Firewall For Preventing Anonymous Proxy Usage

A THESIS

submitted by

ANANT KANDIKUPPA B120519CS

MAHESH SREEKUMAR RAJASREE B120149CS

SRI HARSHA VIPPARTI B120687CS

in partial fulfilment for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

under the guidance of

SUMESH T.A

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
NIT CAMPUS P.O, CALICUT
KERALA, INDIA 673601
April 12, 2016

Abstract

An anonymous proxy is a tool that attempts to make activity on the internet untraceable. Within a corporate world, employees can misuse these anonymous proxies to leak confidential information. This project intends to come up with a solution to prevent anonymous proxy usage. The design and implementation of one such solution has been presented along with the results and future scope.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An anonymous proxy is a tool that attempts to make activity on the Internet untraceable. Proxy servers are usually used within a corporate context to exercise control over outbound Internet traffic and are often combined with caching capabilities to make better use of limited bandwidth. Their use can also be seen in parts of the world where free speech is suppressed. But anonymous proxies also have a darker side. Within a corporate context, employees can gain access to proxies services, such as web proxies, and leak confidential information. There already exist solutions for blocking commonly used proxy services, but there are mechanisms, such as SSH tunneling, by which malicious traffic can be piggybacked onto normal traffic, thus bypassing the defense mechanisms.

There are two kinds of anonymous proxies - one that is specific to a particular protocol say, HTTP, and the other being protocol independent. The advantage of a protocol specific anonymizer is that no extra software is needed. For example, web proxies can be accessed using any web browser. Since they are protocol dependent, firewalls can be designed to identify proxy usage through traffic/packet analysis. Protocol independence can be achieved by creating a tunnel to an anonymizer. In the case of tunneling traffic through VPNs, the client has to install a VPN client and configure it in order to connect to a anonymizer. A naive method to block VPN's would be to restrict traffic only to the standard TCP/UDP ports.

Though firewalls can be designed to identify and block web proxies, the major challenge is to adapt to newer proxies that are being provided. VPNs may also use standard TCP/UDP ports like port 443 to tunnel their traffic to anonymous proxies. In such a case the naive method stated above would fail.

The inability to overcome the challenges associated with web proxies and VPNs make them the most popular forms of anonymous proxy services today. This project therefore focuses on tackling these challenges.

1.1 Problem Statement

The problem is to prevent anonymous proxy usage within an organisations network. This is achieved by a browser extension working along with a backend component that captures all packets, identifies packets originating from browsers and blocks requests routed towards anonymous proxies.

1.2 Literature Survey

Mechanisms to detect commonly used proxies are given by Brozycki in [2]. These include blacklisting, analyzing TCP packets and using regular expression to find known patterns in URLs.

The Request For Comments for the IPSec protocol are presented in [3]. It describes how to provide a set of security services for traffic at the IP layer, in both the IPv4 and IPv6 environments. IPSec is one of the protocols used to tunnel VPN traffic.

A method to attack IPSec is discussed in [1]. The attack exploits the vulnerabilities in particular Diffie-Hellman groups used by Internet Key Exchange which is the main key establishment protocol used for IPsec VPNs.

Chapter 2

Design

We broke down the problem of preventing anonymous proxy usage by considering blocking each of web proxies and VPNs as a separate individual problem.

2.1 Anonymous Web Proxy

An anonymous web proxy is either a dedicated computer or a software running on a computer that acts as an intermediary between the client and the server which the client wants access to. On receiving a request for a resource from a client, the anonymous web proxy forwards this request to the resource server on the client's behalf and returns the response to the client. The advantage of such an anonymous web proxy is that during this entire process, the information that can identify the client remains hidden from the server.

