University of Birmingham

School of Computer Science
Final year project

VPN/HTTP

Demonstration

Accompanying paper

Author: Daniel Jones (1427970)

BSc Computer Science

supervised by
Dr Ian Batten

VPN/HTTP

Daniel Jones

March 16, 2018

Contents

1 Problem Definition

Censorship and monitoring of web traffic is a common occurrence around the world, as is deep packet inspection to ensure that traffic on port 80 is in fact HTTP traffic. On top of this, organisations have been known to man-in-the-middle HTTPS traffic by adding their own certificates to devices owned by the organisation. The aim of this project is to be able to tunnel data over HTTP and make it virtually impossible to tell that data transfer is occurring.

2 Implementation

To embed data in HTTP on its own would be trivial, putting the data to send and receive as the request/response body as data, while this would get around many web filters, it would only require trivial inspection to determine it is in fact not real HTTP traffic. For this reason, I decided that I wanted to insert the data into real websites and realistic browsing patterns, and to do this I decided that the VPN-Server would act as a website mirror when browsed to in the browser.

This means that to an observer, the VPN-Server will appear as a live mirror of a real website that updates as the original does.

When talking about how data is transferred in detail, I will refer to the client uploading, or sending data to the server as the 'Request' side, and when the client downloads, or receives data from the server as the 'Response' side.

Putting data in the Response side of the connection is trivial, as when you make a HTTP request, data is returned, and it is often different even if the same request is sent multiple times. The Request side, however is significantly more difficult. This is because when you are browsing a website, the request doesn't significantly change, and it would be incredibly obvious to an observer if for example the 'User Agent' field continuously changes. Therefore, the changes have to be subtle, and rely on the Header fields being a dictionary, so order doesn't matter, and insensitive to whitespace.

In this project, I implemented three ways of hiding data inside the HTTP protocol and are explained in greater detail below:

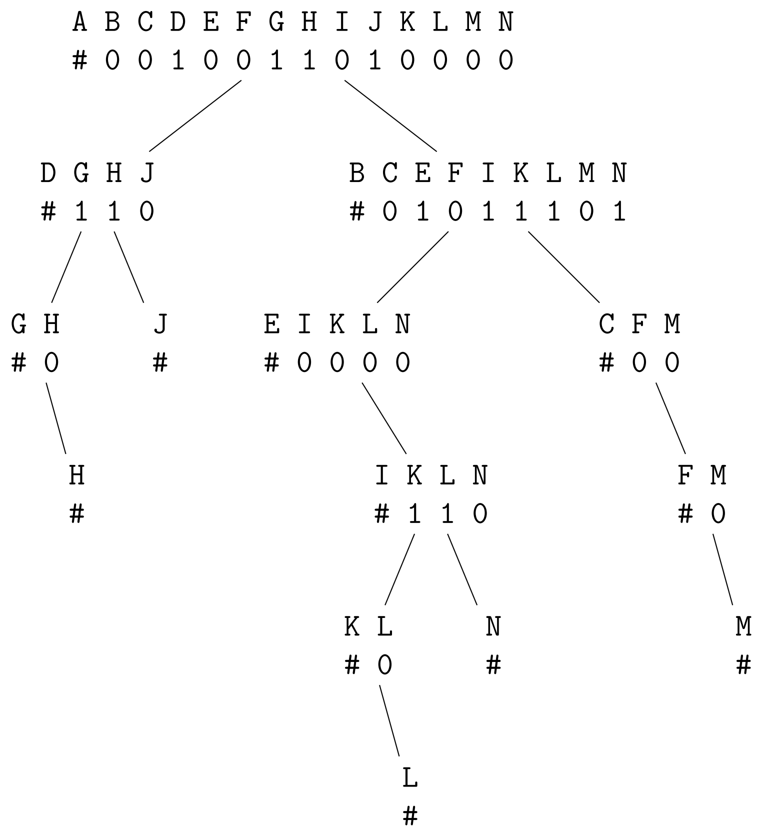
- Adding extra HTML
- Reordering headers
- Appending whitespace to headers

2.1 Adding extra HTML

This is used only on the Response side of the connection, and is performed by adding data into the HTML at certain points. To do this, I wrote a program that can generically encode data into any defined format. The format is defined as a recursive tree, and it has a vague resemblance to Backus Naur form. This format is parsed, and the data to encode is used to select which branch to go down. The benefit of this is that a valid HTML structure is generated.
//TODO image explaining what a total

2.2 Reordering headers

This is only used on the Request side of the connection. HTTP Request headers act as a dictionary, and lots of HTTP monitoring tools sanitise and order the header fields. This means that reordering is a good way of getting data from the client to the server and it being very difficult to spot. Storing data in the order of elements was quite a difficult problem to solve. My solution essentially resembles quicksort however instead of sorting on the data it is sorted by a bitstream.
For example, here, the data 'data' is encoded into the order of:
'A B C D E F G H I J K L M N'
The binary representation of 'data' is:
'01100100 01100001 01110100 01100001'



The theoretical number of bits that can be stored in x elements is:

$$\log_2(x!)$$

And using the method I came up with, I can approach that theoretical maximum. Huh?