CommSec Track

HITB SECCONF AMSTERDAM - 2021

JARM Randomizer: Evading JARM Fingerprinting

Dagmawi Mulugeta

Threat Research Engineer, Netskope

Background

Currently a Threat researcher @ Netskope

Previously
- Researcher @ Cyrisk
- Software Engineer @ Sift Security
- Developer @ ECFMG

MSc in Cybersecurity from Drexel University

Interests: CTFs, exploit development, and cloud apps

Introduction

1) JA3 and JARM: two methods of SSL/TLS Fingerprinting

2) Why JARM is not reliable as a lone tool

3) Server side configurations tweaks result in different JARM fingerprints

4) Present JARM Randomizer, a tool to cycle through JARM fingerprints

1) What are JA3 and JARM?

How does SSL/TLS work?

TLS version and cipher suites are agreed to between client and server prior to any data exchange [1]

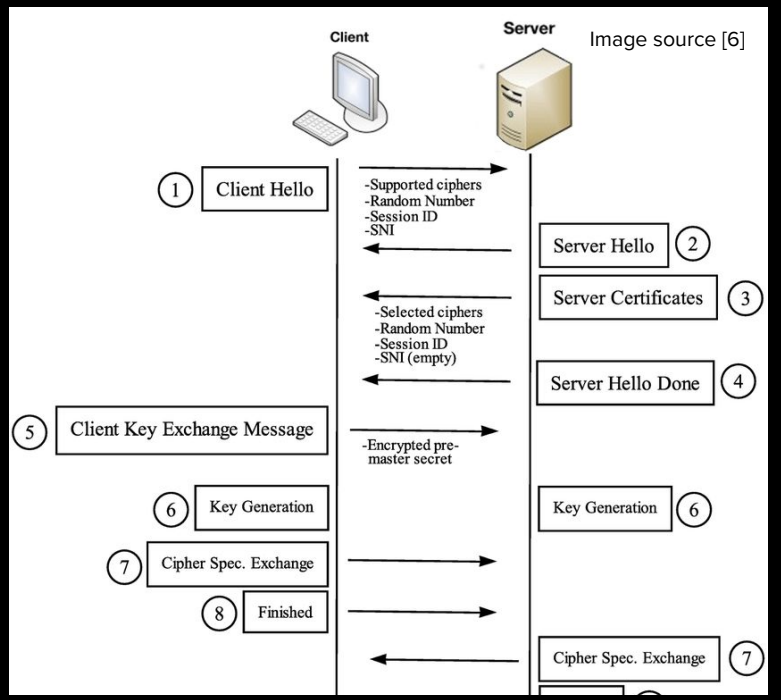


Image source [6]

JA3

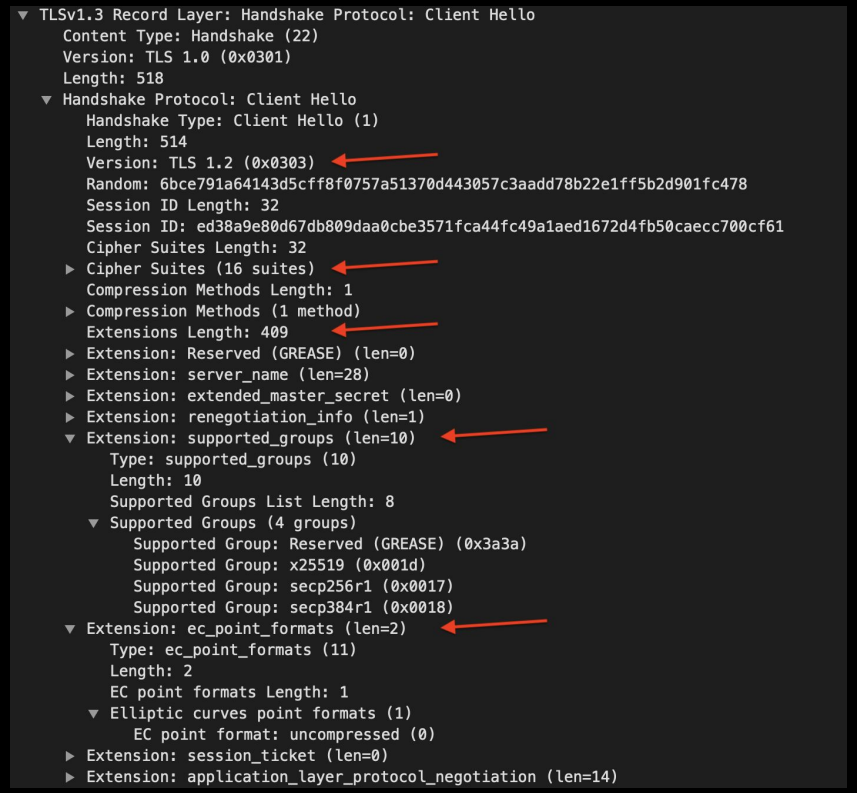
Introduced in 2017 by Salesforce. Found here

