Analyzing a Stealer MSI using msitools

Published: 2022-02-12 · Archived: 2026-04-05 14:28:56 UTC

This post is dedicated to Josh Rickard (@MSAdministrator on Twitter) since his feedback on my blog posts has cut my triage time on MSI files down in a massive way! After writing an analysis of a MSI payload distributing njRAT, Josh hit me up on Twitter to suggest a Python tool he made to analyze MSIs, `msi-utils` with the caveat that it only worked on macOS. I set off to figure out why it only worked on macOS and, long story short, the journey led me to the `msitools` package on Linux. I'll use it in this post to analyze this sample in MalwareBazaar: https://bazaar.abuse.ch/sample/1f7830f0117f694b87ae81caed022c82174f9a8d158a0b8e127154e17d1600cc/.

> 2022-02-14 Edit: `msitools` is now included in REMnux! To get it run `remnux upgrade`.

The `msitools` package isn't installed by default in REMnux, so we have to go install it ourselves. This is easily done using `apt`.

```
sudo apt update
sudo apt install msitools
```

Once the package is installed, we can move on to ripping apart MSI files!

Triage the MSI Sample

The Detect-It-Easy and `file` output confirm we do have a MSI file.

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

remnux@remnux:~/cases/arkei-msi$ file po.msi
po.msi: Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, MSI Installer, Code page: 1252, Title
```

More specifically, the data from `file` indicates the MSI file was created by a tool named "MSI Wrapper (10.0.50.0)". The tool is likely the one from this web site: https://www.exemsi.com/

Enumerating MSI Tables and Streams

We can start the analysis using `msiinfo` to get some information about the file. We definitely want to know what table and stream structures we can expect within the MSI.

```
remnux@remnux:~/cases/arkei-msi$ msiinfo tables po.msi
_SummaryInformation
_ForceCodepage
_Validation
AdminExecuteSequence
AdminUISequence
AdvtExecuteSequence
Binary
Component
Directory
CustomAction
Feature
```

```
FeatureComponents
File
Icon
InstallExecuteSequence
InstallUISequence
LaunchCondition
Media
Property
Registry
Upgrade

remnux@remnux:~/cases/arkei-msi$ msiinfo streams po.msi
Icon.ProductIcon
Binary.bz.CustomActionDll
Binary.bz.WrappedSetupProgram
SummaryInformation
DocumentSummaryInformation
```

As MSI files go, this one isn't particularly complex. Depending on the product used, there can be a lot more data in tables and streams. There are a few things we definitely want to hit during our analysis here. First, we need to examine the contents of the "CustomAction" table at the very least. The CustomAction table is often interesting with malicious installers as adversaries may hide code to execute within the CustomAction table. PurpleFox malware has placed JScript to execute in this table in the past and other malware families have used the table to specify that a malicious DLL should be executed during installation. T

Dumping Tables and Streams

We can dump out all of the contents using `msidump`.

```
remnux@remnux:~/cases/arkei-msi$ msidump -s -t po.msi
Exporting table _SummaryInformation...
Exporting table _ForceCodepage...
Exporting table _Validation...
Exporting table AdminExecuteSequence...
Exporting table AdminUISequence...
Exporting table AdvtExecuteSequence...
Exporting table Binary...
Exporting table Component...
Exporting table Directory...
Exporting table CustomAction...
Exporting table Feature...
Exporting table FeatureComponents...
Exporting table File...
Exporting table Icon...
Exporting table InstallExecuteSequence...
Exporting table InstallUISequence...
Exporting table LaunchCondition...
Exporting table Media...
Exporting table Property...
Exporting table Registry...
Exporting table Upgrade...
Exporting stream Icon.ProductIcon...
Exporting stream Binary.bz.CustomActionDll...
Exporting stream Binary.bz.WrappedSetupProgram...
Exporting stream SummaryInformation...
Exporting stream DocumentSummaryInformation...

remnux@remnux:~/cases/arkei-msi$ ls -l
total 4720
-rw-rw-r-- 1 remnux remnux     243 Feb 12 22:44 AdminExecuteSequence.idt
-rw-rw-r-- 1 remnux remnux     141 Feb 12 22:44 AdminUISequence.idt
-rw-rw-r-- 1 remnux remnux     225 Feb 12 22:44 AdvtExecuteSequence.idt
drwxrwxr-x 2 remnux remnux    4096 Feb 12 22:44 Binary
-rw-rw-r-- 1 remnux remnux     132 Feb 12 22:44 Binary.idt
-rw-rw-r-- 1 remnux remnux     202 Feb 12 22:44 Component.idt
```

```
-rw-rw-r-- 1 remnux remnux    1093 Feb 12 22:44 CustomAction.idt
...
```

Each of the IDT files contain data from the tables, while two folders named “Binary” and “_Streams” hold executable and stream data fetched from the MSI. First up, let’s inspect that CustomAction.idt file.

