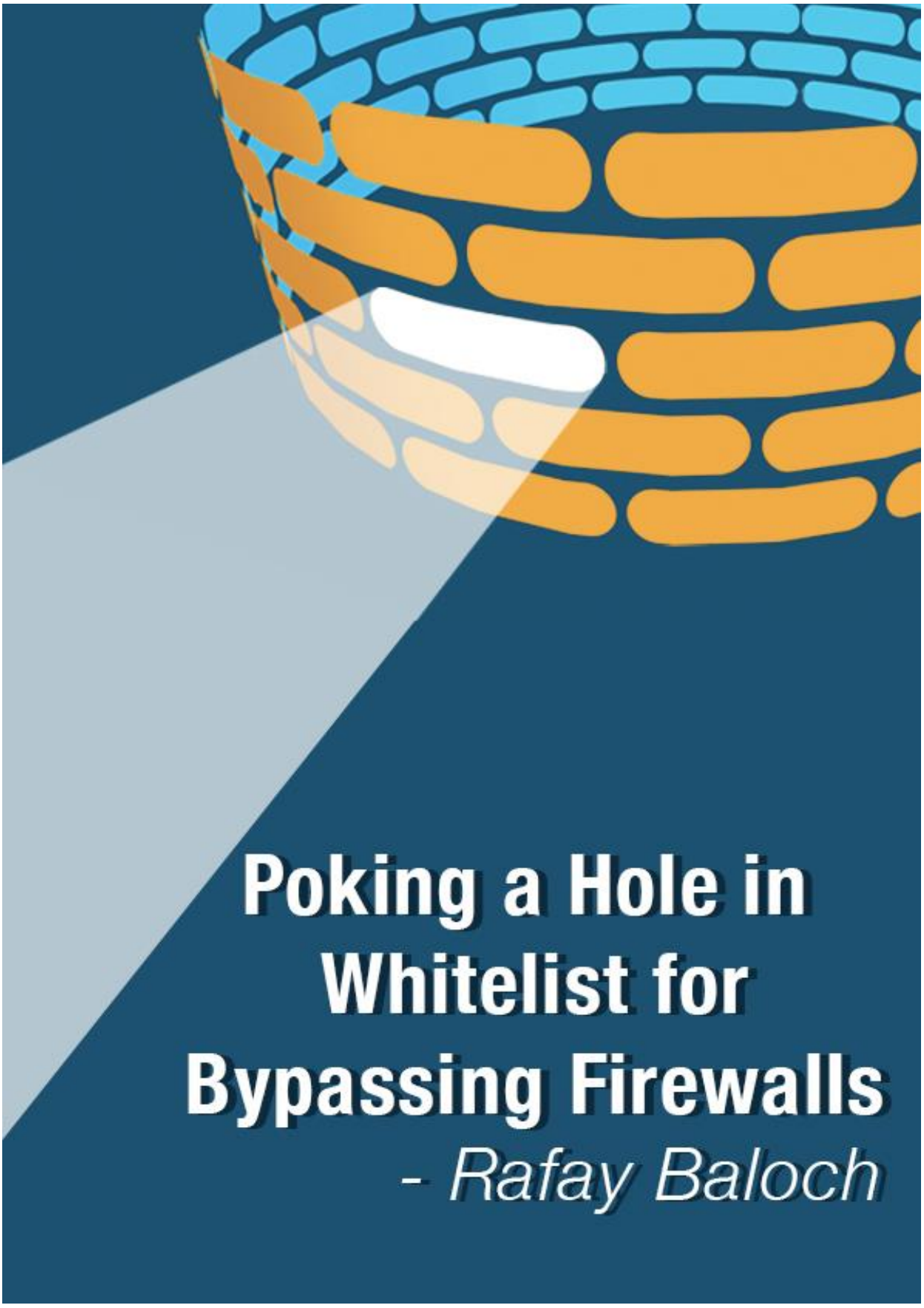

Poking a Hole in
Whitelist for
Bypassing Firewalls
- Rafay Baloch

## Table of Contents

## Abstract

Domain Fronting is a widely popular technique that has been used for evading Firewalls, DPI's and censors. Domain Fronting takes advantage of legitimate high reputation cloud providers, more specifically, Content Delivery Networks (CDN), for evasion. This technique has been commonly used in the wild to circumvent censorship or by malware for establishing a Command and Control C2 channel in restricted network environments.

In this Paper, we look at various forms of Domain Fronting along with few other techniques that can be utilized for circumventing firewalls, Deep Packet Inspection devices and captive portals. We will be dissecting a well-known for bypassing internet censorship bypass known as PSIPHON and will demonstrate how it utilizes Domain Fronting for bypassing Captive Portals.

