njRAT Installed from a MSI

forensicitguy.github.io/njrat-installed-from-msi/

February 3, 2022

By *Tony Lambert*
Posted *2022-02-03* Updated *2022-03-28* *13 min* read

In my last post I walked through the analysis of an unusual MSI file that an adversary had tacked a STRRAT Java ARchive file to the end of the MSI contents. In this post, I want to walk through a more normal MSI sample that an adversary designed to delivery njRAT. If you want to follow along at home, the sample I'm working with is in MalwareBazaar here: https://bazaar.abuse.ch/sample/1f95063441e9d231e0e2b15365a8722c5136c2a6fe2716f3653c260093026354/.

Triaging the File

As usual, let's get started triaging with `file` and `diec`.

```
remnux@remnux:~/cases/njrat-msi$ file mal.msi
mal.msi: Composite Document File V2 Document, Little Endian, Os: Windows,
Version 10.0, MSI Installer, Code page: 1252, Title: Microsoft Visual Studio -
UNREGISTERED - Wrapped using MSI Wrapper from www.exemsi.com 16.6.255.35071,
Subject: Microsoft Visual Studio - UNREGISTERED - Wrapped using MSI Wrapper from
www.exemsi.com, Author: Microsoft Corporation, Keywords: Installer, Template:
x64;1033, Revision Number: {49C681E5-45C4-4467-92EE-456F1E355C5F}, Create
Time/Date: Sun Feb  7 22:37:14 2021, Last Saved Time/Date: Sun Feb  7 22:37:14
2021, Number of Pages: 200, Number of Words: 2, Name of Creating Application:
MSI Wrapper (10.0.50.0), Security: 2

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

From the output it looks like the sample indeed has the magic bytes for a MSI. From the file output, it looks like the adversary may have used an unlicensed MSI Wrapper tool from "exemsi[.]com". This is pretty common, there are multiple free and paid tools to create MSI files and I’ve seen samples where adversaries would essentially download trials from multiple vendors and switch vendors between campaigns. Let’s dive into the MSI contents!

Analyzing the MSI Contents

Just like in the last post, we can use `oledump.py` to view the content streams within this MSI.

```
remnux@remnux:~/cases/njrat-msi$ oledump.py
mal.msi
  1:       136 '\x05DocumentSummaryInformation'
  2:       588 '\x05SummaryInformation'
  3:    669935 '綹枋竤稭逺蔆葱篘甂蝕菥萿韆秊躍'
  4:    212992 '綹枋竤稭逺蠹蠍腤穙绌艜肓蹫'
  5:       672 '軄膷腇膻蚷'
  6:      8555 '軄甀蘮䑞桶蟰躍'
  7:      1216 '軄甀蘮䑞㺢㖋蹫'
  8:        38 '軄瓬秵篢躾'
  9:      2064 '軄盁肕筞蟰荥蠾'
 10:         4 '軄褚艅緟敁筃蘷荥蠾'
 11:        48 '軄竑繙桄篃褆簬篌椕舦篗'
 12:        24 '軄竑繙瘵颺蕥觧窽蹽'
 13:        42 '軄竑蝼訋寘蠒瓾蕥觧窽蹽'
 14:         4 '軄簹蟰蘳㘞腤菋箈蝶躾'
 15:        16 '軄簹蟰蘳蹽'
 16:        14 '軄穑纛躍'
 17:        60 '軄籨繝蠍竤'
 18:         8 '軄綹枋竤'
 19:        18 '軄縺翁蝰蘄艌'
 20:       216 '軄翍蠍肕槶篃褆簬篌椕舦篗'
 21:        48 '軄翍蠍肕瘬颺蕥觧窽蹽'
 22:        12 '軄敁堲艜舦䑞'
 23:        32 '軄莀蘀窮蹽'
 24:        80 '軄萿蓅藤蔢'
 25:       180 '軄蠹蠍腤穙绌艜'
```

Don’t worry about the stream names being unreadable, that’s a common thing in the MSI files I’ve seen. We want to focus on the first two columns. The left column is the stream number and the middle is the size of the stream contents in bytes. We want to analyze the largest streams to the smallest until we start finding streams with no workable data. In this sample, we want to work with streams 3, 4, 6, 7, and 9.

```
remnux@remnux:~/cases/njrat-msi$ oledump.py -a -s 3 mal.msi | head
00000000: 4D 53 43 46 00 00 00 00  EF 38 0A 00 00 00 00 00
MSCF.....8......
00000010: 2C 00 00 00 00 00 00 00  03 01 01 00 01 00 00 00
,...............
00000020: 9B 8E 00 00 47 00 00 00  15 00 00 00 00 38 0A 00
....G........8..
00000030: 00 00 00 00 00 00 3C 54  57 80 20 00 73 65 72 76  ......<TW.
.serv
00000040: 65 72 2E 65 78 65 00 99  0A 33 F0 00 80 00 80 4D
er.exe...3.....M
00000050: 5A 90 00 03 00 00 00 04  00 00 00 FF FF 00 00 B8
Z...............
00000060: 00 00 00 00 00 00 00 40  00 00 00 00 00 00 00 00
.......@........
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
................
00000080: 00 00 00 00 00 00 00 00  00 00 00 80 00 00 00 0E
................
00000090: 1F BA 0E 00 B4 09 CD 21  B8 01 4C CD 21 54 68 69
.......!..L.!Thi
```

In stream 3 we can see the first bytes of content contain the ASCII characters `MSCF`. This is consistent with Cabinet Archive (CAB) files. We can dump out the stream and confirm this with `file`.

