a human readable Reason-Phrase. The status codes are defined as 3-digit integers and there are five general groups of status codes, otherwise called classes of response, categorized by the first digit of the code (Fielding, et al, 1999, pp. 40)

- 1xx: Informational – Request received, continuing process

- 2xx: Success – The action was successfully received, understood, and accepted
- 3xx: Redirection – Further action must be taken in order to complete the request
- 4xx: Client Error – The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error – The server failed to fulfill an apparently valid request

List of the values is extensible, there is around 40 defined and servers can introduce their own implementation specific codes. In this case client should treat any unrecognized value as being equivalent to x00status code class (i.e. not defined code 244 would be interpret as code 200). When server responses with an error code user will be notified and it is possible that a browser will display error code and response phrase, however if the operation is successful (i.e. 2xx code is returned) the content of the status-line will stay secret to the user. A person who wants to send a message in a covert style can use the second and the third digit of the response code to hide some information. Thus 10^2 values could be hidden in a single message from client to server.

Status_Code with payload = (status code class digit) (payload digit) (payload digit)

Furthermore reason phrases listed in HTTP specification are only recommendations and they may be replaced, consequently allowing for hiding unlimited size alphanumeric string in the status-line.

Status-line with payload = HTTP_Version SP
 Status-Code SP
 Payload-As-Alphanumeric-String CRLF

Appendix 3 – Project Presentation

This presentation was given during BCS SGAI Symposium on Intelligence in Security and Forensic Computing, hosted at Napier University, in April 2006.

Symposium on Intelligence in Security and Forensic Computing




Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Supervised by:
Prof. William Buchanan

Zbigniew Kwecka
Matric No. 03008457
BSc (HONS) Networked Computing



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

- HND Electronics – Telecommunication
- HND Electronics – Computer Technologies
- Senior Debug Technician (Electronics)
- Cisco Certified Networking Professional
- BSc(HONS) Networked Computing - 4th year




Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Internet allows heterogeneous systems from around the World to communicate.

Its protocol stack was designed to be as universal as possible.

This allowed for rapid Internet application development.

Trade off: SECURITY

Aim: To investigate Application Layer Data hiding, with special focus on the detection of covert communication.



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Covert Channel:
"Any communication channel that can be exploited ... to transfer information in a manner that violates the systems' security policy" (Rogier, 2004, pp. 3).

"Anything that can be changed by one and seen by another can be used to send data" (Gansberg).

Classification:

- Storage and Timing
- Noisy and Noiseless
- Aggregated and not-aggregated



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Application Layer:

- Data hiding in lower layers of TCP/IP has been under investigation for a number of years now
- Ways of securing the networks from low level access exist
- Data payload scanners implemented in the security software usually filter only the genuine communication channels
- Administrators perceive established outgoing connections as harmless



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Application Layer Envelope

TCP/IP

- Application
- Transport
- Internet
- Network

OSI

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Application Layer Covert Channels

Most vulnerable protocols:

HTTP:

- World Wide Web traffic
- Various auto-updates and feedback software
- Tunnels for other protocols (Firepass, HTTunnel, Corkscrew, etc)

DNS:

- Domain name translation
- DNS Spoofing
- Tunnels: NSTX, Ozyman, VoiceOverDNS
- Covert channels used by botnets and malware

SMTP:

- Mail exchange protocol
- Similar to both HTTP and DNS in operation
- Slower and subject to logging



Centre for Mobile Computing and Security

INVESTOR IN PEOPLE

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:


- Fast
- Relatively easy to investigate, implement and test
- Methods and techniques will apply to SMTP and DNS as well

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Request / Response Line



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS


How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Request Method



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Request Path



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Request Version



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Message Header



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS


How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Header Name



NAPIER UNIVERSITY
EDINBURGH

Centre for Mobile Computing and Security

Application Layer Covert Channels

HTTP:

- Fast
- Relatively easy to investigate, implement and test
- Very similar to SMTP and DNS

How does it work:

- Request-Response architecture
- Client initiates the connection

HTTP Envelope:

```
GET /home.html HTTP/1.1
Host: www.bbc.com
```

Header Value



NAPIER UNIVERSITY
EDINBURGH


Application Layer Covert Channels

Ways to implement covert channels:

- Reordering
- Case Changing
- Optional Headers/Values/Flags
- New Header
- Linear spacing characters
- Modifying server object

Detection:

- Protocol-based
- Signature-based
- Behaviour-based



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:


- Reordering, Example

1st Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```

2nd Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Connection: Keep-Alive
Host: www.bbc.com
```



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:


- Reordering, Example

1st Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```

2nd Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Connection: Keep-Alive
Host: www.bbc.com
```



INVESTOR IN PEOPLE


Application Layer Covert Channels

Ways to implement covert channels:

- Case Changing, Example

Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
ConnECTION: Keep-Alive
```



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:


- Case Changing, Example

Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
ConnECTION: Keep-Alive
```

Letter	C	o	n	n	E	C	t	l	a	n
Hex	0x43	0x6F	0x6E	0x6E	0x45	0x43	0x74	0x49		
Mask	0xDF	0xDF	0xDF	0xDF	0xDF	0xDF	0xDF	0xDF		
Result	0	1	1	1	0	0	1	0		

0x72 "R"



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:


- Optional Headers/Values/Flags, Example

1st Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```

2nd Request

```
GET / HTTP/1.1
Accept: text/xml, */*;q=0.5
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:

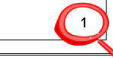
- Optional Headers/Values/Flags, Example

1st Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```

2nd Request

```
GET / HTTP/1.1
Accept: text/xml, */*;q=0.5
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
```



INVESTOR IN PEOPLE


Application Layer Covert Channels

Ways to implement covert channels:

- New Header, Example

Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
Covert-Channel: My Covert Channel
```



INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:
- New Header, Example

```

GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Host: www.bbc.com
Connection: Keep-Alive
Covert-Channel: My Covert Channel
  
```

If a server doesn't recognise a header it MUST be ignored.
Transparent Proxies must forward unknown headers.

INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:
- Linear Spacing Characters, Example

```

GET / HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.bbc.com
Connection: Keep-Alive
  
```

INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:
- Linear Spacing Characters, Example

```

GET[SP]/[SP]HTTP/1.1[CRLF]
Accept:[SP]*/*[HT][SP][SP][HT][SP][SP][SP][CRLF]
Accept-Language:[SP]en-gb[CRLF]
Accept-Encoding:[SP]gzip,[SP]deflate[CRLF]
User-Agent:[SP]Mozilla/4.0[CRLF]
Host:[SP]www.bbc.com[CRLF]
Connection:[SP]Keep-Alive[CRLF]
  
```

[SP] - SPACE - 0
[HT] - TAB - 1
[CRLF] - CR + LF

01001000 = "H"

INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:
- Modifying Server Object, Example

GMT 00:00	
GMT 00:30	GET /news.rss HTTP/1.1
GMT 01:00	GET /news.rss HTTP/1.1
GMT 01:30	
GMT 02:00	GET /news.rss HTTP/1.1
GMT 02:30	
GMT 03:00	GET /news.rss HTTP/1.1
GMT 03:30	

INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:
- Modifying Server Object, Example

GMT 00:00	
GMT 00:30	GET /news.rss HTTP/1.1
GMT 01:00	GET /news.rss HTTP/1.1
GMT 01:30	
GMT 02:00	GET /news.rss HTTP/1.1
GMT 02:30	
GMT 03:00	GET /news.rss HTTP/1.1
GMT 03:30	

SERVER LOG

INVESTOR IN PEOPLE

Application Layer Covert Channels

Ways to implement covert channels:

- Reordering
- Case Changing
- Optional Headers/Values/Flags
- New Header
- Linear spacing characters
- Modifying server object

Detection:

- Protocol-based
- Signature-based
- Behaviour-based

INVESTOR IN PEOPLE

Application Layer Covert Channels

Detection:

- Protocol-based
- Checks for compliance to the protocol's specification
- Fast = low cost
- Will flag few basic covert channel implementations

```

HTTP/1.1 200 OK
Date: Thu, 30 Mar 2006 19:46:22 GMT
Server: Apache/2.0.54 (Unix)
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT
Transfer-Encoding: chunked
Content-Length: 233
Accept-Ranges: bytes
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html
  
```

INVESTOR IN PEOPLE

Application Layer Covert Channels

Detection:

- Protocol-based
- Checks for compliance to the protocol's specification
- Fast = low cost
- Will flag few basic covert channel implementations

```

HTTP/1.1 200 OK
Date: Thu, 30 Mar 2006 19:46:22 GMT
Server: Apache/2.0.54 (Unix)
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT
Transfer-Encoding: chunked
Content-Length: 233
Accept-Ranges: bytes
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html
  
```

These two headers SHOULD NOT be send together


INVESTOR IN PEOPLE

Application Layer Covert Channels

Detection:
Signature-based

- Compares messages to signatures of known covert channels
- Checks against database of protocol's known implementations
- Medium speed = moderate cost
- Flags most tunnelling tools and high-bandwidth covert channels

HTTP/1.1 200 OK
Date: Thu, 30 Mar 2006 19:46:22 GMT
Server: Apache/2.0.54 (Unix)
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT
ETag: "e9-bb533400"
Accept-Ranges: bytes
Content-Length: 233
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html



NAPIER UNIVERSITY
EDINBURGH


Application Layer Covert Channels

Detection:
Signature-based

- Compares messages to signatures of known covert channels
- Checks against database of protocol's known implementations
- Medium speed = moderate cost
- Flags most tunnelling tools and high-bandwidth covert channels

HTTP/1.1 200 OK
Date: Thu, 30 Mar 2006 19:46:22 GMT
Server: Apache/2.0.54 (Unix)
Last-Modified: Mon, 19 Feb 2001 09:41:36 GMT
ETag: "e9-bb533400"
Accept-Ranges: bytes
Content-Length: 233
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html

Apache always uses "title case" to generate message headers



NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Signatures:

Browsers:

Opera:

Internet Explorer:


Firefox:

Netscape:

Automated Software:

Symantec Autoupdate:

Instant messaging application:



NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Internet Explorer:


Accept: /*/*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
Host: www.bbc.com
Connection: Keep-Alive

Firefox and Netscape:

Host: www.bbc.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; Accept: text/xml,application/xml,application/xhtml+xml, Accept-Language: en-us,en;q=0.5 Accept-Encoding: gzip, deflate Connection: keep-alive

Opera:

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT Accept: text/html, application/xml;q=0.9, image/gif, image Accept-Language: en Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0 Connection: Keep-Alive




NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Detection:
Behaviour-based

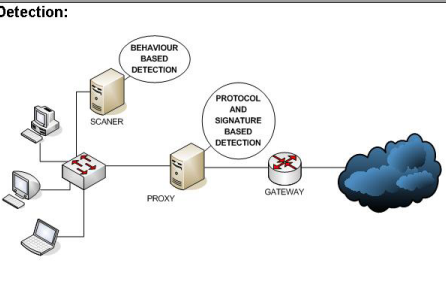

- Users profiling
- Applications profiling
- Traffic profiling
- Requires large number of resources = Expensive
- If used in line may slow down the traffic
- Very efficient, but still not 100% correct



NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Detection:

NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Our plans:


- Development of Secure Browser
- Research signatures of various HTTP implementations
- Recognise outdated parts of those implementations

Areas directly related to the research:

- Information leakage prevention
- Digital surveillance of criminal suspects

Areas where results and techniques described may find use:

- Stopping Distributed Denial of Services
- Anti-Spam Software
- Inline mail filtering for malicious signatures




NAPIER UNIVERSITY
EDINBURGH

Application Layer Covert Channels

Zbigniew Kwecka
Matric No. 03008457
BSc (HONS) Networked Computing

e-mail: 03008457@napier.ac.uk
z.kwecka@gmail.com



NAPIER UNIVERSITY
EDINBURGH

Appendix 4 - Inline Filtering Agent - Code Listing

Classes used by IFA:

- **Form1**
- **ConsoleBuffer**
- *Listener*
- *HttpListener*
- *Client*
- **HttpClinet**

The *Listener*, *HttpListener* and *Client* classes were taken from a Mentalis.org Proxy¹³ and were not modified in this project. Whilst *Form1* and *ConsoleBuffer* were developed specifically for the prototype, *HttpClient* class was rewritten from Mentalis.org Proxy, leaving only the basic request-response handling semantic from the original and introducing prototype's logic.

Form1 and *ConsoleBuffer* classes are specified within *Proxy.cs* file as follows:

```
/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmail.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
using System.Text;

namespace FilterProxy_GUI
{
    /// <summary>
    /// GUI
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button bRun;
        private System.Windows.Forms.ListBox lbConsole;
        private System.Windows.Forms.Button bExit;
        private System.Windows.Forms.Button bShow;
        private System.Windows.Forms.Button bStop;
        private System.Windows.Forms.Timer consoleTimer;
        private System.Windows.Forms.TextBox tbFilter;
        private System.Windows.Forms.Button bSetFilter;
        private System.Windows.Forms.TextBox tbListenerPort;
        private System.Windows.Forms.ComboBox cbListenerAddress;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.CheckBox checkBox2;
        private System.Windows.Forms.CheckBox checkBox3;
        private System.Windows.Forms.CheckBox checkBox4;
        private System.Windows.Forms.CheckBox checkBox5;
        private System.Windows.Forms.CheckBox checkBox6;
        private System.Windows.Forms.CheckBox checkBox7;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox tbHeader;
        private System.ComponentModel.IContainer components;
```

¹³ Autor: KPD-Team; Website: <http://www.mentalis.org/soft/projects/Proxy/>

```

public Form1()
{
    InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}

private Listener listener;
private int bufferRead = 0;

private void bRun_Click(object sender, System.EventArgs e)
{
    string classtype = "FilterProxy_GUI.Http.HttpListener";
    if (classtype == "")
        return;
    else if (Type.GetType(classtype) == null)
    {
        lbConsole.Items.Add("The specified listener class does not exist!");
        return;
    }
    string construct;
    if (cbListenerAddress.SelectedIndex > -1)
    {
        construct = "host:" + cbListenerAddress.SelectedItem.ToString() + ";int:" +
            tbListenerPort.Text.Trim();
    }
    else
    {
        construct = "host:127.0.0.1;int:80";
    }

    object listenObject = CreateListener(classtype, construct);
    if (listenObject == null)
    {
        lbConsole.Items.Add("Invalid construction string.");
        return;
    }

    try
    {
        listener = (Listener)listenObject;
    }
    catch
    {
        lbConsole.Items.Add("The specified object is not a valid Listener object.");
        return;
    }
    try
    {

