Name: Candice Renee Romeo Gomes
ID: 1001427308

8dac49d361285653672e54b88d086411

1. At address 0x00401092 the malware is calling the kernel32.dll library to execute a library function “GetProcAddress”.

2. At address 0x004010A6, the malware is calling Library function ‘VirtualAlloc’. It seems that the callee is trying to allocate address space for the Procedure handle it gets from the ‘GetProcAddress’ library function.

3. All the sub routines are called right after an import is pushed to the stack. Subroutine sub_401360 is called before imports kernel32.dll library functions, sub_401372 is called before imports belonging to adavpi32.dll, sub_401388 is called before imports belonging to user32.dll. We can assume that the malware uses these subroutines to load library functions for the malware file virus.exe

4. The following functions were imported:

   - GetProcAddress
   - VirtualAlloc
   - GetModuleFileNameA
   - ExitProcess
   - CopyFileA
   - GetWindowsDirectory
   - LoadLibraryA
   - RegCreateKeyA
   - SetKeyValueA
   - RegCloseKey
   - MessageBoxA

5. The functions we obfuscated by the malware write so that it would not be picked up by IDA Pro. I use the R key to change the byte to character value.

6. From the strings, we have decoded using IDA Pro, we can assume that this malware open the virus file named ‘/virus.exe’ using Software/Microsoft/Windows/CurrentVersion/Run. This imports several dll functions and creates a registry key that is embedded in the file: Infected!

afdd6b8dae9f522e3e42d4bec4b17716

1. Win/lose function is located at 0x004011BC.

2. The malware flips the characters of the input so that 0x18 becomes 0x81.

3. The encrypted flag is stored at 0x0040303C.

4. Flag: Pra1se_th3_Sun!

b94af4a4d4af6eac81fc135abda1c40c

1. We see that the malware uses GetModuleFilenameA to get the path of the file and then delete the file. We need to change the malware in aa way to allow the code to execute. There are 2 steps to allowing the malware install itself.

```
00402AF3  . B8 2C180000   MOV EAX,182C
00402AF8  . E8 B3030000   CALL b94af4a4.00402EB0
00402AFD  . 837D 08 01    CMP DWORD PTR SS:[EBP+8],1
00402B01  .v75 1A         JNZ SHORT b94af4a4.00402B1D
00402B03  . E8 F8E4FFFF   CALL b94af4a4.00401000
00402B08  . 85C0          TEST EAX,EAX
```

   In line 0x402AFD we can see a comparison operation. Where the command line arguments are being compared with 1. If we can provide additional command line arguments the check in line would satisfy. Therefore, I added argument "-in", this allows the code to move to 0x402B1D.

   Another way to do this is by patching the binary. At address 0x402510 we see that the there are several instructions like ADD, SUM, IMUL and XOR. This suggests that the malware was looking for a unique character or password. If we patch this to make it seem like whichever password entered is excepted, that can allow the malware to install itself.

2. The command line options are for the malware are one of four arguments and a password. The arguments can be:

   - -in: to install the malware
   - -re: to remove the malware
   - -c: to update configuration value in the Windows Registry Key
   - -cc: reverse of the above function, it reads and prints configuration key to the console.

   After analysis of instructions at 0x402510, we can conclude that the password is a string "abcd".

   3. Patching is possible by changing the starting bytes of the function at address 0x402510 so that it would skip the entire password check and return true at the beginning. This is possible by adding the following instructions:

      MOV EAX, 0x1;
      RETN;

      If we convert this into byte sequence we get B8 01 00 00 00 C3.

