Malware Analysis Report: "Practical3.exe"

CAP6137 Malware Reverse Engineering: P0x03

Naman Arora
naman.arora@ufl.edu

April 13, 2021

Contents

1 Executive Summary

The provided binary has been identified as a member of *Ryuk* family of *Ransomwares* [3]. This family of ransomwares gain access to victim systems via phishing [5] or social engineering attacks [6] and are known to be biased towards targeting commercial systems more as compared to personal systems. *FBI* warned against this campaign citing the malware authors as most profitable when compared to other authors of such malwares. Also, this malware family is known to attack *Windows* systems in general and *Windows 10* systems in particular.

The malware binary itself is very small in size, around *170 KB*, and has no significant obfuscation techniques built in. Nevertheless, on dynamic analysis, the malware injects itself into common *Windows* processes and installs a registry key to achieve persistance over reboots. Any new file created or drive *(for eg. USB etc.)* connected to the system is also targeted and encrypted. Before any encryption happens, the malware,

- Stops and kills multiple services and executables generally associated with commercial systems
- Enumerates all drives associated to the system and all the files within them.

The execution of malware is entirely offline and hence once infected, isolating the system from the network does not thwart its any malicious activities. This also means that leaking of sensitive information from the infected system is highly improbable as a result of this infection. After encryption, the malware installs multiple text files named *RyukReadme.txt* acknowledging the data loss, stating the ransom and providing correspondence email and *Bitcoin Wallet* addresses. This ransomware family has, however, built up a reputation of consistently decrypting the data once ransom has been paid.

Independent analysis as well as aggregated opinion from the community very strongly suggests that decrypting the data without paying the ransom amount is not possible. The cryptographic algorithms used for encryption are industry standard and hence extensively secure. The prime recommendation would be to pay the ransom unless,

- the data lost has been backed up and is tested to be recoverable to its fullest extent.
- the lost data cumulatively provides less value than ransom itself.

2 Static Analysis: Primary Executable

2.1 Basic Identification

Attribute	Value
Bits	64
Endianess	Little
Operating System	Microsoft Windows
Class	PE32+
Subsystem	Windows GUI
Size	175616 Bytes
Compiler Timestamp	Tue Aug 14 07:46:26 2018
Compiler	Visual Studio 2015 (Likely)
SHA256 Hash	98ece6bcafa296326654db862140520afc19cfa0b4a76a5950deedb2618097ab