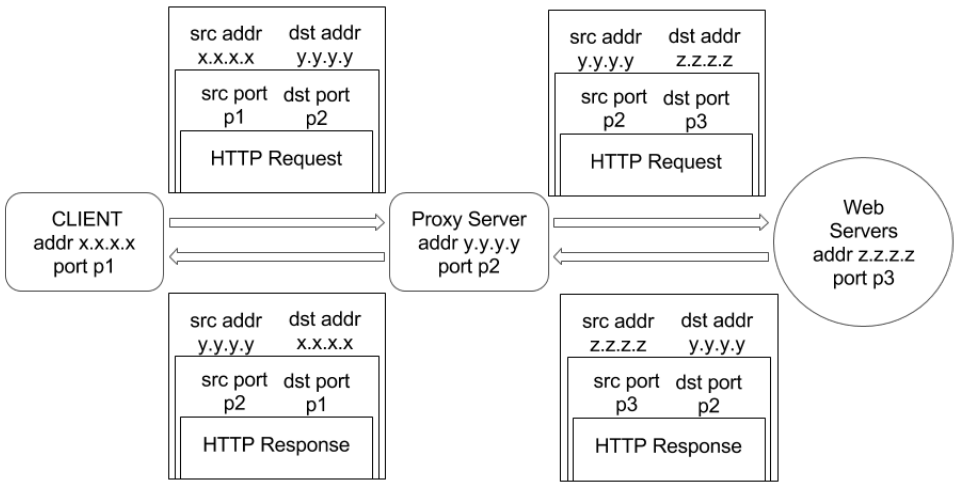

Figure 2.1: Web Proxy workflow

From the literature survey, it became obvious to us that current methods that block URLs using manual blacklisting cannot provide a reliable solution. Further with the widespread usage of strong encryption in network traffic, it is not possible to efficiently extract useful information from the communications. Hence we decided to attack this problem by analysing the packets.

2.1.1 Packet Analysis

To capture traffic generated by web proxies, **mitmproxy**[1] was used. **mitmproxy** is an interactive, SSL-capable man-in-the-middle proxy for HTTP/HTTPS. It allows us to intercept HTTP/HTTPS requests and responses and modify them. Using this tool, the responses of ten free web proxy services were compared with those of the actual server. The following observations were made :

1. The following HTTP header fields were found to be different
   - Server
   - Set Cookie
   - Keep-Alive
   - Content-Length (except for a few cases)
2. The following changes were observed in the packet body
   - The GET/POST form variables contained URLs (user-requested URLs)
   - Additional header and footer were added to the page
   - All the links were modified to hide the actual server hostname
   - In some cases, the page contents were encrypted and were decrypted in the browser using JavaScript.

From the above observations, the following conclusions were made:

- The Server, Set Cookie, Keep-Alive, Content-Length fields in the HTTP response headers would remain unchanged if proxies were not used
- A URL in the GET/POST form data shows that there might be a possibility that the receiver is a proxy server.
- Majority of the response body obtained through a proxy server would match the original response.

2.1.2 Design

Based on the results of the packet analysis, the following mechanism was devised.

1. Obtain URL from HTTP/HTTPS request.
2. If URL is already present in the blacklist, block the request
3. Else, pause the current request
4. Clone the paused request
5. Search for URL-like patterns in the GET/POST request data of the cloned request and replace it with a **reference URL**. The **reference URL** is a URL of an HTML page which contains a large prime number, to uniquely identify the page, and is hosted by us.

[1]https://mitmproxy.org/

6. Send the modified request and wait for response
7. If the response contains the large prime number, then URL is a proxy URL and can be blacklisted
8. Otherwise, the original request can be allowed to proceed normally