```
remnux@remnux:~/cases/njrat-msi$ oledump.py -d -s 3 mal.msi > 3.dat

remnux@remnux:~/cases/njrat-msi$ file 3.dat
3.dat: Microsoft Cabinet archive data, Windows 2000/XP setup, 669935 bytes, 1
file, at 0x2c +A "server.exe", ID 36507, number 1, 21 datablocks, 0x0
compression
```

Sure enough, it looks like we’ve dumped out a CAB file. We’ll get to that in a bit. Let’s finish looking through the other streams.

```
remnux@remnux:~/cases/njrat-msi$ oledump.py -a -s 4 mal.msi | head
00000000: 4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00
MZ..............
00000010: B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00
........@.......
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00
................
00000040: 0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68
........!..L.!Th
00000050: 69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program
canno
00000060: 74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in
DOS
00000070: 6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00
mode....$.......
00000080: FE AE 1E EC BA CF 70 BF  BA CF 70 BF BA CF 70 BF
......p...p...p.
00000090: B3 B7 F4 BF FA CF 70 BF  B3 B7 E5 BF AF CF 70 BF
......p.......p.
```

Stream 4 looks like it contains some executable data with a `MZ` header and DOS stub. We can dump that out and continue.

```
remnux@remnux:~/cases/njrat-msi$ oledump.py -a -s 6 mal.msi | head
00000000: 4E 61 6D 65 54 61 62 6C  65 54 79 70 65 43 6F 6C
NameTableTypeCol
00000010: 75 6D 6E 5F 56 61 6C 69  64 61 74 69 6F 6E 56 61
umn_ValidationVa
00000020: 6C 75 65 4E 50 72 6F 70  65 72 74 79 49 64 5F 53
lueNPropertyId_S
00000030: 75 6D 6D 61 72 79 49 6E  66 6F 72 6D 61 74 69 6F
ummaryInformatio
00000040: 6E 44 65 73 63 72 69 70  74 69 6F 6E 53 65 74 43
nDescriptionSetC
00000050: 61 74 65 67 6F 72 79 4B  65 79 43 6F 6C 75 6D 6E
ategoryKeyColumn
00000060: 4D 61 78 56 61 6C 75 65  4E 75 6C 6C 61 62 6C 65
MaxValueNullable
00000070: 4B 65 79 54 61 62 6C 65  4D 69 6E 56 61 6C 75 65
KeyTableMinValue
00000080: 49 64 65 6E 74 69 66 69  65 72 4E 61 6D 65 20 6F
IdentifierName o
00000090: 66 20 74 61 62 6C 65 4E  61 6D 65 20 6F 66 20 63  f tableName
of c

remnux@remnux:~/cases/njrat-msi$ oledump.py -a -s 7 mal.msi | head
00000000: 00 00 00 00 04 00 06 00  05 00 02 00 00 00 00 00
................
```

```
00000010: 04 00 02 00 06 00 02 00  0B 00 15 00 05 00 05 00
................
00000020: 01 00 2C 00 0A 00 01 00  13 00 02 00 0B 00 06 00
..,.............
00000030: 03 00 02 00 08 00 02 00  09 00 02 00 08 00 02 00
................
00000040: 08 00 02 00 08 00 02 00  08 00 02 00 0A 00 19 00
................
00000050: 0D 00 01 00 0E 00 01 00  03 00 01 00 1E 00 01 00
................
00000060: 01 00 2A 00 15 00 01 00  15 00 01 00 36 00 01 00
..*.........6...
00000070: 24 00 01 00 F5 00 01 00  0F 00 01 00 04 00 09 00
$...............
00000080: 20 00 01 00 15 00 01 00  14 00 07 00 06 00 0C 00
...............
00000090: 42 00 05 00 09 00 15 00  9F 00 05 00 08 00 0C 00
B...............