```

```

        listener.Start();
        lbConsole.Items.Add("Proxy started");
        lbConsole.Items.Add("Listening on" +
            construct.Replace(";int:", ":").Replace("host:", ": "));
        consoleTimer.Enabled = true;
    }
    catch
    {
        Console.WriteLine("Error while staring the Listener.\r\n(Perhaps the specified
            port is already in use?);");
        return;
    }
}
/// <summary>
/// Creates a new Listener obejct from a given listener name and a given listener
    parameter string.
/// </summary>
/// <param name="type">The type of object to instantiate.</param>
/// <param name="cpars"></param>
/// <returns></returns>
public Listener CreateListener(string type, string cpars)
{
    try
    {
        string [] parts = cpars.Split(';');
        object [] pars = new object[parts.Length];
        string oval = null, otype = null;
        int ret;
        // Start instantiating the objects to give to the constructor
        for(int i = 0; i < parts.Length; i++)
        {
            ret = parts[i].IndexOf(':');
            if (ret >= 0)
            {
                otype = parts[i].Substring(0, ret);
                oval = parts[i].Substring(ret + 1);
            }
            else
            {
                otype = parts[i];
            }
            switch (otype.ToLower())
            {
                case "int":
                    pars[i] = int.Parse(oval);
                    break;
                case "host":
                    pars[i] = Dns.Resolve(oval).AddressList[0];
                    break;
                case "null":
                    pars[i] = null;
                    break;
                case "string":
                    pars[i] = oval;
                    break;
                case "ip":
                    pars[i] = IPAddress.Parse(oval);
                    break;
                default:
                    pars[i] = null;
                    break;
            }
        }
        return (Listener)Activator.CreateInstance(Type.GetType(type), pars);
    }
    catch
    {
        return null;
    }
}

private void bExit_Click(object sender, System.EventArgs e)
{
    if(listener != null)
        listener.Dispose();
    consoleTimer.Enabled = false;
}

```

```

    Application.Exit();
}

private void bStop_Click(object sender, System.EventArgs e)
{
    listener.Dispose();
    consoleTimer.Enabled = false;
    lbConsole.Items.Add("Proxy stopped");
}

private void consoleTimer_Tick(object sender, System.EventArgs e)
{
    int a = bufferRead;
    for(int i = a ; i<ConsoleBuffer.buffer.Count;i++ )
    {
        lbConsole.Items.Add(ConsoleBuffer.buffer[i]);
        bufferRead++;
    }
}

private void bSetFilter_Click(object sender, System.EventArgs e)
{
    ConsoleBuffer.filter = tbFilter.Text.Trim();
    ConsoleBuffer.buffer.Add("Filter: " + ConsoleBuffer.filter + " - ADDED");
}

private void Form1_Load(object sender, System.EventArgs e)
{
    ConsoleBuffer.accepted_headers.Add("Accept", "1");
    ConsoleBuffer.accepted_headers.Add("Accept-Encoding", "1");
    ConsoleBuffer.accepted_headers.Add("Accept-Language", "1");
    ConsoleBuffer.accepted_headers.Add("Accept-Charset", "1");
    ConsoleBuffer.accepted_headers.Add("Host", "1");
    ConsoleBuffer.accepted_headers.Add("User-Agent", "1");
    ConsoleBuffer.accepted_headers.Add("Keep-Alive", "1");
    ConsoleBuffer.accepted_headers.Add("Connection", "1");
    ConsoleBuffer.accepted_headers.Add("Proxy-Connection", "1");
    ConsoleBuffer.accepted_headers.Add("If-Modified-Since", "1");
    ConsoleBuffer.accepted_headers.Add("If-None-Match", "1");
    ConsoleBuffer.accepted_headers.Add("x-flash-version", "1");
    ConsoleBuffer.accepted_headers.Add("Cache-Control", "1");
    ConsoleBuffer.accepted_headers.Add("Unless-Modified-Since", "1");
    ConsoleBuffer.accepted_headers.Add("Range", "1");
    ConsoleBuffer.accepted_headers.Add("If-Range", "1");
    ConsoleBuffer.accepted_headers.Add("Pragma", "1");
    ConsoleBuffer.accepted_headers.Add("Content-Length", "1");
    ConsoleBuffer.accepted_headers.Add("Content-Type", "1");
    ConsoleBuffer.accepted_headers.Add("Cookie", "1");
    ConsoleBuffer.accepted_headers.Add("Referer", "1");
    IPEndPoint ipHost = Dns.GetHostByName("");
    IPAddress [] ipHostAddress = ipHost.AddressList;

    for (int i = 0; i < ipHostAddress.Length; i++)
    {
        cbListenerAddress.Items.Add(ipHostAddress[i].ToString ());
    }
    cbListenerAddress.Items.Add("127.0.0.1");
    if(cbListenerAddress.Items.Count > -1)
    {
        cbListenerAddress.SelectedIndex = 0;
    }
}

private void checkBox2_CheckedChanged(object sender, System.EventArgs e)
{
    if(checkBox2.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("User-Agent"))
    {
        ConsoleBuffer.filter_out.Add("User-Agent", "1");
    }
    else if(ConsoleBuffer.filter_out.ContainsKey("User-Agent"))
    {
        ConsoleBuffer.filter_out.Remove("User-Agent");
    }
}

```

```

private void checkBox3_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox3.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Accept"))
    {
        ConsoleBuffer.filter_out.Add("Accept", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("Accept"))
    {
        ConsoleBuffer.filter_out.Remove("Accept");
    }
}

private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox1.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Host"))
    {
        ConsoleBuffer.filter_out.Add("Host", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("Host"))
    {
        ConsoleBuffer.filter_out.Remove("Host");
    }
}

private void checkBox7_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox7.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("All"))
    {
        ConsoleBuffer.filter_out.Add("All", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("All"))
    {
        ConsoleBuffer.filter_out.Remove("All");
    }
}

private void checkBox4_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox4.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Accept-
        Language"))
    {
        ConsoleBuffer.filter_out.Add("Accept-Language", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("Accept-Language"))
    {
        ConsoleBuffer.filter_out.Remove("Accept-Language");
    }
}

private void checkBox5_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox5.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Accept-
        Encoding"))
    {
        ConsoleBuffer.filter_out.Add("Accept-Encoding", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("Accept-Encoding"))
    {
        ConsoleBuffer.filter_out.Remove("Accept-Encoding");
    }
}

private void checkBox6_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox6.Checked == true &&
        !ConsoleBuffer.filter_out.ContainsKey("Connection"))
    {
        ConsoleBuffer.filter_out.Add("Connection", "1");
    }
    else if (ConsoleBuffer.filter_out.ContainsKey("Connection"))
    {
        ConsoleBuffer.filter_out.Remove("Connection");
    }
}

private void button1_Click(object sender, System.EventArgs e)

```

```

    {
        ConsoleBuffer.addedHeader = tbHeader.Text.Trim();
        ConsoleBuffer.buffer.Add("Send: " + ConsoleBuffer.addedHeader+ " - ADDED");
    }

}

/// <summary>
/// An interfac between the GUI and asynchronous operations
/// </summary>
public class ConsoleBuffer
{
    public static ArrayList buffer = new ArrayList();
    public static NameValueCollection agents = new NameValueCollection();
    public static string filter = "";
    public static StringDictionary filter_out = new StringDictionary();
    public static string addedHeader = "";
    public static StringDictionary accepted_headers = new StringDictionary();

}
}

```

HttpClient class is specified within *HttpClient.cs*. This class was originally supplied by the Mentalis.org Proxy implementation, thus the methods, which were not altered for the purpose of IFA prototype implementation are omitted from the following listing of *HTTClient.cs* file:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

/*
ORIGINAL ORIGINAL ORIGINAL
Copyright © 2002, The KPD-Team
All rights reserved.
http://www.mentalis.org/

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Neither the name of the KPD-Team, nor the names of its contributors
  may be used to endorse or promote products derived from this
  software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

using System;
using System.Net;
using System.Text;
using System.Net.Sockets;
using System.Collections;
using System.Collections.Specialized;
using System.Threading;
using System.IO;

```



```

using FilterProxy_GUI;

namespace FilterProxy_GUI.Http {

///<summary>Relays HTTP data between a remote host and a local client.</summary>
///<remarks>This class supports both HTTP and HTTPS.</remarks>
public sealed class HttpClient : Client
{
    ///<summary>Initializes a new instance of the HttpClient class.</summary>
    ///<param name="ClientSocket">The <see cref="Socket">Socket</see> connection
        between this proxy server and the local client.</param>
    ///<param name="Destroyer">The callback method to be called when this Client object
        disconnects from the local client and the remote server.</param>
    public HttpClient(Socket ClientSocket, DestroyDelegate Destroyer) :
        base(ClientSocket, Destroyer) {}

    ///<summary>Gets or sets the HTTP version the client uses.</summary>
    ///<value>A string representing the requested HTTP version.</value>
    private string HttpVersion
    {
        get
        {
            return m_HttpVersion;
        }
        set
        {
            m_HttpVersion = value;
        }
    }

    ///<summary>Starts receiving data from the client connection.</summary>
    public override void StartHandshake()
    {
        try
        {
            ClientSocket.BeginReceive(Buffer, 0, Buffer.Length, SocketFlags.None, new
                AsyncCallback(this.OnReceiveQuery), ClientSocket);
        }
        catch
        {
            Dispose();
        }
    }

    ///<summary>Checks whether a specified string is a valid HTTP query
        string.</summary>
    ///<param name="Query">The query to check.</param>
    ///<returns>True if the specified string is a valid HTTP query, false
        otherwise.</returns>
    private bool IsValidQuery(string Query)
    {
        int index = Query.IndexOf("\r\n\r\n");
        if (index == -1)
            return false;
        HeaderFields = ParseQuery(Query);
        if (HttpRequestType.ToUpper().Equals("POST"))
        {
            try
            {
                int length = int.Parse((string)HeaderFields["Content-Length"]);
                return Query.Length >= index + 6 + length;
            }
            catch
            {
                SendBadRequest();
                return true;
            }
        }
        else
        {
            return true;
        }
    }

    ///<summary>Processes a specified query and connects to the requested HTTP web
        server.</summary>
    ///<param name="Query">A string containing the query to process.</param>

```

```

///<remarks>If there's an error while processing the HTTP request or when connecting
to the remote server, the Proxy sends a "400 - Bad Request" error to the
client.</remarks>
private void ProcessQuery(string Query)
{
    HeaderFields = ParseQuery(Query);
    HeaderFieldsSignature = ParseQuerySignature(Query);
    if (HeaderFields == null || !HeaderFields.ContainsKey("Host"))
    {
        SendBadRequest();
        return;
    }

    //implement filter
    string filterName;
    string filterValue;
    int Ret;
    if(ConsoleBuffer.filter != "")
    {
        Ret = ConsoleBuffer.filter.IndexOf(":");
        if (Ret > 0 && Ret < ConsoleBuffer.filter.Length - 1)
        {
            try
            {
                filterName = ConsoleBuffer.filter.Substring(0, Ret).ToLower();
                filterValue = ConsoleBuffer.filter.Substring(Ret + 1).ToLower().Trim();
                if((filterName == "requestpath" &&
                    RequestedPath.ToLower().IndexOf(filterValue)>=0) ||
                    (filterName == "requesttype" &&
                    HttpRequestType.ToLower().IndexOf(filterValue)>=0) ||
                    (filterName == "requestversion" &&
                    HttpVersion.ToLower().IndexOf(filterValue)>=0))
                {
                    ConsoleBuffer.buffer.Add(ConsoleBuffer.filter + " DETECTED");
                    SendBlockedRequest();
                    return;
                }
            }
            else if(HeaderFields.ContainsKey(filterName) &&
                HeaderFields[filterName].ToLower().IndexOf(filterValue)>=0)
            {
                ConsoleBuffer.buffer.Add(ConsoleBuffer.filter + " DETECTED");
                SendBlockedRequest();
                return;
            }
        }
        catch {}
    }

    //implement signature checking

    string signature = "";
    if(true)
    {
        Ret = ConsoleBuffer.filter.IndexOf(":");
        if (true)
        {
            try
            {
                //Opera
                if(HeaderFieldsSignature.ContainsKey("User-Agent") &&
                    Convert.ToInt16(HeaderFieldsSignature["User-Agent"])==0)
                {
                    if((HeaderFields.ContainsKey("Connection") &&
                        HeaderFields["Connection"].IndexOf("Keep-Alive")>=0)
                        || (HeaderFields.ContainsKey("Proxy-Connection") && HeaderFields["Proxy-
                        Connection"].IndexOf("Keep-Alive")>=0)
                        || HttpVersion == "HTTP/1.0")
                    {
                        signature = "opera";
                    }
                }
            }
        }
    }
}

```

```

//Firefox
else if(HeaderFieldsSignature.ContainsKey("Host") &&
    Convert.ToInt16(HeaderFieldsSignature["Host"])==0)
{
    if((HeaderFields.ContainsKey("Connection") &&
        HeaderFields["Connection"].IndexOf("keep-alive")>=0)
        || (HeaderFields.ContainsKey("Proxy-Connection") && HeaderFields["Proxy-
        Connection"].IndexOf("keep-alive")>=0))
    {
        signature = "firefox";
    }
}

//Explorer
else if(HeaderFieldsSignature.ContainsKey("Accept") &&
    Convert.ToInt16(HeaderFieldsSignature["Accept"])==0)
{
    if((HeaderFields.ContainsKey("Connection") &&
        HeaderFields["Connection"].IndexOf("Keep-Alive")>=0)
        || (HeaderFields.ContainsKey("Proxy-Connection") && HeaderFields["Proxy-
        Connection"].IndexOf("Keep-Alive")>=0))
    {
        signature = "explorer";
    }
}

if(HeaderFields.ContainsKey("User-Agent"))
{
    if(HeaderFields["User-Agent"].IndexOf("MSIE")>=0 && HeaderFields["User-
    Agent"].IndexOf("Opera")<0)
    {
        if(signature=="explorer")
            ConsoleBuffer.buffer.Add("Explorer Signature Match");
        else if(signature != "")
        {
            ConsoleBuffer.buffer.Add("Signature:"+signature+";User-Agent:
            Explorer;MISMATCH");
            ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
        }
        else
        {
            ConsoleBuffer.buffer.Add("Signature:unrecognized;User-Agent:
            Explorer;MISMATCH");
            ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
        }
    }
    else if(HeaderFields["User-Agent"].IndexOf("Firefox")>=0)
    {
        if(signature=="firefox")
            ConsoleBuffer.buffer.Add("Firefox Signature Match");
        else if(signature != "")
        {
            ConsoleBuffer.buffer.Add("Signature:"+signature+";User-Agent:
            Firefox;MISMATCH");
            ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
        }
        else
        {
            ConsoleBuffer.buffer.Add("Signature:unrecognized;User-Agent:
            Firefox;MISMATCH");
            ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
        }
    }
    else if(HeaderFields["User-Agent"].IndexOf("Netscape")>=0)
    {
        if(signature=="firefox")
            ConsoleBuffer.buffer.Add("Netscape Signature Match");
        else if(signature != "")
        {
            ConsoleBuffer.buffer.Add("Signature:"+signature+";User-Agent:
            Netscape;MISMATCH");
            ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
        }
        else
        {

```

```

        ConsoleBuffer.buffer.Add("Signature:unrecognized;User-Agent:
        Netscape;MISSMATCH");
        ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
    }
}
else if(HeaderFields["User-Agent"].IndexOf("Opera")>=0)
{
    if(signature=="opera")
        ConsoleBuffer.buffer.Add("Opera Signature Match");
    else if(signature != "")
    {
        ConsoleBuffer.buffer.Add("Signature:"+signature+";User-Agent:
        Opera;MISSMATCH");
        ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
    }
    else
    {
        ConsoleBuffer.buffer.Add("Signature:unrecognized;User-Agent:
        Opera;MISSMATCH");
        ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
    }
}
else
{
    ConsoleBuffer.buffer.Add("Unrecognized User-Agent");
    ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
}

