Identifying the Use of Anonymising Proxies to Conceal Source IP Addresses

Miller, S., Curran, K., & Lunney, T. (2021). Identifying the Use of Anonymising Proxies to Conceal Source IP Addresses. *International Journal of Digital Crime and Forensics*, *13*(6), 1-20. https://doi.org/10.4018/ijdcf.20211101.oa8

Published in:
International Journal of Digital Crime and Forensics

Publication Status:
Published (in print/issue): 01/11/2021

DOI:
10.4018/ijdcf.20211101.oa8

Document Version
Publisher's PDF, also known as Version of record

International Journal of Digital Crime and Forensics
Volume 13 • Issue 6 • November-December 2021

Identifying the Use of Anonymising Proxies to Conceal Source IP Addresses

Shane Miller, Ulster University, Coleraine, UK

Kevin Curran, Ulster University, Coleraine, UK

Tom Lunney, Ulster University, Coleraine, UK

ABSTRACT

The detection of unauthorised users can be problematic for techniques that are available at present if the nefarious actors are using identity hiding tools such as anonymising proxies or virtual private networks (VPNs). This work presents computational models to address the limitations currently experienced in detecting VPN traffic. The experiments conducted to classify OpenVPN usage found that the neural network was able to correctly identify the VPN traffic with an overall accuracy of 93.71%. These results demonstrate a significant advancement in the detection of unauthorised user access with evidence showing that there could be further advances for research in this field particularly in the application of business security where the detection of VPN usage is important to an organization.

KEYWORDS

Anonymous Proxies, Machine Learning, Security, Virtual Private Networks, VPNs

1. INTRODUCTION

The Internet has become an important part of everyday life and its usage continues to grow as more devices are released that have Internet connectivity. Internet usage in developing countries is especially increasing with the arrival of affordable mobile smartphones (Poushter & Stewart, 2016). As more people use the Internet, governments seek to implement controls on what their citizens can access, either for the protection of said citizens against malware and identity theft or to suppress unacceptable parts of the Internet (Akabogu, 2017; Fiaschi et al., 2017). This leads some people to become concerned for their privacy as they do not want their online activities documented. Due to this and other factors, usage of technologies designed to provide anonymity on the Internet has increased. There has been a rise in trusted platform modules to secure data as well (Ma et al., 2006; Munoz & Fernandez, 2020; Munoz & Mana, 2011; Munoz et al., 2008). Other techniques include monitoring architectures for the cloud and secure agent computing (Indulska et al., 2007; Munoz et al., 2013).

Anonymity technologies allow users of the Internet access to a level of privacy that prevents the recording of information such as IP addresses, which could be used to aid in the identification of the users. Users of these technologies will have varying motivations for why they want to protect their privacy (Muñoz-Gallego & López, 2019; Rudolph et al., 2009). Some use anonymity technologies because they live in a country where their Internet usage is monitored and the websites that they wish to access are blocked. In this situation, the anonymity providing technology helps the user circumvent

DOI: 10.4018/IJDCF.20211101.oa8

the blocks that have been imposed on them. A similar use case is a user preventing their browsing habits from being tracked by their Internet service provider (ISP). Some ISPs track browsing habits to improve the services that they provide while some collect the data so that it can be forwarded on to other third parties. These include advertisers who use it to produce targeted advertisements or possibly security forces who use it to build a profile of the suspects and determine whether they are adhering to a country's laws involving Internet access. Naturally, criminals want to avoid their identity being released to the police. Therefore, they turn to anonymity providing technologies (Kokolakis et al., 2009). Anonymity systems transport network packets over intermediary relays so that no single system other than the original machine has information that could identify the user. Since many people can make use of these intermediary relays at the same time, the connection of the user seeking anonymity is hidden amongst the network traffic of other Internet users (Li et al., 2013). These different use-cases have led to anonymity on the Internet being a divisive topic. On one side, anonymity technologies provide legitimate methods for protecting freedom of speech and privacy, facilitating the transfer of anonymous tips to law enforcement and bypassing state censorship. However, the same technologies can be used to provide protection to criminals who are involved in information and identity theft, spam emailing and even organised terrorism. Additionally, they can be used for network abuse by bypassing Internet usage policies of organisations. This has the potential to expose the internal workings of the organisation to malicious activities.

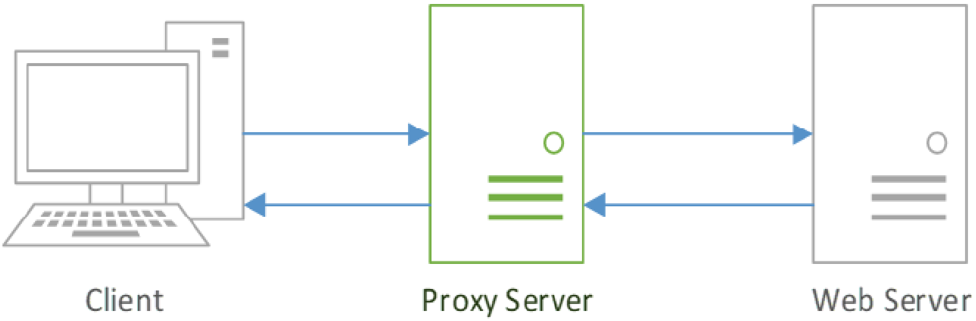
This paper provides an overview of our work to address the limitations of single-hop anonymous communication method classification by proposing a machine learning based approach utilising TCP header information and flow-based TCP statistics. A key goal in implementing this approach will be high accuracy and keeping the number of false positives and false negatives to an absolute minimum. Classifying legitimate network packets as having originated at an anonymous communication system could be catastrophic to an organisation that depends on high volumes of traffic reaching their site. Similarly, classifying anonymous communication traffic as legitimate could open an organisations internal network to malicious activity where the identity of the perpetrator is unknown. Having the ability to accurately determine which class the network traffic falls into can be a step towards allowing a network manager to secure an internal network, especially when combined with other security tools.

2. THE ROLE OF ANONYMOUS PROXIES

Web proxy servers are servers based on the web that fulfil the role of middleman during communications between a client and a server. Proxies normally operate under 2 different protocols, Hypertext Transfer Protocol (HTTP) and Socket Secure (SOCKS) (Ligh et al., 2010). Whilst HTTP is not designed solely for proxy communication, proxies still use it because it supports both encrypted and unencrypted traffic as well as the ability to allow non-HTTP traffic to pass-through a proxy-server (see figure 1). The SOCKS protocol consists of 3 major versions - SOCKSv4, SOCKSv4a and SOCKSv5. SOCKSv4 is a protocol that is designed for proxy-based applications. The other two versions are extensions of it that provide extra features and support for other protocols with SOCKSv5 providing support for the user datagram protocol (UDP), IPv6 and strong authentication (Leech et al., 1996).

