

Target Acquired  
Malware  
200 Points

Both challenges in the Malware category were able to be solved using tools and steps in what is commonly referred to as the “Basic Analysis” phase. This means that no debugger, disassembler, or decompiler is necessary to solve these challenges.

All the tools required to solve this challenge are installed by default with FlareVM, one of the best malware analysis VMs out there, made by Google’s Mandiant: <https://github.com/mandiant/flare-vm>.

The easiest way to solve this challenge is using the strings utility, or rather Floss, since this is a Windows executable. Floss is another tool made by Mandiant that has several string decryption techniques built into it that the regular strings utility doesn’t have, but it only works on PE executables for Windows. Using Floss on the file:

```
C:\Users\admin\Desktop
λ floss -n 5 TargetAcquired.exe > floss.txt
INFO: floss: extracting static strings
finding decoding function features: 100%|██████████| 4/4 [00:00<?, ? functions]
INFO: floss.stackstrings: extracting stackstrings from 3 functions
extracting stackstrings: 100%|██████████| 3/3 [00:00<?, ? stackstrings]
INFO: floss.tightstrings: extracting tightstrings from 1 functions...
extracting tightstrings from function 0x14000c742: 100%|██████████| 1/1 [00:00<?, ? tightstrings]
INFO: floss.string_decoder: decoding strings
emulating function 0x14000c742 (call 1/1): 100%|██████████| 4/4 [00:00<?, ? strings]
INFO: floss: finished execution after 2.73 seconds
INFO: floss: rendering results
```

Looking through the floss.txt results file, nothing of interest can be found. Having very few strings like this is usually a sign that the sample is packed, as there are commonly strings left over from compiling programs that are dug up with Floss. To check whether it is, we can use Detect It Easy (DiE):

Detect It Easy v3.10 [Windows 10 Version 2009] (x86\_64)

File name: C:\Users\admin\Desktop\TargetAcquired.exe

File type: PE64 | File size: 7.50 KiB

Scan: Automatic | Endianness: LE | Mode: 64-bit | Architecture: AMD64 | Type: Console

**PE64**

- Operation system: Windows(Server 2003)[AMD64, 64-bit, Console] S ?
- (Heur)Language: ASMX64 S ?
- Packer: UPX(4.24)[NRV,brute] S ?
- (Heur)Packer: Compressed or packed data[EntryPoint + Imports like UPX (v3.91+) + Sectio... S ?

From the output, we can see that DiE detects that the sample is packed using UPX. Now that we know this, we can simply unpack it using UPX itself:

```
C:\Users\admin\Desktop
λ upx -d TargetAcquired.exe
(Hu)Pac (Compressed) + InjPac + InjectPac + UPX (+3.9(+)) + Sectio... S 7
          Ultimate Packer for eXecutables S 7
          Copyright (C) 1996 - 2025 S 7
UPX 5.0.2      Markus Oberhumer, Laszlo Molnar & John Reiser Jul 20th 2025

  File size      Ratio      Format      Name
-----
  14848 <-    7680     51.72%    win64/pe  TargetAcquired.exe

Unpacked 1 file.
```

Now that it's unpacked, Floss should be able to give us a little more information. Running Floss on the file again results in the following:

```
C:\Users\admin\Desktop
λ floss -n 5 TargetAcquired.exe > floss.txt
INFO: floss: extracting static strings
finding decoding function features: 100%|██████████| 66/66 [00:00<?, ? functions]
INFO: floss.stackstrings: extracting stackstrings from 42 functions
extracting stackstrings: 100%|██████████| 42/42 [00:00<?, ? stackstrings]
INFO: floss.tightstrings: extracting tightstrings from 4 functions...
extracting tightstrings from function 0x140002730: 100%|██████████| 4/4 [00:00<?, ? tightstrings]
INFO: floss.string_decoder: decoding strings
INFO: floss.results: t.brown
INFO: floss.results: StoutCTF{█████████████████████}
INFO: floss.results: C:\Windows\Temp\flag.txt
emulating function 0x140002730 (call 1/1): 100%|██████████| 22/22 [00:00<?, ? emulations]
INFO: floss: finished execution after 7.00 seconds
INFO: floss: rendering results
```

We can see that Floss was able to decode the flag now that the file was unpacked, along with a few other strings.