University of Birmingham

School of Computer Science
Final year project

VPN over HTTP

Project Report

Author: Daniel Jones (1427970)

BSc Computer Science

supervised by
Dr Ian Batten

Submitted in conformity with the requirements
for the degree of Bsc Computer Science
School of Computer Science
University of Birmingham

VPN over HTTP

Daniel Jones

March 30, 2018

Abstract

Problem: VPN traffic is easy to block, and commonly blocked on free public networks.

Solution: HTTP traffic is rarely blocked, so encoding data into HTTP traffic is one possible way to bypass filtering and blocking on public networks.

Conclusion: It is possible to encode data in such a way that it is difficult to detect that this has been done.

All code that was developed can be found at:
`https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/dgj470`

Keywords: VPN, HTTP, Tunnelling, Obfuscation, Steganography

Acknowledgements

I would like to thank my supervisor, Dr Ian Batten for support and guidance throughout the project.

Additionally, I'd like to thank my housemates, friends and family for support throughout both this project and my degree as a whole.

Contents

1 Introduction

The aim of this project is to be able to tunnel data, and by extension operate a VPN over the Hypertext Transfer Protocol (HTTP). There are multiple reasons why this would want to be done, for example:

- To access services that are blocked by the current network
- To hide the fact that blocked services are being accessed
- To maintain privacy regarding services that are being accessed

This project enables connections to be made and transmit data only over the HTTP protocol, and provide all of the benefits highlighted above.

1.1 Paper Overview

1. Introduction
    - Summary of the aims the project intends to fulfill
    - Short overview of the project
2. Existing Work
    - Review of literature surrounding the project
3. Background
    - High level overview of concepts and designs essential for the project:
        - HTTP
        - TCP
        - VPN
        - Privacy
4. Specification
    - Project specification
5. Design
    - High level program architecture and design
    - Configurable options
    - Design choices
6. Implementation

- Detailed information about how each component functions
- High level overview about data flows

7. Testing
    - Testing methodologies
    - Testing strategy
8. Project Management
    - Explaination of how the project was managed
9. Discussion
    - An overview of about what was successful
    - A review of what was learned
10. Conclusion
    - Compare the project to the aims
    - Concludes whether or not the project was successful

2 Existing Work

In this section, I explore some related works, and discuss some literature surrounding Steganography, HTTP, DNS tunneling and detecting tunneled traffic.

There has been a lot of work done on the study of steganography, and using different protocols to hide data. The most work has been done on DNS, as it is often below the radar for firewalls and checking if data is being exfiltrated.

2.1 Image Steganography

Steganography is most commonly used for hiding data in unused or unimportant areas of data[1], and the most common form of data for this is images. This is because all of the colour data in an image is not required for a human to see it, and the human eye is very good at filtering out noise[1].

More advanced approaches to steganography can involve identifying redundant data in images[2] which can be better than changing the least significant bit in an image which can be detected by steganalysis[2].

Steganography that is hidden from computers and is hidden from people are quite different things, and can require quite different approaches. The real challenge is to hide data from both.[2].

2.2 DNS steganography

It is possible to hide data very easily in DNS requests, and this is called DNS Tunneling, and it is often used to get around firewalls and hide which websites are being accessed.[3] This paper highlights the point raised in the previous section, that it is very easy for a human to look at the data and see it's not normal, but non-trivial for a computer. The paper describes how the data is detected, and in doing so describes in depth how the data is encoded and tunneled. DNS tunneling as described in the aformentioned paper has a few advantages and disadvantages. The key advantage is that is can be used in locked down networks, as DNS traffic is often let out, but the main disadvantage is that data transfer is very slow with a lot of overhead.

2.3 HTTP protocol

The HTTP 1.1 Protocol[4] is a protocol that describes how data is sent to and from a client, and it describes many areas where data could be included, HTTP traffic can include: Images, HTML, CSS, Javascript, Binary files, and more.
All of which can be used to hide data.

The headers in HTTP are also a potential vector to hide data, as HTTP Headers are whitespace insensitive, and the order in which the headers are sent do not matter[4].