2.2 Malware Sample Family Identification

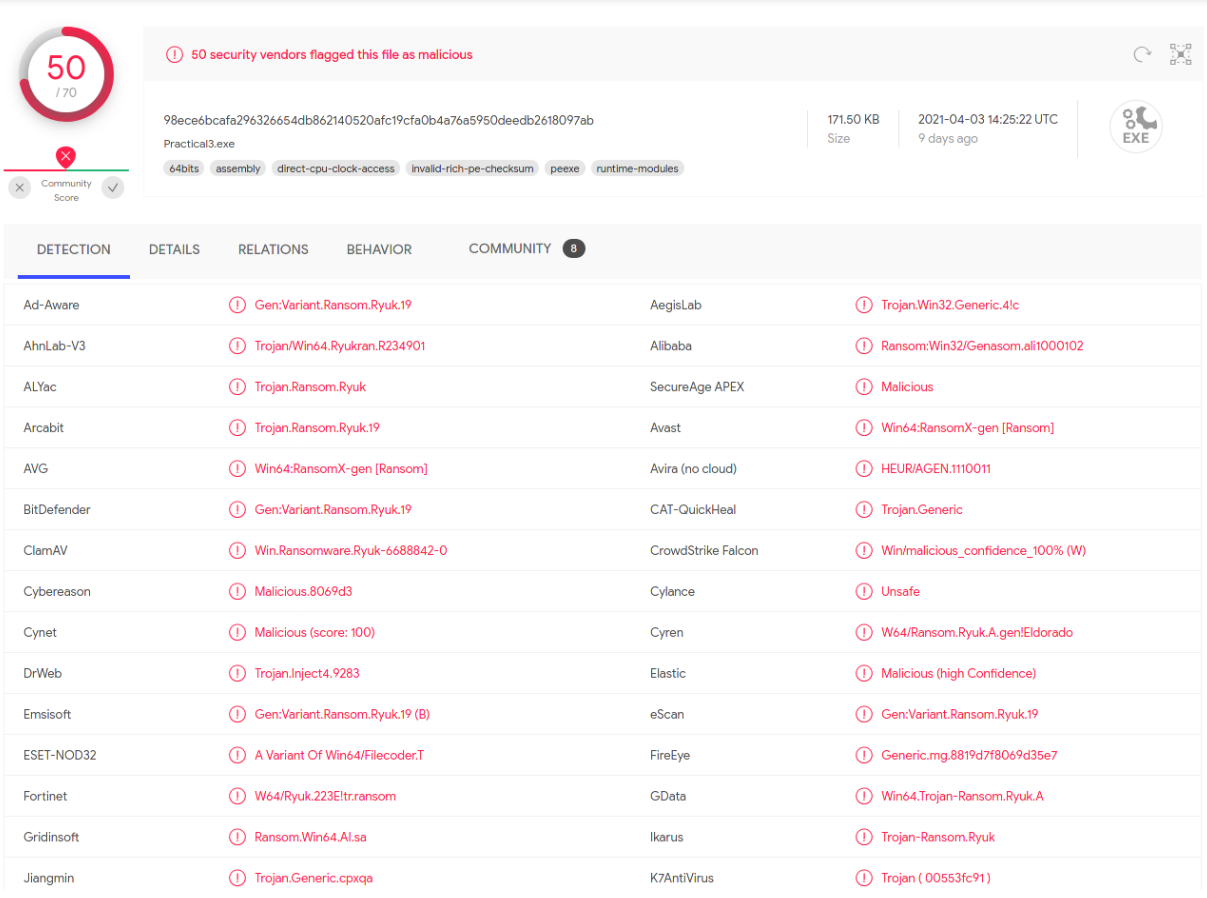


Figure 1: VirusTotal: VirusTotal Scan

The malware is identified to belong to *Ryuk* family of *ransomwares* (Fig. 1). This identification is corroborated by three observations, *viz.*,

- On submitting the sample to *Virustotal* [4], a majority of the *AV* vendors identify it as such.
- Debug file *(pdb)* name within the binary has a path that mentions *Ryuk*.
- On dynamic analysis *(next section)*, it self identifies itself by installing a *RyukReadme.txt* file in each directory.

2.3 PE Headers

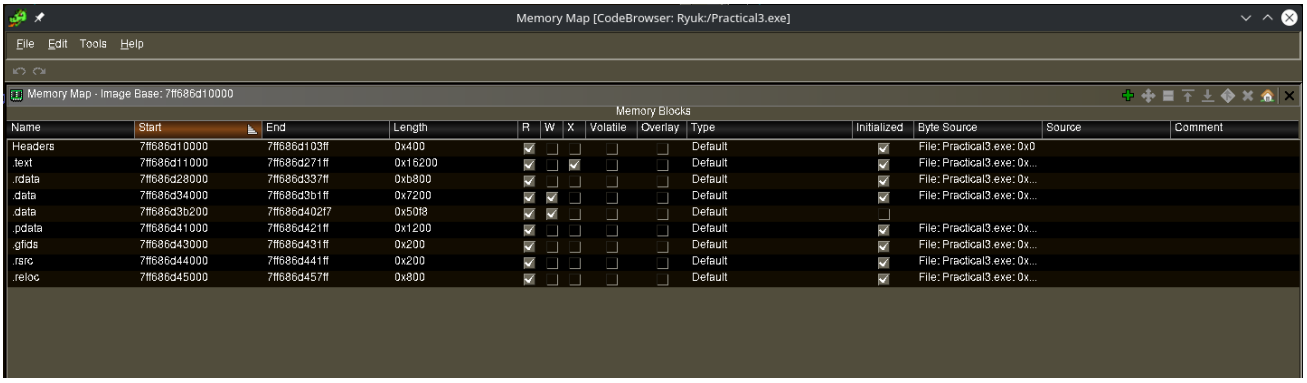


Figure 2: Ghidra: *PE* headers

Sections within the binary show no significant deviation from the norm. All the sections exhibit correct and expected permission sets.