```
Action  Type    Source  Target  ExtendedType
s72     i2      S72     S255    I4
CustomAction    Action
bz.EarlyInstallMain     1       bz.CustomActionDll      _InstallMain@4
bz.EarlyInstallSetPropertyForDeferred1  51      bz.EarlyInstallFinish2  [BZ.INIFILE]
bz.EarlyInstallFinish2  1       bz.CustomActionDll      _InstallFinish2@4
bz.LateInstallPrepare   1       bz.CustomActionDll      _InstallPrepare@4
bz.LateInstallSetPropertyForDeferred1   51      bz.LateInstallFinish1   [BZ.INIFILE]
bz.LateInstallFinish1   3073    bz.CustomActionDll      _InstallFinish1@4
bz.LateInstallSetPropertyForDeferred2   51      bz.LateInstallFinish2   [BZ.INIFILE]
bz.LateInstallFinish2   3073    bz.CustomActionDll      _InstallFinish2@4
bz.CheckReboot  1       bz.CustomActionDll      _CheckReboot@4
bz.UninstallPrepare     1       bz.CustomActionDll      _UninstallPrepare@4
bz.UninstallSetPropertyForDeferred1     51      bz.UninstallFinish1     [BZ.INIFILE]
bz.UninstallFinish1     3073    bz.CustomActionDll      _UninstallFinish1@4
bz.UninstallSetPropertyForDeferred2     51      bz.UninstallFinish2     [BZ.INIFILE]
bz.UninstallFinish2     1025    bz.CustomActionDll      _UninstallFinish2@4
bz.UninstallWrapped     1       bz.CustomActionDll      _UninstallWrapped@4
```

The table contents look relatively normal as far as MSI files go. If there were malicious content here we’d see code chunks that we’d expect to see in JScript or VBScript files. Let’s go take a look at some other interesting tables. The Property table gives some more information.

```
Property        Value
s72     l0
Property        Property
UpgradeCode     {3FF46275-96F9-4EBF-9B1E-50CA97E8DB0E}
ALLUSERS        1
ARPNOREPAIR     1
ARPNOMODIFY     1
ARPPRODUCTICON  ProductIcon
BZ.WRAPPED_REGISTRATION None
BZ.VER  2922
BZ.CURRENTDIR   *SOURCEDIR*
BZ.WRAPPED_APPID        {1FC4DB72-5AB1-4002-B9B0-00FAA9B12D8E}
BZ.COMPANYNAME  EXEMSI.COM
BZ.BASENAME     NEnXoxoXxKaPjctW.exe
BZ.ELEVATE_EXECUTABLE   administrators
BZ.INSTALLMODE  EARLY
BZ.WRAPPERVERSION       10.0.50.0
BZ.EXITCODE     0
BZ.INSTALL_SUCCESS_CODES        0
Manufacturer    My App
ProductCode     {481C9516-0944-4A5D-B8F1-803936B5D792}
ProductLanguage 1033
ProductName     My App
ProductVersion  21.9.9.16
SecureCustomProperties  WIX_DOWNGRADE_DETECTED;WIX_UPGRADE_DETECTED
```

It looks like the CompanyName for this MSI Wrapper is EXEMSI.COM, consistent with what we expected so far. The BaseName property looks to be `NEnXoxoXxKaPjctW.exe`. We haven’t seen this name anywhere else in the tables so far, so I’m going to guess there’s an archive or something inside a stream that contains the executable or content that downloads it. Let’s go look at the _Streams content.

```
remnux@remnux:~/cases/arkei-msi/_Streams$ file *
Binary.bz.CustomActionDll:     PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
Binary.bz.WrappedSetupProgram: Microsoft Cabinet archive data, Windows 2000/XP setup, 2059637 bytes, 1 file, at 0x2c +A "NEnX
DocumentSummaryInformation:    dBase III DBT, version number 0, next free block index 65534
Icon.ProductIcon:              Targa image data - Map 32 x 19866 x 1 +1
SummaryInformation:            dBase III DBT, version number 0, next free block index 65534
```

We have some executable content that looks interesting in _Streams. First, the file `Binary.bz.CustomActionDll` looks like it's a Windows native DLL file. A "custom action DLL" is pretty common to see in MSI files from multiple different products. I commonly see this sort of DLL in MSIs made by AdvancedInstaller tools, and those are usually signed. The second interesting file is `Binary.bz.WrappedSetupProgram`. This looks like a Microsoft CAB file that we can unpack using `7z`.