We will also be exploring how poorly configured whitelists can be abused to circumvent captive portals, Firewalls and Deep Packet Inspection (DPI's) devices. Finally, we will also be releasing a script that can help Vendors audit their whitelists for finding various issues such as Domain Fronting and poorly configured regular expressions.

## 1. Content Delivery Networks Explained

Many companies nowadays are utilizing CDN's (Content Delivery Network) for hosting, caching and delivery of their content. A CDN consists of distributed servers that are geographically dispersed for high availability and maximum performance. The content is delivered based upon the geolocations usually from the nearest server. Let's see how CDN's work with an example:

Take an example where a website www.startv.com is hosted on akamai's CDN, the following is how the browser fetches the webpage:

i) The browser sends a DNS query to www.startv.com and gets its IP address that belongs to Akamai. The IP address can be communicated via all entry points for Akamai.

ii) Once it gets the IP address, it will send a request to that IP address and will include a HOST header pointing to www.startv.com. This is how CDN will find where to route the traffic inside the CDN as many domains can be pointed to the same CDN host.

The following request demonstrates the response when a valid host is found in a CDN:

```
root@kali-emc:~# wget -qO- --header 'host:www.startv.com' 'a1961.g2.akamai.net'
-o '<title>.*</title>'
<title>Imagine More with Star TV Network - Inspiring a Billion Imaginations</tit
```

**Response returned for valid Host in a CDN**

In case of the request below, Akamai returns an error as it is not able to find the route to “nohost.com” since it does not exist within its network.

```
root@kali-emc:~# wget -q -S --header 'host:nohost.com' 'a1961.g2.akamai.net'
  HTTP/1.0 400 Bad Request
  Server: AkamaiGHost
  Mime-Version: 1.0
  Content-Type: text/html
  Content-Length: 206
```

**Response returned for invalid Host in a CDN**

### 1.1 Domain Fronting

The key idea behind domain fronting is to take advantage of the encryption layer to mask our malicious traffic to our target host by making it appear that it is communicating with legitimate whitelisted hosts. On the firewall or DPI, it will appear to be traffic communicating with a legitimate host i.e. a CDN.

CDN’s such as Akamai, Amazon, Azure, Fastly, etc., happen to be a perfect candidate, as one CDN Host can point to multiple websites. There is no other way to block communication with our malicious host apart from blocking the entire CDN. This in turn would lead to collateral damage and could result in other legitimate websites, since many legitimate domains could be mapped to the same CDN host. Domains such as a0.awsstatic.com, s3.amazonaws.com etc. are commonly used by a large number of websites to host content.

Therefore, on the network side it is extremely common to see outbound traffic to a CDN network from almost any potential host. This makes CDN’s a prime target for domain fronting.

To understand how it really works, let’ dissect HTTPS:

In an HTTPS request, the destination domain can appear in only three places, the first being the DNS request which will be used to retrieve the corresponding IP address, second being the SNI (Server Name Indication of TLS protocol) and thirdly the host header. Generally, these three places include the same domain name to avoid confusion.

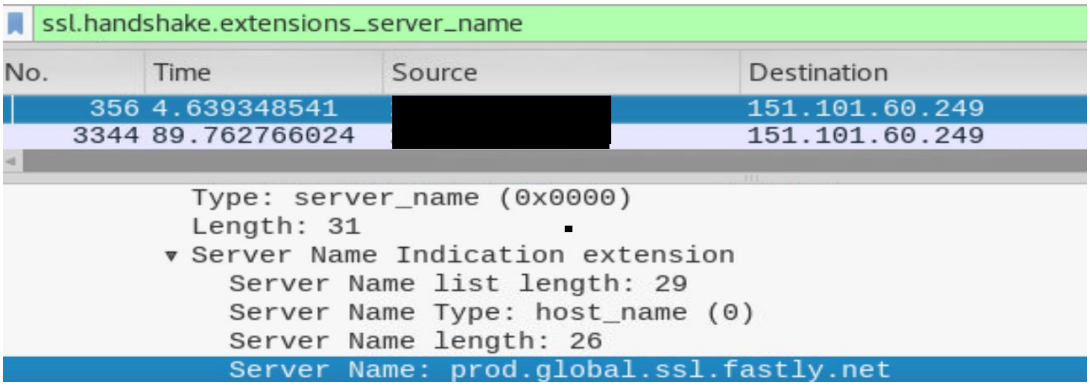
In a domain fronting attack, the hostname in DNS and SNI (Server Name Indication) will be different to the host header in an HTTPS request, since the Host header will be encrypted and only DNS lookup and

SNI will be visible at the network level. The censor does not see if the domain name doesn't match the DNS and SNI fields of the HTTPS request.

```
root@kali-emc:~# wget -qO- --header "Host:fr.foursquare.com" "https://prod.glob
l.ssl.fastly.net" | grep -o "<title.*</title>"
<title>New York | Restau, vie nocturne, divertissement</title>
```

In the above example the DNS Request and SNI (Server Name Indication) field will contain "https://prod.global.ssl.fastly.net" whereas the host header will contain "**fr.foursquare.com**" and will be seen as though the user is accessing prod.global.fastly.net.