Fingerprint Client Hello in a TLS/SSL handshake

These fields are hashed as a fingerprint:

1. TLS Version
2. Ciphers
3. TLS Extensions
4. Supported Groups (Elliptic Curves)
5. Elliptic Curve Point Formats

Quite useful when identifying unusual clients in network



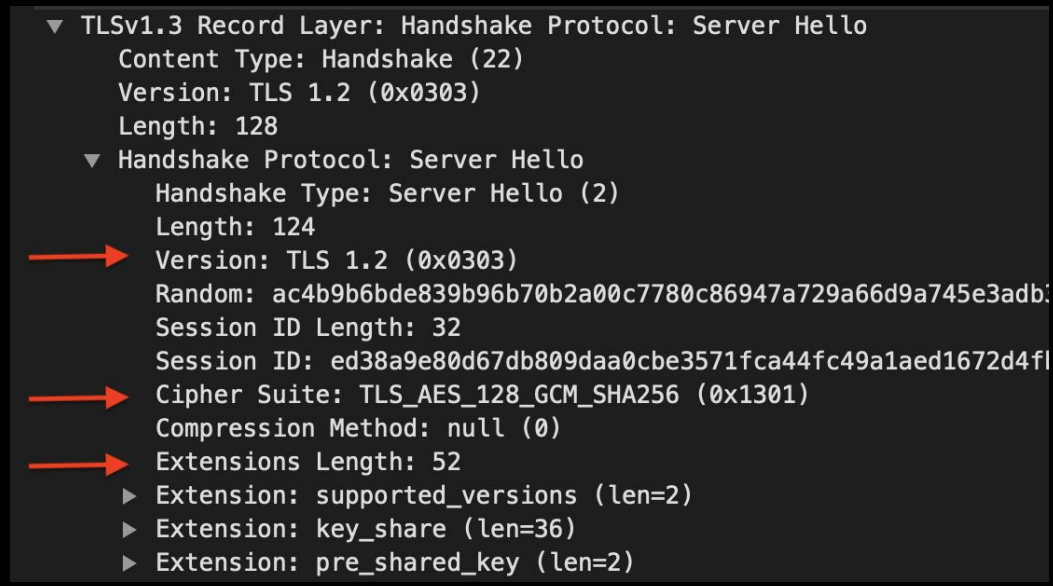
JARM

Introduced in 2020 by Salesforce.

Fingerprint Server Hello in a TLS/SSL handshake

Capture the server's responses:

1. TLS Version
2. Cipher chosen
3. TLS Extensions



JARM Technique

Send ten specially crafted TLS Client Hello packets

PyJARM implementation of Hellos seen below

Ciphers	Version	Cipher Order	GREASE	ALPNs	Support	Extension Order
All	1.1	Forward	No	All	None	Forward
All	1.2	Forward	No	All	1.2	Reverse
All	1.2	Reverse	No	All	1.2	Forward
All	1.2	Top Half	No	All	None	Forward
All	1.2	Bottom Half	No	Rare	None	Forward
All	1.2	Middle Out	Yes	Rare	None	Reverse
All	1.3	Forward	No	All	1.3	Reverse
All	1.3	Reverse	No	All	1.3	Forward
All	1.3	Middle Out	Yes	All	1.3	Reverse
Exclude 1.3	1.3	Forward	No	All	1.3	Forward

JARM Fingerprint

Fingerprint is consecutive 30-character and 32-character long blocks into one hash