}
else if(signature != "")
{
    ConsoleBuffer.buffer.Add("Signature:"+signature+";User-Agent:not
    specified;MISSMATCH");
    ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
}
else
{
    ConsoleBuffer.buffer.Add("Unrecognised signature, User-Agent not
    provided;MISSMATCH");
    ConsoleBuffer.buffer.Add("Source:" + ClientSocket.RemoteEndPoint.ToString());
}

}
catch {}
}

}
int Port;
string Host;
Ret = -1;
if (HttpRequestType.ToUpper().Equals("CONNECT"))
{ //HTTPS
    Ret = RequestedPath.IndexOf(":");
    if (Ret >= 0)
    {
        Host = RequestedPath.Substring(0, Ret);
        if (RequestedPath.Length > Ret + 1)
            Port = int.Parse(RequestedPath.Substring(Ret + 1));
        else
            Port = 443;
    }
    else
    {
        Host = RequestedPath;
        Port = 443;
    }
}
else
{ //Normal HTTP
    Ret = ((string)HeaderFields["Host"]).IndexOf(":");
    if (Ret > 0)
    {
        Host = ((string)HeaderFields["Host"]).Substring(0, Ret);
        Port = int.Parse(((string)HeaderFields["Host"]).Substring(Ret + 1));
    }
    else

```

```

    {
        Host = (string)HeaderFields["Host"];
        Port = 80;
    }
    if (HttpRequestType.ToUpper().Equals("POST"))
    {
        int index = Query.IndexOf("\r\n\r\n");
        m_HttpPost = Query.Substring(index + 4);
    }
}
try
{
    IPEndPoint DestinationEndPoint = new IPEndPoint(Dns.Resolve(Host).AddressList[0],
        Port);
    DestinationSocket = new Socket(DestinationEndPoint.AddressFamily,
        SocketType.Stream, ProtocolType.Tcp);
    if (HeaderFields.ContainsKey("Proxy-Connection") && HeaderFields["Proxy-
        Connection"].ToLower().Equals("keep-alive"))
        DestinationSocket.SetSocketOption(SocketOptionLevel.Socket,
            SocketOptionName.KeepAlive, 1);
    DestinationSocket.BeginConnect(DestinationEndPoint, new
        AsyncCallback(this.OnConnected), DestinationSocket);
}
catch
{
    SendBadRequest();
    return;
}
}
///<summary>Pars(es a specified HTTP query into its header fields.</summary>
///<param name="Query">The HTTP query string to parse.</param>
///<returns>A StringDictionary object containing all the header fields with their
    data.</returns>
///<exception cref="ArgumentNullException">The specified query is null.</exception>
private StringDictionary ParseQuery(string Query)
{
    StringDictionary retdict = new StringDictionary();
    string [] Lines = Query.Replace("\r\n", "\n").Split('\n');
    int Cnt, Ret;
    //Extract requested URL
    if (Lines.Length > 0)
    {
        //Parse the Http Request Type
        Ret = Lines[0].IndexOf(' ');
        if (Ret > 0)
        {
            HttpRequestType = Lines[0].Substring(0, Ret);
            Lines[0] = Lines[0].Substring(Ret).Trim();
        }
        //Parse the Http Version and the Requested Path
        Ret = Lines[0].LastIndexOf(' ');
        if (Ret > 0)
        {
            HttpVersion = Lines[0].Substring(Ret).Trim();
            RequestedPath = Lines[0].Substring(0, Ret);
        }
        else
        {
            RequestedPath = Lines[0];
        }
        //Remove http:// if present
        if (RequestedPath.Length >= 7 && RequestedPath.Substring(0,
            7).ToLower().Equals("http://"))
        {
            Ret = RequestedPath.IndexOf('/', 7);
            if (Ret == -1)
                RequestedPath = "/";
            else
                RequestedPath = RequestedPath.Substring(Ret);
        }
    }
    //parsing of headers follows
    for(Cnt = 1; Cnt < Lines.Length; Cnt++)
    {
        Ret = Lines[Cnt].IndexOf(":");
        if (Ret > 0 && Ret < Lines[Cnt].Length - 1)
        {

```

```

    try
    {

        retdict.Add(Lines[Cnt].Substring(0, Ret), Lines[Cnt].Substring(Ret + 1).Trim());

    }
    catch {}
    }
    return retdict;
}

///<summary>Parses a specified HTTP query into its header fields.</summary>
///<param name="Query">The HTTP query string to parse.</param>
///<returns>A StringDictionary object containing all the header fields with their
    data.</returns>
///<exception cref="ArgumentNullException">The specified query is null.</exception>
private StringDictionary ParseQuerySignature(string Query)
{
    int order = 0;
    StringDictionary retdict = new StringDictionary();
    string [] Lines = Query.Replace("\r\n", "\n").Split('\n');
    int Cnt, Ret;
    //Extract requested URL
    string comparer;
    //parsing of headers follows
    for(Cnt = 1; Cnt < Lines.Length; Cnt++)
    {
        Ret = Lines[Cnt].IndexOf(":");
        if (Ret > 0 && Ret < Lines[Cnt].Length - 1)
        {
            try
            {
                retdict.Add(Lines[Cnt].Substring(0, Ret), order.ToString());
                //comparer =
                System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToTitleCase(Lines[Cn
                    t].Substring(0, Ret));
                comparer = Lines[Cnt].Substring(0, Ret).ToLower();
                comparer =
                System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToTitleCase(comparer
                    );

                if(Lines[Cnt].IndexOf(comparer)<0)
                {
                    ConsoleBuffer.buffer.Add("Invalid Header Name Casing. Source:" +
                        ClientSocket.RemoteEndPoint.ToString());
                }

                if(Lines[Cnt].Substring(Ret + 1)
                    != " " + Lines[Cnt].Substring(Ret + 1).Trim())
                {
                    ConsoleBuffer.buffer.Add("Invalid Linear Spacing. Source:" +
                        ClientSocket.RemoteEndPoint.ToString());
                }

                if(!ConsoleBuffer.accepted_headers.ContainsKey(Lines[Cnt].Substring(0, Ret)))
                {
                    ConsoleBuffer.buffer.Add("Unrecognised Header Detected:
                        "+Lines[Cnt].Substring(0, Ret));
                }
                order++;
            }
            catch {}
        }
    }

    return retdict;
}

```

```

}
///Sends a "400 - Bad Request" error to the client.</summary>
private void SendBadRequest()
{
    string brs = "HTTP/1.1 400 Bad Request\r\nConnection: close\r\nContent-Type:
    text/html\r\n\r\n<html><head><title>400 Bad Request</title></head><body><div
    align=\"center\"><table border=\"0\" cellspacing=\"3\" cellpadding=\"3\"
    bgcolor=\"#C0C0C0\"><tr><td><table border=\"0\" width=\"500\"
    cellspacing=\"3\" cellpadding=\"3\"><tr><td bgcolor=\"#B2B2B2\"><p
    align=\"center\"><strong><font size=\"2\" face=\"Verdana\">400 Bad
    Request</font></strong></p></td></tr><tr><td bgcolor=\"#D1D1D1\"><font
    size=\"2\" face=\"Verdana\"> The proxy server could not understand the HTTP
    request!<br><br> Please contact your network administrator about this
    problem.</font></td></tr></table></center></td></tr></table></div></body></ht
    ml>";

    try
    {
        ClientSocket.BeginSend(Encoding.ASCII.GetBytes(brs), 0, brs.Length,
            SocketFlags.None, new AsyncCallback(this.OnErrorSent), ClientSocket);
    }
    catch
    {
        Dispose();
    }
}

///Sends a "400 - Filtered result" error to the client.</summary>
private void SendBlockedRequest()
{
    string brs = "HTTP/1.1 400 Bad Request\r\nConnection: close\r\nContent-Type:
    text/html\r\n\r\n<html><head><title>Your request has been
    blocked.</title></head><body><div align=\"center\"><table border=\"0\"
    cellspacing=\"3\" cellpadding=\"3\" bgcolor=\"#C0C0C0\"><tr><td><table
    border=\"0\" width=\"500\" cellspacing=\"3\" cellpadding=\"3\"><tr><td
    bgcolor=\"#B2B2B2\"><p align=\"center\"><strong><font size=\"2\"
    face=\"Verdana\">400 Bad Request</font></strong></p></td></tr><tr><td
    bgcolor=\"#D1D1D1\"><font size=\"2\" face=\"Verdana\">Your request was
    blocked and logged by the proxy
    server.<br><br></font></td></tr></table></center></td></tr></table></div></bo
    dy></html>";

    try
    {
        ClientSocket.BeginSend(Encoding.ASCII.GetBytes(brs), 0, brs.Length,
            SocketFlags.None, new AsyncCallback(this.OnErrorSent), ClientSocket);
    }
    catch
    {
        Dispose();
    }
}

///Rebuilds the HTTP query, starting from the HttpRequestType, RequestedPath,
    HttpVersion and HeaderFields properties.</summary>
///

```

```

    }
}
//ret += "Covert-Channel: Covert Data going out\r\n";
foreach(string str in keys)
{
    ret += str;
}
ret += "\r\n";
if (m_HttpPost != null)
    ret += m_HttpPost;
}

return ret;
}
///<summary>Returns text information about this HttpClient object.</summary>
///<returns>A string representing this HttpClient object.</returns>
public override string ToString()
{
    return ToString(false);
}
///<summary>Returns text information about this HttpClient object.</summary>
///<returns>A string representing this HttpClient object.</returns>
///<param name="WithUrl">Specifies whether or not to include information about the
    requested URL.</param>
public string ToString(bool WithUrl)
{
    string Ret;
    try
    {
        if (DestinationSocket == null || DestinationSocket.RemoteEndPoint == null)
            Ret = "Incoming HTTP connection from " +
                ((IPEndPoint)ClientSocket.RemoteEndPoint).Address.ToString();
        else
            Ret = "HTTP connection from " +
                ((IPEndPoint)ClientSocket.RemoteEndPoint).Address.ToString() + " to " +
                ((IPEndPoint)DestinationSocket.RemoteEndPoint).Address.ToString() + " on port " +
                ((IPEndPoint)DestinationSocket.RemoteEndPoint).Port.ToString();
        if (HeaderFields != null && HeaderFields.ContainsKey("Host") && RequestedPath !=
            null)
            Ret += "\r\n" + " requested URL: http://" + HeaderFields["Host"] + RequestedPath;
    }
    catch
    {
        Ret = "HTTP Connection";
    }
    return Ret;
}
// private variables
/// <summary>Holds the value of the HttpQuery property.</summary>
private string m_HttpQuery = "";
/// <summary>Holds the value of the RequestedPath property.</summary>
private string m_RequestedPath = null;
/// <summary>Holds the value of the HeaderFields property.</summary>
private StringDictionary m_HeaderFields = null;
/// <summary>Holds the value of the HeaderFieldsSignature property.</summary>
private StringDictionary m_HeaderFieldsSignature = null;
/// <summary>Holds the value of the HttpVersion property.</summary>
private string m_HttpVersion = "";
/// <summary>Holds the value of the HttpRequestType property.</summary>
private string m_HttpRequestType = "";
/// <summary>Holds the POST data</summary>
private string m_HttpPost = null;
}
}

```


Appendix 5 - HTTP Analyser Foundation - Code Listing

HTTP Analyser Foundation uses *SharpPcap* wrapper to control adapter level packet capture operations of *WinPcap.dll*. The application is made up of two different class files, *Form1.cs* and *ConColl.cs*, both written for the purpose of this project.

Form1.cs:

```
/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmail.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Text.RegularExpressions;
using Tamir.IPLib;
using Tamir.IPLib.Packets;
using System.Text;

namespace HTTPAnalyser
{
    /// <summary>
    /// Form1 is the main window of the HTTPAnalyser.
    /// </summary>
    public class HTTPAnalyser_Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.ListView lvPackets;
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.ComboBox cbAdapters;
        private System.Windows.Forms.ListBox lbHeaders;
        private System.Windows.Forms.MenuItem mFile;
        private System.Windows.Forms.MenuItem mCapture;
        private System.Windows.Forms.MenuItem mcStart;
        private System.Windows.Forms.MenuItem mcStop;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem3;
        private ArrayList headerArray = new ArrayList(); //stores PacketCollections
        private ArrayList sigArray = new ArrayList(); //stores Signatures
        private ArrayList sigSyncArray; //synchronized wrapper
        private System.Windows.Forms.ListView lvCon;
        private System.Windows.Forms.Label label1;
        private PcapDevice device;
        private PcapDeviceList getNetConnections;
        private System.Windows.Forms.CheckBox cbChip;
        private System.Windows.Forms.GroupBox gbDirection;
        private System.Windows.Forms.RadioButton rbToBoth;
        private System.Windows.Forms.RadioButton rbToSrv;
        private System.Windows.Forms.RadioButton rbToCnt;
        private System.Windows.Forms.GroupBox gbView;
        private System.Windows.Forms.RadioButton rbViewFull;
        private System.Windows.Forms.RadioButton rbViewPacket;
        private System.Windows.Forms.CheckBox chHeaders;
        private System.Windows.Forms.GroupBox gbPackets;
        private System.Windows.Forms.MenuItem mhAbout;
        private System.Windows.Forms.MenuItem mhDoc;
        private System.Windows.Forms.CheckBox cbDump;
        private string dumpFile = "";
        private System.Windows.Forms.OpenFileDialog ofdDump;
        private System.Windows.Forms.MenuItem mfOpen;
        private System.Windows.Forms.GroupBox gbAdapter;
        private System.Windows.Forms.MenuItem menuItem2;

        /// <summary>
        /// Required designer variable.
        /// </summary>
    }
}
```

```

private System.ComponentModel.Container components = null;

/// <summary>
/// Default constructor
/// </summary>
public HTTPAnalyser_Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new HTTPAnalyser_Form1());
}

/// <summary>
/// Form_Load - Sets up ListViews and checks for working network adapters
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Load(object sender, System.EventArgs e)
{
    headerSyncArray = ArrayList.Synchronized(headerArray);
    sigSyncArray = ArrayList.Synchronized(sigArray);

    //lvCon columns
    if(lvCon.Width/5 > 20)
        lvCon.Columns.Add("Connection", lvCon.Width /5-20 ,
            HorizontalAlignment.Left);
    else
        lvCon.Columns.Add("Connection", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientPort", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerPort", lvCon.Width /5 , HorizontalAlignment.Left);

    //lvPackets columns
    lvPackets.Columns.Add("No", 30 , HorizontalAlignment.Left);
    if(lvPackets.Width/6 > 53)
        lvPackets.Columns.Add("Port", lvPackets.Width /6 - 53 , HorizontalAlignment.Left);
    else
        lvPackets.Columns.Add("Port", lvPackets.Width /6 , HorizontalAlignment.Left);
    lvPackets.Columns.Add("Flags", lvPackets.Width /6 , HorizontalAlignment.Left);
    lvPackets.Columns.Add("Size (Data Size)", lvPackets.Width /6 ,
        HorizontalAlignment.Left);
    lvPackets.Columns.Add("Date", lvPackets.Width /6 , HorizontalAlignment.Left);
    lvPackets.Columns.Add("SEQ", lvPackets.Width /6 , HorizontalAlignment.Left);
    lvPackets.Columns.Add("ACK", lvPackets.Width /6 , HorizontalAlignment.Left);
    lvPackets.View = View.Details;