The above request would send a request to https://prod.global.ss.fastly.net , however fastly upon receiving the request, will decrypt the communication and forward the request to fr.foursquare.com. The packet capture points the SNI field to prod.global.ssl.fastly.net. The following is a wireshark capture:



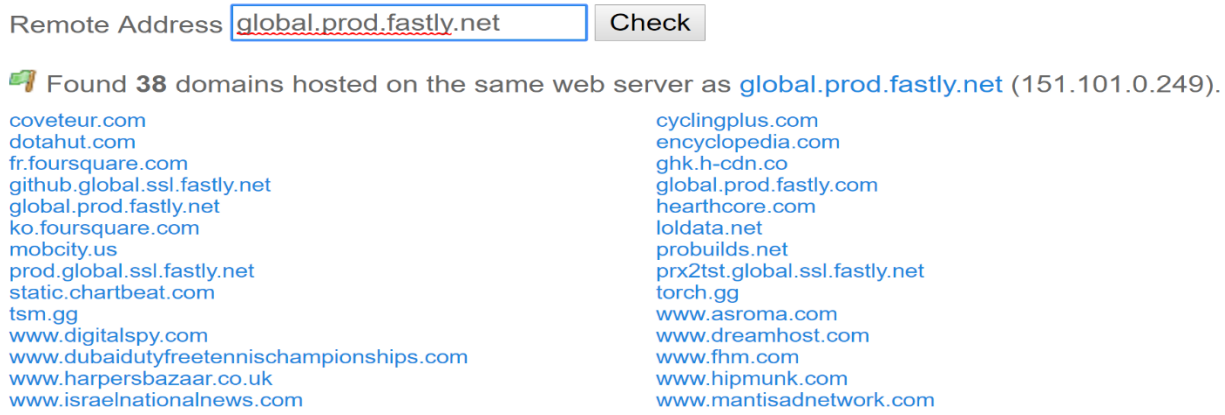

### 1.2 Server Name Indication

The host header field is commonly used in Virtual hosting environments allowing multiple domains to be hosted on one single IP. The Server name indication is a TLS extension which is the TLS equivalent of host header. A host header effectively allows multiple (HTTPS) websites to be hosted on a single IP without having them utilize the same certificate. The SNI is placed inside the client hello request during the initial TLS handshake. The SNI once sent, allows the server to present the browser with the certificate containing the relevant name.

Let's see how this works in practice, let's take an example of the domain a.ssl.fastly.net. The domain points to IP: 151.101.60.249.

```
root@kali-emc:~# host gloal.prod.fastly.net
gloal.prod.fastly.net has address 151.101.60.129
```

To find websites hosted on the IP address, we can perform a reverse IP lookup:




Alternatively, we can also query for "Subject Alternative Name" which allows multiple domains to be used in one SSL certificate.

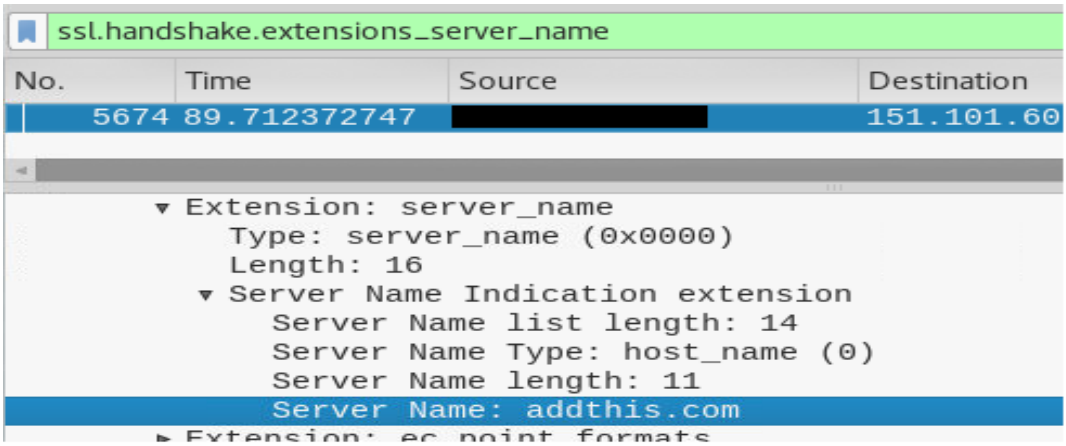
```
root@kali-emc:~# echo | openssl s_client -connect 151.101.60.249:443 | openssl x509  -text | grep DNS:

depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
verify return:1
depth=0 C = US, ST = California, L = San Francisco, O = "Fastly, Inc.", CN = a.ssl.fastly.net
verify return:1
DONE
            DNS:a.ssl.fastly.net, DNS:*.a.ssl.fastly.net, DNS:fast.wistia.com, DNS:purge.fastly.net,
DNS:mirrors.fastly.net, DNS:*.parsecdn.com, DNS:*.fastssl.net, DNS:voxer.com, DNS:www.voxer.com,
DNS:*.firebase.com, DNS:sites.yammer.com, DNS:sites.staging.yammer.com, DNS:*.skimlinks.com,
DNS:*.skimresources.com, DNS:cdn.thinglink.me, DNS:*.fitbit.com, DNS:*.hosts.fastly.net, DNS:control.fastly.net,
DNS:*.wikia-inc.com, DNS:*.perfectaudience.com, DNS:*.wikia.com, DNS:f.cloud.github.com,
DNS:*.digitalscirocco.net, DNS:*.etsy.com, DNS:*.etsystatic.com, DNS:*.addthis.com, DNS:*.addthiscdn.com,
DNS:fast.wistia.net, DNS:raw.github.com,
```

