STRRAT Attached to a MSI File

forensicitguy.github.io/strrat-attached-to-msi/

February 2, 2022

By *Tony Lambert*

Posted *2022-02-02* Updated *2022-03-28* *14 min* read

Adversaries can get really creative with ways to hide and execute payloads. In this post I'll cover one instance where an adversary appended STRRAT to a MSI file to make it look legitimate during analysis. If you want to follow along at home, the sample I'm working with is available in MalwareBazaar here: https://bazaar.abuse.ch/sample/0c3ff324e87c65c09f862f426d137206b5e895e70a85e6831a4aa5cc808a80be/.

Triaging the File

MalwareBazaar says the sample is a MSI file, so let's verify that claim using `file`, `diec`, and `xxd`.

```
remnux@remnux:~/cases/unk-msi$ file delivery.msi
delivery.msi: Composite Document File V2 Document, Little Endian, Os: Windows,
Version 6.1, MSI Installer, Code page: 1252, Title: Installation Database,
Subject: REPLACE, Author: REPLACE, Keywords: Installer, Comments: This installer
database contains the logic and data required to install REPLACE., Template:
Intel;1033, Revision Number: {00000000-0000-0000-0000-000000000000}, Number of
Pages: 200, Number of Words: 0, Security: 4, Create Time/Date: Wed Nov 23
16:25:04 2016, Last Saved Time/Date: Wed Nov 23 16:25:04 2016, Name of Creating
Application: Windows Installer XML v2.0.3719.0 (candle/light)

remnux@remnux:~/cases/unk-msi$ diec delivery.msi
filetype: Binary
arch: NOEXEC
mode: Unknown
endianess: LE
type: Unknown
  installer: Microsoft Installer(MSI)

remnux@remnux:~/cases/unk-msi$ xxd -C delivery.msi | head
00000000: d0cf 11e0 a1b1 1ae1 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 3e00 0400 feff 0c00  ........>.......
00000020: 0600 0000 0000 0000 0100 0000 0100 0000  ................
00000030: 0100 0000 0000 0000 0010 0000 0200 0000  ................
00000040: 0100 0000 feff ffff 0000 0000 0000 0000  ................
00000050: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000060: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000070: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000080: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000090: ffff ffff ffff ffff ffff ffff ffff ffff  ................
```

All of our checks conclude that the file indeed has magic bytes for a MSI installer file. We can also take a look at the file size using `ls` or `exiftool`.

```
remnux@remnux:~/cases/unk-msi$ ls -lh delivery.msi
-rw-r--r-- 1 remnux remnux 658K Feb  3  2022
delivery.msi
```

So the MSI file is about 658K in size. It's a decent enough size to have some executable content within, so let's tear into it.

Analyzing the MSI File

MSI files are structurally similar to MS Office 97-2003 documents. In fact, that's why the `file` command returns `Composite Document File V2 Document` for MSI files as they share the same magic bytes with documents. This is good news because it means we can use `oledump.py` to view and extract content from streams within the MSI file.

```
remnux@remnux:~/cases/unk-msi$ oledump.py
delivery.msi
  1:       528 '\x05SummaryInformation'
  2:       784 '軀臌脦臚蜓'
  3:      8553 '軀甄蘩䑽楠蟰蹕'
  4:      1100 '軀甄蘩䑽狦莚蹫'
  5:        36 '軀瓜栖簣躰'
  6:      2400 '軀盄朐筞蟰荥躅'
  7:        48 '軀竑繣椖篇䄻䍟䓘䋈䑿篗'
  8:        24 '軀竑繣痜䫋蕹䆖䆴蹭'
  9:        48 '軀竑蟋訦寘䕽䠏蕹䆖䆴蹭'
 10:        20 '軀簹蟰䖝嚇䐍䒻䈏蝶躰'
 11:        48 '軀簹蟰䖝蹭'
 12:        14 '軀䆉繬蹕'
 13:       132 '軀篏䌰蠍䟦'
 14:        64 '軀緊蕇䆴䡧'
 15:        30 '軀䋿䈹蠭蕇䡜'
 16:        20 '軀縩簣'
 17:       132 '軀䠢蠍朐槶篇䄻䍟䓘䋈䑿篗'
 18:        42 '軀䠢蠍朐瘬䫋蕹䆖䆴蹭'
 19:        12 '軀䡈筃䒞荥躅'
 20:        60 '軀䡈䔃䑂䑿䑽'
 21:        16 '軀䓑蕹䈓蹭'
 22:        56 '軀䕻䖧䕾䠍'
 23:        20 '軀䡑䕿䐔䇼䖽蹡'
```