    //set menu items
    mcStop.Enabled = false;
    mcStart.Enabled = false;

```

```

//Adaptersc collection
getNetConnections = SharpPcap.GetAllDevices();
for (int i = 0; i < getNetConnections.Count ; i++)
{
    cbAdapters.Items.Add("(" + (i) + ") " + getNetConnections[i].PcapDescription);
}
cbAdapters.Invalidate();
}

/// <summary>
/// Capture Menu Start Click - starts reading from the selected adapter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStart_Click(object sender, System.EventArgs e)
{
    lvPackets.Items.Clear();
    lbHeaders.Items.Clear();
    //conArray.Clear();
    //sigArray.Clear();
    //vnCounter = 0;
    //axPacketXCtrl1.Start();
    if(cbChip.Checked)
    {
        device.PcapOpen(false,1000);
    }
    else
    {
        device.PcapOpen(true,1000);
    }
    device.PcapSetFilter("port 80");
    device.PcapStartCapture();
    mcStart.Enabled = false;
    mcStop.Enabled = true;
    gbAdapter.Enabled = false;
    if(cbDump.Checked && dumpFile != "")
    {
        device.PcapDumpOpen(dumpFile);
    }
    else if(cbDump.Checked)
    {
        MessageBox.Show("Could not open Dump File");
    }
}

/// <summary>
/// Capture Menu Stop Click - stops reading
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStop_Click(object sender, System.EventArgs e)
{
    device.PcapStopCapture();
    device.PcapClose();
    cbAdapters.SelectedIndex = -1;
    mcStart.Enabled = false;
    mcStop.Enabled = false;
    gbAdapter.Enabled = true;
}

/// <summary>
/// OnPacket event handler - builds collection of "conversations" and displays it in
    lvCon
/// </summary>
/// <param name="sender"></param>
/// <param name="aPacket"></param>
private void device_PcapOnPacketArrival(object sender, Packet aPacket)
{
    if(aPacket is TCPpacket)
    {
        TCPpacket tcp = (TCPpacket)aPacket;
        if(tcp.DestinationPort == 80 || tcp.SourcePort == 80)//herefor the offline dump
            handling
    }
}

```

```

{
    int i = 0;
    int key = -1;
    string cntIP;
    string srvIP;
    int cntPort;
    int srvPort;

    if(tcp.DestinationPort == 80)
    {
        cntIP = tcp.SourceAddress;
        srvIP = tcp.DestinationAddress;
        cntPort = tcp.SourcePort;
        srvPort = tcp.DestinationPort;
    }
    else
    {
        cntIP = tcp.DestinationAddress;
        srvIP = tcp.SourceAddress;
        cntPort = tcp.DestinationPort;
        srvPort = tcp.SourcePort;
    }

    lock(sigSyncArray.SyncRoot)
    {
        System.Collections.IEnumerator myEnumerator = sigSyncArray.GetEnumerator();
        while ( myEnumerator.MoveNext() )
        {
            ConColl connection = (ConColl)myEnumerator.Current;
            if( connection.CheckSignature(cntIP,srvIP,cntPort,srvPort))
            {
                connection.Add(aPacket);
                key = i;
                break;
            }
            i++;
        } //end while
    } //end lock

    if (key == (-1) )
    {
        ConColl connection = new ConColl(cntIP,srvIP,cntPort,srvPort);
        connection.Add(aPacket);
        sigSyncArray.Add(connection);
        ListViewItem aItem = new ListViewItem();
        key = sigSyncArray.Count-1;
        aItem.SubItems[0].Text = System.Convert.ToString(key.ToString());
        aItem.SubItems.Add(System.Convert.ToString(cntIP));
        aItem.SubItems.Add(System.Convert.ToString(srvIP));
        aItem.SubItems.Add(System.Convert.ToString(cntPort));
        aItem.SubItems.Add(System.Convert.ToString(srvPort));
        lvCon.Items.Add(aItem);
    } //end if connection array does not exist

    if(device.PcapDumpOpened)
    {
        device.PcapDump(aPacket);
    }
    } //end if source or destination port 80
} //end of is TCP
}

/// <summary>
/// lvPackets Selection - displays packet HTTP level data in lbHeaders
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void lvPackets_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(lvPackets.SelectedItems.Count > 0 && lvPackets.SelectedItems.Count > 0)
    {
        if(rbViewPacket.Checked == true)
        {
            ConColl connection = (ConColl) sigSyncArray[lvCon.SelectedIndices[0]];
            TCPPacket oPacket = (TCPPacket)
                connection.GetPacket(Convert.ToInt16(lvPackets.Items[lvPackets.SelectedIndices[0]].Text));

```

```

lbHeaders.Items.Clear();

string headers = "";
int tcpStart = 14 + 4*(Convert.ToInt16(oPacket.Bytes[14])& 0x0F);//ipstart +
    header lenght
int tcpLenght = (Convert.ToInt16(oPacket.Bytes[tcpStart+12])& 0xF0)/4;
int ipTotal =
    (Convert.ToInt16(oPacket.Bytes[16]))*256+(Convert.ToInt16(oPacket.Bytes[17]))
    ;
if( tcpStart+tcpLenght<oPacket.Bytes.Length)
{
    for(int i=tcpStart+tcpLenght;i<(14+ipTotal); i++)
    {
        if((Convert.ToInt16(oPacket.Bytes[i])>31 &&
            Convert.ToInt16(oPacket.Bytes[i])<127)
            || Convert.ToInt16(oPacket.Bytes[i])==13 ||
            Convert.ToInt16(oPacket.Bytes[i])==10)
        {
            headers = headers + (char)(Convert.ToInt16(oPacket.Bytes[i]));
        }
        else
        {
            headers = headers + (oPacket.Bytes[i]).ToString() + " ";
        }
    }
    Regex r = new Regex("\r\n");
    //string[] header_array = ;

    //lbHeaders.Items.Add(tcpLenght);
    //lbHeaders.Items.Add(Convert.ToInt16(oPacket.DataArray.GetValue(tcpStart+14)));
    foreach(string singleHeader in r.Split(headers))
    {
        if(singleHeader != "")//finds empty line - the start of the content
        {
            lbHeaders.Items.Add(singleHeader);
        }
        else
        {
            lbHeaders.Items.Add("<empty line>");
        }
    }
} //end if rbViewPacket == true
} //end if selected == true
}

/// <summary>
/// Application closing event handler - ensures reading from the adapter is
    stoppedprior closure
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Closing(object sender,
    System.ComponentModel.CancelEventArgs e)
{
    if(mcStop.Enabled == true)
    {
        device.PcapStopCapture();
        device.PcapClose();
    }
}

/// <summary>
/// Shows About messagebox
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mhAbout_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Author: Zbigniew Kwecka\nSupervisor: Dr William Buchanan");
}

/// <summary>
/// cbAdapters selected handler - Changes active adapter

```

```

/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void cbAdapters_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();

        mcStop.Enabled = false;
    }
    if(cbAdapters.SelectedIndex > -1)
    {
        if(getNetConnections[cbAdapters.SelectedIndex] is NetworkDevice)
        {
            mcStart.Enabled = true;
            NetworkDevice netConn =
                (NetworkDevice)getNetConnections[cbAdapters.SelectedIndex];
            device = netConn;
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);
        }
        else
        {
            MessageBox.Show("Selected adapter \nis not suitable \nfor packet sniffing");
            cbAdapters.SelectedIndex = -1;
        }
    }
}

/// <summary>
/// Menu File Exit - Terminates the application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuItem3_Click(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();
    }
    Application.Exit();
}

/// <summary>
/// lvCon selected handler - displays packets of the selected conversation in
/// lvPackets
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void lvCon_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(lvCon.SelectedItems.Count > 0)
    {
        ConColl connection = (ConColl) sigSyncArray[lvCon.SelectedIndices[0]];

        int vnCounter = 0;
        int lastDataSize = 0;
        //long lastSeq = 0;
        long requestedAck = 0;
        //ArrayList requestedAcks = new ArrayList();
        lvPackets.Items.Clear();
        lbHeaders.Items.Clear();
        headerSyncArray.Clear();

        for(int i=0;i<connection.Count();i++)
        {
            TCPPacket oPacket = (TCPPacket) connection.GetPacket(i);

            int flags_byte = 27 + 4*(Convert.ToInt16(oPacket.Bytes[14])& 0x0F);//(ipstart+13)
                + IP header lenght
            int tcpStart = 14 + 4*(Convert.ToInt16(oPacket.Bytes[14])& 0x0F);

```

```

//total lenght - (ethernet + iplenght + tcp header lenght)
int dataSize = oPacket.Bytes.Length -
    (tcpStart+(Convert.ToInt16(oPacket.Bytes[tcpStart+12])& 0xF0)/4);
string flags = "";
long seq, ack = 0;
seq = oPacket.SequenceNumber;

ack = oPacket.AcknowledgmentNumber;
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x20)!=0)
{
    flags = flags + "(URG)";
}
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x10)!=0)
{
    flags = flags + "(ACK)";
}
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x08)!=0)
{
    flags = flags + "(PSH)";
}
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x04)!=0)
{
    flags = flags + "(RST)";
}
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x02)!=0)
{
    flags = flags + "(SYN)";
}
if((Convert.ToInt16(oPacket.Bytes[flags_byte]) & 0x01)!=0)
{
    flags = flags + "(FIN)";
}
//(flags_byte-9) start of the seq number

ListViewItem aItem = new ListViewItem();
//
if(oPacket.DestinationPort == 80)
{
    if(lastDataSize <= 10 && dataSize > 10)
    {
        if(rbViewFull.Checked == true)
        {
            buildHTTP(oPacket);
        }
        aItem.ForeColor = Color.FromName("Red");
        requestedAck = ack;
        //requestedAcks.Add(ack);
    }
    lastDataSize = dataSize;
}
else if(oPacket.SourcePort == 80 && requestedAck == seq && dataSize > 10)
{
    if(rbViewFull.Checked == true)
    {
        buildHTTP(oPacket);
    }
    aItem.ForeColor = Color.FromName("Red");
    for(int j = 0; j<10; j++)
    {
        if(oPacket.Data[j].ToString() == "32" && oPacket.Data[j+1].ToString() == "51")
        {
            lastDataSize = 0;
        }
    }
}
if((rbToBoth.Checked == true || (rbToSrv.Checked == true &&
    oPacket.DestinationPort == 80) || (rbToCnt.Checked == true &&
    oPacket.SourcePort== 80))
    && ((chHeaders.Checked == true && aItem.ForeColor == Color.FromName("Red")) ||
    chHeaders.Checked == false))
{
    aItem.SubItems[0].Text = System.Convert.ToString(vnCounter);
    //aItem.SubItems.Add(System.Convert.ToString(oPacket.SourceIpAddress));
    //aItem.SubItems.Add(System.Convert.ToString(oPacket.DestIpAddress));
}

```

```

        aItem.SubItems.Add(System.Convert.ToString(oPacket.SourcePort)+" =>
            "+System.Convert.ToString(oPacket.DestinationPort));
        aItem.SubItems.Add(System.Convert.ToString(flags));
        aItem.SubItems.Add(System.Convert.ToString(oPacket.Bytes.Length + " (" +
            dataSize + ")"));
        aItem.SubItems.Add(System.Convert.ToString(oPacket.PcapHeader.Date));
        aItem.SubItems.Add(System.Convert.ToString(seq));
        aItem.SubItems.Add(System.Convert.ToString(ack));
        lvPackets.Items.Add(aItem);
        //oPacketColl.Add(oPacket);

    } //end if radio box

    vnCounter++;
} //end for each
if(rbViewFull.Checked == true)
{
    lock(headerSyncArray.SyncRoot)
    {
        foreach(string header in headerSyncArray)
        {
            lbHeaders.Items.Add(header);
        }
    }
}
}

/// <summary>
/// Builds HTTP header list for the bottom lbHeaders
/// </summary>
/// <param name="oPacket"></param>
public void buildHTTP(TCPPacket oPacket)
{
    if(rbToBoth.Checked == true || (rbToSrv.Checked == true && oPacket.DestinationPort
        == 80) || (rbToCnt.Checked == true && oPacket.SourcePort== 80))
    {
        Encoding ASCII = Encoding.ASCII;
        string headers = "";
        int tcpStart = 14 + 4*(Convert.ToInt16(oPacket.Bytes[14])& 0x0F);//ipstart +
            header lenght
        int tcpLenght = (Convert.ToInt16(oPacket.Bytes[tcpStart+12])& 0xF0)/4;
        int ipTotal =
            (Convert.ToInt16(oPacket.Bytes[16]))*256+(Convert.ToInt16(oPacket.Bytes[17]))
            ;
        if( tcpStart+tcpLenght<oPacket.Bytes.Length)
        {
            for(int j=tcpStart+tcpLenght;j<(14+ipTotal); j++)
            {
                if((Convert.ToInt16(oPacket.Bytes[j])>31 &&
                    Convert.ToInt16(oPacket.Bytes[j])<127)
                    || Convert.ToInt16(oPacket.Bytes[j])==13 ||
                    Convert.ToInt16(oPacket.Bytes[j])==10)
                {
                    headers = headers + (char)(Convert.ToInt16(oPacket.Bytes[j]));
                }
                else
                {
                    headers = headers + (oPacket.Bytes[j]).ToString() + " ";
                }
            }
        }

        Regex r = new Regex("\r\n");

        int a = 0;
        foreach(string singleHeader in r.Split(headers))
        {
            if(singleHeader != "")//finds empty line - the start of the content
            {
                headerSyncArray.Add(singleHeader);
                a=0;
            }
            else
            {

```



```

        a++;
        headerSyncArray.Add(" ");
        if((a>1&&oPacket.DestinationPort==80)|| (a>0&&oPacket.SourcePort==80))
        {

            headerSyncArray.Add("-----");

            break;
        }
    }
}
} //end if matches the destination settings
}

private void cbDump_CheckedChanged(object sender, System.EventArgs e)
{
    if(cbDump.Checked == true && dumpFile == "")
    {
        ofdDump.ShowDialog();
        if(ofdDump.FileName != "")
        {
            dumpFile = ofdDump.FileName;
        }
        else
        {
            cbDump.Checked = false;
        }
    }
}

private void mfOpen_Click(object sender, System.EventArgs e)
{
    ofdDump.ShowDialog();
    if(ofdDump.FileName != "")
    {
        mcStart.Enabled = false;
        gbAdapter.Enabled = true;
        cbDump.Checked = false;
        cbAdapters.SelectedIndex = -1;
        lvPackets.Items.Clear();
        lbHeaders.Items.Clear();
        try
        {
            device = SharpPcap.GetPcapOfflineDevice( ofdDump.FileName );
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);

            device.PcapOpen();
            device.PcapStartCapture();
            mcStop.Enabled = true;
        }
        catch(Exception exception)
        {
            MessageBox.Show(exception.Message);
        }
    }
    else
    {
        {
            MessageBox.Show("Wrong input file");
        }
    }
}

private void menuItem2_Click(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == false)
    {
        lvCon.Items.Clear();
        lvPackets.Items.Clear();
        lbHeaders.Items.Clear();
        headerSyncArray.Clear();
    }
}

```

