Extracting Indicators from a Packed Mirai Sample

forensicitguy.github.io/extracting-indicators-from-packed-mirai/

January 4, 2022

By *Tony Lambert*
Posted *2022-01-04* Updated *2022-03-28* *4 min* read

Packing is really commonly used by adversary to stump analysis, so in this post I'm going to look at a sample that is really easy to unpack and get indicators from. In this case the sample is Mirai packed with UPX. If you want to follow along at home, the sample is in MalwareBazaar here:

https://bazaar.abuse.ch/sample/ef11393108bed5f3753d054514b2dddb1a534f3623244ab485c0ed6e2d5ded9e/

Why Just Indicators?

Malware analysis should serve a purpose. In my dayjob on the Red Canary Intelligence team I sometimes have to assess malware for indicators as parts of incidents. Not every adventure ends in assembly code and not every adventure requires a 50-page report.

Identifying UPX Packing

For this Mirai sample, it's easy to detect the UPX packing with `Detect It Easy`.

```
remnux@remnux:~/cases/mirai$ diec
mirai.elf
filetype: ELF32
arch: 386
mode: 32-bit
endianess: LE
type: EXEC
  packer: UPX(3.96)[NRV,brute]
```

To verify it's packed with standard UPX, we can look for `UPX!` (`55 50 58 21`) in the first few bytes:

```
remnux@remnux:~/cases/mirai$ hexdump -C mirai.elf | head
00000000  7f 45 4c 46 01 01 01 03  00 00 00 00 00 00 00 00
|.ELF............|
00000010  02 00 03 00 01 00 00 00  08 ba 04 08 34 00 00 00
|............4...|
00000020  00 00 00 00 00 00 00 00  34 00 20 00 03 00 28 00  |........4.
...(.|
00000030  00 00 00 00 01 00 00 00  00 00 00 00 00 80 04 08
|................|
00000040  00 80 04 08 e6 42 00 00  e6 42 00 00 05 00 00 00
|.....B...B......|
00000050  00 10 00 00 01 00 00 00  00 00 00 00 00 d0 04 08
|................|
00000060  00 d0 04 08 00 00 00 00  20 3c 00 00 06 00 00 00  |........
<......|
00000070  00 10 00 00 51 e5 74 64  00 00 00 00 00 00 00 00
|....Q.td........|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 06 00 00 00
|................|
00000090  04 00 00 00 4c 15 8d 50  55 50 58 21 ec 08 0d 0c
|....L..PUPX!....|
```

Also, we can verify with YARA:

```
remnux@remnux:~/cases/mirai$ yara-rules -s mirai.elf
UPXProtectorv10x2 mirai.elf
0x3a2a:$a0: EB 0E 90 90 90 90 8A 06 46 88 07 47 01 DB 75 07 8B 1E 83 EE FC
11 DB
```

It's not unheard of for adversaries to overwrite artifacts of UPX packing or use custom packers, so when you find a sample with standard UPX it's always time for celebration!

Unpacking The Sample

In this case it's simple to unpack the sample. We're using `upx` on REMnux, but we also need to remember that the command will overwrite the original executable. First, we need to create a backup copy.

```
remnux@remnux:~/cases/mirai$ cp mirai.elf mirai.elf.bak

remnux@remnux:~/cases/mirai$ upx -d mirai.elf
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd
2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     30908 <-     17376   56.22%   linux/i386    mirai.elf

Unpacked 1 file.
```

We can verify the result is executable with `file` and then get our hashes to look up in VT or other sources.

```
remnux@remnux:~/cases/mirai$ file mirai.elf
mirai.elf: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
linked, stripped

remnux@remnux:~/cases/mirai$ diec mirai.elf
filetype: ELF32
arch: 386
mode: 32-bit
endianess: LE
type: EXEC

remnux@remnux:~/cases/mirai$ md5sum mirai.elf
3c246e3a6c146dd823268920918c9b48  mirai.elf
```

Looking for Indicators

The quick and easy triage for indicators can happen with `strings`. Remember, by default it just looks for ASCII and not Unicode, so you need two passes.

```
remnux@remnux:~/cases/mirai$ strings mirai.elf > mirai-strings.txt
remnux@remnux:~/cases/mirai$ strings -eL mirai.elf >> mirai-
strings.txt
```

Examining the strings output, we can find a couple interesting things:

- `35.197.127[.]250`
- `/dev/null`

From here, we can pivot on that IP address as an indicator to see where it leads. We can also possibly look for sandbox reports or execute the malware in a controlled environment. To save some time, I looked into a Joe Sandbox report for the sample. The report included the IP address above plus a few more to try and pivot around. Depending on the requirements for your incident you might also look for obfuscated strings using Ghidra or other tools, but we don't need to for this case.

Thanks for reading!