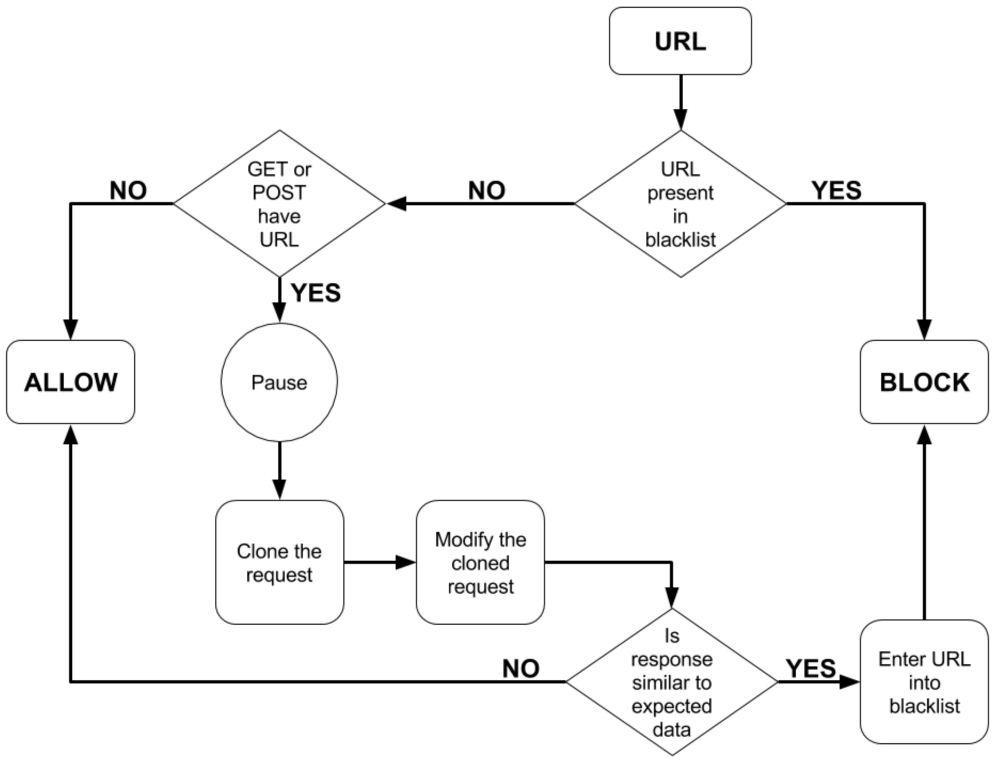

Figure 2.2: Design for blocking anonymous web proxies

Collaborative Blacklisting

Our solution has a feature that allows multiple clients to collaborate to build better blacklists by sharing their individual lists. Each client periodically synchronizes its list with the blacklist present in the central server. The server updates its entries as and when the clients encounter a new blacklisted URL.

Reference URL

As mentioned before, the reference URL points to an HTML page containing a large prime number. Given a URL, a web proxy returns the resource present at the particular URL. To check whether a particular site is a web proxy, we are creating a new request with this reference URL in place of the URL present in the form data. In case the site is a web proxy, the response would contain the HTML page addressed by our reference URL. But, if the site is not a web proxy, it might return a different response. In order to distinguish between these responses, we are using a very large prime number P, say about 100 digits long. Here, P ensures that the possibility of getting false positives is very rare, as it would not be expected to be seen in any other webpage.

2.2 Virtual Private Networks (VPNs)

An anonymous VPN service works in the following way.

- Creates a tunnel interface on the client machine.
- Directs all network traffic through this tunnel interface.
- These packets are encrypted using a protocol such as IPSec and encapsulated within a new transport and network layer.
- These new packets are then sent to the VPN server using the default interface.
- The VPN server decrypts the application layer data to get back the original packet.
- These packets are then forwarded to the intended destinations and the response is then encrypted, encapsulated and sent back to the VPN client.
- The VPN client decrypts the application layer data of the packets and gets the response.

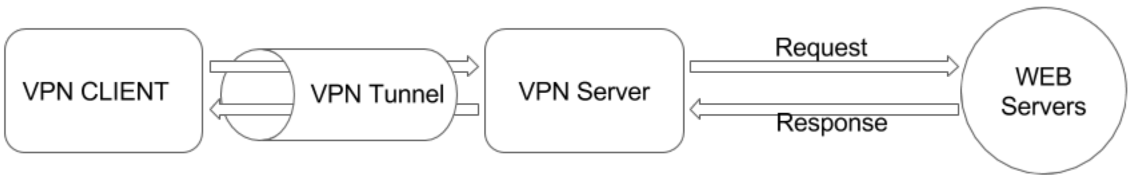

Figure 2.3: VPN workflow

Attacking the cryptography behind VPNs is not feasible with our limited resources, as shown in [1]. Thus, the only usable information that can be extracted is the network layer header which includes the source and destination IP address

2.2.1 Packet Analysis

For understanding VPN traffic, packets were analyzed using **Wireshark**. **Wireshark** is a free network protocol analyzer that allows us to capture and filter packets.Using this tool, we observed that:

- Destination IP is different from the IP of the original host
- Contents of other layers are obfuscated

From these observations, we can conclude that usage of VPNs can be identified by differences in the destination IP addresses.

2.2.2 Design

We propose the following approach to detect and block VPN usage:

1. Obtain the URL of a HTTP/HTTPS request
2. The extension sends the domain name of the URL to the backend component.
3. At the backend, all the source IP addresses of the incoming packets are stored in a record
4. On receiving the response, the extension queries the backend to know whether the IP corresponding to the domain name is present in the records
5. If it is not present, that implies that the packets corresponding to this request, were routed through a different path. Such a response is blocked and not shown to the user.
6. Otherwise, the response is allowed to be displayed by the browser