2.4 A case against Packing

A packed *PE* executable, in the context of malicious binaries, is used to obfuscate behavior in hopes of evading static analysis and *AV* detection. The binary sample, though, attempts no such evasion. It is very unlikely that the given sample is packed or encrypted in any way given the following observations,

- The sheer quantity of imports gathered from auto-analysis within *Ghidra*, some of which are generally associated with malicious behavior (Fig 3, 4 and 5)
- Multiple strings indicating malicious behavior

2.5 Interesting Imports

Following are the interesting *DLLS* that are imported,

Advapi32.dll

The malware sample imports functions like *AdjustTokenPrivileges, GetTokenInformation, OpenProcessToken etc.* indicate some token privilege escalation (Fig. 3). Such a behavior indicate the possibility of process injection, which on further analysis, pans out.

Kernel32.dll

The functions imported from this library can be further subdivided in terms of their intended use as,

- Functions like *OpenProcess, VirtualAllocEx, VirtualFreeEx, CreateRemoteThread and WriteProcessMemory* indicate a strong possibility, in conjunction with above *token* related imports, of process injection.
- Functions like *FindFirstFileExA, FindNextFileA and SetFilePointerEx* indicate towards filesystem enumeration.
- Functions like *GetProcAddress and LoadLibraryA* indicate dynamic resolution of function references which, in conjunction with strong traits of process injection, give a bigger picture.

Refer to figure 4.

Shell32.dll

The functions imported from this library *ShellExecuteA and ShellExecuteW*, on further analysis, reveal to be used in stopping system processes and executables (Fig 5).

2.6 Interesting Code Constructs

The Stopping of System Services

Before any significant exhibition of malicious behavior, the malware sample uses the shell execution commands to stop a list of predefined processes and executables. This list is contained within the binary and can be referenced here [2] and [1]. It uses *"taskkill /IM $taskName /F"* and *"net stop $serviceName /y"* template commands to achieve this. All the tasks and executables are generally associated with commercial systems which further strengthens the argument that this malware targets businesses rather than private individuals. The function responsible for this execution is at offset ***7ff686d11250***.

Process Injection

The malware sample, after stopping common services and processes, injects itself into virtual memory section of multiple *Windows* services. Primary function that implements this injection receives a list of arguments including the process name of the victim process and its *PID*. This function then calculates its own size and allocates memory within the victim process and writes itself to that memory area. Following that, it calls *CreateRemoteThread* to execute a function within its own memory area located at offset ***7ff686d119b0***. The function responsible for all this is at offset ***7ff686d126f0***.

Dynamic Function Address Resolution

Once the remote process injection is successful and the remote thread has begun the routine which the malware sample requested, a function located at offset ***7ff686d16850*** is then called. This function is responsible for dynamically resolving addresses to a series of function that, down the line, rest of the malware uses. Some function that are resolved by the said function are absent from the list of imports statically available for investigation (talked about in the dynamic analysis section).