```

        sigSyncArray.Clear();
    }
}
}
}

```

ConColl.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmail.com
 * Napier University, Edinburgh
 */

using System;
using System.Collections;

namespace HTTPAnalyser
{
    /// <summary>
    /// Summary description for ConColl.
    /// </summary>
    public class ConColl
    {
        private string cntIP;
        private string srvIP;
        private int cntPort;
        private int srvPort;
        private ArrayList packets;
        private ArrayList synPackets;

        public ConColl(string aCntIP, string aSrvIP, int aCntPort, int aSrvPort)
        {
            if(aCntIP != "" && aSrvIP != "" && aCntPort != 0 && aSrvPort != 0)
            {
                cntIP = aCntIP;
                srvIP = aSrvIP;
                cntPort = aCntPort;
                srvPort = aSrvPort;
                packets = new ArrayList();
                synPackets = ArrayList.Synchronized(packets);
            }
        }

        public int Count()
        {
            return synPackets.Count;
        }

        public object GetPacket(int aIndex)
        {
            if(aIndex < synPackets.Count)
            {
                return synPackets[aIndex];
            }
            else
            {
                return null;
            }
        }

        public void Add(object aPacket)
        {
            if(aPacket != null)
                synPackets.Add(aPacket);
        }

        public bool CheckSignature(string aCntIP, string aSrvIP, int aCntPort, int aSrvPort)
        {

```

```
    if (cntIP == aCntIP && srvIP == aSrvIP && cntPort == aCntPort && srvPort ==  
        aSrvPort)  
    {  
        return true;  
    }  
    else  
    {  
        return false;  
    }  
}  
  
}  
}
```

Appendix 6 – Browser Timer - Code Listing

Form1.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
using System.IO;
using System.Threading;

namespace HTTPBrowser
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button button1;
        private AxSHDocVw.AxWebBrowser axWebBrowser1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.Button button3;
        private System.ComponentModel.IContainer components;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.Resources.ResourceManager resources = new
                System.Resources.ResourceManager(typeof(Form1));

```

```

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}
int read = 0;
int current = 0;
int arraySize=200;
object notUsed = null;
string [] sites;
DateTime startTime;
TimeSpan timeTaken;
double average = 0;
long sum = 0;
StreamWriter sw;

private void button1_Click(object sender, System.EventArgs e)
{
    try
    {
        string file = "sites.txt";
        string txt="";

        if (File.Exists(file))
        {
            StreamReader SW = new StreamReader(file);

            while ((txt=SW.ReadLine())!=null && read < arraySize)
            {
                sites[read]=txt;
                //textBox2.Text += txt + Environment.NewLine;
                read++;
            }
            SW.Close();
        }

        if(checkBox1.Checked == true)
        {
            if(textBox1.Text != "")
            {
                sw = new StreamWriter(textBox1.Text,false);

            }
            else
            {
                sw = new StreamWriter("default_output.txt",false);
            }
        }
        button1.Enabled = false;
        checkBox1.Enabled = false;
        textBox1.Enabled = false;
        button2.Enabled = true;
        button3.Enabled = true;
        Thread thdNavigate = new Thread(new ThreadStart(nav));

        thdNavigate.Start();

    }
    catch(Exception ex)
    {
    }
}

public void nav()
{
    if(button1.Enabled)
        return;
    if(current < read )//&& sites[current] != Environment.NewLine)
    {

```

```

        startTime = DateTime.Now;

        axWebBrowser1.Navigate(sites[current], ref notUsed, ref notUsed, ref notUsed, ref
            notUsed);
        current++;
    }
    else
    {
        MessageBox.Show(sum.ToString() + " " + read.ToString());
        average = sum/read;
        MessageBox.Show("Average time taken: " + average.ToString());
        sw.Close();
        button1.Enabled = true;
        checkBox1.Enabled = true;
        textBox1.Enabled = true;
        button2.Enabled = false;
        button3.Enabled = false;
    }
}

public string getHTTP(string aURL)
{
    HttpWebRequest httpRequest;
    HttpWebResponse httpResponse;
    string body = "";
    Stream responseStream;
    string responseHeader;
    Byte[] RecvBytes = new Byte[Byte.MaxValue];
    Int32 bytes;

    httpRequest = (HttpWebRequest) WebRequest.Create(aURL);
    httpResponse = (HttpWebResponse) httpRequest.GetResponse();
    responseStream = httpResponse.GetResponseStream();
    responseHeader = httpResponse.GetResponseHeader("Content-Type");

    while(true)
    {
        bytes = responseStream.Read(RecvBytes, 0, RecvBytes.Length);
        if(bytes <= 0) break;
        body += System.Text.Encoding.UTF8.GetString(RecvBytes, 0, bytes);
    }
    return httpResponse.StatusDescription + responseHeader + body;
}

private void Form1_Load(object sender, System.EventArgs e)
{
    sites = new string[arraySize];
}

private void axWebBrowser1_DocumentComplete(object sender,
    AxSHDocVw.DWebBrowserEvents2_DocumentCompleteEvent e)
{
    timeTaken = (DateTime.Now - startTime);
    textBox2.Text += e.uRL.ToString() + " - Time taken: " + timeTaken.ToString() +
        Environment.NewLine;
    if(timeTaken.TotalMilliseconds > 0)
    {
        sum += Convert.ToInt64(timeTaken.TotalMilliseconds);
        if(sw != null)
        {
            sw.WriteLine(e.uRL.ToString()+"\t\t"+timeTaken.ToString());
        }
    }
    nav();
}

private void button2_Click(object sender, System.EventArgs e)
{
    button1.Enabled = true;
    checkBox1.Enabled = true;
    textBox1.Enabled = true;
    button2.Enabled = false;
    button3.Enabled = false;
    sw.Close();
}

```

```
    }  
  
    private void checkBox1_CheckedChanged(object sender, System.EventArgs e)  
    {  
  
    }  
  
    private void button3_Click(object sender, System.EventArgs e)  
    {  
        nav();  
    }  
  
    }  
}
```

Appendix 7 – Browser Caller - Code Listing

Form1.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
using System.IO;
using System.Threading;
using System.Diagnostics;
using System.Text;

namespace HTTPBrowser
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.Timer triggerProcess;
        private System.Windows.Forms.GroupBox gbBrowser;
        private System.Windows.Forms.RadioButton rbFirefox;
        private System.Windows.Forms.RadioButton rbIEExplorer;
        private System.Windows.Forms.RadioButton rbOpera;
        private System.Windows.Forms.RadioButton rbNetscape;
        private System.Windows.Forms.TextBox tbSites;
        private System.Windows.Forms.Timer browserDelay;
        private System.ComponentModel.IContainer components;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        /// <summary>
        /// The main entry point for the application.

```



```

/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}
int read = 0;
int current = 0;
int arraySize=1000;
object notUsed = null;
string [] sites;
DateTime startTime;
TimeSpan timeTaken;
double average = 0;
string target= "";
long sum = 0;

bool secondExecution = false;

private void button1_Click(object sender, System.EventArgs e)
{
    button2.Enabled = true;
    checkBox1.Enabled = false;

    string file = Application.StartupPath+"\\sites.txt";
    string txt="";
    current = 0;

    if (File.Exists(file))
    {
        StreamReader SW = new StreamReader(file);

        while ((txt=SW.ReadLine())!=null && read < arraySize)
        {
            sites[read]=txt;
            //textBox2.Text += txt + Environment.NewLine;
            read++;
        }
        SW.Close();
    }
    else
    {
        tbSites.Text += Environment.NewLine + "ERROR: Error reading sites file";
    }
    triggerProcess.Enabled = true;
    button1.Enabled = false;
    gbBrowser.Enabled = false;
    tbSites.Text = "";
}

public void nav()
{
    if(current < read )//&& sites[current] != Environment.NewLine)
    {
        startTime = DateTime.Now;

        current++;
    }
    else
    {
        MessageBox.Show(sum.ToString() + " " + read.ToString());
        average = sum/read;
        MessageBox.Show("Average time taken: " + average.ToString());
    }
}

public string getHTTP(string aURL)
{
    HttpWebRequest httpRequest;
    HttpWebResponse httpResponse;
    string body = "";
    Stream responseStream;
    string responseHeader;

```

```

Byte[] RecvBytes = new Byte[Byte.MaxValue];
Int32 bytes;

httpRequest = (HttpWebRequest) WebRequest.Create(aURL);
httpResponse = (HttpWebResponse) httpRequest.GetResponse();
responseStream = httpResponse.GetResponseStream();
responseHeader = httpResponse.GetResponseHeader("Content-Type");

while(true)
{
    bytes = responseStream.Read(RecvBytes,0,RecvBytes.Length);
    if(bytes<=0) break;
    body += System.Text.Encoding.UTF8.GetString(RecvBytes,0,bytes);
}
return httpResponse.StatusDescription + responseHeader + body;
}

private void Form1_Load(object sender, System.EventArgs e)
{
    sites = new string[arraySize];
}

private void axWebBrowser1_DocumentComplete(object sender,
    AxSHDocVw.DWebBrowserEvents2_DocumentCompleteEvent e)
{
    timeTaken = (DateTime.Now - startTime);
    tbSites.Text += e.uRL.ToString()+ " - Time taken: " + timeTaken.ToString() +
        Environment.NewLine;
    if(timeTaken.TotalMilliseconds >0)
    {
        sum += Convert.ToInt64(timeTaken.TotalMilliseconds);
    }
    nav();
}

private void button2_Click(object sender, System.EventArgs e)
{
    triggerProcess.Enabled = false;
    browserDelay.Enabled = false;
    button1.Enabled = true;
    gbBrowser.Enabled = true;
    button2.Enabled = false;
    checkBox1.Enabled = true;
}

private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{
}

private void triggerProcess_Tick(object sender, System.EventArgs e)
{
    triggerProcess.Enabled = false;
    if(button2.Enabled == false)
        return;
    if(tbSites.Text.Length + 30 > tbSites.MaxLength)
        tbSites.Text = "";

    StreamWriter hostFile = null;
    if(checkBox1.Checked == true)
    {
        if(File.Exists("C:\\WINDOWS\\system32\\drivers\\etc\\hosts"))
        {
            hostFile = new StreamWriter("C:\\WINDOWS\\system32\\drivers\\etc\\hosts",false);
        }
        else
        {
            hostFile = new StreamWriter("C:\\WINNT\\system32\\drivers\\etc\\hosts",false);
        }
    }
    if(current < read )//&& sites[current] != Environment.NewLine)

```

```

{
    target = sites[current];
    if(secondExecution == false && checkBox1.Checked == true)
    {

        hostFile.WriteLine("127.0.0.1\tlocalhost");
        hostFile.WriteLine("192.168.1.7\twww.filteringproxy.com");
        hostFile.Close();
        secondExecution = true;
    }
    else if(secondExecution == true && checkBox1.Checked == true)
    {

        hostFile.WriteLine("127.0.0.1\tlocalhost");
        hostFile.WriteLine("192.168.1.8\twww.filteringproxy.com");
        hostFile.Close();
        current++;
        secondExecution = false;
    }
    else
    {

        current++;
    }
    ProcessStartInfo startInfo;

    try
    {
        if(rbFirefox.Checked == true)
        {
            System.Diagnostics.Process[] p =System.Diagnostics.Process.GetProcesses();
            for(int i=0 ;i<p.Length;i++)
            {
                if (p[i].ProcessName.ToLower()=="firefox")
                {
                    p[i].CloseMainWindow();
                    p[i].WaitForExit(60000);
                }
            }

            startInfo = new ProcessStartInfo("C:\\\\PROGRA~1\\\\MOZILL~1\\\\FIREFOX.EXE");
            startInfo.Arguments = "-url \\""+target+"\"";
            Process.Start(startInfo);
            tbSites.Text += Environment.NewLine + target;
            triggerProcess.Enabled = true;
        }
        else if(rbIEExplorer.Checked == true)
        {
            System.Diagnostics.Process[] p =System.Diagnostics.Process.GetProcesses();
            for(int i=0 ;i<p.Length;i++)
            {
                if (p[i].ProcessName.ToLower()=="iexplore")
                {
                    p[i].CloseMainWindow();
                    p[i].WaitForExit(60000);
                    if(!p[i].HasExited)
                        p[i].Kill();
                }
            }

            startInfo = new ProcessStartInfo("IExplore.EXE");
            startInfo.Arguments = target;
            Process.Start(startInfo);
            tbSites.Text += Environment.NewLine + target;
            triggerProcess.Enabled = true;
        }
        else if(rbOpera.Checked == true)
        {
            System.Diagnostics.Process[] p =System.Diagnostics.Process.GetProcesses();
            for(int i=0 ;i<p.Length;i++)
            {
                if (p[i].ProcessName.ToLower()=="opera")
                {
                    p[i].CloseMainWindow();
                    p[i].WaitForExit(60000);
                    MessageBox.Show("Opera kill about to be executed");
                    if(!p[i].HasExited)

```

```

        {
            p[i].Kill();
        }
    }

    startInfo = new ProcessStartInfo("c:\\Progra~1\\Opera\\Opera.exe");

    startInfo.Arguments = target;
    Process.Start(startInfo);
    tbSites.Text += Environment.NewLine + target;
    triggerProcess.Enabled = true;
}
else if(rbNetscape.Checked == true)
{
    System.Diagnostics.Process[] p = System.Diagnostics.Process.GetProcesses();
    for(int i=0 ;i<p.Length;i++)
    {
        if (p[i].ProcessName.ToLower()=="netscape")
        {
            p[i].CloseMainWindow();
            p[i].WaitForExit(600000);
            MessageBox.Show("Netscape kill about to be executed");
            if(!p[i].HasExited)
            {
                p[i].Kill();
            }
        }
    }

    startInfo = new ProcessStartInfo("C:\\Program Files\\Netscape\\Netscape
        Browser\\netscape.exe");

    startInfo.Arguments = target;
    Process.Start(startInfo);
    tbSites.Text += Environment.NewLine + target;
    triggerProcess.Enabled = true;
}
else
{
    triggerProcess.Enabled = false;
    button1.Enabled = true;
    gbBrowser.Enabled = true;
}

}
catch
{
    (
        System.ComponentModel.Win32Exception noBrowser)
    {
        if (noBrowser.ErrorCode==-2147467259)
            MessageBox.Show(noBrowser.Message);
    }
    catch (System.Exception other)
    {
        MessageBox.Show(other.Message);
    }
}
else
{
    triggerProcess.Enabled = false;
    button1.Enabled = true;
    gbBrowser.Enabled = true;
}
}

private void button3_Click(object sender, System.EventArgs e)
{
    System.Diagnostics.Process[] p = System.Diagnostics.Process.GetProcesses();
    for(int i=0 ;i<p.Length;i++)
    {
        if (p[i].ProcessName=="Opera") p[i].CloseMainWindow();
    }
}