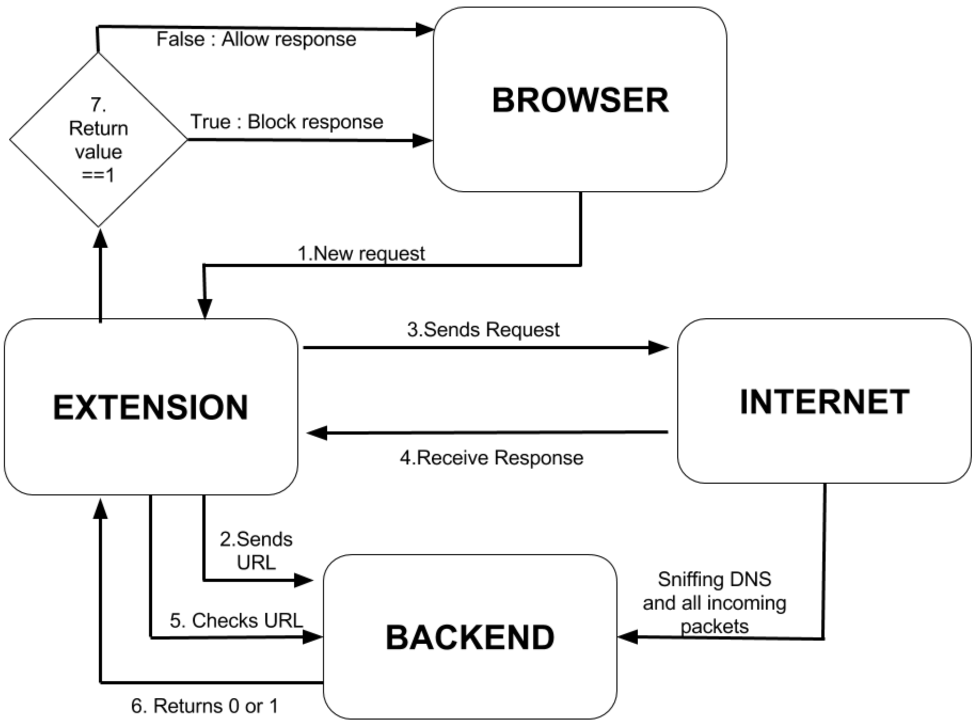

Figure 2.4: Design for blocking VPNs

For allowing access to legitimate VPNs, a whitelist can be maintained to specify the IP addresses of the VPN servers which can be legitimately accessed from within the organization.

Chapter 3

Implementation

3.1 Anonymous Web Proxies

The initial implementation included a browser extension that would block proxy websites. While doing so it was found out that there was a lack of API support available for modifying the HTTP request body and for cloning the requests.
mitmproxy provides all the desired features plus the added benefit of using it as a transparent proxy which allows our application to block HTTP/S requests originating from any application in our system, rather than restricting it to only web browsers.
The application was therefore modified, replacing the extension with an inline script running within mitmproxy.

3.1.1 GET/POST Requests

In order to filter out suspicious requests, our application searches for url like patterns in the GET/POST request data. The following regular expression is used to search for such patterns

```
(((https?:\/\/)?(www\.)?)?([a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6})\b
([-a-zA-Z0-9@:%_\+.~#?&//=]*))
```

When a suspicious POST request is made to the proxy server, it returns a redirect URL. The resource at this URL is then fetched by our application. In order to be able to fetch the resource, the GET request should bear a cookie value that the proxy server expects. The application copies and updates the cookie fields in the GET request depending on the previous communication with the proxy server.

3.1.2 Blacklisting

The application maintains a blacklist of URLs to block proxy sites that were encountered in the past in a SQLite database. The peculiarity of our problem is that the vast majority of the URLs encountered would be legitimate and queries to the database in such cases would slow down such traffic to a large extent. This calls for an additional data structure that allows for fast lookups and has low space requirements. One such data structure is a **Bloom Filter**.

Bloom filter is a probabilistic data structure to determine if an element is either definitely absent from the set or maybe present in the set. Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees, tries, hash tables, or simple arrays or linked lists of the entries.

An empty Bloom filter is a bit array of m bits, all set to 0. It also has k different hash functions defined, each of which maps some set element to one of the m array positions with a uniform random distribution. To add an element, it should be fed to each of the k hash functions to get k array positions which are set to 1. To query for an element, the element is fed to each of the k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set and if all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

Given a Bloom filter with m bits and k hash functions, both insertions and membership testing are O(k). The false positive error rate is approximately $(1 - e^{-kn/m})^k$ where k = number of hash functions, n = number of elements already inserted into the filter and m = number of bits of the filter.

