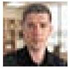
Spoofing JARM signatures. I am the Cobalt Strike server now!

By Stefan Grimminck

Published: 2020-12-25 · Archived: 2026-04-06 00:04:55 UTC



3 min read

Dec 25, 2020

TL;DR: JARM is very useful fingerprinting tool, but can be deceived by replaying server hello's from other services.

Press enter or click to view image in full size

The JARM scanner created by

is quite an effective tool for system fingerprinting. It uses the Server Hello responses from a TLS handshake to generate a signature. These can then be used to find similar software or services. Ideal for finding C2 or other malicious servers that implement TLS. So, It doesn't come as a surprise that Shodan.io uses this fingerprinting mechanism in their scanners. Read the Salesforce post for more information about the JARM library, scanner and its uses.

The question, then, arises: Is it possible to spoof these JARM signatures? Let's find out! Salesforce stated in their post that scanning a Cobalt Strike server would result in the following signature `07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1`

That this signature isn't Cobalt Strike specific, was revealed in the Cobalt Strike blog. Let's still use it as a starting point anyway.

First I used the list of addresses published by Salesforce to find a server with a matching hash. I scanned it using jarmscan and created a packet capture of the response. The ssl handshake (filter: `ssl.handshake.type == 1`) filter in Wireshark will display all TLS client hello's sent by the scanner.

Press enter or click to view image in full size

No.	Time	Sc	Source Port	Protocol	De	Length	Info
4	0.123392	…	64773	TLSv1.2	…	470	Client Hello
15	1.380676	…	64774	TLSv1.2	…	470	Client Hello
26	1.632238	…	64775	TLSv1.2	…	402	Client Hello
38	1.884997	…	64776	TLSv1.2	…	388	Client Hello
50	2.146878	…	64777	TLSv1.2	…	429	Client Hello
62	2.408787	…	64778	TLSv1.1	…	470	Client Hello
75	2.671207	…	64779	TLSv1.3	…	483	Client Hello
88	2.923828	…	64780	TLSv1.3	…	483	Client Hello
101	3.177630	…	64781	TLSv1.2	…	473	Client Hello
111	3.428488	…	64782	TLSv1.3	…	456	Client Hello

Wireshark capture of 10 TLS Client Hello's

And in turn the "Cobalt Strike" server will return its Server Hello's. These are used by jarmscan to generate a unique signature (filter: `ssl.handshake.type == 2`).

Press enter or click to view image in full size

No.	Time	Sc	Source Port	Protocol	De	Length	Info
6	0.245771	…	443	TLSv1.2	…	147	Server Hello
17	1.506351	…	443	TLSv1.2	…	147	Server Hello
28	1.758667	…	443	TLSv1.2	…	147	Server Hello
41	2.015779	…	443	TLSv1.2	…	147	Server Hello
52	2.278894	…	443	TLSv1.2	…	147	Server Hello
65	2.537956	…	443	TLSv1.1	…	147	Server Hello
78	2.797243	…	443	TLSv1.3	…	250	Hello Retry Request
90	3.050883	…	443	TLSv1.3	…	218	Hello Retry Request
114	3.553612	…	443	TLSv1.3	…	218	Hello Retry Request

Wireshark capture of 10 TLS Server Hello's

These Server Hello's are the packets we want to replay. This can easily be done by setting up a TCP server listening for the specific Client Hello's, then replaying their corresponding Server Hello's captured from the alleged Cobalt Strike server. A rather lazy, but effective approach.

Get Stefan Grimminck's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

I scanned the server on three separate occasions and found the duplicate bytes for every request. I used these bytes to identify each specific Client Hello.

Luckily Wireshark has an option to display packets as C Arrays. This made it pretty easy to get the Server Hello's working in my Golang spoofing application.