3 Dynamic Analysis

3.1 Network Based Analysis

3.2 File System Based Analysis

4 Indicators of Compromise

4.1 Network Based

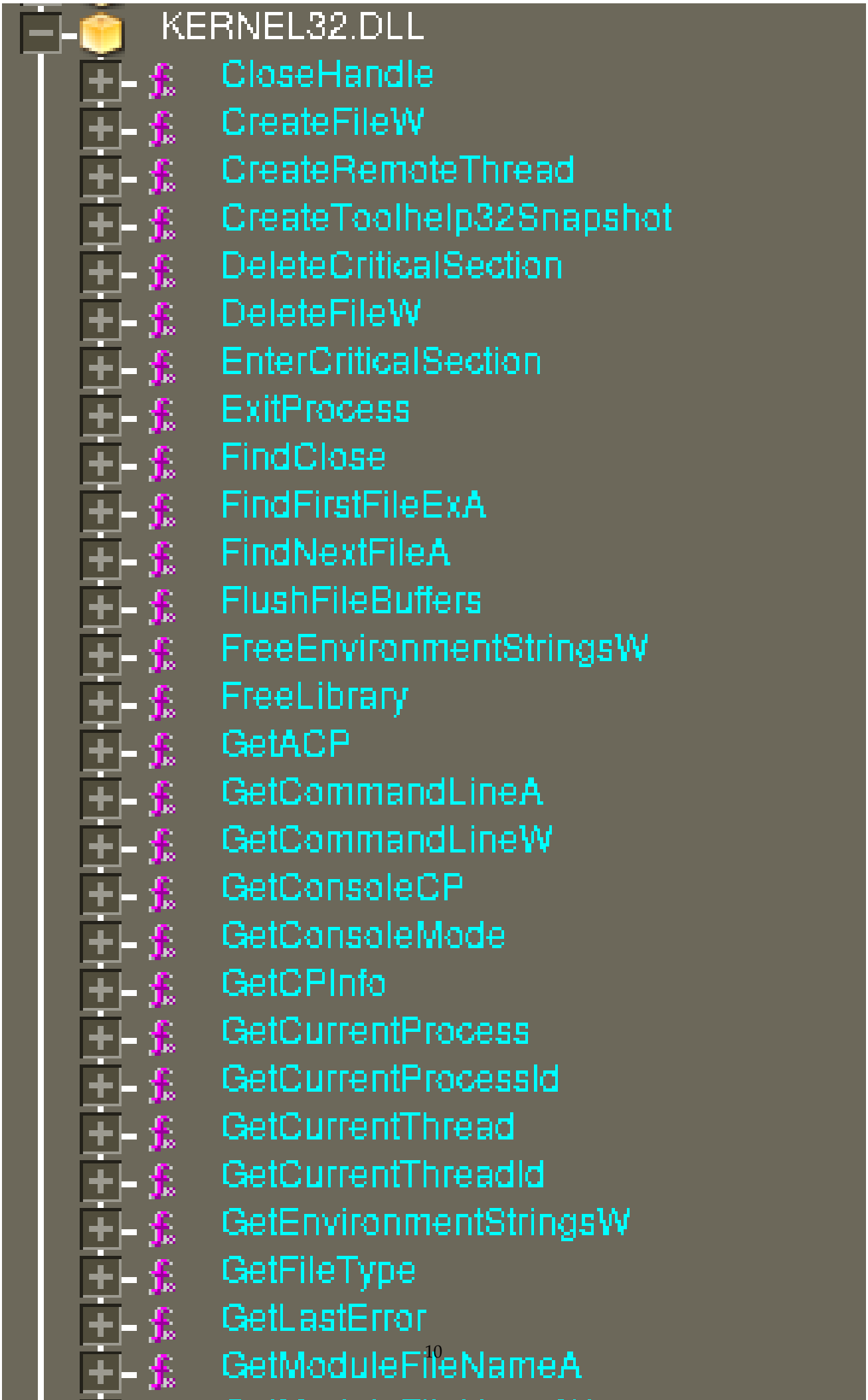
4.2 Host Based

4.3 *YARA* Rule

```
rule practical2_rat {
        meta:
                description = "Detect Practical2.exe RAT"
                author = "Naman Arora"
                date = "2021-03-17"
                hash = "9633d0564a2b8f1b4c6e718ae7ab48be921d435236a403cf5e7ddfbfd4283382"
        strings:
                $pdb = "C:\\Users\\W7H64\\Desktop\\VCSamples-master\\VC2010Samples\\ATL\\General\\AtlCon\\bitcoin coinjoin op.pdb" fullword ascii
                $ops = {c6 04 0a c2 b8 01 00 00 00 c1 e0 00 8b 4d 84 c6 04 01 10 b8 01 00 00 00 d1 e0 8b 4d 84 c6 04 01 00 b8 01 00 00 00 6b c8 03 8b 55 84 c6 04 0a 90}
        condition:
                uint16(0) == 0x5a4d and filesize < 1500MB and all of them
}
```

5 Appendix A: Screenshots

Figure 3: Ghidra: *Advapi32.dll* Imports


KERNEL32.DLL
CloseHandle
CreateFileW
CreateRemoteThread
CreateToolhelp32Snapshot
DeleteCriticalSection
DeleteFileW
EnterCriticalSection
ExitProcess
FindClose
FindFirstFileExA
FindNextFileA
FlushFileBuffers