```

```

    }

    private void browserDelay_Tick(object sender, System.EventArgs e)
    {
        browserDelay.Enabled = false;
        if(button2.Enabled == false)
            return;
        try
        {
            if(rbOpera.Checked == true)
            {
                System.Diagnostics.Process[] p =System.Diagnostics.Process.GetProcesses();
                for(int i=0 ;i<p.Length;i++)
                {
                    if (p[i].ProcessName=="Opera") p[i].CloseMainWindow();
                }

                p = System.Diagnostics.Process.GetProcesses();
                for(int i=0 ;i<p.Length;i++)
                {
                    if (p[i].ProcessName=="Opera") p[i].Kill();
                }
                ProcessStartInfo startInfo = new
                    ProcessStartInfo("c:\\Progra~1\\Opera\\Opera.exe");

                startInfo.Arguments = target;
                Process.Start(startInfo);
                tbSites.Text += Environment.NewLine + target;

            }
            else if(rbNetscape.Checked == true)
            {
                System.Diagnostics.Process[] p =System.Diagnostics.Process.GetProcesses();
                for(int i=0 ;i<p.Length;i++)
                {
                    if (p[i].ProcessName=="netscape") p[i].CloseMainWindow();
                }

                p = System.Diagnostics.Process.GetProcesses();
                for(int i=0 ;i<p.Length;i++)
                {
                    if (p[i].ProcessName=="netscape") p[i].Kill();
                }
                ProcessStartInfo startInfo = new ProcessStartInfo("C:\\Program
                    Files\\Netscape\\Netscape Browser\\netscape.exe");

                startInfo.Arguments = target;
                Process.Start(startInfo);
                tbSites.Text += Environment.NewLine + target;
            }
            triggerProcess.Enabled = true;
        }
        catch(System.ComponentModel.Win32Exception noBrowser)
        {
            if (noBrowser.ErrorCode==2147467259)
                MessageBox.Show(noBrowser.Message);
        }
        catch (System.Exception other)
        {
            MessageBox.Show(other.Message);
        }
    }

}

```

Appendix 8 – Data Hiding Proxy - Code Listing

Proxy.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
using System.Text;

namespace FilterProxy_GUI
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button bRun;
        private System.Windows.Forms.ListBox lbConsole;
        private System.Windows.Forms.Button bExit;
        private System.Windows.Forms.Button bShow;
        private System.Windows.Forms.Button bStop;
        private System.Windows.Forms.Timer consoleTimer;
        private System.Windows.Forms.TextBox tbFilter;
        private System.Windows.Forms.TextBox tbListenerPort;
        private System.Windows.Forms.ComboBox cbListenerAddress;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.CheckBox checkBox2;
        private System.Windows.Forms.CheckBox checkBox3;
        private System.Windows.Forms.CheckBox checkBox4;
        private System.Windows.Forms.CheckBox checkBox5;
        private System.Windows.Forms.CheckBox checkBox7;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.RadioButton radioButton1;
        private System.Windows.Forms.RadioButton radioButton2;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button button1;
        private System.ComponentModel.IContainer components;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
        }
    }
}

```

```

    }
}
base.Dispose( disposing );
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}

private Listener listener;
private int bufferRead = 0;

private void bRun_Click(object sender, System.EventArgs e)
{
    string classtype = "FilterProxy_GUI.Http.HttpListener";
    if (classtype == "")
        return;
    else if (Type.GetType(classtype) == null)
    {
        lbConsole.Items.Add("The specified listener class does not exist!");
        return;
    }
    string construct;
    if (cbListenerAddress.SelectedIndex > -1)
    {
        construct = "host:" + cbListenerAddress.SelectedItem.ToString() + ";int:" +
            tbListenerPort.Text.Trim();
    }
    else
    {
        construct = "host:127.0.0.1;int:80";
    }

    object listenObject = CreateListener(classtype, construct);
    if (listenObject == null)
    {
        lbConsole.Items.Add("Invalid construction string.");
        return;
    }

    try
    {
        listener = (Listener)listenObject;
    }
    catch
    {
        lbConsole.Items.Add("The specified object is not a valid Listener object.");
        return;
    }
    try
    {
        listener.Start();
        lbConsole.Items.Add("Proxy started");
        lbConsole.Items.Add("Listening on" +
            construct.Replace(";int:", ":").Replace("host:", ": "));
        consoleTimer.Enabled = true;
    }
    catch
    {
        Console.WriteLine("Error while staring the Listener.\r\n(Perhaps the specified
            port is already in use?)");
        return;
    }
}

/// <summary>
/// Creates a new Listener obejct from a given listener name and a given listener
/// parameter string.
/// </summary>
/// <param name="type">The type of object to instantiate.</param>
/// <param name="cpars"></param>

```

```

/// <returns></returns>
public Listener CreateListener(string type, string cpars)
{
    try
    {
        string [] parts = cpars.Split(';');
        object [] pars = new object[parts.Length];
        string oval = null, otype = null;
        int ret;
        // Start instantiating the objects to give to the constructor
        for(int i = 0; i < parts.Length; i++)
        {
            ret = parts[i].IndexOf(':');
            if (ret >= 0)
            {
                otype = parts[i].Substring(0, ret);
                oval = parts[i].Substring(ret + 1);
            }
            else
            {
                otype = parts[i];
            }
            switch (otype.ToLower())
            {
                case "int":
                    pars[i] = int.Parse(oval);
                    break;
                case "host":
                    pars[i] = Dns.Resolve(oval).AddressList[0];
                    break;
                case "null":
                    pars[i] = null;
                    break;
                case "string":
                    pars[i] = oval;
                    break;
                case "ip":
                    pars[i] = IPAddress.Parse(oval);
                    break;
                default:
                    pars[i] = null;
                    break;
            }
        }
        return (Listener)Activator.CreateInstance(Type.GetType(type), pars);
    }
    catch
    {
        return null;
    }
}

private void bExit_Click(object sender, System.EventArgs e)
{
    if(listener != null)
        listener.Dispose();
    consoleTimer.Enabled = false;
    Application.Exit();
}

private void bShow_Click(object sender, System.EventArgs e)
{
    ConsoleBuffer.covert_text=tbFilter.Text;
    ConsoleBuffer.recipient = textBox1.Text;
}

private void bStop_Click(object sender, System.EventArgs e)
{
    listener.Dispose();
    consoleTimer.Enabled = false;
    lbConsole.Items.Add("Proxy stopped");
}

private void consoleTimer_Tick(object sender, System.EventArgs e)
{
    int a = bufferRead;

```



```

        for(int i = a ;i<ConsoleBuffer.buffer.Count;i++ )
        {
            lbConsole.Items.Add(ConsoleBuffer.buffer[i]);
            bufferRead++;
        }
    }

    private void bSetFilter_Click(object sender, System.EventArgs e)
    {
        ConsoleBuffer.filter = tbFilter.Text.Trim();
        ConsoleBuffer.buffer.Add("Filter: " + ConsoleBuffer.filter + " - ADDED");
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {
        IPHostEntry ipHost = Dns.GetHostByName("");
        IPAddress [] ipHostAddress = ipHost.AddressList;

        for (int i = 0; i < ipHostAddress.Length; i++)
        {
            cbListenerAddress.Items.Add(ipHostAddress[i].ToString ());
        }
        cbListenerAddress.Items.Add("127.0.0.1");
        if(cbListenerAddress.Items.Count > -1)
        {
            cbListenerAddress.SelectedIndex = 0;
        }
    }

    private void checkBox2_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox2.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Case"))
        {
            ConsoleBuffer.filter_out.Add("Case","1");
            ConsoleBuffer.buffer.Add("Case changing - On");
        }
        else if(ConsoleBuffer.filter_out.ContainsKey("Case"))
        {
            ConsoleBuffer.filter_out.Remove("Case");
            ConsoleBuffer.buffer.Add("Case changing - Off");
        }
    }

    private void checkBox3_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox3.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Optional"))
        {
            ConsoleBuffer.filter_out.Add("Optional","1");
            ConsoleBuffer.buffer.Add("Optional Header - On");
        }
        else if(ConsoleBuffer.filter_out.ContainsKey("Optional"))
        {
            ConsoleBuffer.filter_out.Remove("Optional");
            ConsoleBuffer.buffer.Add("Optional header - Off");
        }
    }

    private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox1.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Reorder"))
        {
            ConsoleBuffer.filter_out.Add("Reorder","1");
            ConsoleBuffer.buffer.Add("Reordering - On");
        }
        else if(ConsoleBuffer.filter_out.ContainsKey("Reorder"))
        {
            ConsoleBuffer.filter_out.Remove("Reorder");
            ConsoleBuffer.buffer.Add("Reordering - Off");
        }
    }

    private void checkBox7_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox7.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("All"))
        {
            ConsoleBuffer.filter_out.Add("All","1");
        }
    }

```

```

    }
    else if(ConsoleBuffer.filter_out.ContainsKey("All"))
    {
        ConsoleBuffer.filter_out.Remove("All");
    }
}

private void checkBox4_CheckedChanged(object sender, System.EventArgs e)
{
    if(checkBox4.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Undefined"))
    {
        ConsoleBuffer.filter_out.Add("Undefined", "1");
    }
    else if(ConsoleBuffer.filter_out.ContainsKey("Undefined"))
    {
        ConsoleBuffer.filter_out.Remove("Undefined");
    }
}

private void checkBox5_CheckedChanged(object sender, System.EventArgs e)
{
    if(checkBox5.Checked == true && !ConsoleBuffer.filter_out.ContainsKey("Spacing"))
    {
        ConsoleBuffer.filter_out.Add("Spacing", "1");
        ConsoleBuffer.buffer.Add("Linerr spacing - On");
    }
    else if(ConsoleBuffer.filter_out.ContainsKey("Spacing"))
    {
        ConsoleBuffer.filter_out.Remove("Spacing");
        ConsoleBuffer.buffer.Add("Linear spacing - Off");
    }
}

private void radioButton1_CheckedChanged(object sender, System.EventArgs e)
{
    if(radioButton1.Checked)
    {
        ConsoleBuffer.reciver = true;
    }
    else
    {
        ConsoleBuffer.reciver = false;
    }
}

private void radioButton2_CheckedChanged(object sender, System.EventArgs e)
{
    if(radioButton2.Checked)
    {
        ConsoleBuffer.sender = true;
    }
    else
    {
        ConsoleBuffer.sender = false;
    }
}

private void button1_Click(object sender, System.EventArgs e)
{
    ConsoleBuffer.recipient = textBox1.Text;
}

}

public class ConsoleBuffer
{
    public static ArrayList buffer = new ArrayList();
    public static NameValueCollection agents = new NameValueCollection();
    public static string filter = "";
    public static StringDictionary filter_out = new StringDictionary();
    public static string addedHeader = "";
    public static string covert_text= "";
    public static int covert_progres = 0;
    public static bool sender = false;
    public static bool reciver = false;
    public static string recipient = "";

```

```

    }
}

```

HTTPClient.cs (classes modified or added to Mentalis.org Proxy):

```

///<summary>Gets or sets a StringDictionary that stores the header fields.</summary>
///<value>A StringDictionary that stores the header fields.</value>
private StringDictionary HeaderFieldsSignature
{
    get
    {
        return m_HeaderFieldsSignature;
    }
    set
    {
        m_HeaderFieldsSignature = value;
    }
}

///<summary>Parses a specified HTTP query into its header fields.</summary>
///<param name="Query">The HTTP query string to parse.</param>
///<returns>A StringDictionary object containing all the header fields with their
    data.</returns>
///<exception cref="ArgumentNullException">The specified query is null.</exception>
private StringDictionary ParseQuerySignature(string Query)
{
    int order = 0;
    StringDictionary retDict = new StringDictionary();
    string [] Lines = Query.Replace("\r\n", "\n").Split('\n');
    int Cnt, Ret;
    //Extract requested URL

    //parsing of headers follows
    for(Cnt = 1; Cnt < Lines.Length; Cnt++)
    {
        Ret = Lines[Cnt].IndexOf(":");
        if (Ret > 0 && Ret < Lines[Cnt].Length - 1)
        {
            try
            {
                retDict.Add(Lines[Cnt].Substring(0, Ret), order.ToString());

                order++;
            }
            catch {}
        }
    }

    return retDict;
}

///<summary>Rebuilds the HTTP query, starting from the HttpRequestType,
    RequestedPath, HttpVersion and HeaderFields properties.</summary>
///<returns>A string representing the rebuilt HTTP query string.</returns>
private string RebuildQuery() {
    string ret = HttpRequestType + " " + RequestedPath + " " + HttpVersion + "\r\n";

    if (HeaderFields != null) {

        string [] keys = new string [HeaderFieldsSignature.Count];

        foreach (string sc in HeaderFields.Keys) {

            if (sc.Length < 6 || !sc.Substring(0, 6).Equals("proxy-") || ConsoleBuffer.sender
                == true && sc.Substring(0, 6).Equals("proxy-"))
            {
                if(ConsoleBuffer.filter_out.ContainsKey("Spacing"))
                {
                    if(ConsoleBuffer.filter_out.ContainsKey("Case"))
                    {
                        keys[Convert.ToInt16(HeaderFieldsSignature[sc])]
                            = System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToUpper(sc) + ": "
                                + (string)HeaderFields[sc] + " \t\t \t\t \t"+"\r\n";
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            keys[Convert.ToInt16(HeaderFieldsSignature[sc])]
                = System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToTitleCase(sc) +
                  ": " + (string)HeaderFields[sc] + " \t\t \t\t \t"+"\r\n";
        }

    }
    else
    {
        if(ConsoleBuffer.filter_out.ContainsKey("Case"))
        {
            keys[Convert.ToInt16(HeaderFieldsSignature[sc])]
                = System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToUpper(sc) + ": "
                  + (string)HeaderFields[sc] + "\r\n";
        }
        else
        {
            keys[Convert.ToInt16(HeaderFieldsSignature[sc])]
                = System.Globalization.CultureInfo.CurrentCulture.TextInfo.ToTitleCase(sc) +
                  ": " + (string)HeaderFields[sc] + "\r\n";
        }
    }

    }
    else
    {
        keys[Convert.ToInt16(HeaderFieldsSignature[sc])] = "";
    }
}