During a normal HTTP interaction, a client will communicate directly with a server over HTTP. When a web proxy is involved the client will instead send its traffic to the proxy, which itself will communicate with the target server on the client's behalf. This means that web proxies have two roles to fulfil themselves, that of a HTTP client and that of a HTTP server. This is because the client is sending request messages to the proxy that are intended for the target web server. The proxy server must be able to handle and process those requests properly and the subsequent responses to facilitate a successful connection with the client. At the same time, the proxy itself must send requests to the target server, therefore it must be able to send requests and receive responses just like a normal HTTP client. SOCKS proxies operate at a lower level of the OSI layer model than HTTP, as shown in Figure 2 and differs in their operation. Where an HTTP proxy acts as a middle man or stepping stone

Figure 1. HTTP Proxy Interaction



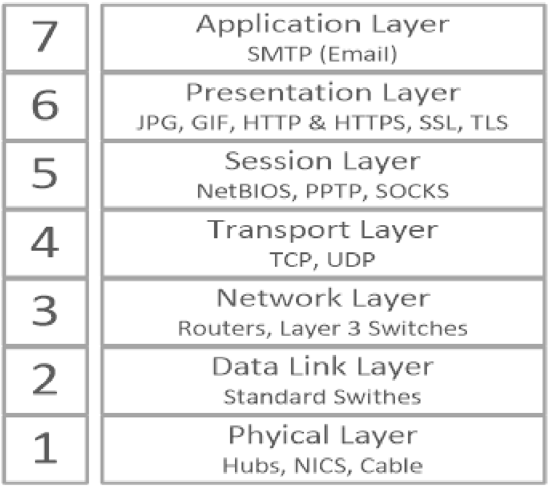
between a client and server by forwarding the HTTP requests, SOCKS proxies relay communications via TCP connections at a firewall gateway to allow a user application transparent access through the firewall (Lee, 2005). To use a SOCKS proxy connection, a client must have SOCKS client and server software installed on the user's machine. This can be in the form of an application such as PuTTY or a web browser, or it can be installed in the TCP/IP stack. The client software's main function is to redirect network packets into a SOCKS tunnel. The SOCKS client then initiates a connection to a SOCKS server. The proxy server then acts as the client and communicates with an external web server. This external server is only aware of the proxy server and not the original client that initiated the connection.

There are two overall types of proxy server; those dedicated to a single client and those that are shared among many clients (Stallings & Lawrie, 2008). Proxy servers that are dedicated to a single client are referred to as private proxies and those that are available to multiple clients are public or shared proxies. Private proxies perform a few specialised tasks, mainly when they are run directly on client computers. ISP services run small proxies to provide certain services such as extended browser features and to host advertising. Public or shared proxies are more common as they are usually accessible from the Internet. These types of proxy are more popular due to their accessible nature and are easier and cheaper to administer. There are also several sub-types of proxy in addition to the two overall types. These sub-type Proxies have a designated role or job. These roles include Content Filters, Document Access Controllers, Security Firewalls, Web Caches, Reverse Proxies, Content Routers, Transcoders and Anonymizing Proxies. A number of these are usually limited to enterprise networks, but some can be found on home networks as built-in modules of the gateway router supplied by an Internet Service Provider (ISP) and others, like anonymizing proxies, can be found openly on the Internet.

2.1 Content Filters

Content filter proxies provide administrative control over client transactions as they happen at the application protocol layer of the network stack (Kumar et al., 2014). This is commonly used in both commercial and non-commercial organisations, especially in schools as a method to control access to the Internet. Requests may be filtered using several methods, such as URL blacklists, URL regex filtering or content keyword filtering. More in-depth filtering can be accomplished by analysing the content of requests to discern whether they should be allowed or not. If the requested URL passes these filters, the filter proxy then proceeds to fetch the website, usually over HTTP. On the return path, dynamic filtering can be applied to block specific elements of a webpage, such as embedded videos or JavaScript. A drawback to content filtering proxies is that they cannot, normally, scan websites that

Figure 2. OSI 7 Layer Model



7	Application Layer SMTP (Email)
6	Presentation Layer JPG, GIF, HTTP & HTTPS, SSL, TLS
5	Session Layer NetBIOS, PPTP, SOCKS
4	Transport Layer TCP, UDP
3	Network Layer Routers, Layer 3 Switches
2	Data Link Layer Standard Swithes
1	Phyical Layer Hubs, NICS, Cable

are transmitted over an encrypted, HTTPS session where the chain of trust for the website in question has not been tampered with. The chain of trust refers to the use of certificates in the implementation of SSL/TLS connections. These certificates are issued by root Certificate Authorities (CAs) who can attest to the legitimacy of the website. However, content proxies can generate their own root certificate and inject that into the communications as a trusted root certificate. With this certificate in place, the content filter can decrypt requests to scan the normally encrypted content. In this situation, the filter is effectively operating what is known as a Man in The Middle (MiTM) attack.

2.2 Document Access Controllers

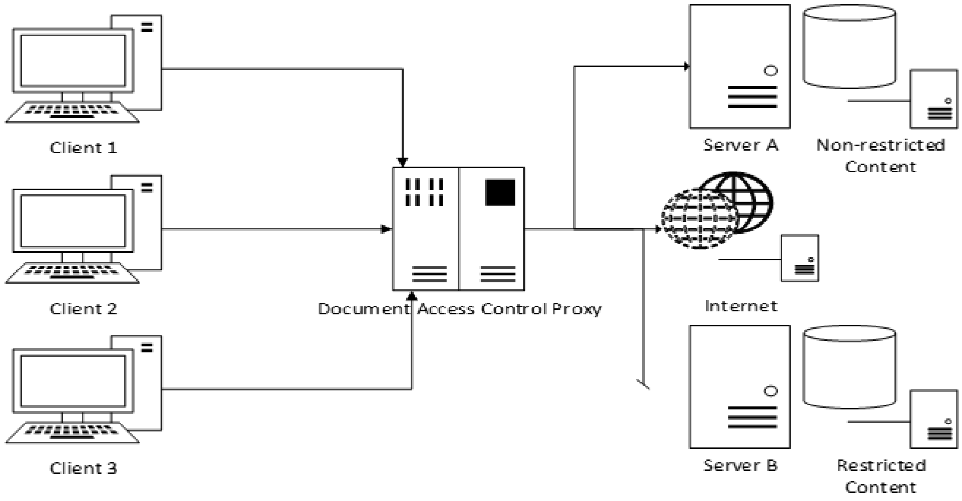
Document access controllers are proxy servers that have the role of implementing a uniform access control policy across a larger network of web servers and resources (Stallings & Lawrie, 2008). This eliminates the need for multiple access control systems and simplifies the administration of access control as all the controls can be configured on a centralized proxy server. This is particularly useful when used in a large data centre that makes use of servers of many different types and models which all have slightly different methods of applying access control.

Figure 3 shows an example network which contains a Document access controller. It shows three client pcs that are connected to the controller and it show some example content that the proxy controls access to. Server A contains non-restricted content therefore all of the clients can access this resource. Some of the clients will require access to the Internet and the controller regulates this access, only allowing the clients that have the required permissions to access it. Server B contains restricted content and by default none of the clients can access this unless they have been authenticated and have the required permissions.

2.3 Security Firewalls

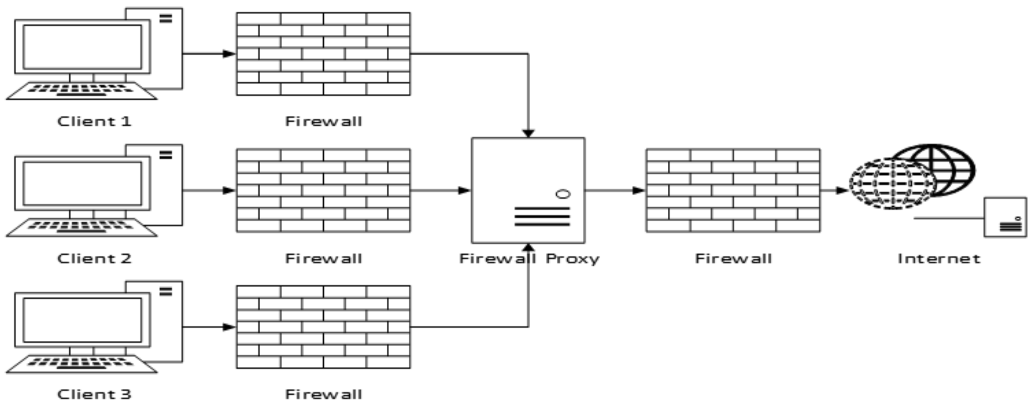
A significant security problem for business type networks is hostile or unwanted access by users or software (Kumar et al., 2014). Unwanted user access (an intrusion) can be in the form of unauthorised logon to a machine or gaining the ability to perform higher privilege actions than what is normally authorised (see figure 4). Unwanted software access can take the form of a virus, Trojan horse or other form of malware. A firewall is defined as a component or set of components that restrict access between a protected network and external networks (Ali et al., 2015). There are a few different