2.4 Detecting tunneled DNS traffic

There are a variety of ways to detect tunneled traffic, from entropy analysis to performing DNS requests[3]. Another way of performing the lookup is to do character frequency analysis[5]. Frequency analysis looks at the difference in frequency of letters in domain names/english words and in random data. It is similar to but not quite the same as entropy analysis, and it is also more effective[5].

3 Background

This section provides an overview of all of the technologies mentioned in the remainder of the project.

3.1 HTTP

HTTP (Hyper Text Transfer Protocol) is the most important protocol in this project and as such needs to be covered in detail. `HTTP/1.1` is defined in `RFC2616`[4] and all of the information about HTTP comes from that document unless otherwise specified. HTTP is a Request/Response protocol, which means the client Requests data from the server, and the server returns it.
An example request is as follows:

```
GET / HTTP/1.1
Host: localhost
User-Agent: curl/7.58.0
Accept: */*
```

In this request, there is first the request line, and then a series of headers. The headers are a set of key-value pairs which form a dictionary that tells the server extra information about the client.

An example response is as follows:

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.6.4
Date: Tue, 20 Mar 2018 14:25:01 GMT
Content-type: text/html
Content-Length: 10088
Last-Modified: Tue, 13 Mar 2018 15:27:09 GMT

<!doctype html>
<html>
<head>
...
```

In this request the response line is sent, along with some headers which provide important meta-data about the response itself, which is then included when the headers are finished.

3.1.1 HTTP Encoding types

HTTP has a variety of different encoding types and methods. These are specified in the `Transfer-Encoding` and `Content-Encoding`, both of which are used to perform compression and other encoding of data.

`Transfer-Encoding` allows for `Chunked` encoding[6], which splits the page or portions of data up into multiple sections, and sends the length for each section separately. This is done as then servers can send data as it is generated, rather than having to buffer the entire page.

3.2 TCP

TCP (Transfer Control Protocol) is defined in `RFC793`[7] and all of the information about TCP comes from there unless otherwise specified. TCP is a connection based protocol that allows for reliable data transfer between a server and a client.
TCP connections are a connection between two sockets, one active and one passive.
A passive socket is listening, and is typically a `server` whereas an active socket is connecting and is typically a `client`.

3.3 Steganography

Steganography is defined as[8]:

> `the art or practice of concealing a message, image, or file within another message, image, or file`

An example of this could be encoding data into an image, or into a HTTP stream.

3.4 Obfuscation

To obfuscate is defined as[9]:

> `to be evasive, unclear, or confusing`

Obfuscation is simply the act of being `evasive, unclear, or confusing`.

When applied to computers, this is similar to steganography and the two terms are often used interchangeably, however steganography is the art of concealing and disguising data, whereas obfuscation just makes data difficult to read and understand.

3.5 Encryption

> `Encryption is the process of converting data to an unrecognizable or 'encrypted' form. It is commonly used to protect sensitive information so that only authorized parties can view it[10].`

There are many types of encryption, AES and RSA are two major examples.

It's generally referred to as bad practice to '`roll your own crypto`'[11], which simply means that it is a bad idea to write a bespoke cryptographic algorithm, or to try and come up with your own cryptographic scheme.

3.6 Privacy

Privacy on the internet is often overlooked, however, it is vitally important[12]. Almost everything that is done on the internet is tracked by multiple parties: Internet service providers (ISPs), the websites visited, DNS servers, third party servers and many more. These parties could all be gathering information in order to profile and build a model to predict behavoir. This can then be used for highly targeted advertising which is not always desirable for the end user. An extreme example of this was a case in 2012 where a retailer inadvertently told a father his daughter was pregnant by sending her coupons for baby clothes and cribs.[13] This case was first reported in early 2012, and since then statistical models have become significantly more complex and intrusive. This is demonstrated by Google recently allowing you to mark Adverts as '`knowing too much`'[14].

3.7 Blocking of services