4. Host based indicators:
   - HKLM\Software\Microsoft \XPS\ Configuration
   - XYZ Manager Service

5. At address 0x401E60 we see a function comparing a string to the following instructions:

   - SLEEP: sleeps for some seconds.
   - UPLOAD: creates a file with a certain filename and prints contents it receives from a specified port.
   - DOWNLOAD: sends a file of a certain filename at a specified port.
   - NOTHING: nothing performed
   - CMD: executes a command on the command prompt and then sends that information using port mentioned

6. Network based signatures:
   - http://www.practicalmalwareanalysis.com/;
   - HTTP/1.0 GET

251f4d0caf6eadae453488f9c9c0ea95

1. We find several imports using IDA Pro and string ‘cmd’ is found in Ollydb

2. Nothing is found, the malware terminates without doing anything.

3.

```
.text:0040113A    mov     byte ptr [ebp-1AFh], 'q'
.text:00401141    mov     [ebp+var_1AE], 'a'
.text:00401148    mov     [ebp+var_1AD], 'z'
.text:0040114F    mov     [ebp+var_1AC], '2'
.text:00401156    mov     [ebp+var_1AB], 'w'
.text:0040115D    mov     [ebp+var_1AA], 's'
.text:00401164    mov     [ebp+var_1A9], 'x'
.text:0040116B    mov     [ebp+var_1A8], '3'
.text:00401172    mov     [ebp+var_1A7], 'e'
.text:00401179    mov     [ebp+var_1A6], 'd'
.text:00401180    mov     [ebp+var_1A5], 'c'
.text:00401187    mov     [ebp+var_1A4], 0
.text:0040118E    mov     [ebp+var_1A0], 'o'
.text:00401195    mov     byte ptr [ebp-19Fh], 'c'
.text:0040119C    mov     [ebp+var_19E], 'l'
.text:004011A3    mov     [ebp+var_19D], '.'
.text:004011AA    mov     [ebp+var_19C], 'e'
.text:004011B1    mov     [ebp+var_19B], 'x'
.text:004011B8    mov     [ebp+var_19A], 'e'
.text:004011BF    mov     [ebp+var_199], 0
.text:004011C6    mov     ecx, 8
.text:004011CB    mov     esi, offset unk_405034
.text:004011D0    lea     edi, [ebp+var_1F0]
.text:004011D6    rep movsd
gram control flow 004011D8    movsb
.text:004011D9    mov     [ebp+var_1B8], 0
.text:004011E3    mov     [ebp+Filename], 0
.text:004011EA    mov     ecx, 43h
.text:004011EF    xor     eax, eax
.text:004011F1    lea     edi, [ebp-2FFh]
.text:004011F7    rep stosd
.text:004011F9    stosb
.text:004011FA    push    10Eh            ; nSize
.text:004011FF    lea     eax, [ebp+Filename]
.text:00401205    push    eax             ; lpFilename
.text:00401206    push    0               ; hModule
.text:00401208    call    ds:GetModuleFileNameA
.text:0040120E    push    5Ch             ; int
.text:00401210    lea     ecx, [ebp+Filename]
.text:00401216    push    ecx             ; char *
.text:00401217    call    _strrchr
.text:0040121C    add     esp, 8
```

From the figures above we see that at instruction 0x0401128 the malware moves bytes into local variables and then followed by library call to GetModuleFileNameA, _strrchr and _strcmp. This can mean that the malware is scanning for occurrence of a character and from the current file name. Below we can see the file names in the stack ready for comparison. From this we can assume that we can get the file to run by naming it ‘ocl.exe’

```
0012FC68  0040123B  RETURN to 251f4d0c.0040123B from 251f4d0c.004014C0
0012FC6C  0012FDE0  ASCII "ocl.exe"
0012FC70  0012FCB5  ASCII "251f4d0caf6eadae453488f9c9c0ea95"
0012FC74  7C910208  ntdll.7C910208
0012FC78  FFFFFFFF
0012FC7C  00000000
0012FC80  445C3A43
0012FC84  6D75636F
```

4. Two strings are being built using local variables. This string is “1qaz2wsx3edc” and “ocl.exe”. The malware designer want to make this string unrecognizable during basic static analysis and to prevent IDA Pro from reading it as a file name.