Now let's use the server-name modifier to query one of the domains hosted on the same IP such as addthis.com.

```
root@kali-emc:~# echo | openssl s_client -connect 151.101.60.249:443 -servername addthis.com | openssl x509 -subject
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
verify return:1
depth=0 C = US, ST = California, L = San Francisco, O = "Fastly, Inc.", CN = a.ssl.fastly.net
verify return:1
subject= /C=US/ST=California/L=San Francisco/O=Fastly, Inc./CN=a.ssl.fastly.net
```

If we look at the following capture, we can see clearly the SNI extension which was communicated as a part of the initial TLS handshake in the ClientHello message that **addthis.com** is the virtual it wishes to communicate with.




### 1.3 Frontless Fronting

In case of frontless domain fronting, the domain name is only placed in the host header. In that case even the DNS request is not visible and on the censor it is seen as though the client has established a normal SSL/TLS connection to the website. This can be useful in a scenario where the censor is known to block SNI. In this case, the censor has to block the entire IP address which effectively means that all websites hosted on the same IP address (especially in case of a CDN) will be blocked and hence will result in collateral damage.

```
root@kali-emc:~# wget -qO- --header="Host:fr.foursquare.com" https://151.101.36.249 --no-check-certificate | grep -o "<title>.*</title>"
<title>New York | Restau, vie nocturne, divertissement</title>
```

## 2. DNS Encryption and Domain Fronting

DNS uses UDP/TCP protocols to send requests, the requests are sent in plain text which allows censors/ISPs to monitor the websites visited. The TLS over DNS rfc7858 has been under discussion for a while and Google is also planning to implement it in future android versions **[1]**. Once all computers start to speak TLS over DNS, it will allow a more effective domain fronting whereby the only part that will be visible to the censor would be the SNI part.

However, there are other places where DNS names are also being exposed such TLS certificate messages. TLS 1.3 protocol designers are also working on ways to encrypt DNS names and prevent exposure **[2]**. Moreover, discussion about encryption and preventing exposure of SNI is also under consideration for TLS 1.3.

Upon implementation of RFC 7858, It will be even more difficult to detect domain fronting as the DNS and SNI fields will remain encrypted by default.

## 3. Captive Portals

Captive portals are a form of "Network Access Protection" that are commonly used by organizations for either controlled access or to collect the data from users. Users, when connected to a wireless network, are presented with a page that either requires authentication or would like users to agree with "Acceptable Usage Policy" in order to connect to the internet.

Captive portals are implemented in various ways. At a high level, all the requested domains prior to authentication are redirected to the captive portal address unless specifically instructed not to do so. This is accomplished by using a technique known as DNS hijacking.

When a DNS request for unauthenticated clients is sent to example.com, the firewall will hijack the request and respond with the captive portal's IP address. Since the DNS request will be stored in the DNS cache, this would mean that the browser will return the captive portal address even after authentication, as the DNS cache will be poisoned. To limit it, the "Time to Live' field is set to 0.

### 3.1 Passthroughs/Whitelists

Captive portals often add different hosts to whitelists that might be needed to facilitate a certain business requirement. For instance, the captive portal owner would like to offer free browsing to Facebook, Google etc. while charging for other websites. To accomplish that, Facebook and its corresponding subdomains must be added to the whitelist.

Addition of domains to the whitelist introduces the possibility of various ways to circumvent captive portals. We will discuss these various ways of abusing whitelists to circumvent protection around captive portal, firewall or DPI's (Deep Packet Inspection).

### 3.2 Bypassing Captive Portals Using Domain Fronting

Depending upon its implementation, Captive portals can be circumvented in many possible ways most notably being ICMP and DNS tunneling. Tunneling is moving one protocol into another. Assuming a real-world scenario where DNS, ICMP traffic is being monitored by a Deep Packet inspection device, we can utilize domain fronting to circumvent captive portals.