Figure 3. Document Access Controller Example



types of firewall. These are: packet filtering firewalls, stateful inspection firewalls, application-level gateway, circuit-level gateway and proxy firewalls. Packet filtering firewalls apply a set of rules to incoming and out-coming IP packets, any packets that adhere to those rules are forwarded on to their destination and any that do not are discarded (Kokolakis et al., 2009). A stateful inspection firewall reviews the same packet information as a packet filtering firewall, but also records information about the TCP connections that are sending and receiving the packets as well. Some also keep track of the TCP connection sequence numbers to prevent attacks that depend on the imposters using a sequence number, such as session hijacking. An Application-level gateway acts as a relay for application-level traffic. A user will contact the gateway using a TCP/IP application such as FTP or Telnet and the gateway asks for the name of the remote host to be accessed. When the user responds with the name of the remote host and provides a valid ID and authentication information, the gateway contacts the application on the remote host and relays application data between the two. If the gateway does not support the application, the service is not supported and cannot be forwarded across the firewall. Application-level gateways tend to be more secure than packet filtering firewalls as they only scrutinise a few allowable applications. A circuit-level gateway can be a stand-alone system, or it can be part of a specialised function performed by an application-level gateway. A circuit-level gateway operates in much the same way as an application-level gateway however, once it sets up the TCP connections, it does not examine the contents. The security function consists of determining which connections will be allowed (Ghai & Verma, 2015). Proxy firewalls are a network security application that filters network packets at the application layer (Singh et al., 2011) and they are the most secure type of firewall at the expense of speed and functionality of the network because they can limit the applications that are supported on the network. Proxy firewalls act just like standard proxy servers in that they act as an intermediary between a client computer and a destination web server. They are also the only machine on a proxy firewall protected network to have a direct connection to the Internet. This means that any other machine that wants to access a resource from the Internet will have to use the proxy firewall as a gateway. As the proxy firewall receives every request travelling to and from the network, it is able to filter and log requests based upon inspection of their packets. An added benefit to this type of proxy is that content that is being requested by multiple clients can be cached locally

Figure 4. Firewall Proxy separating clients from Internet



to increase the access time for the content. However, on networks with large amounts of traffic, the proxy firewall could be the cause of a reduction in performance due to the creation of a bottleneck or increasing the risk to the network by becoming a single point of failure.

2.4 Web Caches

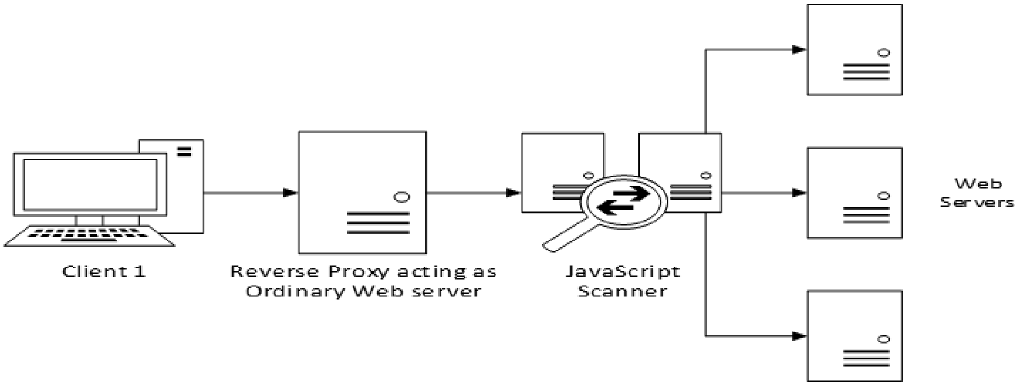
Alongside security proxy firewalls, there are also dedicated web caching proxy servers. Web caching is the temporary storage of popular remote web resources on a local server (Cobb & ElAarag, 2008). These proxy caches are used to reduce the strain of repeated requests for the same resource on web servers and network bandwidth providers (Durumeric, 2017). Caching proxies can be configured in one of two ways. The first way positions one or more servers on the network between a web server or (more commonly in a datacentre) a group of web servers and incoming network traffic from the Internet. This is designed to reduce the load on the web servers. The second configuration is designed with a focus on reducing congestion on the network. With this approach the proxy cache is located on the same network as the client machines making requests. As the request comes in, the proxy first determines if it has the requested material is stored locally. If the material is stored locally, it replies to the quest and delivers the material. If not, it initiates a connection with the web server to access the material and fetches the materials on behalf of the client and potentially caches it.

2.5 Reverse Proxy

A reverse proxy is a server that transparently hands-off requests to another server. Contrary to the normal operation of a proxy server where the server acts as an intermediary between clients and servers, the reverse proxy itself appears to the client as a web server, acting as an intermediary for its associated servers to be contacted by any client even if the servers are behind a firewall (see figure 5).

Reverse proxies can also be used as a load-balancer among several back-end servers or to provide caching for a single server to reduce the load of commonly requested data on the target server. In this latter implementation the reverse proxy can be referred to as a server accelerator (Stallings & Lawrie, 2008). The popular Content Distribution Network (CDN) and Distributed Denial of Service (DDoS) protection company Cloudflare provides its services by acting as a reverse proxy (Wurzinger, 2009). When a client visits a site that is protected by Cloudflare’s services, instead of connecting directly to the web server that is hosting the website, the client connects to one of Cloudflare’s servers which serves a cached version of the site or proxies the connection to the origin server. Another use case of reverse proxies can be in the mitigation of attacks against websites. Kurose, (2014) describe their

Figure 5. Reverse Proxy with JavaScript Scanner acting as a load balancer



method of mitigating Cross Site Scripting (XSS) attacks using a reverse proxy to relay traffic to and from the web server that is being protected. Each response made by the web server is forwarded by the reverse proxy to a JavaScript scanning component which scans the response for harmful scripts. If detected the proxy blocks the response from being delivered and instead notifies the client of the attempted attack.

2.6 Content Router

Content Routers are proxy servers that can redirect requests as part of an information-centric network (ICN). In a simple ICN setting, requests for content are generated and sent by end users. Each request consists of a content name, whole controls access to the content and the location from which it can be accessed (Wong, 2011). These requests are forwarded among content routers towards the location and controller of the content. Before a content router forwards a content request, it first checks its own local cache store for the requested content. If the content router has the content stored locally, the router itself can satisfy the request for that content, sending it to the requestor. As content is sent along its path from controller to requestor, each content router on the way stores a copy of the content in its local cache. This ensures that repeat requests for that same content can be detoured to content routers which may have the requested content, helping reduce the congestion on the network that would be created by having to forward requests directly to the content's controlling server (Chang & Chen, 2003).

2.7 Transcoder

Transcoding can be defined as the transformation that is used to convert a multimedia object from one form to another (Cardellini et al., 2000). Based on where the actual transcoding takes place, different technologies can be classified as belonging to server-based, client-based or proxy-based approaches. Transcoding proxy servers can modify the format of content before it is delivered to the destination. They can convert images from one filetype to another to reduce size and modify the image itself to make it fit onto different types of screen, for example when accessing a desktop website from a smartphone or similar device. They also can modify text files in a similar method, even translating the text into different languages based on the country ID of the user requesting the content (Stallings & Lawrie, 2008). This can be particularly useful when attempting to provide content for an international community where everyone may not understand or speak the original language of the content.