5. At call to subroutine 0x00401089, we see 2 items in the stack,

```
004012A3  .v75 0A        JNZ SHORT ocl.004012AF
004012A5  . B8 01000000  MOV EAX,1
004012AA  .vE9 27010000  JMP ocl.004013D6
004012AF  > 8D8D 10FEFFFF LEA ECX,DWORD PTR SS:[EBP-1F0]
004012B5  . 51           PUSH ECX                          Arg2
004012B6  . 8D95 50FEFFFF LEA EDX,DWORD PTR SS:[EBP-1B0]
004012BC  . 52           PUSH EDX                          Arg1
004012BD  . E8 C7FDFFFF  CALL ocl.00401089                 ocl.00401089
004012C2  . 83C4 08      ADD ESP,8
004012C5  . 8945 F8      MOV DWORD PTR SS:[EBP-8],EAX
004012C8  . 8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
00401089=ocl.00401089

Address   Hex dump                                         ASCII
00405000  00 00 00 00 00 00 00 00 00 00 00 00 F8 27 40 00  ............°'@.
00405010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00405020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00405030  63 6D 64 00 46 06 16 54 42 05 12 1B 47 0C 07 02  cmd.F♠▬TB↕↨↑G.•☻
00405040  5D 1C 00 16 45 16 01 1D 52 0B 05 0F 48 02 08 09  ]∟..E▬☺↔R♂♣▬H☻◘○
00405050  1C 14 1C 15 00 00 00 00 00 00 00 00 00 00 00 00  ∟¶∟§............
00405060  DD 16 40 00 01 00 00 00 05 00 00 C0 0B 00 00 00  ▌▬@.☺...♣..└♂...

0012FC6C  0012FDD0  Arg1 = 0012FDD0 ASCII "1qaz2wsx3edc"
0012FC70  0012FD90  Arg2 = 0012FD90
0012FC74  7C910208  ntdll.7C910208
0012FC78  FFFFFFFF
0012FC7C  00000060
0012FC80  445C3A43
0012FC84  6D75636F
0012FC88  73746E65
```

- String ‘1qaz2wsx3edc’
- Pointer to address 0x0012FD90

6.