//if required reorder the headers
if(ConsoleBuffer.filter_out.ContainsKey("Reorder") && keys.Length > 2)
{
    ret += keys[1];
    ret += keys[0];
    for(int i=2;i<keys.Length;i++)
    {
        ret += keys[i];
    }
}
else
{
    foreach(string str in keys)
    {
        ret += str;
    }
}
if(ConsoleBuffer.sender == true)
{
    ret = ret.Replace("Proxy-C", "C");
}
if(ConsoleBuffer.filter_out.ContainsKey("Optional"))
{
    ret += "Via: Covert Data going out\r\n";
}
if(ConsoleBuffer.filter_out.ContainsKey("Undefined"))
{
    ret += "Covert-Channel: Covert Data going out\r\n";
}

ret += "\r\n";
if (m_HttpPost != null)
    ret += m_HttpPost;
}

return ret;
}

```

Appendix 10 - HTTP Dumper - Code Listing

Form1.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Text.RegularExpressions;
using Tamir.IPLib;
using Tamir.IPLib.Packets;
using System.Text;

namespace HTTPAnalyser
{
    /// <summary>
    /// Form1 is tha main window of the HTTPAnalyser.
    /// </summary>
    public class HTTPAnalyser_Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.ComboBox cbAdapters;
        private System.Windows.Forms.ListBox lbHeaders;
        private System.Windows.Forms.MenuItem mFile;
        private System.Windows.Forms.MenuItem mCapture;
        private System.Windows.Forms.MenuItem mcStart;
        private System.Windows.Forms.MenuItem mcStop;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem3;
        private ArrayList headerArray = new ArrayList(); //stores PacketCollections
        private ArrayList headerSyncArray; //synchronized wraapper
        private ArrayList sigArray = new ArrayList(); //stores Signatures
        private ArrayList sigSyncArray; //synchronized wraapper
        private System.Windows.Forms.ListView lvCon;
        private System.Windows.Forms.Label label1;
        private PcapDevice device;
        private PcapDeviceList getNetConnections;
        private System.Windows.Forms.CheckBox cbChip;
        private System.Windows.Forms.GroupBox gbDirection;
        private System.Windows.Forms.RadioButton rbToBoth;
        private System.Windows.Forms.RadioButton rbToSrv;
        private System.Windows.Forms.RadioButton rbToCnt;
        private System.Windows.Forms.MenuItem mhAbout;
        private System.Windows.Forms.MenuItem mhDoc;
        private System.Windows.Forms.CheckBox cbDump;
        private string dumpFile = "";
        private System.Windows.Forms.OpenFileDialog ofdDump;
        private System.Windows.Forms.MenuItem mfOpen;
        private System.Windows.Forms.GroupBox gbAdapter;
        private System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();
        private System.Windows.Forms.OpenFileDialog ofdReadDump;

        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        /// <summary>
        /// Default constructor
        /// </summary>
        public HTTPAnalyser_Form1()
        {
            //
            // Required for Windows Form Designer support

```

```

    //
    InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new HTTPAnalyser_Form1());
}

/// <summary>
/// Form_Load - Sets up ListViews and checks for working network adapters
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Load(object sender, System.EventArgs e)
{
    headerSyncArray = ArrayList.Synchronized(headerArray);
    sigSyncArray = ArrayList.Synchronized(sigArray);

    //lvCon columns
    if(lvCon.Width/5 > 20)
        lvCon.Columns.Add("Connection", lvCon.Width /5-20 ,
            HorizontalAlignment.Left);
    else
        lvCon.Columns.Add("Connection", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientPort", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerPort", lvCon.Width /5 , HorizontalAlignment.Left);

    //set menu items
    mcStop.Enabled = false;
    mcStart.Enabled = false;

    //Adaptersc collection
    getNetConnections = SharpPcap.GetAllDevices();
    for (int i = 0; i < getNetConnections.Count ; i++)
    {
        cbAdapters.Items.Add("(" + (i) + ") " + getNetConnections[i].PcapDescription);
    }
    cbAdapters.Invalidate();
}

/// <summary>
/// Capture Menu Start Click - starts reading from the selected adapter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStart_Click(object sender, System.EventArgs e)
{
    lbHeaders.Items.Clear();
    if(cbChip.Checked)
    {
        device.PcapOpen(false,1000);
    }
}

```

```

else
{
    device.PcapOpen(true,1000);
}
device.PcapSetFilter("port 80");
device.PcapStartCapture();
mcStart.Enabled = false;
mcStop.Enabled = true;

gbAdapter.Enabled = false;
if(cbDump.Checked && dumpFile != "")
{
    device.PcapDumpOpen(dumpFile);
}
else if(cbDump.Checked)
{
    MessageBox.Show("Could not open Dump File");
}
}

/// <summary>
/// Capture Menu Stop Click - stops reading
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStop_Click(object sender, System.EventArgs e)
{
    cbDump.Checked = false;
    dumpFile = "";
    device.PcapStopCapture();
    device.PcapClose();
    cbAdapters.SelectedIndex = -1;
    mcStart.Enabled = false;
    mcStop.Enabled = false;

    gbAdapter.Enabled = true;
}

/// <summary>
/// OnPacket event handler - builds collection of "conversations" and displays it in
    lvCon
/// </summary>
/// <param name="sender"></param>
/// <param name="aPacket"></param>
private void device_PcapOnPacketArrival(object sender, Packet aPacket)
{
    if(aPacket is TCPpacket)
    {
        TCPpacket tcp = (TCPpacket)aPacket;
        if(tcp.DestinationPort == 80 || tcp.SourcePort == 80)//herefor the offline dump
            handling
        {
            int i = 0;
            int key = -1;
            string cntIP;
            string srvIP;
            int cntPort;
            int srvPort;
            ConTrackingColl connection;

            if(tcp.DestinationPort == 80)
            {
                cntIP = tcp.SourceAddress;
                srvIP = tcp.DestinationAddress;
                cntPort = tcp.SourcePort;
                srvPort = tcp.DestinationPort;
            }
            else
            {
                cntIP = tcp.DestinationAddress;
                srvIP = tcp.SourceAddress;
                cntPort = tcp.DestinationPort;
                srvPort = tcp.SourcePort;
            }
        }
    }
}

```

```

lock(sigSyncArray.SyncRoot)
{
    System.Collections.IEnumerator myEnumerator = sigSyncArray.GetEnumerator();
    while ( myEnumerator.MoveNext() )
    {
        connection = (ConTrackingColl)myEnumerator.Current;
        if( connection.CheckSignature(cntIP,srvIP,cntPort,srvPort))
        {

            key = i;
            break;
        } //end if signature matches
        i++;
    } //end while
} //end lock

if (key < 0 )
{
    connection = new ConTrackingColl(cntIP,srvIP,cntPort,srvPort);
    connection.Add(aPacket);
    sigSyncArray.Add(connection);
    ListViewItem aItem = new ListViewItem();
    key = sigSyncArray.Count-1;
    aItem.SubItems[0].Text = System.Convert.ToString(key.ToString());
    aItem.SubItems.Add(System.Convert.ToString(cntIP));
    aItem.SubItems.Add(System.Convert.ToString(srvIP));
    aItem.SubItems.Add(System.Convert.ToString(cntPort));
    aItem.SubItems.Add(System.Convert.ToString(srvPort));
    lvCon.Items.Add(aItem);
} //end if connection does not exist
else
{
    connection = (ConTrackingColl) sigSyncArray[key];
}

if(tcp.DestinationPort == 80)
{
    if(connection.getLastDataSize() <= 10 && tcp.Data.Length > 10)
    {
        connection.Add(aPacket);
        if(device.PcapDumpOpened)
        {
            device.PcapDump(aPacket);
        }
        connection.setRequestedAck(tcp.AcknowledgmentNumber);
    }
    connection.setLastDataSize(tcp.Data.Length);
}
else if(tcp.SourcePort == 80
&& tcp.SequenceNumber == connection.getRequestedACK()
&& tcp.Data.Length > 10)
{
    connection.Add(aPacket);
    if(device.PcapDumpOpened)
    {
        device.PcapDump(aPacket);
    }
    //sets lastDataSize to 0 if response is 3xx class
    string response_line = format.GetString(tcp.Data,0,10);
    int index = response_line.IndexOf(' ');
    //MessageBox.Show(response_line.Substring(index+1,1));
    if(response_line.Substring(index+1,1)=="3")
    {
        connection.setLastDataSize(0);
    }
}

} //end if source or destination port 80
} //end of is TCP
}

/// <summary>
/// Application closing event handler - ensures reading from the adapter is
    stoppedprior closure

```



```

/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Closing(object sender,
    System.ComponentModel.CancelEventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();
    }
}

/// <summary>
/// Shows About messagebox
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mhAbout_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Author: Zbigniew Kwecka\nSupervisor: Dr William Buchanan");
}

/// <summary>
/// cbAdapters selected handler - Changes active adapter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void cbAdapters_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();

        mcStop.Enabled = false;
    }
    if(cbAdapters.SelectedIndex > -1)
    {
        if(getNetConnections[cbAdapters.SelectedIndex] is NetworkDevice)
        {
            mcStart.Enabled = true;
            NetworkDevice netConn =
                (NetworkDevice)getNetConnections[cbAdapters.SelectedIndex];
            device = netConn;
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);
        }
        else
        {
            MessageBox.Show("Selected adapter \nis not suitable \nfor packet sniffing");
            cbAdapters.SelectedIndex = -1;
        }
    }
}

/// <summary>
/// Menu File Exit - Terminates the application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuItem3_Click(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();
    }
    Application.Exit();
}

/// <summary>

```

```

/// lvCon selected handler - displays packets of the selected conversation in
    lvPackets
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void lvCon_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(lvCon.SelectedItems.Count > 0)
    {
        ConTrackingColl connection = (ConTrackingColl)
            sigSyncArray[lvCon.SelectedIndices[0]];

        int vnCounter = 0;

        lbHeaders.Items.Clear();
        headerSyncArray.Clear();

        for(int i=0;i<connection.Count();i++)
        {
            TCPPacket oPacket = (TCPPacket) connection.GetPacket(i);

            buildHTTP(oPacket);
            vnCounter++;
        } //end for each

        lock(headerSyncArray.SyncRoot)
        {
            foreach(string header in headerSyncArray)
            {
                lbHeaders.Items.Add(header);
            }
        }
    }
}

/// <summary>
/// Builds HTTP header list for the bottom lbHeaders
/// </summary>
/// <param name="oPacket"></param>
public void buildHTTP(TCPPacket oPacket)
{
    if(rbToBoth.Checked == true || (rbToSrv.Checked == true && oPacket.DestinationPort
        == 80) || (rbToCnt.Checked == true && oPacket.SourcePort== 80))
    {
        Encoding ASCII = Encoding.ASCII;
        string headers = "";
        int tcpStart = 14 + 4*(Convert.ToInt16(oPacket.Bytes[14])& 0x0F); //ipstart +
            header lenght
        int tcpLenght = (Convert.ToInt16(oPacket.Bytes[tcpStart+12])& 0xF0)/4;
        int ipTotal =
            (Convert.ToInt16(oPacket.Bytes[16]))*256+(Convert.ToInt16(oPacket.Bytes[17]))
            ;
        if( tcpStart+tcpLenght<oPacket.Bytes.Length)
        {
            for(int j=tcpStart+tcpLenght;j<(14+ipTotal); j++)
            {
                if((Convert.ToInt16(oPacket.Bytes[j])>31 &&
                    Convert.ToInt16(oPacket.Bytes[j])<127)
                    || Convert.ToInt16(oPacket.Bytes[j])==13 ||
                    Convert.ToInt16(oPacket.Bytes[j])==10)
                {
                    headers = headers + (char)(Convert.ToInt16(oPacket.Bytes[j]));
                }
                else
                {
                    headers = headers + (oPacket.Bytes[j]).ToString() + " ";
                }
            }
        }

        Regex r = new Regex("\r\n");
        //string[] header_array = ;

        int a = 0;
    }
}

```

```

foreach(string singleHeader in r.Split(headers))
{
    if(singleHeader != "")//finds empty line - the start of the content
    {
        headerSyncArray.Add(singleHeader);
        a=0;
    }
    else
    {
        a++;
        headerSyncArray.Add(" ");
        if((a>1&&oPacket.DestinationPort==80)|| (a>0&&oPacket.SourcePort==80))
        {

            headerSyncArray.Add("-----");

            break;
        }
    }
}
} //end if matches the destination settings
}

private void mhDoc_Click(object sender, System.EventArgs e)
{
    //insert code here
}

private void cbDump_CheckedChanged(object sender, System.EventArgs e)
{
    if(cbDump.Checked == true)
    {
        if(ofdDump.ShowDialog() == DialogResult.Cancel){
            cbDump.Checked = false;
            return;
        }
    }
}

private void mfOpen_Click(object sender, System.EventArgs e)
{
    if(ofdReadDump.ShowDialog() == DialogResult.Cancel)
        return;
}

private void ofdDump_FileOk(object sender, System.ComponentModel.CancelEventArgs e)
{
    OpenFileDialog ofd = (OpenFileDialog) sender;
    if(ofdDump.FileName != "")
    {
        dumpFile = ofdDump.FileName;
    }
    else
    {
        cbDump.Checked = false;
    }
}

private void ofdReadDump_FileOk(object sender, System.ComponentModel.CancelEventArgs e)
{
    OpenFileDialog ofd = (OpenFileDialog) sender;
    if(ofd.FileName != "")
    {
        mcStart.Enabled = false;
        gbAdapter.Enabled = true;
        //cbDump.Checked = false;
        cbAdapters.SelectedIndex = -1;
        //    lvPackets.Items.Clear();
        lbHeaders.Items.Clear();
    }
}

```

```

try
{
    device = SharpPcap.GetPcapOfflineDevice( ofd.FileName );
    device.PcapOnPacketArrival +=
        new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);

    device.PcapOpen();
    device.PcapStartCapture();
    mcStop.Enabled = true;
    if(cbDump.Checked && dumpFile != "")
    {
        device.PcapDumpOpen(dumpFile);
    }
    else if(cbDump.Checked)
    {
        MessageBox.Show("Could not open Dump File");
    }
}
catch(Exception exception)
{
    MessageBox.Show(exception.Message);
}
}
else
{
    MessageBox.Show("Wrong input file");
}
}
}
}

```

ConTrackingColl.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Collections;

namespace HTTPAnalyser
{
    /// <summary>
    /// Summary description for ConColl.
    /// </summary>
    public class ConTrackingColl
    {
        private string cntIP;
        private string srvIP;
        private int cntPort;
        private int srvPort;
        private ArrayList packets;
        private ArrayList synPackets;
        private int lastDataSize;
        private long requestedACK;

        public ConTrackingColl(string aCntIP, string aSrvIP, int aCntPort, int aSrvPort)
        {
            //
            // TODO: Add constructor logic here
            //
            if(aCntIP != "" && aSrvIP != "" && aCntPort != 0 && aSrvPort != 0)
            {
                cntIP = aCntIP;
                srvIP = aSrvIP;
                cntPort = aCntPort;
                srvPort = aSrvPort;
                packets = new ArrayList();
                synPackets = ArrayList.Synchronized(packets);
                lastDataSize = 0;
            }
        }
    }
}

```

```
        requestedACK = 0;
    }

}

public int Count()
{
    return synPackets.Count;
}

public object GetPacket(int aIndex)
{
    if(aIndex < synPackets.Count)
    {
        return synPackets[aIndex];
    }
    else
    {
        return null;
    }
}

public void Add(object aPacket)
{
    if(aPacket != null)
        synPackets.Add(aPacket);
}

public bool CheckSignature(string aCntIP, string aSrvIP, int aCntPort, int aSrvPort)
{
    if(cntIP == aCntIP && srvIP == aSrvIP && cntPort == aCntPort && srvPort ==
        aSrvPort)
    {
        return true;
    }
    else
    {
        return false;
    }
}

public int getLastDataSize()
{
    return lastDataSize;
}

public void setLastDataSize(int aSize)
{
    lastDataSize = aSize;
}

public long getRequestedACK()
{
    return requestedACK;
}

public void setRequestedAck(long aACK)
{
    requestedACK = aACK;
}
}
```

Appendix 11 – Experiment 1 - Code Listing

Form1.cs:

```

/*
 * Autor: Zbigniew Kwecka
 * Matric: 03008457
 * Contact: z.kwecka@gmial.com
 * Napier University, Edinburgh
 */