Within the `oledump.py` output, you can ignore the stream names. After working with numerous MSI files in `oledump` I've found that having unreadable stream names is fairly common. Instead, let's focus on the first two columns. The left column is a stream number and the middle is a size value in bytes. From here, we can inspect the individual streams using `oledump`. To do this, work from the stream with the largest size value downward until you start encountering empty-ish streams. In this sample, we want to inspect streams 3, 6, 4, and 2. Stream 1 should always contain summary information about the MSI file.

```
remnux@remnux:~/cases/unk-msi$ oledump.py -a -s 3 delivery.msi | head
00000000: 4E 61 6D 65 54 61 62 6C  65 46 65 61 74 75 72 65
NameTableFeature
00000010: 46 65 61 74 75 72 65 5F  50 61 72 65 6E 74 54 69
Feature_ParentTi
00000020: 74 6C 65 44 65 73 63 72  69 70 74 69 6F 6E 44 69
tleDescriptionDi
00000030: 73 70 6C 61 79 4C 65 76  65 6C 44 69 72 65 63 74
splayLevelDirect
00000040: 6F 72 79 5F 41 74 74 72  69 62 75 74 65 73 41 70
ory_AttributesAp
00000050: 70 6C 69 63 61 74 69 6F  6E 44 65 73 6B 74 6F 70
plicationDesktop
00000060: 53 68 6F 72 74 63 75 74  50 72 6F 67 72 61 6D 53
ShortcutProgramS
00000070: 68 6F 72 74 63 75 74 55  70 67 72 61 64 65 43 6F
hortcutUpgradeCo
00000080: 64 65 55 70 67 72 61 64  65 4B 65 79 52 6F 6F 74
deUpgradeKeyRoot
00000090: 52 65 67 69 73 74 72 79  44 45 4C 45 54 45 5F 54
RegistryDELETE_T
```

In stream 3 we can observe string values that might be related to a configuration file or something else related to the installation process. Since this is our largest stream, this is already a bit problematic. If the installer contains EXE binary content, the binary will likely be larger than a text file. After dumping out this stream with `oledump.py -d` we can look at the contents and see that it doesn't contain any scripting commands either. This dumping process is easier to do than show in a post, so I'll leave that for a reader exercise :).

Where's the Malware???