2.8 Anonymous Proxies

Anonymity technologies allow Internet users to maintain their privacy by preventing the collection of identifying information such as IP addresses. Due to an increasing awareness of what is shared and collected online, Internet users are growing more concerned with their privacy and are turning to the use of technologies such as anonymous proxies (Edman & Yener, 2009; Miller et al., 2015a). Anonymous Proxies are one of the easier to deploy technologies for anonymity on the Internet and are generally access through a web browser (Miller et al., 2015b). They are proxy servers that are based on the open Internet and are accessible to the general userbase of the Internet. The aim of an anonymous proxy is to make a user's Internet activity untraceable by acting as an intermediary between the user's client pc and the rest of the Internet. Anonymity is provided by the proxy server by processing client requests using the proxy server's IP address rather than the user's IP address. The server relays requests from the user to their destinations and delivers the responses back to the user. This provides a basic level of anonymity. However, the proxy servers can see both the source (client IP address) and destination (resource IP address) and therefore can track user activities and what is being relayed between the source and destination. There are multiple ways of setting up a proxy server. Two examples of some of the more popular technologies for setting up proxy servers are PHP and Common Gateway Interface (CGI) based scripts. Both provide the required functionality that anonymous proxy servers rely on and they have the benefit of being supported across multiple operating systems. Glype is a PHP based script and is one of the most common and popular web proxy scripts available on the Internet. Setting up a proxy server using the Glype proxy is accomplished by downloading the script files from the Glype website and then relocating those files to the correct directories on the webserver. This may appeal to user's who have access to web capable servers that are located on a different local network, such as owning a Virtual Server hosted by a server hosting company. However, a simpler option would be to access one of the many existing proxy sites already available. A study done in 2011 on the geo-location of public proxy servers found that there were 7,246 proxy servers available (Leberknight et al., 2010). A list found on the Glype proxy website listed 3,389 unique servers that were running the Glype script (Annessi & Schmiedecker, 2016; Poh & Divakaran, 2021; Zhang et al., 2020). A more recent list showed that there were 50,824 individual web proxies. This presents a problem when trying to block access to these proxies because there are so many that when one server is blocked, it's a simple case of accessing another proxy server. The difficulty lies in compiling a complete list to add to an IP block list or Access Control List. Due to the ease of setting up new proxy servers, new proxy servers are being added all the time. URL filtering is defeated by the proxy server's use of encoding to obfuscate and hide the actual URL from the filters. Some PHP based proxies also make use of the simple ROT13 encoding scheme, which is based on the Caesar encryption cipher using a key of 13. CGI proxies make use of the Common Gateway Interface which is a standardised protocol created to enable web servers to execute console/terminal style applications. The most common use of these is to generate web pages dynamically each time a request for that web page is received. CGI proxies use this type of script to perform the act of proxying a connection. Proxy clients send a request containing the URL of the website they wish to visit embedded in the data portion of an HTTP request. The proxy server pulls the destination information from the embedded data and uses it to send its own HTTP request to the destination. Whenever the results are returned from the destination web server, this is then forwarded to the proxy user (Huang & Cao, 2020). While Glype proxies enable URL obfuscation by default, the CGIProxy script does not. ROT13 encoding can be enabled by removing the line comments for the methods proxy_encode() and proxy_decode() in the script. The script also provides support for custom encoding code to be added such as hexadecimal encoding.

Table 1.TCP Header

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td colspan="32">Sequence number</td></tr>
<tr><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td colspan="4">Data offset</td><td colspan="3"><i>Reserved</i></td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td colspan="32">Options (if <i>data offset</i> > 5. Padded at the end with “0” bytes if necessary.)...</td></tr>
</table>

3. DETECTION OF ANONYMISING PROXIES

The client machine used to initiate connections and send requests through the web proxies is a virtual machine (VM) hosted using the desktop virtualisation software VirtualBox. The host system used to run the VM is equipped with a quad core Intel i7 processor and 24GB of DDR3 RAM. The VM has access to 4 threads from the processor and 6GB of RAM. The operating system chosen for the VM was Ubuntu 16.04 and this was later upgraded to 17.10. A Linux operating system was chosen because of the ease of automation for the capture of data and then packaging it into a suitable format. It was also a preferred choice due to the ease of programming with python using the built-in terminal command prompt. For capturing the network data, the VirtualBox network interfaces needed to be set up. VirtualBox provides up to eight virtual PCI Ethernet cards for each virtual machine. For each card, the individual hardware that is virtualised and the mode in which it is virtualised can be selected, with respect to the physical interface on the host machine. Each of the virtual network hardware types represents a different physical hardware PCI Ethernet card, with each card having different compatibilities with various operating systems. For the purposes of capturing network data from an Ubuntu VM, the Intel PRO/1000 MT Desktop virtual network card was left as the default choice. Each network adapter can also be configured to operate in a different mode. The mode selected for capturing the network data was the Bridged Networking mode. When this mode is enabled, the VM connects directly to the host machines network card and exchanged packets directly, circumventing the host operating systems network stack. The proxy used was Glype.

3.1 Dataset

The machine learning algorithm that was selected for classification of proxy network traffic was the“Two-Class Neural Network”. When training neural networks, a dataset containing example data is required. The data included in the dataset can either be labelled or unlabelled, but for the purposes of training a neural network, which is a supervised learning model, the data needs to be labelled. For the purposes of these experiments, there are two labels. Data generated from anonymising web proxies is given a label of ‘1’ and traffic that is not generated from the proxies is given a label of ‘0’. The first step in compiling training and testing datasets is gathering the actual raw data. The data being used for these experiments will be in the form of Transmission Control Protocol (TCP) network packets excluding the payload section. Table 1 shows a representation of a TCP header, giving an overview of what is transmitted by the protocol.

Table 2 shows the fields of the TCP header and a short description of what the purpose of the field is. Also included in the data will be fields from the IP header. However, these fields are for organising the data and won’t be included in the neural network training. This is to ensure that the neural network does not overfit the data by focusing on the IP addresses of the web proxies.

Table 2. List of TPC fields and description of their function

TCP Field	Usage
Source Port	The Source Port is the port number used by the computer sending the TCP segment and is usually a number above 1024 (but not always).
Destination Port	The Destination Port is the port number used by the computer receiving the TCP packet and is usually a number below 1024 (but not always).
Sequence Number	The sequence number helps the TCP software on both sides keep track of how much data has been transferred & to put the data back into the correct order if it is received in wrong order & to request data when it's been lost in transit.
Acknowledgement number	The acknowledgement number acknowledges receipt of data that has been received, if any. It also indicates the value of next sequence number expected.
Data Offset	Specifies the size of the TCP header in 32-bit words
Reserved	Set aside for future use and should be zero
URG	**Urgent Flag:** Used to indicate if "urgent" data is contained in the packet
ACK	**Acknowledgement Flag**: Used during 3-way handshake and data transfers.
PSH	**Push Flag**: Used for TCP push, which returns the buffer to the user application. Used primarily in streaming.
RST	**Reset Flag:** Used to reset a TCP connection
SYN	**Synchronise Flag:** Used during 3-way handshake
FIN	Indicates end of the TCP session
Window	Number of octets in the TCP header
Checksum	This field is used by the receiver to verify the integrity of the data in the TCP payload and rejects data that fails the CRC check.
Urgent Pointer	Points to end of "urgent" data in the packet. only exists if the URG flag is set.
Options	Used to indicate the options used, if any.
Padding	Used to ensure that the TCP header ends on a 32-bit boundary.
Data	segment of data from the user, such as part of an email or web page.