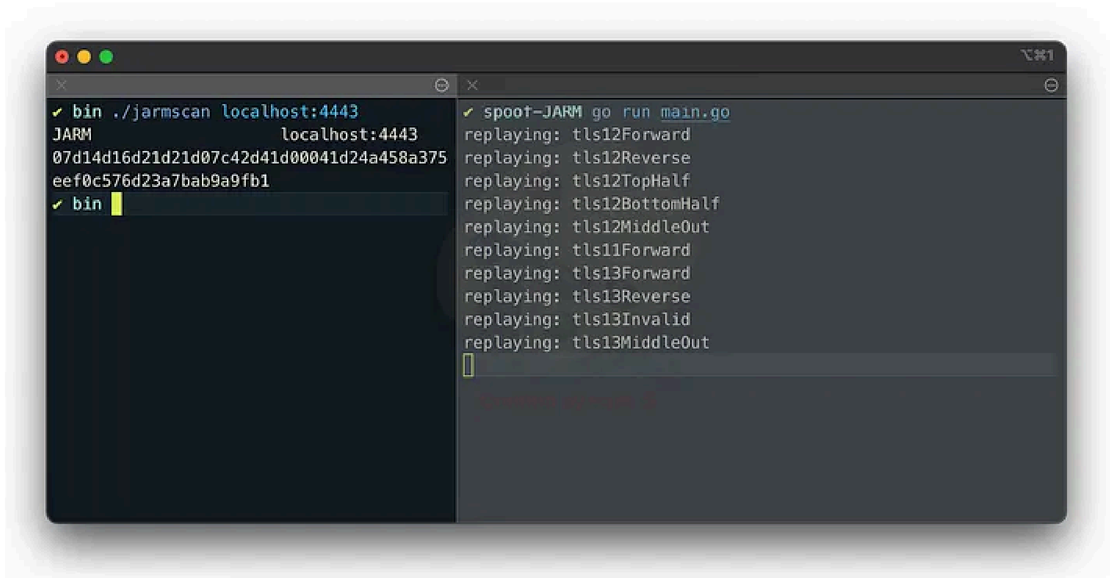
Press enter or click to view image in full size

By replaying these responses, slowly but steadily the fingerprint can be rebuilt.

```
if bytes.Contains(request, [] byte {
     0x00, 0x8c, 0x1a, 0x1a, 0x00, 0x16, 0x00, 0x33, 0x00,
     0x67, 0xc0, 0x9e, 0xc0, 0xa2, 0x00, 0x9e, 0x00, 0x39,
     0x00, 0x6b, 0xc0, 0x9f, 0xc0, 0xa3, 0x00, 0x9f, 0x00,
     0x45, 0x00, 0xbe, 0x00, 0x88, 0x00, 0xc4, 0x00, 0x9a,
     ... ...
 }) {
     fmt.Println("replaying: tls12Forward")
     conn.Write([] byte {
         0x16, 0x03, 0x03, 0x00, 0x5a, 0x02, 0x00, 0x00,
         0x56, 0x03, 0x03, 0x17, 0xa6, 0xa3, 0x84, 0x80,
         0x0b, 0xda, 0xbb, 0x3d, 0xe9, 0x3e, 0x92, 0x65,
         0x9a, 0x68, 0x7d, 0x70, 0xda, 0x00, 0xe9, 0x7c,
         ... ...
     })
 }
```

A full signature can be faked after implementing a reply for all ten different requests.

Press enter or click to view image in full size



(Mis)usage of spoofed signatures

You’re probably thinking: So what? What is the use of spoofed TLS fingerprints? They could be used by malicious actors to hide their applications when tools like JARM scanners are deployed to identify services in a network or on the internet. It can also be used for good. A honeypot replaying the fingerprint of a specific service can be used to setup a digital smokescreen for attackers.

Notes

- jarmscan (jarm-go) is not a product of Salesforce. They’ve published JARM a Python based JARM scanner implementation. Jarmscan (the scanner used here) is a Golang based implementation by @RumbleDiscovery

---

Source: https://grimminck.medium.com/spoofing-jarm-signatures-i-am-the-cobalt-strike-server-now-a27bd549fc6b