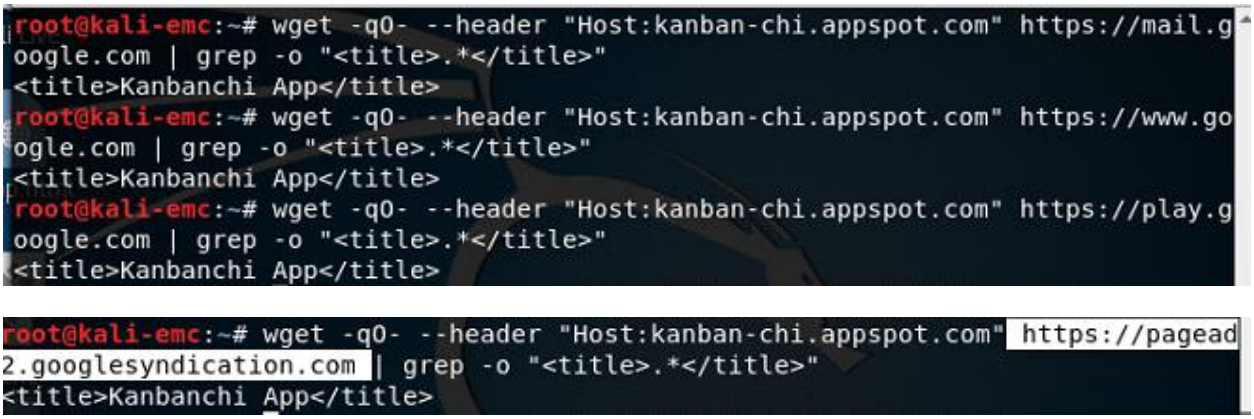
Captive portals will normally query a domain name and permit the associated IP address when a host is permitted in the whitelist. If a host points to multiple IP addresses all of them will be permitted. For instance. From my machine, www.disney.com resolves to **23.214.98.69** and happens to be an alias for **matterhornsecure.edgekey.net** which is an alias for **e13055.e12.akamaiedge.net.** If www.disney.com is permitted in the whitelist, this will permit **23.214.98.69** which would effectively mean that all virtual hosts on the same IP will be allowed.


Since host "**e13055.e12.akamaiedge.net**" is a part of Akamai's CDN network. Several domains will be mapped to one single CDN host. Assuming a scenario where our SSH server is also behind the same IP and is part of the same network. It will allow us to tunnel the traffic to our SSH server through the whitelisted host www.disney.com as it is part of the same CDN. As discussed before, the host header containing our reflector address will be encrypted under the TLS packet and hence the captive portal, DPI or the censor will allow the traffic.

During our research, we discovered that several tools used for circumventing internet censorship such as Psiphon, Latern are using Akamai's CDN for handling their traffic. The idea is to host tons of reflectors in Akamai's CDN to get maximum mapping to different Akamai's hosts. Chances are that the whitelisted websites are also hosted on the same CDN host or may be served from the same frontal address and hence allowing them to bypass the traffic.

Akamai currently hosts more than 20% of all web traffic which leads to more than a 20% possibility that one of the whitelisted hosts belong to Akamai's CDN **[3].** Similarly, all Google services are behind the same IP address. If a captive portal allows access to Google.com and any other Google services, it will also allow access to Google's App engine (appspot.com) which effectively means that we can host our "reflector" on Google app engine and forward our traffic to bypass captive portals. A real-world scenario would be in the case of captive portals running adsense on its default pages. Google ads normally run from googlesyndication.com which would require it to be on the whitelist and hence will also allow appspot. It's also worth mentioning that the majority of other ad-networks utilize akamai's infrastructure to server their content.

The following screenshot demonstrates that the Kanban-chi.appspot.com domain can be served from several Google frontals.