3.2 Packet Capture

To gather network packets that originated from a proxy service for use with the dataset, a list of such proxy sites needed to be collected. The best source for this was the various "proxy lists" available on the internet. These websites collect together a recent list of known proxy servers that are available for use with an interest in advertising their own services. Some of the lists only show sites that require payment to access the proxy service, some sites list a mix of paid and free proxy services and others list only those that are free from charge. One such list is operated by a company called UpsideOut who operate their own proxy service called Proxify. At the time of writing, the site is listing 50,824 different web proxies. Through trial and error, it was discovered that not every site on this list is online, however a list of sites that were accessible at the time of the experiments was gathered. Generating the network traffic required for the dataset involves using the proxy sites to visit websites. Doing this manually would have taken a large amount of time so a solution was developed to automate the browsing. It was decided that the scripting language Python would be used for development. Python has great support for working with networks and automation of functions, which is exactly what is required to generate this dataset. Familiarity with the language also played a part as there are other languages which are useful for the purposes of handling data, such as R, but would have taken time to learn. The python library selenium includes a package called Splinter which allows a python script to interact with an installed web browser. Splinter allows a python script to interact with elements contained within the HTML code of websites, such as filling out text fields or clicking buttons. URL addresses are provided in the form of a string containing the full address, for example: "https://www.website.com/". The first thing that is accomplished is browsing to the site. What site is being browsed to is determined by the for loops position in the string array. The script is then instructed to sleep for two seconds to allow the site to fully load before moving on to the next part. This next part finds the text input field of the proxy server by its CSS id, which was determined to be "input" on most of the Glype, PHP and CGI proxy servers. The site that is the target site for the proxies is "www.whatismyipaddress.com". This is a website that displays the connecting machine's IP address, which in the case of the script would be the IP address of the proxy server that is acting as an intermediator.

The script is then instructed to wait for one second before finding and clicking the submit button which initiates the connection to the target site. Some of the proxy sites used do not support the use of SSL encryption, so when browsing to a website that does make use of encryption, which the target site does, the proxy will display a warning page informing the user that the connection will not be encrypted. Below the warning, a button is provided which allows the user to continue despite the lack of security. The script checks for this warning and if it is found, it proceeds to locate the continue button and click it. The capture of the packets and compilation into a Comma Separated Value (CSV) dataset is also handled with a python script. There are various packet capture tools available. The script makes use of python's socket library which allows for the programming of various sockets. Sockets are one of the most popular methods of inter process communication and are used extensively in network communications. Browsers make use of sockets whenever they attempt to connect to a website. First the destination file for the data is defined, in this case it is "vpntraffictest.csv". This creates a Comma Separated Value (CSV) file in the same directory that the packet capture script is stored in. This file needs to be created before the writer object is created so that the writer knows where its output destination is otherwise an error will occur. The writer is created by calling the writer from the CSV library and passing the outputFile variable to it. Before the packet capture begins, the CSV writer writes the first row of the dataset which are the names of each part of the TCP packets that are being captured. Once a packet has been captured and processed by the script, the details of the packet are written to the output file. The traffic that is being investigated is Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) web traffic, however the packet capture script captures all TCP traffic. To strip out the unwanted traffic and only record the HTTP/HTTPS traffic, the TCP source port is used. An if statement instructs the CSV writer to only write the details of the TCP header to the output file if the source port is either port 80 (representing HTTP traffic) or port 443 (representing HTTPS).

Training a binary class classification algorithm requires two different classes of data to be provided. As a comparison to the proxy network packet data, packets were captured from the same system without the use of a proxy. To do this, the automated web browsing python script which first visited a proxy site then used the proxy to browse to "whatismyipaddress.com" was modified. The code instructing the browser to visit the proxy site were removed. The URL list was populated with a variety of sites from the Alexa top 500 index. The script then instructs the browser to visit each of the sites repeatedly until the execution is manually cancelled. The same packet capture script that is used to capture the proxy network packets is also used to capture the non-proxy data and write it out to a CSV file.

4. EVALUATION

Azure Machine Learning studio is a cloud service that provides an IDE-like workspace to allow for easier building, testing and deployment of predictive analytic models. Models can be constructed by dragging and dropping dataset and analysis modules into a workspace area. Modules can be added iteratively to help pinpoint problems. Predictive analysis helps you predict what will happen in the future. It is used to predict the probability of an uncertain outcome. Azure offers various types of statistical and machine learning algorithms aimed at predictive analysis such as neural networks, boosted decision trees and linear regression. Azure outlines a 5-step process to building an experimental predictive model: gather raw data, pre-process the data to clear the data of missing values or other mistakes, define the features that the model will be trained on, choose and train a learning algorithm, test the algorithm (Wurzinger, 2009). Once the model is trained and is predicting the test data accurately it can be deployed as a web service. Azure replaces the original dataset with a module to allow input from the web. Using the C#, python or R programming languages in conjunction with the URL of the deployed web service and a generated key, data can be sent to the web service to be analysed.

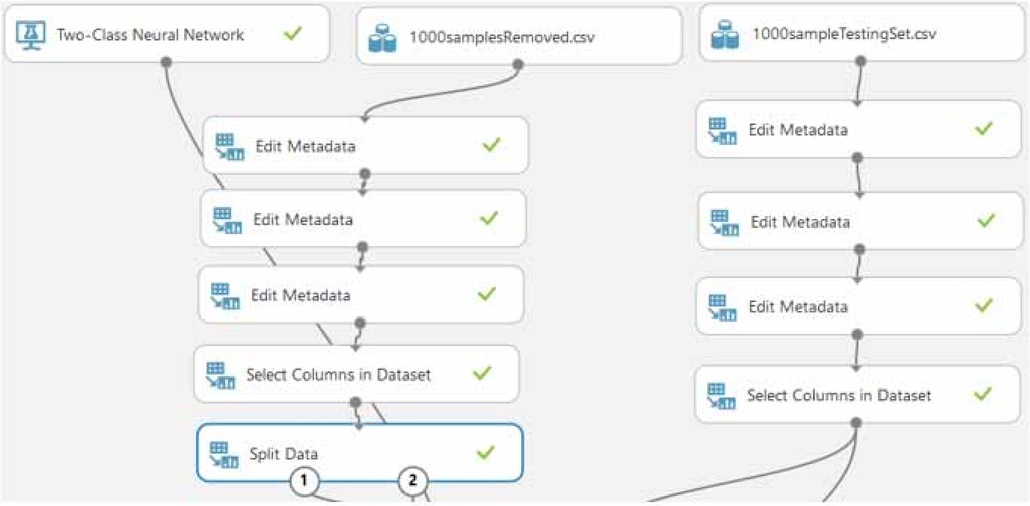
4.1 Methodology

There have been several recent studies that have made use of Azure's machine learning studio. Kurose, (2014) proposed a generalized flow within Azure that would accept multi-class and binary classification datasets and process them to maximise the overall classification accuracy. Two sets of experiments were run. The first was to benchmark the Azure machine learning platform using three public datasets. The second was to evaluate the proposed generalized flow using a generated multi-class dataset. The results showed that the generalized flow improved accuracy in all but one of the comparisons with prior work. (Pathak et al., 2015) describes a methodology to obtain a real-time view of traffic issues in a city using status updates, tweets and comments on social media networks using state of the art machine learning. The machine learning capability was provided by Azure machine learning studio. Data from various social networks is polled continuously by a worker role process hosted in Azure. The machine learning studio is used to process the data and analyse the text being imported. For the experiment they annotated 1100 social network feeds for 4 cities. This data was then split into training, cross validation and testing datasets that were used to train and test the machine learning algorithms in Azure machine learning studio. Classification accuracy for one social network ranged from 90-95% whereas on another the accuracy was just higher than 75%. Wong, (2011) proposed a method aimed at grading short test answers using machine learning techniques implemented in Azure. Experiments were run using 152 samples of student answers to a computer science question. The experiment showed that the system was able to grade all of the answers correctly after testing. Chang & Chen, (2003) proposed an anti-fraud web service that employed machine learning algorithms for predictive analytics in order to reduce the costs of infrastructure and software. Azure machine learning studio was used to provide the machine learning aspect. When building the machine learning model in Azure, they experimented with several algorithms for two-class classification. Using Azure's built in Score Model module, they were able to achieve an accuracy of 88% and went on to publish the model as a web service that can perform anti-fraud activities whilst reducing the cost of such a service to virtually zero.