```
remnux@remnux:~/cases/arkei-msi/_Streams$ 7z x Binary.bz.WrappedSetupProgram

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz (806EA

Scanning the drive for archives:
1 file, 2059637 bytes (2012 KiB)

Extracting archive: Binary.bz.WrappedSetupProgram
--
Path = Binary.bz.WrappedSetupProgram
Type = Cab
Physical Size = 2059637
Method = LZX:21
Blocks = 1
Volumes = 1
Volume Index = 0
ID = 5658

Everything is Ok

Size:       2094224
Compressed: 2059637

remnux@remnux:~/cases/arkei-msi/_Streams$ ls -l
total 4408
-rw-rw-r-- 1 remnux remnux  212992 Feb 12 22:44  Binary.bz.CustomActionDll
-rw-rw-r-- 1 remnux remnux 2059637 Feb 12 22:44  Binary.bz.WrappedSetupProgram
-rw-rw-r-- 1 remnux remnux 2094224 Feb  9 14:17  NEnXoxoXxKaPjctW.exe
```

After a life-affirming message from `7z`, the tool successfully unpacked `NEnXoxoXxKaPjctW.exe` from the CAB. This is the EXE we were looking for after it was mentioned in Property.idt! Thus ends the MSI triage!

Triage the EXE

Using Detect-It-Easy to identify the file helped find a stumbling block.

```
remnux@remnux:~/cases/arkei-msi/_Streams$ diec NEnXoxoXxKaPjctW.exe
filetype: PE32
arch: I386
mode: 32-bit
endianess: LE
type: GUI
```

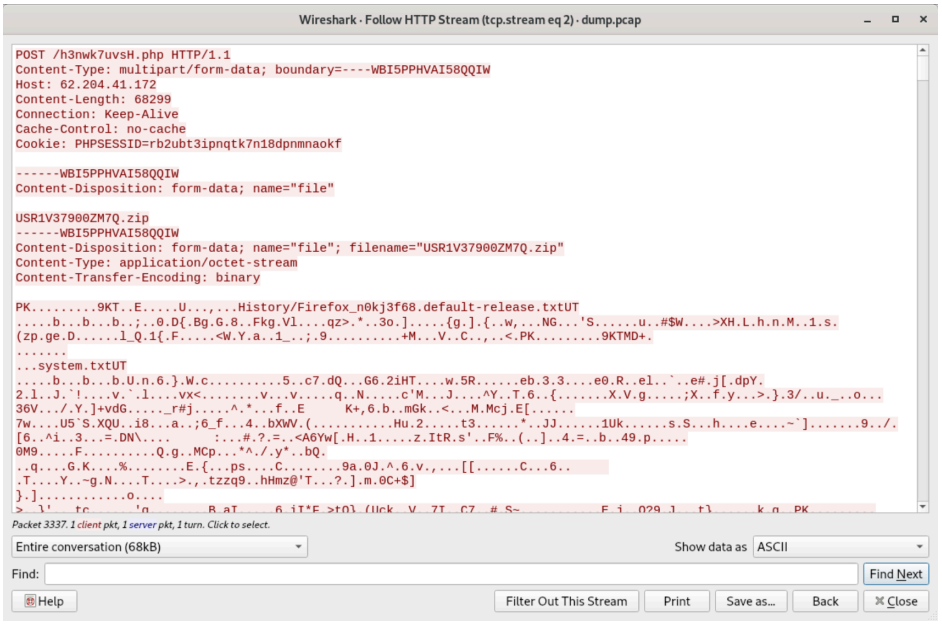
```
7       protector: Obsidium(-)[-]
8       linker: unknown(2.30)[GUI32]
```

The EXE is protected/packed using Obsidium, a commercial packing tool. There's probably a way to unpack it statically or with a debugger, but that's going to take more effort than I want to put in tonight. The best way from here forward for me will be to lean on a sandbox report.

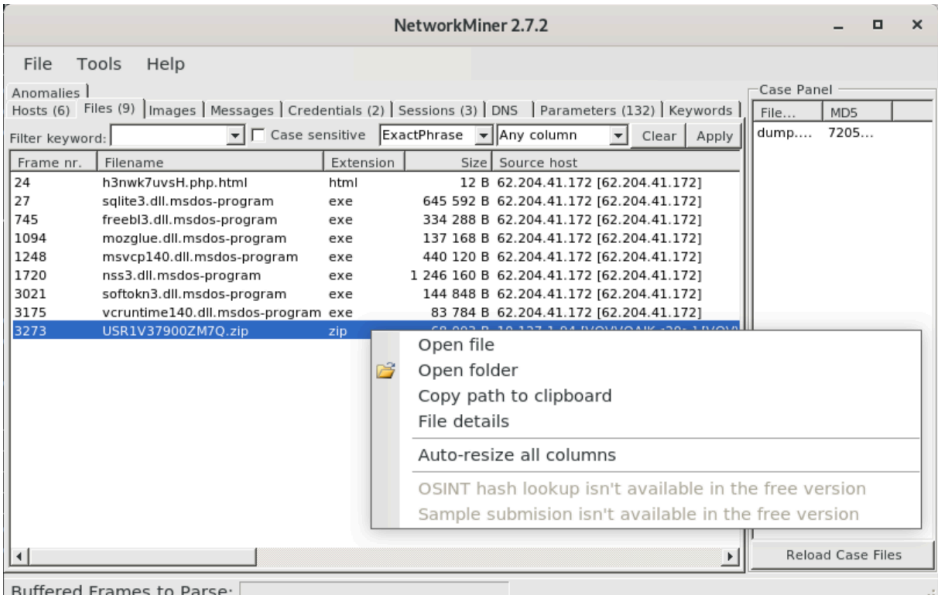
How do we know it's Arkei Stealer?

The Tria.ge report for the sample indicates it found evidence of Arkei in the memory dump of process ID 1312, which corresponds to `NEnXoxoXxKaPjctW.exe`. Let's inspect that memory dump with some YARA rules and see what we can find. After running the sample through YARA rules from the `yara-rules` and `ditekshen` repositories, I couldn't find a match. I assume at this point that the YARA rule is internal/private to Hatching. So let's see if we can find intelligence overlaps in the sandbox telemetry.

The network activity from the report indicates the sample downloaded `mozglue.dll`, `sqlite3.dll`, `nss3.dll`, `freebl3.dll`, and a couple others. These DLLs are commonly downloaded and loaded into memory by stealers as they provide functionality to decrypt sensitive data within Mozilla Firefox and Chromium-based web browsers. This is common to Vidar and Arkei, and these two families are similar enough that one may be forked from the other. The network telemetry in the sandbox PCAP can also be helpful since it looks like there was a POST request. We can take a look at the data in Wireshark. In Wireshark, we want to filter on `http` protocol only so we can immediately find that POST request.To see the content of the POST, we can right-click on the POST request and follow the HTTP stream. Once we do that, we can see it looks like the malware uploaded a ZIP archive named `USR1V37900ZM7Q.zip`.



Since this file is uploaded over HTTP, it stands to reason that we can carve it out of the network traffic and inspect the contents. We can do this easily with NetworkMiner. Once you open the PCAP in NetworkMiner, all the files get automatically reassembled and written to disk. To inspect one, right-click on the file and either "Open File" or "Open Folder".



After unpacking the ZIP we can find just a few files.