FreeEnvironmentStringsW
FreeLibrary
GetACP
GetCommandLineA
GetCommandLineW
GetConsoleCP
GetConsoleMode
GetCPInfo
GetCurrentProcess
GetCurrentProcessId
GetCurrentThread
GetCurrentThreadId
GetEnvironmentStringsW
GetFileType
GetLastError
GetModuleFileNameA

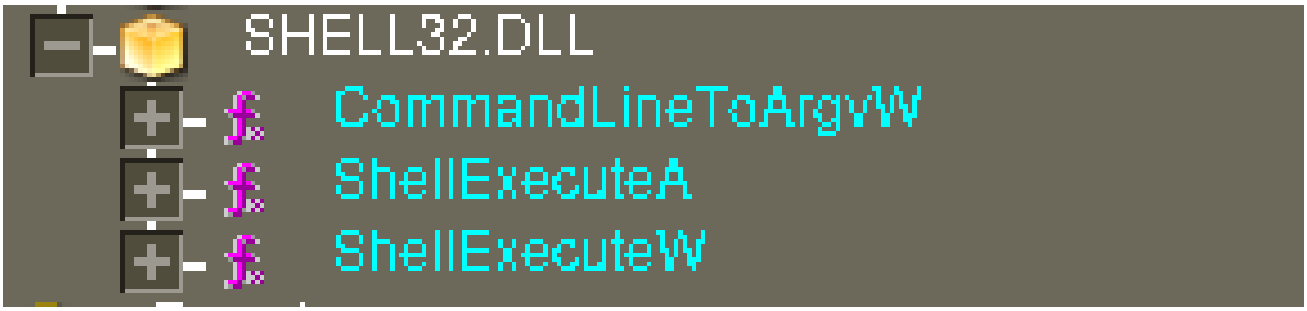


Figure 5: Ghidra: *Shell32.dll* Imports

References

[1] Naman Arora. *Ryuk Processes Stopped*. https://gist.github.com/r0ck3r008/1271cd18978a5419b99e19b194 [Online; accessed 13-April-2021].

[2] Naman Arora. *Ryuk Services Stopped*. https://gist.github.com/r0ck3r008/3f9196e4810174011db41708c0 [Online; accessed 13-April-2021].

[3] Federal Bureau of Investigation. *Ransomware*. https://www.fbi.gov/scams-and-safety/common-scams-and-crimes/ransomware. [Online; accessed 11-April-2021].

[4] VirusTotal. *Ryuk Family Identification*. https://www.virustotal.com/gui/file/98ece6bcafa296326654db86 detection. [Online; accessed 12-April-2021].

[5] WikiMedia. *Phishing*. https://en.wikipedia.org/wiki/Phishing. [Online; accessed 11-April-2021].

[6] WikiMedia. *Social Engineering*. https://en.wikipedia.org/wiki/Social_engineering_security. [Online; accessed 11-April-2021].