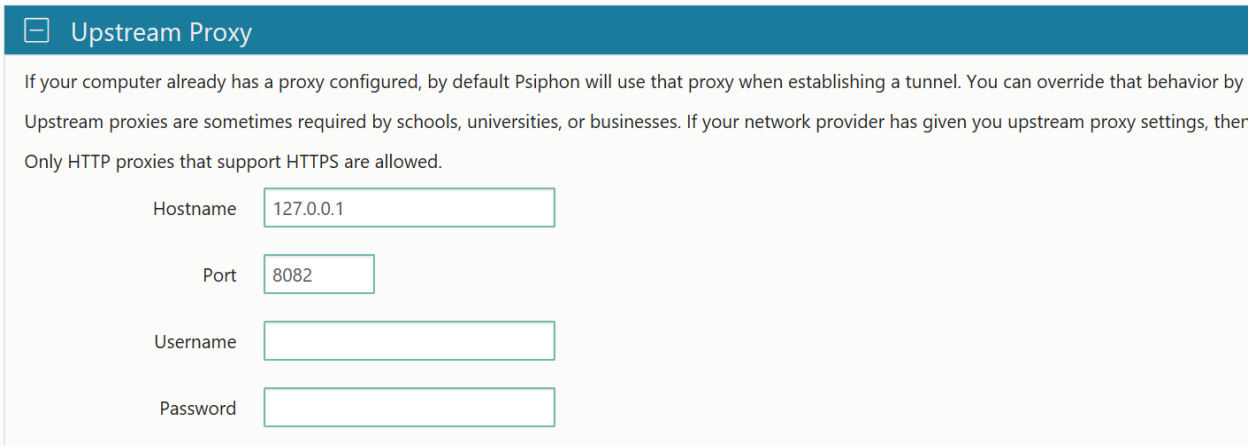



## 4. Psiphon Analysis

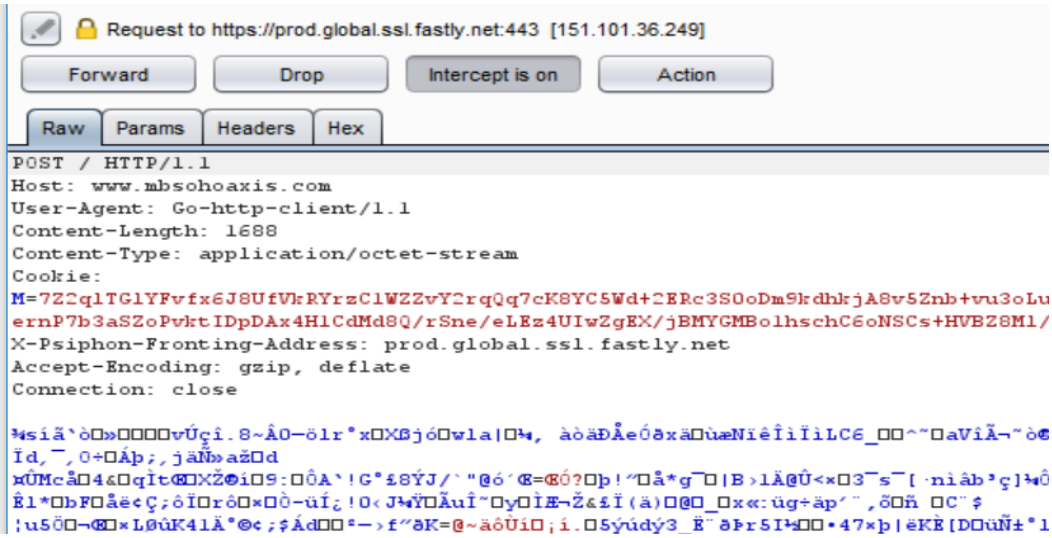
Psiphon is an internet censorship circumvention tool, which utilizes a combination of various techniques in order to circumvent censorship. Psiphon consists of multiple transport mechanisms such as SSH, VPN and HTTP Proxy for communication.

However, primarily it utilizes obfuscated SSH for communication which also helps Psiphon to hide its fingerprints from DPI's. Its network is geographically dispersed and consists of thousands of Proxy servers. To prevent enumeration of a large number of Psiphon servers that it connects to, Psiphon utilizes a mechanism known as "**Obfuscated Server List (OSL)**".

Psiphon happens to be one of the first implementations of the Domain Fronting technique and can be utilized to bypass several captive portals and censors. To understand the inner-working of Psiphon, we set a listener on Burp proxy at local port 8082 and configured Psiphon to use it as upstream proxy.




Psiphon starts by sending a request to "prod.global.ssl.fastly.net" which resolves to 151.101.36.249 as we are well aware that most of the websites hosted by fastly use "**151.101.36.249**" as a frontal. Therefore, there is a huge probability that it will be whitelisted. As you may also see from the screenshot below, Psiphon also adds an additional header i.e. X-Psiphon-Fronting-Address which points to **prod.global.ssly.fastly.net.** In case of the censor, these headers will not be visible and will be encapsulated inside SSL/TLS packets.




In case of fastly, edge server will not process if SNI and Host header are not matched. However, in that case, Frontless fronting can be utilized. Psiphon primarily utilizes fastly for hosting Obuscated Server List (OSL). [4]

OSL is a mechanism that is used to distribute servers' lists that are used to establish connection. The OSL only distributes server lists to the clients that satisfy certain conditions and prevents a single client to enumerate lists of all servers. As clients meet behavioral conditions they are seeded with "Server List

Obfuscation Keys" (SOK). These lists are stored and later concatenated and assembled in order to decrypt the OSL files which will contain the list of the servers.