```
remnux@remnux:~/cases/arkei-msi/network$ tree -a
.
├── History
│   └── Firefox_n0kj3f68.default-release.txt
├── screenshot.jpg
├── system.txt
└── USR1V37900ZM7Q.zip
```

Just searching for “screenshot.jpg” + “system.txt” on Google will yield some hits on Oski stealer, and one on Vidar. This isn’t surprising as Oski reportedly shares some code with Vidar and Arkei.

The Firefox and screenshot information will likely be self-explanatory, so let’s start with `system.txt`. Stealers commonly capture system configuration data within a text file and sometimes leave specific toolmarks/artifacts inside those files. For example, Raccoon often leaves its own name in a systeminfo text file. Previous versions of Vidar, such as in fumik0’s analysis, seem to store system information in a file named `information.txt` instead. This makes me think we’re actually somewhere in Oski stealer territory since it allegedly shares some code with Arkei. Lastly, there’s also a possibility this could be Mars stealer based on its similarity to Oski in this analysis as with Oski, there is a `system.txt` file.

So why does any of this matter? It matters a bit for threat intelligence tracking and attribution to developers. This also shows the great challenge in threat intelligence of trying to interpret malware analysis findings and detection details when many malware tools fork from one another and share artifacts. In worst case scenarios analysts can make assessments based on severely flawed or dated information. In many cases, though, the data is “close enough” to still be useful. This can be seen in the Tria.ge report: no matter what the stealer is, the configuration is still parsed.

Thanks for reading!

Source: https://forensicitguy.github.io/analyzing-stealer-msi-using-msitools/