remnux@remnux:~/cases/njrat-msi$ oledump.py -a -s 9 mal.msi | head
00000000: 06 00 06 00 06 00 06 00  06 00 06 00 06 00 06 00
................
00000010: 06 00 06 00 0A 00 0A 00  22 00 22 00 22 00 29 00
........".".".).
00000020: 29 00 29 00 2A 00 2A 00  2A 00 2B 00 2B 00 2F 00
).).*.*.*.+.+./.
00000030: 2F 00 2F 00 2F 00 2F 00  2F 00 35 00 35 00 35 00
/./././././.5.5.5.
00000040: 3D 00 3D 00 3D 00 3D 00  3D 00 4D 00 4D 00 4D 00
=.=.=.=.=.M.M.M.
00000050: 4D 00 4D 00 4D 00 4D 00  4D 00 5C 00 5C 00 61 00
M.M.M.M.M.\.\.a.
00000060: 61 00 61 00 61 00 61 00  61 00 61 00 61 00 6F 00
a.a.a.a.a.a.a.o.
00000070: 6F 00 72 00 72 00 72 00  73 00 73 00 73 00 74 00
o.r.r.r.s.s.s.t.
00000080: 74 00 77 00 77 00 77 00  77 00 77 00 77 00 82 00
t.w.w.w.w.w.w...
00000090: 82 00 86 00 86 00 86 00  86 00 86 00 86 00 90 00
................
```

Streams 6, 7, and 9 have either some string data or not much recognizable contents. If we start running into issues, dumping stream 6 might be a decent idea to see if there are scripting commands within, but that's not necessary right now.

Extracting the CAB File

Extracting the contents of the CAB is really easy. Just use `7z`. Extracting the contents unpacks `server.exe`, which appears to be a .NET binary.

```
remnux@remnux:~/cases/njrat-msi$ 7z x 3.dat
Extracting archive: 3.dat
--
Path = 3.dat
Type = Cab
Physical Size = 669935
Method = None
Blocks = 1
Volumes = 1
Volume Index = 0
ID = 36507

Everything is Ok

Size:       669696
Compressed: 669935

remnux@remnux:~/cases/njrat-msi$ file server.exe
server.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS
Windows

remnux@remnux:~/cases/njrat-msi$ diec server.exe
filetype: PE32
arch: I386
mode: 32-bit
endianess: LE
type: GUI
  library: .NET(v4.0.30319)[-]
  compiler: VB.NET(-)[-]
  linker: Microsoft Linker(48.0)[GUI32]
```

The final step for this branch of analysis will be to decompile the .NET malware to its source. For this, I like to use `ilspycmd`.

```
remnux@remnux:~/cases/njrat-msi$ ilspycmd server.exe >
server.decompiled.cs

remnux@remnux:~/cases/njrat-msi$ head server.decompiled.cs
using System;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Diagnostics;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
```

Sure enough, it looks like we got some readable C# code!

What about that other EXE/DLL?

The other DLL we pulled from stream 4 might still be relevant, so let’s look into it. We can get a pretty good idea of the DLL’s functionality using a combination of `pedump` and strings from `floss`.

```
remnux@remnux:~/cases/njrat-msi$ pedump --exports 4.dat

=== EXPORTS ===

# module "MsiCustomActions.dll"
# flags=0x0  ts="2021-02-07 22:37:10"  version=0.0
ord_base=1
# nFuncs=10  nNames=10

  ORD ENTRY_VA  NAME
    1     a5d0  _CheckReboot@4
    2     a510  _InstallFinish1@4
    3     a740  _InstallFinish2@4
    4     a9d0  _InstallMain@4
    5     a4a0  _InstallPrepare@4
    6     abc0  _InstallRollback@4
    7     ac80  _SubstWrappedArguments@4
    8     b280  _UninstallFinish1@4
    9     b6e0  _UninstallFinish2@4
    a     ac90  _UninstallPrepare@4
```

The exported functions in the DLL look like they might be related to generic installation activity. In addition, the DLL thinks it has a module name of `MsiCustomActions.dll`. Nothing really stands out as suspicious, let’s take a look at output from `floss` that has been ranked with `stringsifter`.

```
remnux@remnux:~/cases/njrat-msi$ floss -q 4.dat | rank_strings >
ranked_floss.txt
remnux@remnux:~/cases/njrat-msi$ less ranked_floss.txt

files.cab
C:\ss2\Projects\MsiWrapper\MsiCustomActions\Release\MsiCustomActions.pdb
- UNREGISTERED - Wrapped using MSI Wrapper from www.exemsi.com
SOFTWARE\EXEMSI.COM\MSI Wrapper
-R files.cab -F:* files
msiwrapper.ini
cmd.exe
SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
msi.dll
Error setting security.
Remove cabinet file
QuietUninstallString is
UninstallString is
Protection failed.
Removal of protection failed.
Focus is
SELECT `Data` FROM `Binary` WHERE `Name` = '%s'
ShellExecuteEx failed (%d).
Error setting security. Exit code %d.
...
```

There are loads of strings in this binary that seem consistent with being an installation component. The debugging PDB file is named with a MSI-related path. The vendor of the MSI Wrapper is mentioned in the DLL as well. It would be nice if the binary was signed, but we can’t always get what we want.

Wrapping up, if you want to dive deeper into that njRAT server.exe process, start with the decompiled code output from `ilspycmd` and have fun. Thanks for reading!