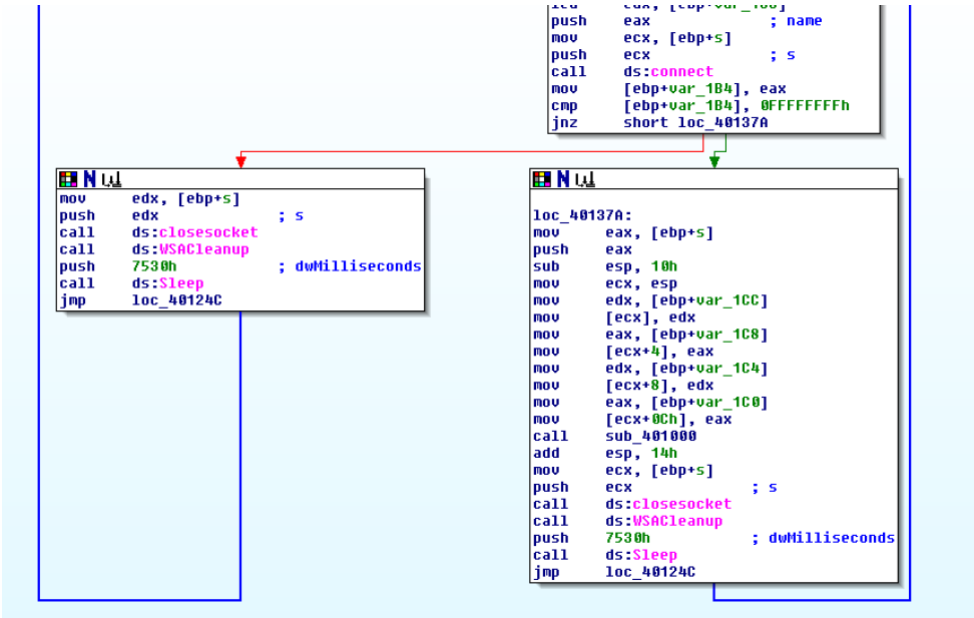
```
00401114  . 888C05 00FFFFF MOV BYTE PTR SS:[EBP+EAX-100],CL
0040111B  .^EB B7          JMP SHORT ocl.004010D4
0040111D  > 8D85 00FFFFFF  LEA EAX,DWORD PTR SS:[EBP-100]
00401123  . 5F             POP EDI
00401124  . 8BE5           MOV ESP,EBP
00401126  . 5D             POP EBP
00401127  . C3             RETN
00401128  $ 55             PUSH EBP
00401129  . 8BEC           MOV EBP,ESP
0040112B  . 81EC 04030000  SUB ESP,304
00401131  . 56             PUSH ESI
00401132  . 57             PUSH EDI
00401133  . C685 50FEFFFF  MOV BYTE PTR SS:[EBP-1B0],31
0040113A  . C685 51FEFFFF  MOV BYTE PTR SS:[EBP-1AF],71
00401141  . C685 52FEFFFF  MOV BYTE PTR SS:[EBP-1AE],61
00401148  . C685 53FEFFFF  MOV BYTE PTR SS:[EBP-1AD],7A
0040114F  . C685 54FEFFFF  MOV BYTE PTR SS:[EBP-1AC],32
00401156  . C685 55FEFFFF  MOV BYTE PTR SS:[EBP-1AB],77
0040115D  . C685 56FEFFFF  MOV BYTE PTR SS:[EBP-1AA],73
00401164  . C685 57FEFFFF  MOV BYTE PTR SS:[EBP-1A9],78
Stack address=0012FB64, (ASCII "www.practicalmalwareanalysis.com")
EAX=0000001F
Jump from 004010EA
```

From the figure above we can see that the domain name at 0x40111D is www.practicalmalwareanalysis.com

7. The loop below shows us that the malware is XORing string ‘www.practicalmalwareanalysis.com’ with string ‘1qaz2wsx3edc’ to obfuscate the domain name.

```
004010C2  . 8985 FCFEFFFF  MOV DWORD PTR SS:[EBP-104],EAX
004010C8  . C785 F8FEFFFF  MOV DWORD PTR SS:[EBP-108],0
004010D2  .v EB 0F         JMP SHORT ocl.004010E3
004010D4  > 8B8D F8FEFFFF  MOV ECX,DWORD PTR SS:[EBP-108]
004010DA  . 83C1 01        ADD ECX,1
004010DD  . 898D F8FEFFFF  MOV DWORD PTR SS:[EBP-108],ECX
004010E3  > 83BD F8FEFFFF  CMP DWORD PTR SS:[EBP-108],20
004010EA  .v 7D 31         JGE SHORT ocl.0040111D
004010EC  . 8B55 0C        MOV EDX,DWORD PTR SS:[EBP+C]
004010EF  . 0395 F8FEFFFF  ADD EDX,DWORD PTR SS:[EBP-108]
004010F5  . 0FBE0A         MOVSX ECX,BYTE PTR DS:[EDX]
004010F8  . 8B85 F8FEFFFF  MOV EAX,DWORD PTR SS:[EBP-108]
004010FE  . 99             CDQ
004010FF  . F7BD FCFEFFFF  IDIV DWORD PTR SS:[EBP-104]
00401105  . 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
00401108  . 0FBE1410       MOVSX EDX,BYTE PTR DS:[EAX+EDX]
0040110C  . 33CA           XOR ECX,EDX
0040110E  . 8B85 F8FEFFFF  MOV EAX,DWORD PTR SS:[EBP-108]
00401114  . 888C05 00FFFF  MOV BYTE PTR SS:[EBP+EAX-100],CL
0040111B  .^EB B7          JMP SHORT ocl.004010D4
0040111D  > 8D85 00FFFFFF  LEA EAX,DWORD PTR SS:[EBP-100]
```

