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 \(^1\)
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

<table>
<thead>
<tr>
<th>Ciphers</th>
<th>Version</th>
<th>Cipher Order</th>
<th>GREASE</th>
<th>ALPNs</th>
<th>Support</th>
<th>Extension Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>1.1</td>
<td>Forward</td>
<td>No</td>
<td>All</td>
<td>None</td>
<td>Forward</td>
</tr>
<tr>
<td>All</td>
<td>1.2</td>
<td>Forward</td>
<td>No</td>
<td>All</td>
<td>1.2</td>
<td>Reverse</td>
</tr>
<tr>
<td>All</td>
<td>1.2</td>
<td>Reverse</td>
<td>No</td>
<td>All</td>
<td>1.2</td>
<td>Forward</td>
</tr>
<tr>
<td>All</td>
<td>1.2</td>
<td>Top Half</td>
<td>No</td>
<td>All</td>
<td>None</td>
<td>Forward</td>
</tr>
<tr>
<td>All</td>
<td>1.2</td>
<td>Bottom Half</td>
<td>No</td>
<td>Rare</td>
<td>None</td>
<td>Forward</td>
</tr>
<tr>
<td>All</td>
<td>1.2</td>
<td>Middle Out</td>
<td>Yes</td>
<td>Rare</td>
<td>None</td>
<td>Reverse</td>
</tr>
<tr>
<td>All</td>
<td>1.3</td>
<td>Forward</td>
<td>No</td>
<td>All</td>
<td>1.3</td>
<td>Reverse</td>
</tr>
<tr>
<td>All</td>
<td>1.3</td>
<td>Reverse</td>
<td>No</td>
<td>All</td>
<td>1.3</td>
<td>Forward</td>
</tr>
<tr>
<td>All</td>
<td>1.3</td>
<td>Middle Out</td>
<td>Yes</td>
<td>All</td>
<td>1.3</td>
<td>Reverse</td>
</tr>
<tr>
<td>Exclude 1.3</td>
<td>1.3</td>
<td>Forward</td>
<td>No</td>
<td>All</td>
<td>1.3</td>
<td>Forward</td>
</tr>
</tbody>
</table>
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
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]

<table>
<thead>
<tr>
<th>JARM + Other Intel</th>
<th>JARM as a lone tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Useful to provide information around attacker infrastructure</td>
<td>Results in high FPs</td>
</tr>
<tr>
<td>Tough to evade when combined with other detections</td>
<td>Easily evadable via Proxy or Load Balancer</td>
</tr>
</tbody>
</table>
3) How do changes in server-side configuration affect JARM?
## Configuration Changes

Tested on macOS using [PyJARM](#) 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

<table>
<thead>
<tr>
<th>TLS</th>
<th>Cipher</th>
<th>JARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
<tr>
<td>5</td>
<td>Cipher</td>
<td>CAMELLIA128-SHA, JARM</td>
</tr>
</tbody>
</table>
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
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]

<table>
<thead>
<tr>
<th>jarm</th>
<th>binary edge</th>
<th>shodan</th>
<th>tls_cipher_pair_instances</th>
<th>red team tools</th>
<th>sample servers with similar JARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ad...4e8</td>
<td>3172</td>
<td>12176</td>
<td>1</td>
<td>0</td>
<td>IVFRT-NIC; Microsoft-IIS/8.0; Vault 1.0</td>
</tr>
<tr>
<td>2ad...6eb</td>
<td>477</td>
<td>47434</td>
<td>1</td>
<td>5</td>
<td>Apache; CradlepointHTTPService/1.0.0; EZproxy</td>
</tr>
<tr>
<td>29d...6eb</td>
<td>167</td>
<td>3130</td>
<td>1</td>
<td>0</td>
<td>AIS Streaming Server 9.0.2; Asterisk/13.36.0; Asterisk/16.14.1</td>
</tr>
<tr>
<td>29d...4e8</td>
<td>46</td>
<td>578</td>
<td>1</td>
<td>0</td>
<td>ReactPHP/1; Zattoo/20210413.121332</td>
</tr>
<tr>
<td>000...b64</td>
<td>37</td>
<td>553</td>
<td>18</td>
<td>0</td>
<td>Apache; Apache/2.4.29 (Ubuntu); Apache/2.4.37 (centos) OpenSSL/1.1.1c</td>
</tr>
</tbody>
</table>
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 -> 3fd3fd0003fd3fd00042d42d000000ad9bf51cc3f5a1e29eeceb81d0c7b06eb
[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 -> 08d08d00008d08d00042d42d0000007320ccd9701dbcc0704a4f866f0cfd9
[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 -> 0bd0bd0000bd0bd0bd0bd0bdcdfe7f0b77f33e9e6b7374a546c1af73
[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 -> 22d22d00022d22d00042d42d000000ad9bf51cc3f5a1e29eeceb81d0c7b06eb
[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


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
[8] https://tarshpartnership.co.uk/career-advice/interview-tips-whats-your-biggest-weakness/
[9] https://github.com/CUCyber/ja3transpor
[12] https://cucyber.net/