After checking out the remaining large streams we can't find executable content in the MSI.

```
remnux@remnux:~/cases/unk-msi$ oledump.py -a -s 6 delivery.msi | head
00000000: 05 00 05 00 05 00 05 00  05 00 05 00 05 00 05 00
................
00000010: 11 00 11 00 11 00 11 00  11 00 11 00 11 00 14 00
................
00000020: 14 00 14 00 14 00 14 00  14 00 3D 00 3D 00 3D 00
..........=.=.=.
00000030: 3D 00 3D 00 3D 00 44 00  44 00 44 00 50 00 50 00
=.=.=.D.D.D.P.P.
00000040: 50 00 50 00 50 00 50 00  50 00 50 00 50 00 50 00
P.P.P.P.P.P.P.P.
00000050: 50 00 50 00 50 00 50 00  50 00 50 00 60 00 60 00
P.P.P.P.P.P.`.`.
```

```
00000060: 60 00 60 00 60 00 60 00  68 00 68 00 68 00 6A 00
`.`.`.`.h.h.h.j.
00000070: 6A 00 6A 00 73 00 73 00  73 00 87 00 87 00 87 00
j.j.s.s.s.......
00000080: 88 00 88 00 88 00 8A 00  8A 00 8C 00 8C 00 8C 00
................
00000090: 8C 00 8C 00 8C 00 8C 00  8C 00 90 00 90 00 90 00
................

remnux@remnux:~/cases/unk-msi$ oledump.py -a -s 4 delivery.msi | head
00000000: E9 FD 00 00 04 00 04 00  05 00 02 00 00 00 00 00
................
00000010: 00 00 00 00 07 00 16 00  0E 00 02 00 05 00 02 00
................
00000020: 0B 00 06 00 07 00 02 00  05 00 04 00 0A 00 06 00
................
00000030: 0A 00 08 00 0B 00 04 00  0F 00 03 00 0F 00 03 00
................
00000040: 0B 00 03 00 07 00 0F 00  03 00 02 00 04 00 02 00
................
00000050: 08 00 0F 00 0F 00 01 00  26 00 07 00 0E 00 02 00
........&.......
00000060: 06 00 02 00 08 00 06 00  0A 00 02 00 0A 00 02 00
................
00000070: 05 00 05 00 0A 00 08 00  0E 00 01 00 07 00 16 00
................
00000080: 0B 00 01 00 0B 00 03 00  0C 00 0C 00 0E 00 01 00
................
00000090: 0B 00 01 00 11 00 01 00  0E 00 01 00 0F 00 01 00
................

remnux@remnux:~/cases/unk-msi$ oledump.py -a -s 2 delivery.msi | head
00000000: 05 00 05 00 05 00 05 00  05 00 05 00 05 00 05 00
................
00000010: 11 00 11 00 11 00 11 00  11 00 11 00 11 00 14 00
................
00000020: 14 00 14 00 14 00 14 00  14 00 3D 00 3D 00 3D 00
..........=.=.=.
00000030: 3D 00 3D 00 3D 00 44 00  44 00 44 00 50 00 50 00
=.=.=.D.D.D.P.P.
00000040: 50 00 50 00 50 00 50 00  50 00 50 00 50 00 50 00
P.P.P.P.P.P.P.P.
00000050: 50 00 50 00 50 00 50 00  50 00 50 00 60 00 60 00
P.P.P.P.P.`.`.
00000060: 60 00 60 00 60 00 60 00  68 00 68 00 68 00 6A 00
`.`.`.`.h.h.h.j.
00000070: 6A 00 6A 00 73 00 73 00  73 00 87 00 87 00 87 00
j.j.s.s.s.......
00000080: 88 00 88 00 88 00 8A 00  8A 00 8C 00 8C 00 8C 00
................
00000090: 8C 00 8C 00 8C 00 8C 00  8C 00 90 00 90 00 90 00
................
```

Clearly there is some form of content here otherwise AV vendors wouldn't call the file malicious. Also, if we go back and look at the `oledump` output we can see the stream sizes, when summed up, fall significantly short of 658K. In situations like this I prefer to use `xxd` and `less` combined to browse the hex and ASCII side-by-side to look for patterns.