For our project, we are using the pybloomfiltermmap library for Python. The hash functions used in python-bloomfilter are either sha512, sha384, sha256, sha1 or md5 (depending on the size of the bloom filter that we are using and the error rate) and are independent and uniformly distributed. The application in our case uses a constant space bloom filter having 10000000 bits and 0.01 false positive error rate.

3.1.3 Reference site

As described in 2.1.2, the reference HTML page which contains the random 100 digit prime number is currently hosted at `athena.nitc.ac.in/anant_b120519cs/refURL`.

3.1.4 Encrypted Pages

Some proxy sites offer the option of returning proxied pages in an encrypted form. The pages returned by the proxy contain JavaScript code that decrypts the contents when executed in a web browser. This form of encrpytion is able to fool our application as our reference prime would not be present as is in the response text. In order to get around this problem, we need to be able to execute JavaScript code within our application itself, and then look for the reference prime in the page. We have accomplished this using PhantomJS and Selenium, which allow us to test web applications from within our Python script.

3.1.5 Collaborative blacklisting

A central server is maintained which holds the URLs of all the web proxies encountered by the client machines which are connected to it via a persistent connection using web sockets. Whenever a client connects to the server, it downloads a copy of the current blacklist and updates its bloomfilter and database. Every time one of the clients encounters a new web proxy, it notifies the central server which pushes the update to all the connected clients. The web socket client code is blocking in nature, hence it runs on a separate thread that checks for updates from the server and also updates the server whenever a new web proxy is encountered.

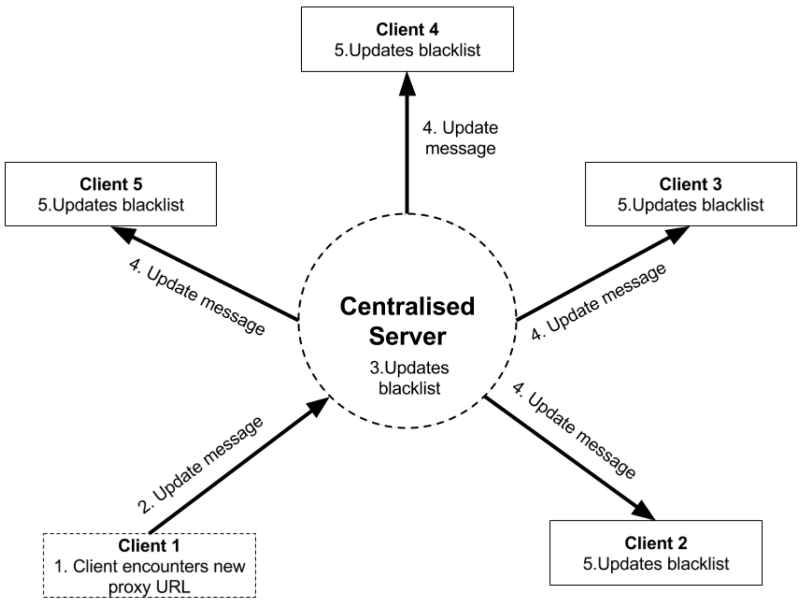

Figure 3.1: Collaborative Blacklisting

3.2 Virtual Private Networks (VPNs)

3.2.1 Browser Extension

The browser extension intercepts two kinds of events

- onBeforeRequest - This event is triggered before the browser request is sent to the network layer. At this point, the extension sends an asynchronous HTTP POST request to the backend, with the requestID, domain name of the request URL and the time at which the event was triggered, as its content.
- onHeadersReceived - This event is triggered when the response headers are received by the browser. At this point, the browser would have had opened a connection to the resource server. Thus, the extension queries the backend component through a synchronous HTTP GET request using the requestID and the time at which this event was triggered as the parameters. The request is redirected to an error page if the result of the query is 1 and is allowed through otherwise.

The requestID is a unique identifier generated by the browser that allows us to match responses to the corresponding requests.

3.2.2 Backend Component

HTTP Server

A HTTP server is used to communicate with the browser extension using HTTP messages. This server performs the following functions:

- Stores the domain name D and request time t_{rq} corresponding to a requestID, as sent by the browser extension via a HTTP POST request.

- On receiving a HTTP GET request containing a requestID R and the time when response headers were received t_{rp}, from the extension, retrieves the domain name D and the request time t_{rq} corresponding to R. Converts D to an IP address, IP, using its DNS cache. Obtains the time t_s at which the IP was observed in the sniffed packets from the IP Store. If $t_{rq} <= t_s <= t_{rp}$, returns 0 otherwise returns 1.
- Serves the page to be displayed when VPN usage is suspected.

