Malware Analysis Report: "Practical2.exe"

CAP6137 Malware Reverse Engineering: P0x02

Naman Arora
naman.arora@ufl.edu

March 18, 2021

Contents

1 Executive Summary

2 Static Analysis: Primary Executable

2.1 Basic Identification

Attribute	Value
Bits	32
Endianess	Little
Operating System	Microsoft Windows
Class	PE32
Subsystem	Windows CUI
Size	1446912
Compiler Timestamp	Thu Dec 10 02:47:43 2020
Compiler	Visual Studio
SHA256 Hash	9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382

2.2 Malware Sample Family Identification

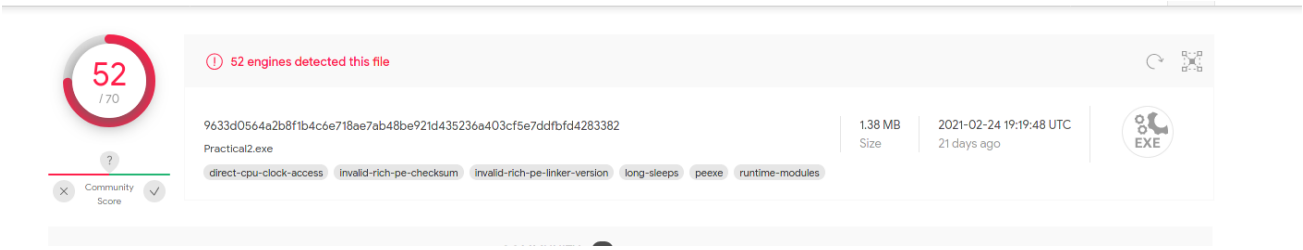

Figure 1: VirusTotal: VirusTotal Scan

The given PE file, on being uploaded to VirusTotal, is identified as a variant of *AveMariaRAT* family (Fig. 1) As seen later in the *dynamic analysis* section, another in-memory PE when dumped and analysed on VirusTotal, is identified to belong to *WarZoneRAT* family.

2.3 PE Sections

```
[0x004045d4]> iS entropy
[Sections]

nth paddr        size vaddr        vsize perm entropy    name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000400  0xb5200 0x00401000  0xb6000 -r-x 5.41704949 .text
1   0x000b5600  0x29800 0x004b7000  0x2a000 -r-- 3.65648569 .rdata
2   0x000dee00  0x79c00 0x004e1000  0x7e000 -rw- 4.61074896 .data
3   0x00158a00   0x1600 0x0055f000   0x2000 -r-- 4.58260461 .idata
4   0x0015a000    0x400 0x00561000   0x1000 -rw- 0.01117382 .tls
5   0x0015a400    0x200 0x00562000   0x1000 -r-- 0.06116285 .00cfg
6   0x0015a600    0x600 0x00563000   0x1000 -r-- 2.14085818 .rsrc
7   0x0015ac00   0x6800 0x00564000   0x7000 -r-- 6.09421433 .reloc
```

Figure 2: Rizin: Section-wise Entropy

2.3.1 The *.text*, *.rdata*, *.idata*, *.rsrc* and *.reloc* sections

These commonly found PE sections within the executable show no peculiar characteristics in terms of entropy, virtual sizes and permissions.

2.3.2 The *.data* Section

This section, although not peculiar either, on static analysis reveals that it is referenced in the identified main function. On further analysis of the function, a unpacking loop is encountered thus hinting towards the section being the store of packed data.

2.3.3 The *.tls* Section

Presence of this section generally hints towards thread execution before *entrypoint* is reached in the context of malicious binaries. This binary, however, shows no such execution. Thus, the reason for the presence of this section cannot be corroborated during the current analysis.

2.3.4 The *.00cfg* Section

The presence of this unusual section (*Control Flow Guard*) seems to be explained as an artifact of the *Visual studio compiler*. This guess is supported by

- Very small size of the section *0x200*.
- Almost all bytes being zeros.
- All the references to this section (Fig. 6) seem to originate from *Ghidra* identified library functions with exception to one which does not show much promise on followup.