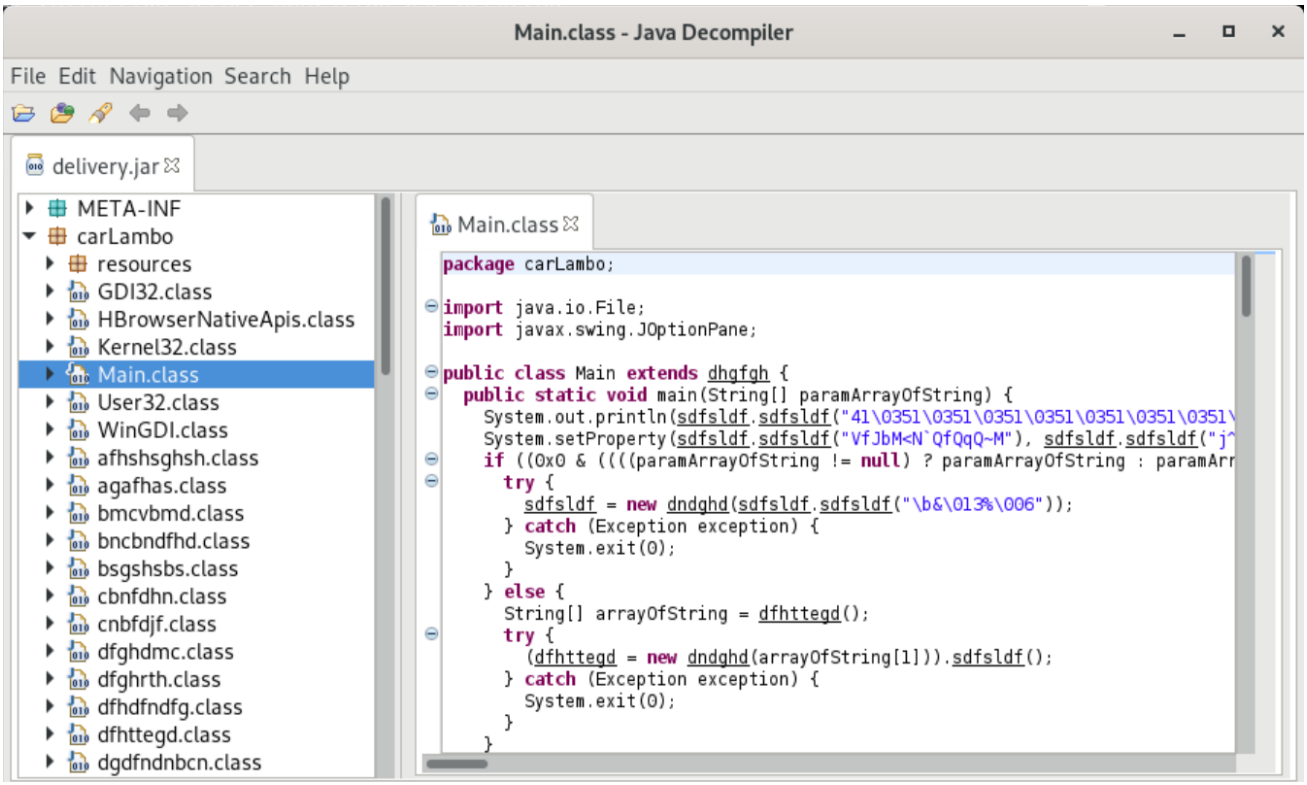
```
remnux@remnux:~/cases/unk-msi$ xxd -C delivery.msi | less
...

00027600: 4d5a 9000 0300 0000 0400 0000 ffff 0000
MZ..............
00027610: b800 0000 0000 0000 4000 0000 0000 0000
........@.......
00027620: 0000 0000 0000 0000 0000 0000 0000 0000
................
00027630: 0000 0000 0000 0000 0000 0000 0001 0000
................
00027640: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468
........!..L.!Th
00027650: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program
canno
00027660: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in
DOS
00027670: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000
mode....$.......
00027680: 532f 81a2 174e eff1 174e eff1 174e eff1
S/...N...N...N..

...
```

Using the search feature of `less` and the string “MZ” we found some executable content. We can try to unpack that in a minute after we get done browsing the bytes. Once we get toward the end of the MSI file we can see some more interesting strings.

```
000a4540: 6c61 7373 504b 0102 1400 1400 0808 0800
lassPK.........
000a4550: 8719 3f54 59cb 777b a40a 0000 ca11 0000  ..?
TY.w{........
000a4560: 1800 0000 0000 0000 0000 0000 0000 fd40
...............@
000a4570: 0100 6361 724c 616d 626f 2f64 6668 6466
..carLambo/dfhdf
000a4580: 6e64 6667 2e63 6c61 7373 504b 0102 1400
ndfg.classPK....
000a4590: 1400 0808 0800 8719 3f54 3ceb b0cc 3704  ........?
T<...7.
000a45a0: 0000 4b07 0000 1600 0000 0000 0000 0000
..K.............
000a45b0: 0000 0000 e74b 0100 6361 724c 616d 626f
.....K..carLambo
000a45c0: 2f78 6276 6378 6e78 2e63 6c61 7373 504b
/xbvcxnx.classPK
000a45d0: 0506 0000 0000 4100 4100 6c11 0000 6250
......A.A.l...bP
```

In the strings we can see some references to “carLambo” and “.class” files. These are significant findings because “.class” files are Java bytecode files. We can hypothesize that there is likely Java executable content attached to this MSI file. This sort of thing has happened before, in fact there was a bit of a hubbub back in 2019 when VirusTotal talked about it in a blog post. I also published one then, doing a bit of tinkering.