4.2 Two-Class Neural Network

The algorithm that was selected for classification of proxy network traffic was the Azure module "Two-Class Neural Network". There is also a module provided for a multiple class application however that does not apply for this task. The trainer mode is the parameter that sets the algorithm up for one of the two scenarios that were mentioned. It contains two options, "Single Parameter" and "Parameter Range". The Single Parameter option allows the user to enter a single value for each of the parameters whereas the Parameter Range option allows for multiple value ranges to be used. The latter was the selected option for the proxy classification problem as the optimal parameter values were unknown beforehand. This is then combined with the module "Tune model hyperparameters" module which performs a parameter sweep over the specified settings and learns an optimal set of hyperparameters. This process is referred to as "tuning". The specification of the hidden layer can either be a fully connected instance, as selected, or it can be defined using a custom script written in the Net# language. The default, fully connected case uses a pre-defined specification for the hidden layer.

Figure 6. Dataset preparation



This results in a neural network which has one hidden layer, an output layer that is fully connected to the hidden layer which is in turn fully connected to the input layer. The number of nodes in the input layer equals the number of features used in the training data and the number of nodes in the hidden layer is defined by the user in the parameter option. The number of nodes in the output layer equals the number of classes, which for a two-class network means that all inputs will map to one of two nodes. The custom definition script is useful for when a more complex network is required, such as when deep learning is being implemented.

4.3 Dataset Upload and Preparation

To use a dataset with Azure it first needs to be uploaded to the machine learning studio. Azure supports multiple dataset types including CSV files. Before uploading the entire dataset samples randomly removed which were then used to compile a separate testing dataset. This left a training dataset of 5954 samples and a testing set of 1002 samples which were to be kept separate from the training process. Once uploaded, the datasets can then be added to the Azure interface and can then be further prepared for training and testing the neural network. The first preparation steps are in adjusting the metadata of the dataset so the model understands what fields are training features and what are class labels in the data. This is metadata that would not be readily available as part of the CSV format. The metadata is adjusted using the “Edit Metadata” module. In this module, the field that is to be edited is first selected and then there are four changes that can be made. The first deals with the data type of the field, for example, string or integer. If the field is already defined as being the data type required or simply does not need a type associated with it, there is the option to leave it unchanged. This applies to the first three modifications available. The second defines the field as either being a categorical field or not. If the field is already as required there is the option to leave it unchanged. The third modification allows the user to change whether the field contains a feature, a label or a weight. Again, there is the option to leave it unchanged and there are also options available to clear a previous definition. The fourth and final modification allows the user to enter new column names for the selected fields.

For feature selection, Azure machine learning studio provides three modules: “Filter Based Feature Selection”, “Fisher Linear Discriminant Analysis” and “Permutation Feature Importance”. For the purposes of this experiment however, feature selection was performed manually as there were only twelve features to select from. Some features were removed from the training dataset because they did not offer any value to the training. These were the Version number, Protocol identifier and the Time To Live (TTL). These fields contained the same value for each row of the dataset and therefore would have affected the computation time with no benefit gained. The source and destination IP addresses were removed as these could very well cause the network to overfit to the problem. From there, trial and error techniques were used to ascertain what features would cause the network to either overperform or underperform. The module used to select and remove features from the training is the "Select Columns in Dataset" module. All of the desired training features plus the classification label are selected using this module. The features that were used to train the network were: Source Port number, Destination Port number, TCP Sequence number, TCP Acknowledgement number, TCP Flag, Data size in bytes and whether HTTPS or HTTP were used. The final step before training and tuning commenced was to split the training dataset to provide both a training dataset and a training validation dataset. This is accomplished using the "Split Data" module and an 80/20 split was used which resulted in a final training dataset of 4763 and a validation set of 1191. Figure 6 shows the dataset preparation as it is represented in Azure. Also shown on the right-hand side is the 1002 sample testing dataset which is also going through the same preparation as the training dataset. This is so it can be successfully tested against as otherwise Azure will throw an error.

4.4 Training and Testing

Figure 9 shows the entire experiment. Once the data has been fully prepared, it is connected to the tuning module as the training dataset. The two-class neural network model is also connected to the tuning module at this time. At this point the experiment can be run with the tuning module's default settings, however these settings were modified slightly. By default, the tuning module will only perform 5 parameter sweeps. This was changed to 50 sweeps so the final model could be as close to the best model as possible. Training time for this model lasted for approximately one hour. This could be considered a consequence of the use of Azure, specifically the use of the free workspace, as there was no control over what hardware was used to train the model. The studio workspace used is in the South-Central US region which could also add latency to the connection, further slowing the process. It is possible to purchase a subscription to Azure which unlocks a far greater feature set. This was deemed unnecessary for this experiment as the resources provided were enough.

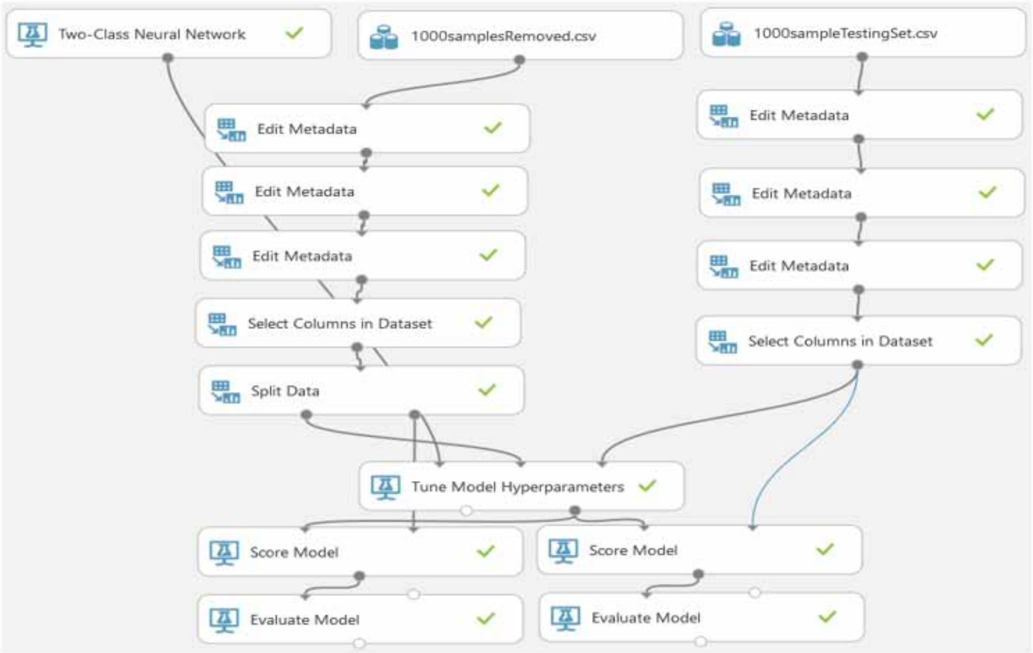
Once tuning is finished, the tuning module outputs a trained version of the best model it found. For this experiment, the best model used a learning rate of 0.0441313, a Squared Error loss function and was trained over 433 iterations. To score and evaluate the model based on the validation dataset and then the separate training dataset, the "Score" and "Evaluate" modules are used sequentially. The score module scores the classification predictions for the trained model and the outputs those results as a scored dataset. The scored dataset is then passed to the evaluate module which calculates a range of metrics based on the results. These metrics include: Accuracy, Precision, Recall, F-score, Area Under Curve (AUC), Average Log Loss and Training Log Loss.