When you connect to Public Wi-Fi, often the provider will limit what content you have access to, often by blocking 'ports' or preventing a blacklist of websites.
HTTP is run over port 80, and this is often one of the only ports available. ISP's have also been known to block certain websites.

3.8 Data Tunnelling

Data Tunnelling is a term for transmitting data in a different form to how it is usually transmitted. A VPN is an example of this, as is an SSH (Secure SHell) tunnel.

3.9 Binary

Binary is how a computer represents data, and it is a series of 1's and 0's, each one is called a bit. All characters have a binary representation, and typically characters are made up of 8 bits. Some file formats may not have a concept of characters, and instead may utilise a stream, which means data is processed one bit at a time.

3.10 VPN

A VPN or Virtual Private Network which is defined in `RFC2764`[15], is a piece of software, split into a server and a client, where the server and the client (or clients) form a network which is both virtual and private.
For a network to be private, it needs to be encrypted so no one else can see the data, and for it to be virtual, it only has to exist inside computers, and not physically connected by cables.

In practical terms this means that data is tunnelled from one computer to another, generally by use of `tun` or `tap` devices. `IP` packets can be read from and written to a

`tun` device.
`Ethernet` frames can be read from and written to a `tap` device.
Which ever device type is used, the packets/frames are transmitted through an existing connection to a remote client or server, which will also read and write from a corresponding device.

A typical example of a VPN is OpenVPN, `https://openvpn.net/`.

4 Specification

4.1 Functional Requirements

1. The system must allow duplex communication with a remote server and local client
2. All communication between the user and the server must be encapsulated within the HTTP Protocol
3. The user must be able to browse websites
4. The user must be able to connect to a remote server, over SSH for example
5. The server and client must expose a TCP port which can be used for multiple applications
6. The server must act as a valid HTTP server serving a valid web page if navigated to
7. The server must act as a mirror to another HTTP server when serving HTTP to both the client and any other clients that may discover it
8. The system should not encrypt data traveling over it, data should be assumed to be already encrypted
9. The client should poll the server for data
10. When the client or server has data to send, the client should increase the frequency of requests
11. The system should be able to serve multiple clients at a time without interference

4.2 Non-functional Requirements

1. The extra data encapsulated within the HTTP protocol must be done in such a way that is looks like typical HTTP traffic and is not easy or obvious to spot.
2. The connection speed must be such that browsing the internet is possible without undue difficulty.
3. If the connection drops, the server and client must work to re-establish the connection

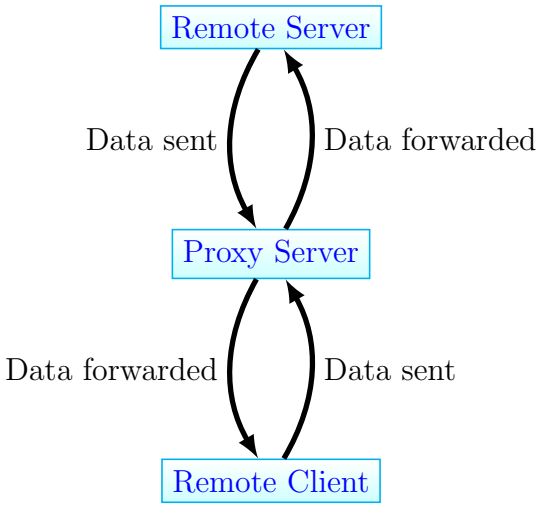
5 Design

This section sets out the High level program architecture and design, along with areas that are configurable and the different design choices that were made in the development of the project.

During the design of the project, efforts were made to keep the entire program modular, so all sections are reusable and replaceable.

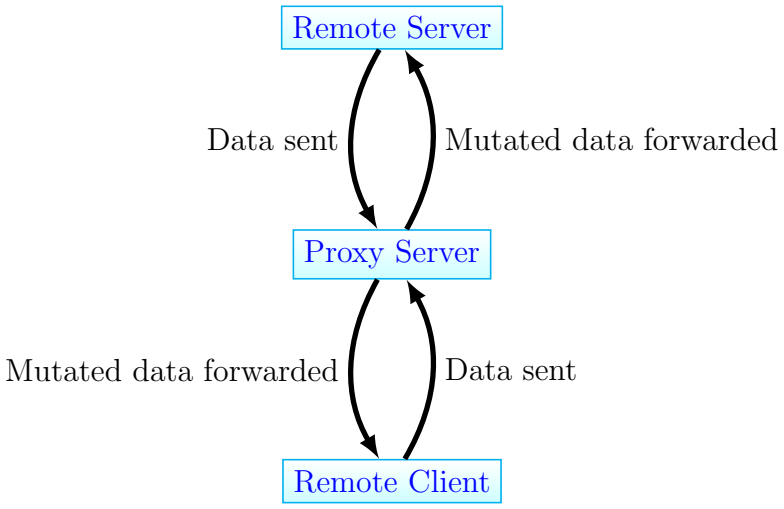
5.1 Server Overview

The server of the project is split into two parts.
The actual 'server' that runs, which acts as a manager for the two duplex connections that are active. The server (called the proxy) creates a TCP listening socket, and then when that socket is connected to it, it creates a connection to the remote server. In the most basic configuration, data is read from one socket and written directly to the other, acting as a TCP-Forwarder:



The server runs two threads, one from the client to the server, and one in the return direction.

One of the reasons I chose this design is because it allows traffic to be '`mutated`' on the fly:



The other reasons I chose this design are:

- It allows the lower level to deal with keeping the connection alive, and the upper levels can assume that if they are running, both of the TCP sockets are alive.
- It allows ease of customisation, because the mutators can be easily switched out.
- It allows the lover level to provide a 'data storage' to the upper threads, which is provided thread safe and reliably so they can access current data about the session.

5.1.1 Configurable Options

In the main 'proxy' part of the program, there are lots of options that are customisable. Here is an example configuration file:

```
config:
    name: localhost-http
    remote: localhost
    remote-port: 8000
    local-port: 80
    test-request: |+
        GET / HTTP/1.1
        Host: localhost
        User-Agent: Test-Agent


    test-response: HTTP/1.0 2\d\d.*
    test-response-length: 16
    proxy-mutator-location: mutators/http.py
    proxy-mutate-receive: receive
    proxy-mutate-send: send
```

The config file is in yaml format.

The '`name`' field is just a unique identifier.
The '`remote`' and '`remote-port`' specify the remote host and port to connect to.
The '`local-port`' specifies the local port to listen on.
The '`test-request`' provides an example request to send to the remote server to check it is active, and the first '`test-response-length`' bytes are read from the socket and compared to the regular expression defined in '`test-response`'.
'`proxy-mutator-location`' is the location of the file that stores the mutator code. '`proxy-mutate-receive`' and '`proxy-mutate-send`' are the names of the functions that are called when data is received from the remote server and needs mutating before sending to the client, and vice versa.

Because of all of this configuration testing connections, debugging, and attempting new things were made significantly simpler. It has also left significant scope for future expansion of the project.

5.2 Obfuscation Overview

Early on in the project, I created a '`generic`' obfuscation program.
The program reads a configuration file, called '`pages`', which sets out the structure the obfuscated data should fit into. It uses a custom configuration file format, for a number of reasons:

- Easy to extend
- Concise
- Easy to understand

An example of the file format is as follows:

```
PAGE#0.1
DELIM <-

bin <-0
bin <-1

byte <- %bin%bin%bin%bin%bin%bin%bin%bin
*out <- %byte
```

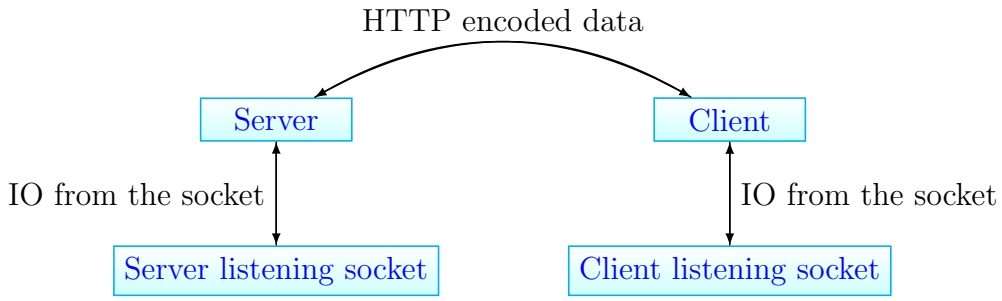
This is a very simple example that will simply print out a byte of the input at a time, but in binary. The reason I designed the obfuscation like this is because it allows for easy expansion, and testing. The obfuscation is completely reversible, and more details about how it works will be explained in a later section.

5.3 HTTP Layer

The HTTP Layer consists of a '`mutator`' (as defined in the previous section), and a corresponding client. The HTTP layer understands the HTTP Protocol, and how data can be inserted into it. The layer also calls the '`obfuscater`' which is used to insert data into the HTTP protocol.

5.3.1 Listening Ports

As a part of the HTTP layer (on both server and client), a TCP socket is listening, which is where all data that is transferred from client to server (and vice versa) is read from and written to.

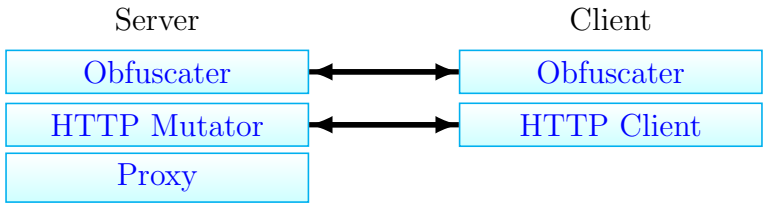


Using listening ports instead of directly interfacing with an application means that there is significantly more flexibility.

5.4 Modular Sections Overview

The overall design of the project is to be very modular, so things can be switched out and replaced.

Below is a diagram showing all of the parts that could be switched out and replaced between the server and client.



Both the server and the client have a HTTP section and an Obfuscater section. Either of these sections (on both the server and the client) could be switched out with an equivalent pair, and it would continue to function correctly without changing other parts of the program.

6 Implementation

6.1 Programming Language Choice

For the project, I opted to use Python for a number of reasons:

- Quick to develop and prototype
- Generally Safe
- Solid language design and features

On top of this, I did not know any python when I started the project, and I wanted to learn something new, which I definitely feel I did.
Python also has a wealth of useful libraries.

6.2 Project Breakdown

6.2.1 Proxy

hello there

6.2.2 Obfuscator

6.2.3 HTTP Mutator

6.2.4 HTTP Client

7 Testing

8 Project Management

9 Discussion

10 Conclusion

11 Bibliography

References

[1] Johnson Neil F and Jajodia Sushil. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, Feb 1998.

[2] Provos N and Honeyman P. Hide and seek: an introduction to steganography. *IEEE Security & Privacy*, 99:32–44, June 2003.

[3] Greg Farnham. Detecting dns tunneling. *SANS Institute Reading Room*, Feb 2013.

[4] Fielding R, Irvine UC, Gettys J, Compaq/W3C, Mogul J, Compaq, Frystyk H, W3C/MIT, Masinter L, Xerox, Leach P, Microsoft, and Berners-Lee T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.

[5] Born Kenton and Gustafson Dr. David. Detecting dns tunnels using character frequency analysis. *CoRR*, abs,1004.4358, 2010.

[6] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, RFC Editor, June 2014.

[7] University of Southern California Information Sciences Institute. TRANSMISSION CONTROL PROTOCOL. RFC 7230, RFC Editor, September 1981.

[8] Merriam-Webster Online:. Steganography, February 2018.

[9] Merriam-Webster Online:. Obfuscate, February 2018.

[10] P Christensson. Encryption, November 2014.

[11] Bruce Schneier. Memo to the amateur cipher designer, October 1998.

[12] Bruce Schneier. The value of privacy, May 2006.

[13] CHARLES DUHIGG. How companies learn your secrets, February 2012.

[14] AATIF SULLEYMAN. Google lets you report ads that know too much about you, September 2017.

[15] B. Gleeson, A. Lin, Nortel Networks, J. Heinanen, Telia Finland, G. Armitage, A. Malis, and Lucent Technologies. A Framework for IP Based Virtual Private Networks. RFC 2764, RFC Editor, February 2000.

12 Appendices