8. In the figure below after call to connection to socket, we see the malware calling sub routine 0x401000.



Here we see the malware calling library function CreateProcessA with string ‘cmd’ as an argument. This is a reverse shell which is spawned to receive commands from an attacker. The malware use the StartUpInfo’s commands hStdInput, hStdOutput and hStdError to direct the shell commands and responses to the socket.

```
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 58h
.text:00401006                 mov     [ebp+var_14], 0
.text:0040100D                 push    44h             ; size_t
.text:0040100F                 push    0               ; int
.text:00401011                 lea     eax, [ebp+StartupInfo]
.text:00401014                 push    eax             ; void *
.text:00401015                 call    _memset
.text:0040101A                 add     esp, 0Ch
.text:0040101D                 mov     [ebp+StartupInfo.cb], 44h
.text:00401024                 push    10h             ; size_t
.text:00401026                 push    0               ; int
.text:00401028                 lea     ecx, [ebp+hHandle]
.text:0040102B                 push    ecx             ; void *
.text:0040102C                 call    _memset
.text:00401031                 add     esp, 0Ch
.text:00401034                 mov     [ebp+StartupInfo.dwFlags], 101h
.text:0040103B                 mov     [ebp+StartupInfo.wShowWindow], 0
.text:00401041                 mov     edx, [ebp+arg_10]
.text:00401044                 mov     [ebp+StartupInfo.hStdInput], edx
.text:00401047                 mov     eax, [ebp+StartupInfo.hStdInput]
.text:0040104A                 mov     [ebp+StartupInfo.hStdError], eax
.text:0040104D                 mov     ecx, [ebp+StartupInfo.hStdError]
.text:00401050                 mov     [ebp+StartupInfo.hStdOutput], ecx
.text:00401053                 lea     edx, [ebp+hHandle]
.text:00401056                 push    edx             ; lpProcessInformation
.text:00401057                 lea     eax, [ebp+StartupInfo]
.text:0040105A                 push    eax             ; lpStartupInfo
.text:0040105B                 push    0               ; lpCurrentDirectory
.text:0040105D                 push    0               ; lpEnvironment
.text:0040105F                 push    0               ; dwCreationFlags
.text:00401061                 push    1               ; bInheritHandles
.text:00401063                 push    0               ; lpThreadAttributes
.text:00401065                 push    0               ; lpProcessAttributes
.text:00401067                 push    offset CommandLine ; "cmd"
.text:0040106C                 push    0               ; lpApplicationName
.text:0040106E                 call    ds:CreateProcessA
.text:00401074                 mov     [ebp+var_14], eax
```

795f093a536f118fb4c34fcedfa42165 + 795f_driver.sys

1.

7:05:2...	795f093a536f1...	3444	RegCloseKey	SUCCESS		HKLM\System\CurrentControlSet\Control\Session Manager
7:05:2...	795f093a536f1...	3444	QueryNameInform...	BUFFER OVERFL...	Name: \D	C:\Documents and Settings\Administrator\Desktop\Lab5\79
7:05:2...	795f093a536f1...	3444	QueryNameInform...	SUCCESS	Name: \Documents and Settings\Administrator\Deskto...	C:\Documents and Settings\Administrator\Desktop\Lab5\79
7:05:2...	795f093a536f1...	3444	RegSetValue	SUCCESS	Type: REG_BINARY, Length: 80, Data: 45 AC F3 5B ...	HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
7:05:2...	795f093a536f1...	3444	SetEndOfFileInfor...	SUCCESS	EndOfFile: 8,192	C:\WINDOWS\system32\config\software.LOG
7:05:2...	795f093a536f1...	3444	SetEndOfFileInfor...	SUCCESS	EndOfFile: 8,192	C:\WINDOWS\system32\config\software.LOG
7:05:2...	795f093a536f1...	3444	Thread Exit	SUCCESS	Thread ID: 3696, User Time: 0.0000000, Kernel Time: ...	

