A Red Teamer Plays with JARM

blog.cobaltstrike.com/2020/12/08/a-red-teamer-plays-with-jarm/

Raphael Mudge December 8, 2020

I spent a little time looking into Saleforce's JARM tool released in November. JARM is an active tool to probe the TLS/SSL stack of a listening internet application and generate a hash that's unique to that specific TLS/SSL stack.

One of the initial JARM fingerprints of interest relates to Cobalt Strike. The value associated with Cobalt Strike is:

07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1

To generate a JARM fingerprint for an application, use the JARM python tool:

python3 jarm.py [target] -p [port]

I opted to dig into this, because I wanted to get a sense of whether the fingerprint is Cobalt Strike or Java.

Cobalt Strike's JARM Fingerprint is Java's JARM Fingerprint

I started my work with a hypothesis: Cobalt Strike's JARM fingerprint is Java's JARM fingerprint. To validate this, I created a simple Java SSL server application (listens on port 1234) in Sleep.

```
import javax.net.*;
import javax.net.ssl.*;

$factory = [SSLServerSocketFactory getDefault];
$server  = [$factory createServerSocket: 1234];
[$server setSoTimeout: 0];

if (checkError($error)) {
warn($error);
}

while (true) {
$socket = [$server accept];
[$socket startHandshake];
[$socket close];
}
```

I ran this server from Java 11 with:

java -jar sleep.jar server.sl

I assessed its JARM fingerprint as:

00000000000000000042d41d00041d7a6ef1dc1a653e7ae663e0a2214cc4d9

Interesting! This fingerprint does not match the supposed Cobalt Strike fingerprint. Does this mean we're done? No.

The current popular use of JARM is to fingerprint web server applications listening on port 443. This implies that these servers have a certificate associated with their TLS communications. Does this change the above JARM fingerprint? Let's setup an experiment to find out.

I generated a Java keystore with a self-signed certificate and I directed my simple server to use it:

```
keytool -keystore ./exp.store -storepass 123456 -keypass 123456 -genkey -keyalg RSA -dname "CN=,OU=,O=,L=,S=,C="
java -Djavax.net.ssl.keyStore=./exp.store -Djavax.net.ssl.keyStorePassword=123456 -jar sleep.jar server.sl
```

The JARM result:

07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1

Interesting. We've validated that the above JARM fingerprint is specific to a Java 11 TLS stack.

Another question: is the JARM fingerprint affected by Java version? I setup several experiments and validated that yes, different major Java versions have different JARM fingerprints in the above circumstance.

How many Java-native Web servers are on the internet?

Part of the value of JARM is to turn the internet haystack into something smaller for an analyst to sift through. I wanted to get a sense of how much Java is on the internet. Fortunately, this analysis was easy thanks to some timely and available data. Silas Cutler had scanned the internet for port 443 and obtained JARM values for each of these hosts. This data was made available as an SQLite database too. Counting through this data was a relatively easy exercise of:

sqlite> .open jarm.sqlite
sqlite> select COUNT(ip) FROM jarm WHERE hash = “[hash here]”;

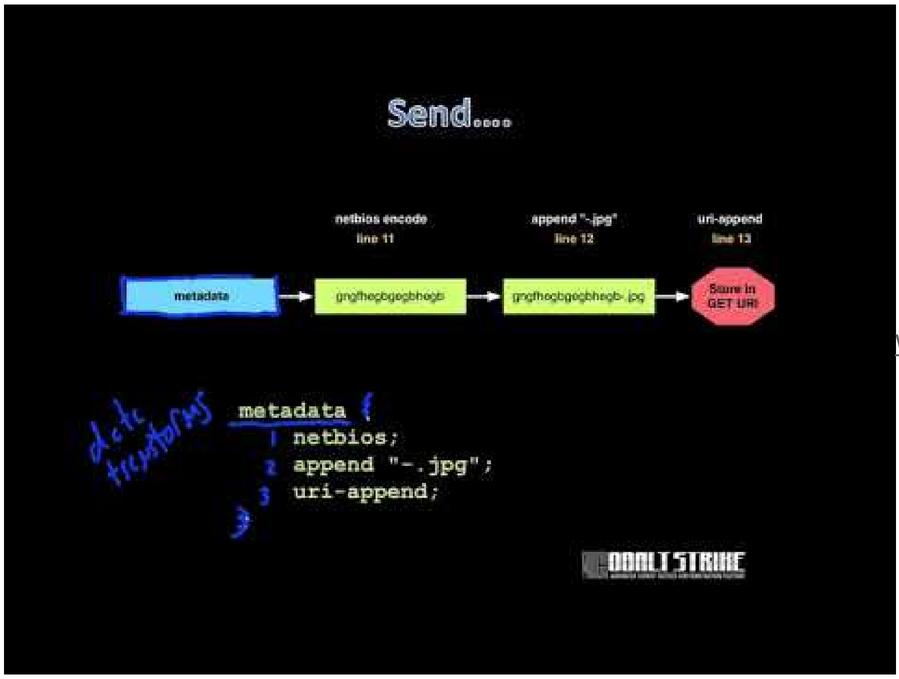
Here’s what I found digging through this data:

Application	Count	JARM Hash
Java 1.8.0	21,099	07d14d16d21d21d07c07d14d07d21d9b2f5869a6985368a9dec764186a9175
Java 1.9.0	9	05d14d16d04d04d05c05d14d05d04d4606ef7946105f20b303b9a05200e829
Java 11.05	2,957	07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1
Java 13.01	0	2ad2ad16d2ad2ad22c42d42d00042d58c7162162b6a603d3d90a2b76865b53

I went a slight step further with this data. I opted to convert the Java 11.05 data to hostnames and eyeball what appeared as interesting. I found several mail servers. I did not investigate which application they are. I found an instance of Burp Intruder (corroborating Salesforce’s blog post). I also found several instances of Oracle Peoplesoft as well. These JARM hashes are a fingerprint for Java applications, in general.

Closing Thoughts

For defenders, I wouldn’t act on a JARM value as proof of application identity alone. For red teamers, this is a good reminder to think about pro-active identification of command and control servers. This is a commoditized threat intelligence practice. If your blue team uses this type of information, there are a lot of options to protect your infrastructure. Part 3 of Red Team Operations with Cobalt Strike covers this topic starting at 1h 26m 15s:



```
metadata {
    netbios;
    append "-.jpg";
    uri-append;
}
```

Watch Video At:

https://youtu.be/Z8n9bIPAIao

JARM is a pretty cool way to probe a server and learn more about what it’s running. I’d love to see a database of JARM hashes and which applications they map to as a reconaissance tool. The C2 fingerprinting is a neat application of JARM too. It’s a good reminder to keep up on your infrastructure OPSEC.