The following request retrieves the server list from fastly's CDN:

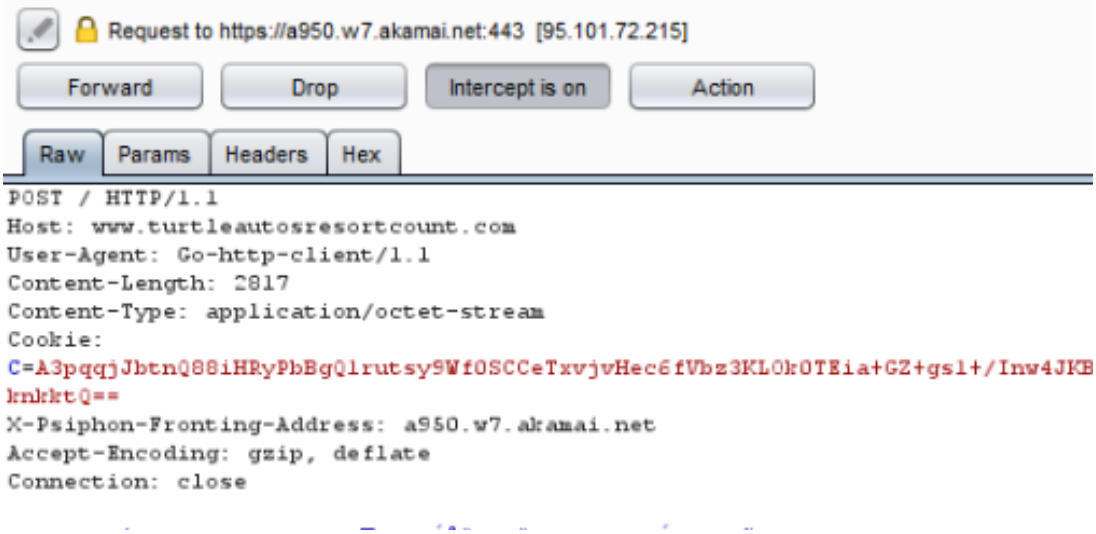

As mentioned before, most of the fastly subdomains are served from the same frontal i.e. 151.101.112.249.

```
root@kali-emc:~# host www.herbmxdiincorporated.com.global.prod.fastly.net
www.herbmxdiincorporated.com.global.prod.fastly.net has address 151.101.60.249
```

List of Identified Subdomains Serving OSL Files

https://www.partychildesmulti.com.global.prod.fastly.net/
www.herbmxdiincorporated.com.global.prod.fastly.net/
https://www.raceegguxdas.com.global.prod.fastly.net/
www.partychildesmulti.com.global.prod.fastly.net/

Once, the OSL server list is decrypted and extracted, Psiphon tries establishing a connection to these servers. Psiphon primarily tries establishing obfuscated SSH handshake, in case if fails it then tries VPN or HTTP(s) tunneling. Psiphon primarily utilizes Akamai servers to establish the tunnel. Psiphon utilizes port 22,53,80 and 443 for establishing a connection. Port 53, 80 and 443 are normally allowed for captive portals. The following screenshot demonstrates the utilize of Akamai's CDN as frontal to forward the traffic.




At times, we have also seen Psiphon using frontless fronting. For frontless fronting, Psiphon utilizes Akamai and Cloudflare CDN.


## 5. Bypassing DPI's & Captive Portals Using Poorly Used Regex

During my research, another issue I found was the poor use of regular expressions for matching whitelisted domains. This is more common with ISPs where DPIs are often utilized to manage bandwidth, packages and restrict content. A lot of service providers offer "**Social Media Bundles**" whereby they allow unlimited access to Facebook, Instagram etc. This is mostly managed on DPI whereby facebook.com, Instagram and other social media are added to the whitelist.

The problem arises when the whitelist is misconfigured, for instance, there was a specific instance whereby the service provider was using the following regular expression to match the twitter server.

**([\w\d]+\.)?twitter\.com**

As you may notice, the problem with the regular expression is that it does not contain an end of line sequence but will also match **twitter.com.pk**. In that case, an adversary can simply register any domain **twitter.com.pk** and deploy a proxy/reflector to forward all the traffic.


Regex missing end of sequence

Instead of registering twitter.com.pk domain, we can also choose to create subdomains for any domain we own. For instance, **twitter.com.rafaybaloch.com** which can also be used to host a reflector. The same applies to captive portals.

The correct way of matching a subdomain will be to use the end of line sequence which would be as follows:

**([\w\d]+\.)?twitter\.com$**


"$" in regex indicates end of line sequence

Another problem with the above regular expression (**([\w\d]+\.)?twitter\.com$**) is that it will match all the subdomains. In case of any one of the subdomains pointing to Akamai CDN, it will allow tools such as Psiphon already utilizing dozens of Akamai frontals to establish a connection.