4.5 Results

The results gathered from the separate testing set are shown in figure 8 along with the confusion matrix. The ROC curve is shown in figure 9. The AUC for the test was 0.988

The results in figure 8 show an overall classification accuracy of 94.6% for 1002 samples. There were only 13 false positive classifications, which is relatively low. However, there were 41 false negatives which is quite high. This may be due to the higher amount of negative (i.e. non-proxy) samples than positive samples at 473 positives to 529 negatives. In the context of this model being used as a proxy detection system on a live network, 41 proxy packets in every 1002 (0.04%) packets would possibly avoid detection and 13 in every 1002 (0.013%) normal, non-proxy packets would be possibly detected erroneously. Overall there is the possibility for approximately 50 errors in every 1000 packets. Whilst that may warrant worry, it is something that could possibly be improved upon

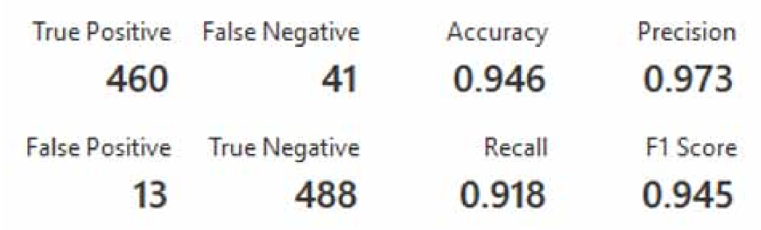
Figure 7. Fully connected experiment



and overall the results seem be an indication that the model has the capability of detecting proxy network packets.

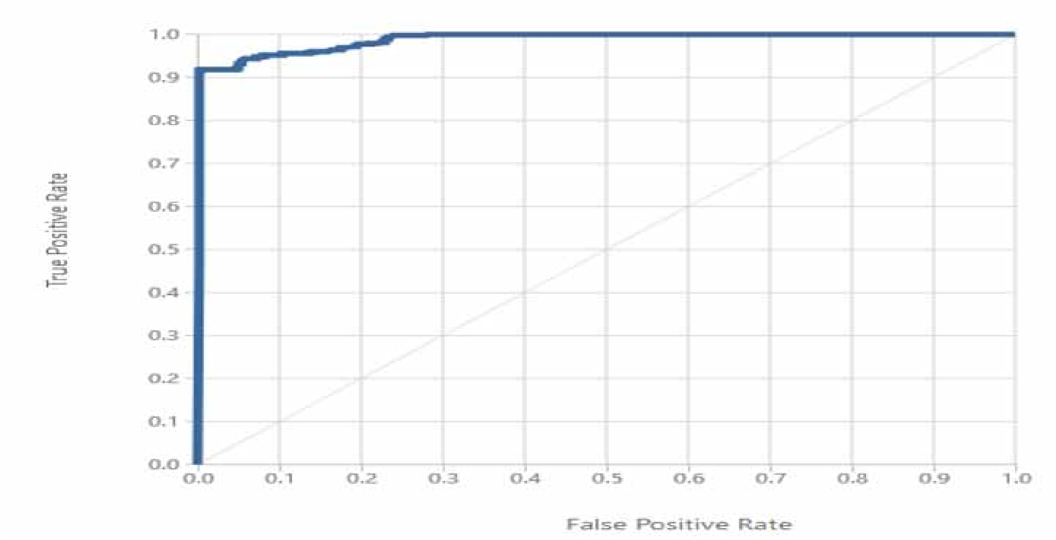
A drawback of this approach is that the dataset used is relatively small at approximately 7000 samples. Unfortunately, due to the nature of the packet capture and the tools available, the capture was time consuming. Another potential drawback is the use of TCP header details as training features. As there are only 12 features total, which are then reduced to 7, there is not much leeway given to the possibility that some features may not be suitable for classification at times, depending on the network conditions.

Figure 8. Confusion Matrix and Results



True Positive	False Negative	Accuracy	Precision
460	41	0.946	0.973
False Positive	True Negative	Recall	F1 Score
13	488	0.918	0.945

Figure 9. ROC Curve



5. CONCLUSION

The aim was to investigate methods that would aid in the detection of anonymising web proxies that are being used to hide an attacker's identity. While proxies and VPNs have legitimate uses, such as connecting to a business network from a remote location, they are still abused by criminals who use them to commit crimes whilst remaining undetected and unidentified (McKeague & Curran, 2018). We showed here a neural network model that was capable of classifying Anonymising Proxy network traffic versus normal traffic not being routed through an Anonymising Proxy. The chapter begins by outlining the hardware setup used to generate network traffic and the reasons for the choices made. Then the chapter begins describing the steps taken to generate the dataset needed to train and test the neural network model. This involved using the Proxy client described in the hardware setup section to generate and capture network traffic from anonymising web traffic sources and network traffic from non-Proxy sources. The capture used automated browsing and capture scripts that were written in Python. The features of the dataset resulting from the network traffic capture took the form of TCP header details. This dataset was used alongside the Microsoft Azure Machine Learning Studio to train, tune and test a binary class Multi-layered Perceptron Neural Network. As tuning of the model progressed, it was found that some features were causing the model to overfit to the data and these were then removed. Once overfitting was reduced a completed model could be trained and was tested on data from the dataset that had purposely been kept separate from the training and tuning processes to reduce any bias that may have formed in the model. The results of the validation test showed that the model was capable of classifying network traffic as either Anonymising Proxy traffic or as non-Proxy traffic and the concluding results section details the results obtained. The datasets captured in the process of the work undertaken for the thesis have been of varying sizes, with the largest being approximately 11000 samples. Time played a large part in how much data was feasible to capture. Future work could be done to increase the size of the datasets to further test the strength of the models created and the neural network model developed is only one of the many machine learning algorithms available. It would be worth investigating other algorithms to compare them against the neural network model to see whether there can be any improvement in either training time or in overall accuracy.

NOTE

On behalf of all authors, the corresponding author states that there is no conflict of interest.

REFERENCES

Akabogu, C. (2017). Implications of mass media censorship on the individual and the Nigerian society. *International Journal of Communication, 1*(1), 66-74. Available at: https://journal.ijcunn.com/index.php/IJC/article/view/45

Ali, A. A., Darwish, S. M., & Guirguis, S. K. (2015). An Approach for Improving Performance of a Packet Filtering Firewall Based on Fuzzy Petri Net. *Journal of Advances in Computer Networks*, *3*(1), 67–74. doi:10.7763/JACN.2015.V3.144

Annessi, R., & Schmiedecker, M. (2016). NavigaTor: Finding faster paths to anonymity. *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P),* 214-226.