- First half made of TLS versions and ciphers chosen to each ClientHello
- Second half represents a truncated SHA256 hash of the server-side extensions

Domain	JARM Fingerprint [4]
google.com	27d40d40d29d40d1dc42d43d00041d4689ee210389f4f6b4b5b1b93f92252d
youtube.com	27d40d40d29d40d1dc42d43d00041d4689ee210389f4f6b4b5b1b93f92252d
blogger.com	27d40d40d29d40d1dc42d43d00041d4689ee210389f4f6b4b5b1b93f92252d
facebook.com	27d27d27d29d27d1dc41d43d00041d741011a7be03d7498e0df05581db08a9
instagram.com	27d27d27d29d27d1dc41d43d00041d741011a7be03d7498e0df05581db08a9
oculus.com	29d29d20d29d29d21c41d43d00041d741011a7be03d7498e0df05581db08a9

Cipher picked and TLS version, SHA256 of TLS extensions

2) What are the challenges with JARM?

JARM weaknesses

Heavily dependent on [3]:

- Operating system and version
- Packages and libraries
- Other custom configurations

E.g., the JARM for Cobalt Strike, a popular red team tool, is actually the JARM for Java 11 TLS stack [5]

JARM + Other Intel	JARM as a lone tool
Useful to provide information around attacker infrastructure	Results in high FPs
Tough to evade when combined with other detections	Easily evadable via Proxy or Load Balancer

3) How do changes in server-side configuration affect JARM?

Configuration Changes

Tested on macOS using PyJARM[11] for fingerprinting

Used SSL in Python to cycle through the TLS Versions and Ciphers and fingerprinted the server

Result is a list of different JARMs

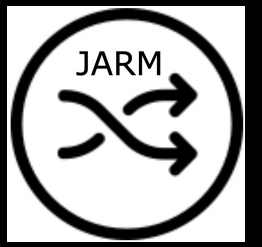
```
TLS --> 5, Cipher --> CAMELLIA128-SHA, JARM --> 0cd0cd0000cd0cd0000cd0cd0cd0cdcdfe7f0b77f33e9e6b7374a546c1af73
TLS --> 2, Cipher --> ECDHE-RSA-AES256-GCM-SHA384, JARM --> 2ad2ad0002ad2ad00042d42d000000ad9bf51cc3f5a1e29eecb81d0c7b06eb
TLS --> 2, Cipher --> AES128-GCM-SHA256, JARM --> 13d13d00013d13d00042d42d0000007320ccd9701dbccd7024a4f866f0cfd9
TLS --> 2, Cipher --> AES128-SHA, JARM --> 06d06d00006d06d06c42d42d000000b5e8d55ec3d127d54c131ba2d199aa34
TLS --> 2, Cipher --> CAMELLIA128-SHA, JARM --> 0cd0cd0000cd0cd0cc42d42d000000b5e8d55ec3d127d54c131ba2d199aa34
TLS --> 5, Cipher --> ECDHE-RSA-AES256-SHA, JARM --> 22d22d00022d22d00022d22d22d22d02098c5f1b1aef82f7daaf9fed36c4e8
TLS --> 5, Cipher --> AES128-SHA, JARM --> 06d06d00006d06d00006d06d06d06dcdfe7f0b77f33e9e6b7374a546c1af73
TLS --> 2, Cipher --> ECDHE-RSA-AES256-SHA, JARM --> 22d22d00022d22d22c42d42d000000faabb8fd156aa8b4d8a37853e1063261
TLS --> 2, Cipher --> AES128-SHA, JARM --> 06d06d00006d06d06c42d42d000000b5e8d55ec3d127d54c131ba2d199aa34
TLS --> 2, Cipher --> CAMELLIA256-SHA, JARM --> 10d10d00010d10d10c42d42d000000b5e8d55ec3d127d54c131ba2d199aa34
TLS --> 5, Cipher --> CAMELLIA128-SHA256, JARM --> 17d17d00017d17d00017d17d17d17dcdfe7f0b77f33e9e6b7374a546c1af73
```

4) How can we use this to evade JARM?