Packet Sniffer

The packet sniffer sniffs incoming packets from the default network interface to store the source IP addresses. Also, it extracts DNS response records from the network and updates the DNS cache of our application.

IP Store

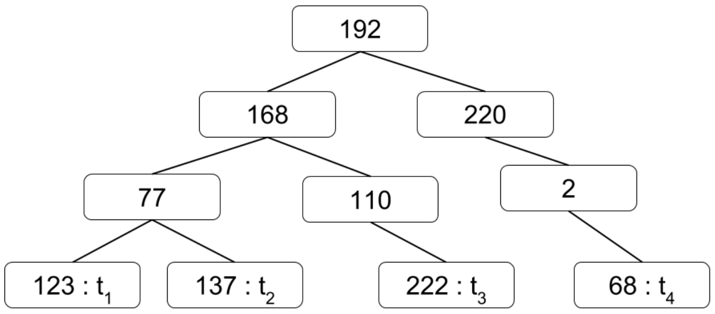

Figure 3.2: IP Store Data-structure

IP Store is a tree like structure with 4 levels. Each level is used to represent an octet of an IPv4 address. The leaf node of every IP address representation stores the time at which it was seen in the sniffed packets. This data structure provides $O(1)$ query and insertion.

DNS Cache

The primary motivation behind creating our a DNS cache within our application is that the IP addresses returned by a DNS query changes with time. In such a situation, we need to ensure that the IP addresses that we obtain are the same as those which the browser has obtained. The DNS cache is a mapping from a domain name to a list of IP addresses that are associated with it, that is updated by the sniffer on sniffing DNS response records.

Chapter 4

Results

4.1 Anonymous Web Proxy

4.1.1 Effectiveness

To test the effectiveness of our solution, we tested our application against the following proxy sites :

- hide.me
- hidester.com
- proxysite.com
- filterbypass.me
- unblockmyweb.com

Table 4.1: Effectiveness of the application

Proxy Site	None	Encrypt URL	Encrypt Page
hide.me	Blocked	Blocked	Blocked
hidester.com	N.A	N.A	Blocked
proxysite.com	N.A	Blocked	N.A
filterbypass.me	Blocked	Blocked	Blocked
unblockmyweb.com	N.A	Blocked	N.A

As seen in Table 4.1 the application is able to block all proxy sites irrespective of the encryption.

4.1.2 Efficiency

For analysing the efficiency of our solution, we considered the following cases:

1. Normal request without any URL like patterns in the form data.
2. Request with URL like patterns but not a proxy request.

Table 4.2: Efficiency (page load time in s)

Cases	Baseline	Fresh Request	Repeated Request
Case 1	1.32	2.06	N.A
Case 2	2.62	4.48	N.A
Case 3	N.A	13.63	0.14

3. Request with URL like patterns to a proxy server.

The page load times seen in Table 4.2 were observed in firefox browser with an empty cache.

4.2 Virtual Private Networks(VPNs)

Our solution is able to block requests made from a Chrome browser running our extension, in most of the cases. The one case where our solution fails is when the browser already has the IP address of a domain in its cache. In such a case a DNS query would not have been sent into the network and the sniffer would not be able to sniff the DNS responses and populate the DNS cache entries, thereby blocking legitimate access to the web pages through the browser.

Chapter 5

Conclusions

The aim of this project is to prevent anonymous proxy usage either through web proxies or through VPN service. During the course of this project, it was observed that achieving the above objective using passive methods like packet analysis, was not possible and some level of interaction with the end user was required. The solution that we implemented would work within a restricted environment such as that of an organization wherein the users do not have administrative privileges. There are certain scenarios where this solution can fail such as when the URL in the request body is encrypted, our solution would fail to identify URL like patterns in the form data thereby allowing the proxy request to proceed.

The design could be modified in the future by moving all the backend components to a dedicated centralized system, that can allow for easy maintenance and control. Secondly, the efficieny of the web proxy blocking system can be further improved by creating a standalone application, rather than using an inline script within mitmproxy. A component that can classify web pages based on the content can be plugged in to this application to enable it to filter web pages intelligently.

Bibliography

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice.

[2] J Brozycki. Detecting anonymous proxy usage (2008).

[3] S. Kent and K. Seo. Security architecture for the internet protocol. RFC 4301, RFC Editor, December 2005.