Figure 3: Ghidra: references to the *.00cfg* section

```
i = 0;
while (i <= count + -1) {
  allocatedMem[i] = (code)~(byte)*(undefined2 *)(s_VirtualAddress + ((count + -1) -
                  /* pbstrPath != 0 && ppTypeLib != 0 */
                  /* pbstrPath != 0 && ppTypeLib != 0 */
  MessageBoxA((HWND)0x0,"","",0);
  i = i + 1;
}
j = 0;
while (j < 9600000) {
  k = 0;
  while (k < 0x400) {
    local_90 = count;
    if (count < 0x1f5) {
      local_94 = 0;
    }
    else {
      local_94 = 100;
    }
    k = k + 1;
  }
  j = j + 1;
}
l = 0;
while (l < local_90) {
  allocatedMem[l] = (code)((byte)allocatedMem[l] ^ local_70[l % local_94]);
  l = l + 1;
}
```

Figure 4: Ghidra: Disassembly of unpacking

2.4 A case for Packing

A very strong case for packing can be made for this binary given the following observations,

- The identified *main* function exhibits a series of byte operations on data pointed to by the *.data* section.
- Immediately preceding the manipulations, a call to *VirtualAlloc* can be intercepted.
- The manipulated bytes from *.data* section are stored in the allocated memory section.
- After the said manipulations, the memory section is called as a function.
- The said allocated section, on analysis and after being manipulated, exhibits a presence of *shell code* and a *PE* header preceding code at repeatedly reproducible offsets and sizes.