Long story short, you can append Java Archive (JAR) files to MSI files without hindering the operation of either file type. Java reads the JAR from end to front and `msiexec` reads from front to end. In fact, we can rename the MSI file to `delivery.jar` and view the Java contents using JD-GUI in REMnux.



Extracting the Java Class Files

As an alternate path to extract the Java class files, you can also use `binwalk`.

```
remnux@remnux:~/cases/unk-msi$ binwalk -e delivery.msi

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
...
85504         0x14E00         Microsoft executable, portable (PE)
...
161280        0x27600         Microsoft executable, portable (PE)
...
658644        0xA0CD4         Zip archive data, at least v2.0 to extract, name:
carLambo/dhgfgh.class
661635        0xA1883         Zip archive data, at least v2.0 to extract, name:
carLambo/ngdnbn.class
664267        0xA22CB         Zip archive data, at least v2.0 to extract, name:
carLambo/xbxcv.class
664829        0xA24FD         Zip archive data, at least v2.0 to extract, name:
carLambo/dfhdfndfg.class
667623        0xA2FE7         Zip archive data, at least v2.0 to extract, name:
carLambo/xbvcxnx.class
673230        0xA45CE         End of Zip archive, footer length: 22
```

By taking this approach the `binwalk` tool also extracts the relevant EXE files.

```
remnux@remnux:~/cases/unk-msi/_delivery.msi.extracted$ ll
total 220
drwxrwxr-x 4 remnux remnux  4096 Feb  2 22:25 ./
drwxrwxr-x 3 remnux remnux  4096 Feb  2 22:25 ../
-rw-rw-r-- 1 remnux remnux  5120 Apr  7  2014
_75C053BA9648745475BB4EE6B8141822
-rw-rw-r-- 1 remnux remnux 96740 Feb  2 22:25 8CC00.cab
-rw-rw-r-- 1 remnux remnux 90596 Feb  2 22:25 8E400.zip
drwxrwxr-x 3 remnux remnux  4096 Feb  2 22:28 carLambo/
-rw-rw-r-- 1 remnux remnux  4662 Apr  7  2014
_CFADD94CF3B345E684381C7F3A1EEE82
drwxrwxr-x 2 remnux remnux  4096 Feb  2 22:28 META-INF/
```

From here we can work with the extracted files as we please! The executable content in `_75C053BA9648745475BB4EE6B8141822` isn't malicious and seems to just be part of the MSI "background noise" here. Under the carLambo folder we can browse and decompile class files as desired using JD-GUI, `cfr`, or another tool.

```
remnux@remnux:~/cases/unk-msi/_delivery.msi.extracted/carLambo$ cfr Main.class |
head
/*
 * Decompiled with CFR 0.149.
 */
package carLambo;

import carLambo.dhgfgh;
import carLambo.dndghd;
import carLambo.fgssdg;
import carLambo.sbsbgsrg;
import carLambo.sdfsldf;
```

How do we know it's STRRAT??

Alright, the first piece of evidence that this malware is STRRAT is that the entire Java package inside the JAR file is named "carLambo". Googling `"carLambo" java` gives you loads of sandbox reports and some analysis reports pointing to STRRAT.

The next piece of evidence is the structure of the JAR file. There is a `resources/config.txt` file. This is consistent with STRRAT. In addition, despite other classes being obfuscated, there are deobfuscated classes referring to Windows USER32, WinGDI, Kernel32, and HBrowserNativeApis keywords. These are also consistent with STRRAT based on documentation from Jai Minton.

Alright, that's where I end tonight so I can go to bed. Hopefully in the next few days I'll put together another post on analyzing this STRRAT sample some more so we can get its C2 address and such. Thanks for reading!