Cardellini, V., Yu, P. S., & Huang, Y.-W. (2000). Collaborative proxy system for distributed Web content transcoding. *Proceedings of the ninth international conference on Information and knowledge management - CIKM '00*, 520–527. doi:10.1145/354756.354861

Chang, C. Y., & Chen, M. S. (2003). On exploring aggregate effect for efficient cache replacement in transcoding proxies. *IEEE Transactions on Parallel and Distributed Systems*, *14*(6), 611–624. doi:10.1109/TPDS.2003.1206507

Cobb, J., & ElAarag, H. (2008). Web proxy cache replacement scheme based on back-propagation neural network. *Journal of Systems and Software*, *81*(9), 1539–1558. doi:10.1016/j.jss.2007.10.024

Durumeric, Z. (2017). The Security Impact of HTTPS Interception. *Network and Distributed Systems Symposium (NDSS'17),* 44-52. https://jhalderm.com/pub/papers/interception-ndss17.pdf

Edman, M., & Yener, B. (2009). On anonymity in an electronic society. *ACM Computing Surveys*, *42*(1), 1–35. doi:10.1145/1592451.1592456

Fiaschi, D., Giuliani, E., & Nieri, F. (2017). Overcoming the liability of origin by doing no-harm: Emerging country firms' social irresponsibility as they go global. *Journal of World Business. JAI*, *52*(4), 546–563. doi:10.1016/j.jwb.2016.09.001

Ghai, S., & Verma, A. (2015). Network Security Using Divergent Firewall Technologies. *IITM Journal of Information Technology*, *1*(2), 29–38.

Gourley, D., & Totty, B. (2002). *HTTP: The Definitive Guide* (1st ed.). O'Reilly.

Huang, S., & Cao, Z. (2020). Detecting Malicious Users Behind Circuit-Based Anonymity Networks. *IEEE Access: Practical Innovations, Open Solutions*, *8*, 208610–208622. doi:10.1109/ACCESS.2020.3038141

Kumar, S., Jkhann, O., Purushotham, M., Sreenivasula, G., & Rama, G. (2014). Overview of Emerging Trends in Network Security and Cryptography. *IJEIR, 3*(1), 51–56. Available at: http://www.ijeir.org/index.php/issue?view=publication&task=show&id=250

Kurose, J. (2014). Information-centric networking: The evolution from circuits to packets to content. *Computer Networks. Elsevier*, *66*, 112–120. doi:10.1016/j.comnet.2014.04.002

Leberknight, C., Chiang, M., Harold, P., & Wong, F. (2010). A taxonomy of Internet censorship and anti-censorship. *Fifth International Conference on Fun with Algorithms*, 52-64. Available at: https://www.princeton.edu/~chiangm/anticensorship.pdf

Lee, Y.-D. (2005). *SOCKS: A protocol for TCP proxy across firewalls*. NEC Systems. Available at https://ftp.icm.edu.pl/packages/socks/socks4/SOCKS4.protocol

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., & Jones, L. (1996). *SOCKS Protocol Version 5.* RFC: 1928. Available at: https://tools.ietf.org/html/rfc1928

Li, B., Erdin, E., Gunes, M. H., Bebis, G., & Shipley, T. (2013). An overview of anonymity technology usage. *Computer Communications. Elsevier*, *36*(12), 1269–1283. doi:10.1016/j.comcom.2013.04.009

Ligh, M., Adair, S., Hartstein, B., & Richard, M. (2010). *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishers.

López, J., Maña, A., & Muñoz, A. (2006). A Secure and Auto-configurable Environment for Mobile Agents in Ubiquitous Computing Scenarios. In J. Ma, H. Jin, L. T. Yang, & J. J. P. Tsai (Eds.), Lecture Notes in Computer Science: Vol. 4159. *Ubiquitous Intelligence and Computing. UIC 2006*. Springer. doi:10.1007/11833529_99

Luotonen, A., & Altis, K. (1994). World Wide Web proxies. *Computer Networks and ISDN Systems*, *27*(1), 147–154. doi:10.1016/0169-7552(94)90128-7

Maña, A., Muñoz, A., & Serrano, D. (2007). Towards Secure Agent Computing for Ubiquitous Computing and Ambient Intelligence. In J. Indulska, J. Ma, L. T. Yang, T. Ungerer, & J. Cao (Eds.), Lecture Notes in Computer Science: Vol. 4611. *Ubiquitous Intelligence and Computing. UIC 2007*. Springer. doi:10.1007/978-3-540-73549-6_117

McKeague, J., & Curran, K. (2018, April). Detecting the use of anonymous proxies. *International Journal of Digital Crime and Forensics*, *10*(2), 74–94. doi:10.4018/IJDCF.2018040105

Miller, S., Curran, K., & Lunney, T. (2015a). Securing the internet through the detection of anonymous proxy usage. *2015 World Congress on Internet Security, WorldCIS 2015*, 153–158. doi:10.1109/WorldCIS.2015.7359434

Miller, S., Curran, K., & Lunney, T. (2015b). Traffic Classification for the Detection of Anonymous Web Proxy Routing. *IJISR*, *5*(1), 538–545. doi:10.20533/ijisr.2042.4639.2015.0061

Munoz, A., & Fernandez, E. (2020). TPM, a pattern for an architecture for trusted computing. *EuroPLoP '20: Proceedings of the European Conference on Pattern Languages of Programs 2020*, 1–8. doi:10.1145/3424771.3424781

Munoz, A., & Mana, A. (2011). TPM-based protection for mobile agents. *Security and Communication Networks*, *4*(1), 45–60. doi:10.1002/sec.158

Munoz, A., Mana, A., & González, J. (2013). Dynamic Security Properties Monitoring Architecture for Cloud Computing. *Security Engineering for Cloud Computing: Approaches and Tools*. 10.4018/978-1-4666-2125-1.ch001

Munoz, A., Sanchez Cid, F., Khoury, P., & Compagna, L. (2008). *XACML as a Security and Dependability Pattern for Access Control in AmI environments*. Developing Ambient Intelligence., doi:10.1007/978-2-287-78544-3_14

Muñoz-Gallego, A., & López, J. (2019). *A Security Pattern for Cloud service certification*. Semantic Scholar.

Poh, G., & Divakaran, D. (2021). *A Survey of Privacy-Preserving Techniques for Encrypted Traffic Inspection over Network Middleboxes*. arXiv:2101.04338.

Poushter, J., & Stewart, R. (2016). *Smartphone ownership and internet usage continues to climb in emerging economies*. Pew Research Centre. Available at: https://www.pewresearch.org/wp-content/uploads/sites/2/2016/02/pew_research_center_global_technology_report_final_february_22__2016.pdf

Rudolph, C., Compagna, L., Carbone, R., & Munoz, A. (2009). *Verification of S&D Solutions for network communications and devices*. Security and Dependability for Ambient Intelligence. doi:10.1007/978-0-387-88775-3_9

Sánchez-Cid, F., Maña, A., Spanoudakis, G., Kloukinas, C., Serrano, D., & Muñoz, A. (2009). Representation of Security and Dependability Solutions. In S. Kokolakis, A. Gómez, & G. Spanoudakis (Eds.), *Security and Dependability for Ambient Intelligence. Advances in Information Security* (Vol. 45). Springer. doi:10.1007/978-0-387-88775-3_5

Singh, D. D., Kumar, S., & Kapoor, S. (2011). An Explore View of Web Caching Techniques. *International Journal of Advances in Engineering Sciences*, *1*(3), 38–43. Retrieved November 8, 2018, from http://www.rgjournals.com/index.php/ijse/article/view/175

Stallings, W., & Lawrie, B. (2008). *Computer security* (2nd ed.). Pearson Education. Available at http://williamstallings.com/ComputerSecurity/

Wong, W. (2011). Content routers: Fetching data on network path. *IEEE International Conference on Communications*, 1–6. doi:10.1109/icc.2011.5963111

Wurzinger, P. (2009). SWAP: Mitigating XSS attacks using a reverse proxy. *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, 33–39. doi:10.1109/IWSESS.2009.5068456

Zhang, X., Ma, X., Han, B., & Li, W. (2020). An Uncertainty-Based Traffic Training Approach to Efficiently Identifying Encrypted Proxies. *2020 12th International Conference on Advanced Infocomm Technology (ICAIT),* 95-99. doi:10.1109/ICAIT51223.2020.9315573