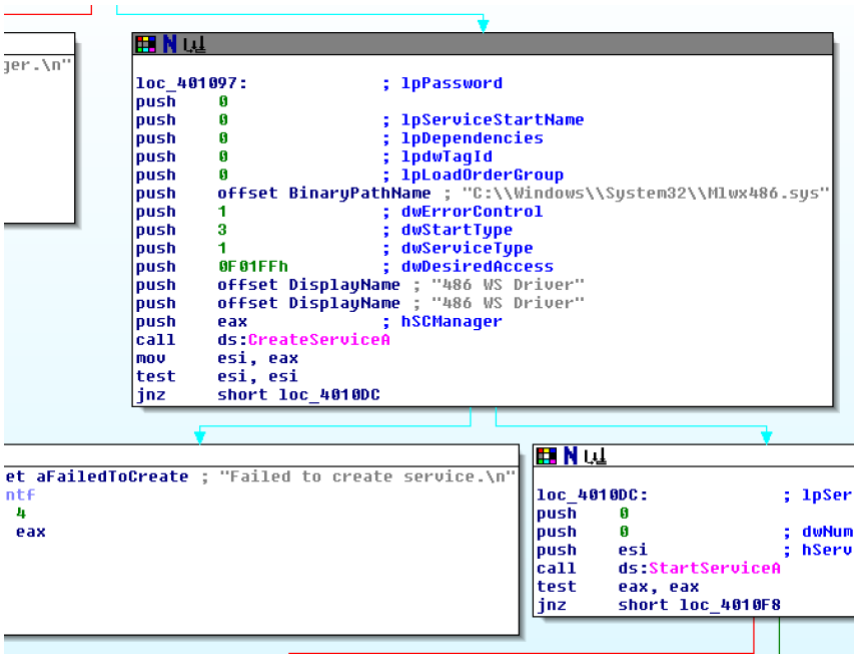
When I monitored the execution using Procmon, I learned that there was only a single "RegSetValue " operation performed where the malware sets a value to 'HKLM\SOFTWARE\Cryptography\RNG\Seed'

2. We can start by setting breakpoint at the ControlService at address 0x401080. Just before this address we see that the malware pushes 1 to the stack. We find that this means "SERVICE_CONTROL_STOP". This function is used to unload the driver. Next, we load the .sys file to Windbg, and set a breakpoint at the "DriverUnload" function at offset 0xf7c47486, we see that is being triggered when we restart the malware.

3. The malware is used to load a driver that is used to make changes to the registry key and set values of the Windows XP firewall. It disables the firewall of the victim's computer so that it cannot be turned back on and then unloads the driver file.

3f3a29ca2467d2d05feac9d233366f45

1. The malware creates file C:\Windows\System32\Mlwx486.sys. This is shown below. It uses CreateFileA and WriteFile library functions to do this.



2. There is a kernel component. The kernel is loaded using service name “486 WS Driver”.

3. The malware is a rootkit. It uses a SSDT hook to patch the file name stored in the file directory. This patch function overwrites the filenames that begin with “Mlwx”. This is done to hide the files.

f72d773f13ceb6b842a9d29c56f8880f + f72d_driver.sys

1. This malware loads driver f72d_driver.sys and then displays an advertisement to the user every 30 seconds on Internet Explorer. The driver is also used to hide the advertisement for 30 seconds.

2. The only way to stop the malware from running is by restarting the computer. If we do this the driver will not be loaded.

3. The driver unlinks a process from the process list by hiding it using the DeviceIoControl requests to modify the processes in the linked list.