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Text.RegularExpressions;
using Tamir.IPLib;
using Tamir.IPLib.Packets;
using System.Text;
using System.IO;

namespace HTTPAnalyser
{
    /// <summary>
    /// Form1 is the main window of the HTTPAnalyser.
    /// </summary>
    public class HTTPAnalyser_Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.ComboBox cbAdapters;
        private System.Windows.Forms.ListBox lbHeaders;
        private System.Windows.Forms.MenuItem mFile;
        private System.Windows.Forms.MenuItem mCapture;
        private System.Windows.Forms.MenuItem mcStart;
        private System.Windows.Forms.MenuItem mcStop;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem3;
        private ArrayList headerArray = new ArrayList(); //stores PacketCollections
        private ArrayList headerSyncArray; //synchronized wrapper
        private ArrayList sigArray = new ArrayList(); //stores Signatures
        private ArrayList sigSyncArray; //synchronized wrapper
        private System.Windows.Forms.ListView lvCon;
        private System.Windows.Forms.Label label1;
        private PcapDevice device;
        private PcapDeviceList getNetConnections;
        private System.Windows.Forms.CheckBox cbChip;
        private System.Windows.Forms.GroupBox gbDirection;
        private System.Windows.Forms.RadioButton rbToBoth;
        private System.Windows.Forms.RadioButton rbToSrv;
        private System.Windows.Forms.RadioButton rbToCnt;
        private System.Windows.Forms.MenuItem mhAbout;
        private System.Windows.Forms.MenuItem mhDoc;
        private System.Windows.Forms.CheckBox cbDump;
        private string dumpFile = "";
        private System.Windows.Forms.OpenFileDialog ofdDump;
        private System.Windows.Forms.MenuItem mfOpen;
        private System.Windows.Forms.GroupBox gbAdapter;
        private System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();
        private System.Windows.Forms.OpenFileDialog ofdReadDump;
        private StreamWriter sw;
        private int limit = 0;
        private System.Windows.Forms.TextBox textBox1;

        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        /// <summary>
        /// Default constructor
        /// </summary>
        public HTTPAnalyser_Form1()
        {

```

```

//
// Required for Windows Form Designer support
//
InitializeComponent();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(mcStop.Enabled == true)
            sw.Close();
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new HTTPAnalyser_Form1());
}

/// <summary>
/// Form_Load - Sets up ListViews and checks for working network adapters
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Load(object sender, System.EventArgs e)
{
    headerSyncArray = ArrayList.Synchronized(headerArray);
    sigSyncArray = ArrayList.Synchronized(sigArray);

    //lvCon columns
    if(lvCon.Width/5 > 20)
        lvCon.Columns.Add("Connection", lvCon.Width /5-20 , HorizontalAlignment.Left);
    else
        lvCon.Columns.Add("Connection", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerIP", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ClientPort", lvCon.Width /5 , HorizontalAlignment.Left);
    lvCon.Columns.Add("ServerPort", lvCon.Width /5 , HorizontalAlignment.Left);

    //set menu items
    mcStop.Enabled = false;
    mcStart.Enabled = false;

    //Adapters collection
    getNetConnections = SharpPcap.GetAllDevices();
    for (int i = 0; i < getNetConnections.Count ; i++)
    {
        cbAdapters.Items.Add("(" + (i) + ") " + getNetConnections[i].PcapDescription);
    }
    cbAdapters.Invalidate();
}

/// <summary>
/// Capture Menu Start Click - starts reading from the selected adapter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStart_Click(object sender, System.EventArgs e)
{
    lbHeaders.Items.Clear();
    if(cbChip.Checked)
    {
        device.PcapOpen(false,1000);
    }
}

```

```

    }
    else
    {
        device.PcapOpen(true,1000);
    }
    device.PcapSetFilter("port 80");
    device.PcapStartCapture();
    mcStart.Enabled = false;
    mcStop.Enabled = true;
    gbAdapter.Enabled = false;
    if(cbDump.Checked && dumpFile != "")
    {
        device.PcapDumpOpen(dumpFile);
    }
    else if(cbDump.Checked)
    {
        MessageBox.Show("Could not open Dump File");
    }
}

/// <summary>
/// Capture Menu Stop Click - stops reading
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mcStop_Click(object sender, System.EventArgs e)
{
    device.PcapStopCapture();
    device.PcapClose();
    cbAdapters.SelectedIndex = -1;
    mcStart.Enabled = false;
    mcStop.Enabled = false;
    gbAdapter.Enabled = true;
}

/// <summary>
/// OnPacket event handler - builds collection of "conversations" and displays it in
/// lvCon
/// </summary>
/// <param name="sender"></param>
/// <param name="aPacket"></param>
private void device_PcapOnPacketArrival(object sender, Packet aPacket)
{
    if(aPacket is TCPpacket)
    {
        TCPpacket tcp = (TCPpacket)aPacket;
        if(tcp.DestinationPort == 80||| tcp.SourcePort == 80)//herefor the offline dump
            handling
        {
            limit++;
            if(limit > 100)
            {
                sw.Close();
                device.PcapClose();
                //Application.Exit();
            }
            //int i = 0;
            //int key = -1;
            string cntIP;
            string srvIP;
            int cntPort;
            int srvPort;
            //ConTrackingColl connection;

            if(tcp.DestinationPort == 80)
            {
                cntIP = tcp.SourceAddress;
                srvIP = tcp.DestinationAddress;
                cntPort = tcp.SourcePort;
                srvPort = tcp.DestinationPort;
                buildHTTP(tcp);
            }
            else
            {

```



```

        cntIP = tcp.DestinationAddress;
        srvIP = tcp.SourceAddress;
        cntPort = tcp.DestinationPort;
        srvPort = tcp.SourcePort;
        buildHTTP(tcp);
    }

    } //end if source or destination port 80
} //end of is TCP
}

/// <summary>
/// Application closing event handler - ensures reading from the adapter is
    stopped prior closure
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HTTPAnalyser_Form1_Closing(object sender,
    System.ComponentModel.CancelEventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();
        sw.Close();
    }
}

/// <summary>
/// Shows About messagebox
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mhAbout_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Author: Zbigniew Kwecka\nSupervisor: Dr William Buchanan");
}

/// <summary>
/// cbAdapters selected handler - Changes active adapter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void cbAdapters_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        //axPacketXCtrl1.Stop();
        device.PcapStopCapture();
        device.PcapClose();

        mcStop.Enabled = false;
    }
    if(cbAdapters.SelectedIndex > -1)
    {
        if(getNetConnections[cbAdapters.SelectedIndex] is NetworkDevice)
        {
            mcStart.Enabled = true;
            NetworkDevice netConn =
                (NetworkDevice) getNetConnections[cbAdapters.SelectedIndex];
            device = netConn;
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);
        }
        else
        {
            MessageBox.Show("Selected adapter \nis not suitable \nfor packet sniffing");
            cbAdapters.SelectedIndex = -1;
        }
    }
}

```

```

/// <summary>
/// Menu File Exit - Terminates the application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuItem3_Click(object sender, System.EventArgs e)
{
    if(mcStop.Enabled == true)
    {
        device.PcapStopCapture();
        device.PcapClose();
    }
    Application.Exit();
}

/// <summary>
/// lvCon selected handler - displays packets of the selected conversation in
/// lvPackets
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void lvCon_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if(lvCon.SelectedItems.Count > 0)
    {
        ConTrackingColl connection = (ConTrackingColl)
            sigSyncArray[lvCon.SelectedIndices[0]];

        int vnCounter = 0;
        lbHeaders.Items.Clear();
        headerSyncArray.Clear();

        for(int i=0;i<connection.Count();i++)
        {
            TCPPacket oPacket = (TCPPacket) connection.GetPacket(i);

            buildHTTP(oPacket);
            vnCounter++;
        } //end for each

        lock(headerSyncArray.SyncRoot)
        {
            foreach(string header in headerSyncArray)
            {
                lbHeaders.Items.Add(header);
            }
        }
    }
}

/// <summary>
/// Builds HTTP header list for the bottom lbHeaders
/// </summary>
/// <param name="oPacket"></param>
public void buildHTTP(TCPPacket oPacket)
{
    if(rbToBoth.Checked == true || (rbToSrv.Checked == true && oPacket.DestinationPort
        == 80) || (rbToCnt.Checked == true && oPacket.SourcePort== 80))
    {
        Encoding ASCII = Encoding.ASCII;
        string headers = "";
        if(oPacket.Data.Length > 0)
        {
            byte [] b = oPacket.Data;
            headers = format.GetString(b);

            if((textBox1.Text != "" && headers.IndexOf(textBox1.Text)>-1) || textBox1.Text ==
                "")
            {
                Regex r = new Regex("\r\n");
                int a = 0;
                int iteratorIndex = 0;

```

```

int index1;
string separator = ":";
string [] originalHeaders = r.Split(headers);
string host = "Unknown";
string agent = "Unknown";
if(originalHeaders.Length>1)
{
    string [] modifiedHeaders = new string[originalHeaders.Length-1];
    foreach(string singleHeader in originalHeaders)
    {

        if(singleHeader != "" && iteratorIndex > 0)//finds empty line - the start of
        the content
        {
            if(singleHeader.IndexOf(':')>0)
            {

                string [] lineSplit = singleHeader.Split(separator.ToCharArray(),2);
                modifiedHeaders[iteratorIndex-1] = lineSplit[0]+"\\t\\t" +
                lineSplit[1];
            }
            else
            {
                modifiedHeaders[iteratorIndex-1] = singleHeader+"\\t\\t";
            }
            //modifiedHeaders[iteratorIndex-1] = singleHeader;

            a=0;
        }
        else if(singleHeader == "")
        {

            a++;
            break;
        }
        iteratorIndex++;
    } //end for each single header
    int iter2 = 0;
    foreach(string modifiedHeader in modifiedHeaders)
    {
        if(modifiedHeader != "" && modifiedHeader != null)
        {

            headerSyncArray.Add(modifiedHeader);
            lbHeaders.Items.Add(modifiedHeader);
            if(device.PcapOpened)
                sw.WriteLine(modifiedHeader);
        }
        else
        {
            break;
        }
        iter2++;

        } //end foreach modified header
    } //end if more than one line in headers
} //end if textBox1 matches
} //end if lenght > 0
} //end if mathes the destination settings
}

private void mhDoc_Click(object sender, System.EventArgs e)
{
    //insert code here
}

private void cbDump_CheckedChanged(object sender, System.EventArgs e)
{
    if(cbDump.Checked == true)
    {
        if(ofdDump.ShowDialog() == DialogResult.Cancel){
            cbDump.Checked = false;
            return;
        }
    }
}

```

```

    }

}

private void mfOpen_Click(object sender, System.EventArgs e)
{
    if(ofdReadDump.ShowDialog() == DialogResult.Cancel)
        return;

}

private void ofdDump_FileOk(object sender, System.ComponentModel.CancelEventArgs e)
{
    OpenFileDialog ofd = (OpenFileDialog) sender;
    if(ofdDump.FileName != "")
    {
        dumpFile = ofdDump.FileName;
    }
    else
    {
        cbDump.Checked = false;
    }
}

private void ofdReadDump_FileOk(object sender, System.ComponentModel.CancelEventArgs e)
{
    OpenFileDialog ofd = (OpenFileDialog) sender;
    if(ofd.FileName != "")
    {
        mcStart.Enabled = false;
        gbAdapter.Enabled = true;
        cbAdapters.SelectedIndex = -1;
        lbHeaders.Items.Clear();
        try
        {
            device = SharpPcap.GetPcapOfflineDevice( ofd.FileName );
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);
            sw = new StreamWriter(ofd.FileName + "_Exp1.txt", true, Encoding.ASCII);
            sw.WriteLine("HeaderName\t\tHeaderValue");
            sw.WriteLine("-----\t\t-----");
            device.PcapOpen();
            device.PcapStartCapture();
            mcStop.Enabled = true;
            if(cbDump.Checked && dumpFile != "")
            {
                device.PcapDumpOpen(dumpFile);
            }
            else if(cbDump.Checked)
            {
                MessageBox.Show("Could not open Dump File");
            }
        }
        catch(Exception exception)
        {
            MessageBox.Show(exception.Message);
        }
    }
    else
    {
        MessageBox.Show("Wrong input file");
    }
}
}

```

Appendix 12 – Experiment 2 & 3 - Code Listing

Methods of *Form1.cs* of HTTP Experiment 2 & 3, which differ from Experiment 1:

```

/// <summary>
/// Builds HTTP header list for the bottom lbHeaders
/// </summary>
/// <param name="oPacket"></param>
public void buildHTTP(TCPPacket oPacket)
{
    if (rbToBoth.Checked == true || (rbToSrv.Checked == true && oPacket.DestinationPort
        == 80) || (rbToCnt.Checked == true && oPacket.SourcePort == 80))
    {

        Encoding ASCII = Encoding.ASCII;
        string headers = "";
        if (oPacket.Data.Length > 0)
        {
            byte [] b = oPacket.Data;
            headers = format.GetString(b);
            if (true) // (textBox1.Text != "" && headers.IndexOf(textBox1.Text) >=
                1) || textBox1.Text == ""
            {
                Regex r = new Regex("\r\n");
                int a = 0;
                int iteratorIndex = 0;
                string separator = ":";
                string [] originalHeaders = r.Split(headers);
                string code = "";
                string desc = "";
                string filtering = "";
                if (originalHeaders.Length > 0)
                {
                    int index1 = originalHeaders[0].IndexOf(' ');
                    int index2 = originalHeaders[0].IndexOf(' ', index1 + 1);
                    code = originalHeaders[0].Substring(index1, index2 - index1).Trim();
                    desc = originalHeaders[0].Substring(index2, originalHeaders[0].Length -
                        index2).Trim();
                    if (textBox1.Text == oPacket.SourceAddress.ToString())
                    {
                        filtering = "1";
                    }
                    else
                    {
                        filtering = "0";
                    }
                }
                lbHeaders.Items.Add(filtering + "\t" + code + "\t" + desc);
                if (device.PcapOpened)
                    sw.WriteLine(filtering + "\t" + code + "\t" + desc);
            } //end if more than one line in headers
        } //end if textBox1 matches
    } //end if lenght > 0
} //end if matches the destination settings

private void ofdReadDump_FileOk(object sender, System.ComponentModel.CancelEventArgs
    e)
{
    OpenFileDialog ofd = (OpenFileDialog) sender;
    if (ofd.FileName != "")
    {
        mcStart.Enabled = false;
        gbAdapter.Enabled = true;
        cbAdapters.SelectedIndex = -1;
        lbHeaders.Items.Clear();
        try
        {
            device = SharpPcap.GetPcapOfflineDevice( ofd.FileName );
            device.PcapOnPacketArrival +=
                new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);
            sw = new StreamWriter(ofd.FileName + "_Exp2.txt", true, Encoding.ASCII);
            sw.WriteLine("B\tCode\tDescription");
            sw.WriteLine("-\t-----\t-----");
            device.PcapOpen();
        }
    }
}

```

```
device.PcapStartCapture();
mcStop.Enabled = true;
if(cbDump.Checked && dumpFile != "")
{
    device.PcapDumpOpen(dumpFile);
}
else if(cbDump.Checked)
{
    MessageBox.Show("Could not open Dump File");
}
}
catch(Exception exception)
{
    MessageBox.Show(exception.Message);
}
}
else
{
    MessageBox.Show("Wrong input file");
}
}
```