2.5 Interesting Imports

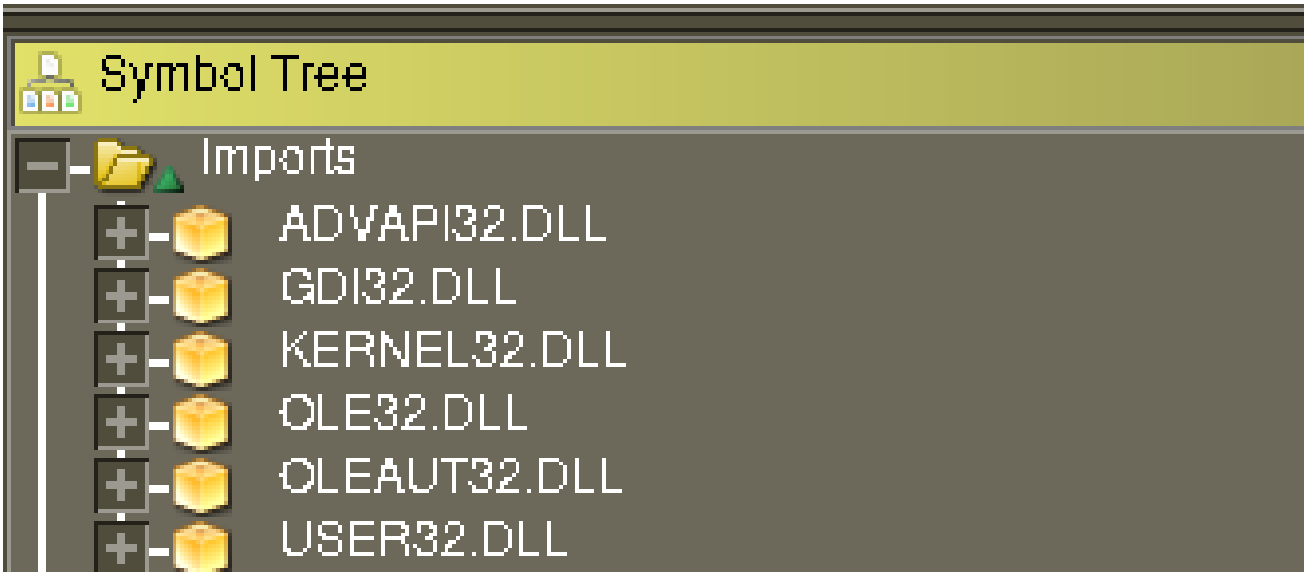

Figure 5: Ghidra: Imports tree

2.5.1 Imports from *Kernel32.dll*

Imports like *VirtualAlloc and VirtualFree* in combination with *VirtualProtect* strongly indicate runtime memory injection preceding change in injected region's permissions to *executable*. Presence of *FreeConsole* seems to corroborate the assumption that this is a *CUI* program, given this function is used to unlink from the parent process.

2.6 Imports from *user32.dll*

An import from this library, *viz.*, *MessageBoxA* is peculiar. This is due to the fact that, in *main* function, the permissions of memory containing code for this import is updated from *PAGE_EXECUTE_READ* to *PAGE_EXECUTE_READWRI* and is subsequently the code is replaced with a *return 0x10000* call. This function is then invoked multiple times during the unpacking process and the string *"pbstrPath != 0 && ppTypeLib != 0"* is pushed as twice arguments. The reason behind this could not be identified during this analysis (Fig 4).

3 Static Analysis: Dynamically Unpacked Shell Code

3.1 Basic Identification

Attribute	Value
Bits	32
Endianess	Little
Operating System	Microsoft Windows
Class	PE32
Subsystem	Windows CUI
Size	1446912
Compiler Timestamp	Thu Dec 10 02:47:43 2020
Compiler	Visual Studio
SHA256 Hash	9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382

3.2 Sample Family Identification

3.3 Shell Code Sections

3.3.1 The *.text*, *.rdata*, *.idata*, *.rsrc* and *.reloc* sections

These commonly found PE sections within the executable show no peculiar characteristics in terms of entropy, virtual sizes and permissions.

3.3.2 The *.data* Section

This section, although not peculiar either, on static analysis reveals that it is referenced in the identified main function. On further analysis of the function, a unpacking loop is encountered thus hinting towards the section being the store of packed data.

3.3.3 The *.tls* Section

Presence of this section generally hints towards thread execution before *entrypoint* is reached in the context of malicious binaries. This binary, however, shows no such execution. Thus, the reason for the presence of this section cannot be corroborated during the current analysis.

3.3.4 The *.00cfg* Section

The presence of this unusual section (*Control Flow Guard*) seems to be explained as an artifact of the *Visual studio compiler*. This guess is supported by

- Very small size of the section *0x200*.
- Almost all bytes being zeros.
- All the references to this section (Fig. 6) seem to originate from *Ghidra* identified library functions with exception to one which does not show much promise on followup.

3.4 Interesting Imports

4 Static Analysis: Dynamically Unpacked PE Executable

4.1 Basic Identification

Attribute	Value
Bits	32
Endianess	Little
Operating System	Microsoft Windows
Class	PE32
Subsystem	Windows CUI
Size	1446912
Compiler Timestamp	Thu Dec 10 02:47:43 2020
Compiler	Visual Studio
SHA256 Hash	9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382

Figure 6: Ghidra: references to the *.00cfg* section

4.2 Sample Family Identification

4.3 PE Sections

4.3.1 The *.text*, *.rdata*, *.idata*, *.rsrc* and *.reloc* sections

4.3.2 The *.data* Section

4.3.3 The *.tls* Section

4.3.4 The *.00cfg* Section

4.4 Interesting Imports

5 Static Analysis: Dynamically Unpacked DLL

5.1 Basic Identification

Attribute	Value
Bits	32
Endianess	Little
Operating System	Microsoft Windows
Class	PE32
Subsystem	Windows CUI
Size	1446912
Compiler Timestamp	Thu Dec 10 02:47:43 2020
Compiler	Visual Studio
SHA256 Hash	9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382

8 Appendix A: Memory Dump string analysis screenshots