Evading SSL/TLS fingerprinting

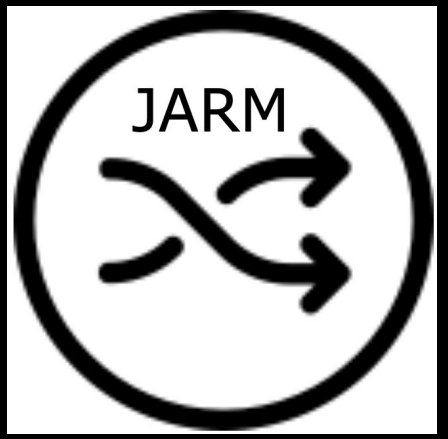
JA3Transport [9] : allowing offensive Go tools to make HTTPS requests using a custom fingerprint

JARM Randomizer: cycling through supported server-side configurations



JARM Randomizer

At the heart of it, it cycles through supported TLS version + Ciphers



Dependencies

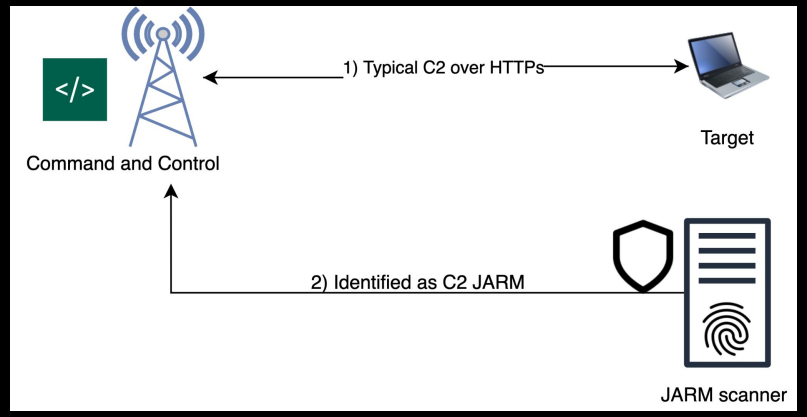
- Pipenv
- Python 3.9
- PyJARM
- Shodan
- Pybinaryedge

Current Features

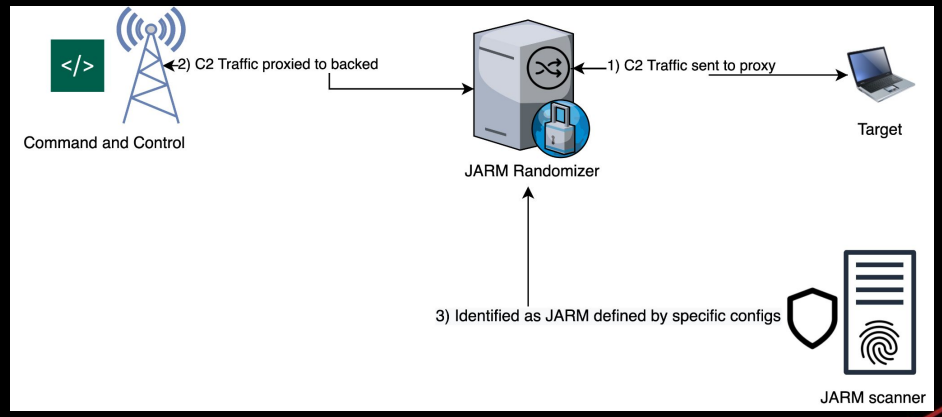
- Iterate and identify valid configurations during setup
- Query usage on BinaryEdge and Shodan
- Cross check against a red team tool list
- Cycle setting to rotate configs at specified intervals

Placement

- Python proxy that is placed in front of a C2 server
- Use this, alongside tools like Cobalt Strike, to evade fingerprinting of the tool itself



Without Randomizer



With Randomizer

Setup

Required step to identify supported configurations

Also, lay groundwork to run proxy

```
[x] Grabbing the list of ciphers that are supported on this system
[X] Finding all the possible JARMs
[x] Validating tls 2 and cipher AES128-GCM-SHA256
127.0.0.1 - - [11/May/2021 17:32:25] "GET /http://google.com HTTP/1.1" 200 -
...
[X] There are 27 possible JARMS across 70 TLS - Cipher pairs
[X] Grabbing the metrics for the JARMs...might take a while for long list of JARMs
[x] Run python3 ./main.py to start the proxy server
```