## 6. Using Cross Site Scripting for Bypassing Captive Portals

Cross Site scripting (XSS) is still the most widely found vulnerability in a Web applications **[6]**. XSS occurs when the user supplied input is not properly filtered or sanitized before its reflected to the user. In case if any whitelisted website is vulnerable to a XSS vulnerability, it can be used to proxy traffic to any

blocked host as the traffic will appear to come from a whitelisted domain. This will only work if the outgoing request from a Whitelisted domain is not restricted on the portal.

One example is "**Stagecoach Greater Manchester**" captive portal which was vulnerable to a reflected XSS vulnerability. The vulnerability was found inside captive portal's script itself. The following POC injects an iframe into the webpage to load content from a website that is not a part of the whitelist.

**POC**

***https://portal.moovmanage.com/stagecoach-manchester/m/connect.php?res=tmgm"><iframe src="http://www.blocked.com">***

Another scenario is where the captive portal's pages are vulnerable to XSS. In that case we can use captive portal itself to proxy the traffic which will most likely be allowed as the traffic will appear to come from the captive portal itself instead of a client.

## 7. Whitelist Auditor

By now, it's safe to assume that the root cause of all the vulnerabilities is somehow related to a misconfigured whitelist. The larger the whitelist, the more chances that one of the domains might be hosted on a CDN and may allow tools like Psiphon, Latern or TOR to work. To help Captive Portal vendors and Firewall/DPI administrators, we have written a tool which will audit the whitelist for the following:

- **i)** The tool will check if the domain is part of a CDN network, this will be done by querying the Canonical Name Record (up to two levels) and will match if it points to a CDN network.
- **ii)** The tool also performs a reverse DNS lookup to check if an IP address is part of a CDN network.
- **iii)** The tool will check for poorly configured regular expressions that might lead to bypasses.

The tool consists of two different files "Auditor.py" and "sample.txt". The sample.txt file will be with whitelisted domains. The sample.txt must be placed in the same directory as the auditor.py. Once, the analysis is completed the tool will output an excel sheet, which will contain whitelisted domains that should be reviewed. The following is what the output will look like:

```
C:\Users\Pentest_User\Desktop\Cname>Auditor.py
Finding Canonical Domain Name Process  Started
Finding Canonical Domain Name Process  Finished
Started Updating results in Excel File
Finished Updating results in Excel File
```

| URL | CNAME(First Layer) | CNAME(Second Layer) |
|---|---|---|
| *.icservices.mtnsat.com | | |
| *.cellularatsea.com | | |
| *.wmsatsea.com | | |
| *.adisneycruise.disney.go.com | wdpr.vo.llnwd.net. | |
| *.bookwdw.reservations.disney.go.com | | |
| *.disneycruise.com | | |
| *.disneycruise.disney.go.com | origin.dcl.wdpro.disney.com. | disneycruise.disney.go.com-v1.edgekey.net. |
| http://clients3google.com/generate_204* | | |
| *.acrobits.cz | | |
| http://connectivitycheck.android.com/generate_204* | | |
| http://connectivitycheck.gstatic.com/generate_204* | | |
| *.disneyvacationclub.disney.go.com | | |
| *.cdn.trackjs.com | cdn.trackjs.netdna-cdn.com. | |
| *.capture.trackjs.com | | |
| http://www.apple.com/library/test/success.html* | | |
| *.analytics.disney.go.com | | |
| origin.dcl.wdpro.disney.com | disneycruise.disney.go.com-v1.edgekey.net. | e651.e12.akamaiedge.net. |
| http://clients3.google.com/generate_204* | clients.l.google.com. | |
| *.godaddy.com | | |
| *.shopdisney.com | | |

Domains highlighted in orange are poorly configured regular expressions, whereas the ones highlighted in red are a part of a CDN.

Please note that not all CDN hosts are utilized by circumvention tools, therefore the administrator needs to manually review the domains pointed.

**Whitelist Auditor Download Link - https://github.com/rafaybaloch/whitelist-auditor**

## Acknowledgements

The author is highly indebted to “Gowdhaman Mohan” for his assistance with tool. “Tamara Naudi” from Acunetix for proof reading, Farhan Azam Memon for and Muhammad Gazzaly for designing the cover.

## References

1. https://android-review.googlesource.com/#/q/topic:dns-dev-opt+(status:open+OR+status:merged
2. https://indico.dns-oarc.net/event/24/session/6/contribution/24/material/slides/0.pptx
3. http://www.akamai.com/dl/technical_publications/ network_overview_osr.pdf
4. https://github.com/Psiphon-Labs/psiphon-tunnel-core/psiphon/common/osl
5. https://medium.com/@0x0luke/stagecoach-greater-manchesters-wifi-login-portal-is-vulnerable-to-reflected-xss-leading-to-91c76a2bb1ea
6. https://www.acunetix.com/acunetix-web-application-vulnerability-report-2016/