Analysis

From a macOS:

- 31 possible JARMS across 48 TLS - Cipher pairs
- Table below shows top 5 when sorted by occurrence in Shodan
- JARM for red team tools obtained from this repo[14]

jarm	binary edge	shodan	tls_cipher_pair_instances	red team tools	sample servers with similar JARM
2ad...4e8	3172	12176	1	0	IVFRT-NIC; Microsoft-IIS/8.0; Vault 1.0
2ad...6eb	477	47434	1	5	Apache; CradlepointHTTPService/1.0.0; EZproxy
29d...6eb	167	3130	1	0	AIS Streaming Server 9.0.2; Asterisk/13.36.0; Asterisk/16.14.1
29d...4e8	46	578	1	0	ReactPHP/1; Zattoo/20210413.121332
000...b64	37	553	18	0	Apache; Apache/2.4.29 (Ubuntu); Apache/2.4.37 (centos) OpenSSL/1.1.1c

Use cases

Serve with one preferred JARM

```
ubuntu@ip-172-31-41-225:~/jarm_randomizer$ pipenv run python3 ./main.py

[x] Selected configs: TLS -> 2, Cipher -> ECDHE-RSA-CHACHA20-POLY1305, JARM -> 3fd3fd0003fd3fd00042d42d000000ad9bf51cc3f5a1e29eecb81d0c7b06eb
[x] Server running on https://0.0.0.0:8443 forever...
```

Cycle through JARMs

```
ubuntu@ip-172-31-41-225:~/jarm_randomizer$ pipenv run python3 ./main.py

[x] Selected configs: TLS -> 2, Cipher -> AES256-SHA, JARM -> 08d08d00008d08d00042d42d0000007320ccd9701dbccd7024a4f866f0cfd9
[x] Cycle mode selected: server running on https://0.0.0.0:8443 for 5 secs

[x] Selected configs: TLS -> 5, Cipher -> AES256-SHA256, JARM -> 0bd0bd0000bd0bd0000bd0bd0bd0bdcdfe7f0b77f33e9e6b7374a546c1af73
[x] Cycle mode selected: server running on https://0.0.0.0:8443 for 5 secs

[x] Selected configs: TLS -> 2, Cipher -> ECDHE-RSA-AES256-SHA, JARM -> 22d22d00022d22d00042d42d000000ad9bf51cc3f5a1e29eecb81d0c7b06eb
[x] Cycle mode selected: server running on https://0.0.0.0:8443 for 5 secs
```

Limitations

1) Finite number of signatures
2) Client compatibility check required
3) C2 traffic can still be identified by other methods
4) Not exhaustive, but rather a step to fully evade fingerprinting

Recognition & Open Source

Salesforce Engineering [4] for JA3/JARM

CU Cyber[12] for working on JA3Transport

JARM Randomizer can be found here

Conclusion

Takeaways

1. JA3 and JARM: two methods of SSL/TLS Fingerprinting
2. JARM is not reliable as a lone tool to fingerprint servers
3. Server side configurations tweaks result in different JARM fingerprints
4. JARM Randomizer, a tool to cycle through JARM configurations

Keep an eye out on our blog for latest TLS Fingerprinting research & tools

Contact

Twitter: @dagmulu

Linkedin: dmulugeta

Github: jarm_randomizer

Future updates on our blog

Thank You

For your attention

References

[1] https://securitytrails.com/blog/jarm-fingerprinting-tool

[2] https://www.exoprise.com/2019/07/29/monitor-ssl-expiration-spoofing-changes/

[3] https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake/

[4] https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a

[5] https://blog.cobaltstrike.com/2020/12/08/a-red-teamer-plays-with-jarm/

[6] https://www.researchgate.net/figure/TLS-handshake-protocol_fig1_298065605

[7] https://medium.com/cu-cyber/impersonating-ja3-fingerprints-b9f555880e42

[8] https://tarshpartnership.co.uk/career-advice/interview-tips-whats-your-biggest-weakness/

[9] https://github.com/CUCyber/ja3transpor

[10] https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[11] https://github.com/PaloAltoNetworks/pyjarm

[12] https://cucyber.net/

[13] http://draw.io/

[14]https://github.com/cedowens/C2-JARM