Win32.Infostealer.Dexter

StarDust Variant

Malware Analysis Report by Jake McLellan

Contents

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, PEview, 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, PEview indicates that the .text virtual size and raw size are similar, which is an indicator that the code likely isn't packed.

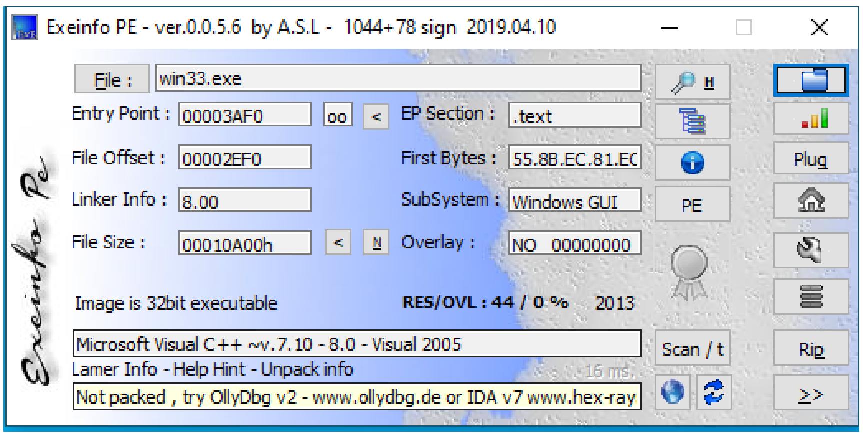


Figure 1: Exeinfo PE output

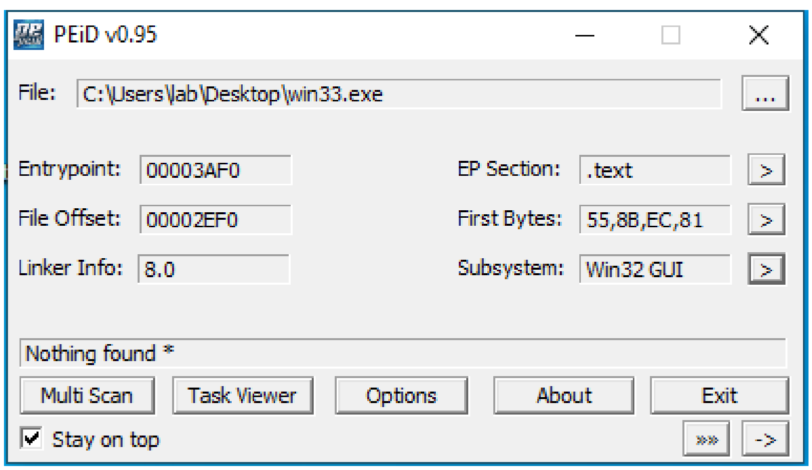


Figure 2: PEiD output

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: PEview virtual size versus size of raw data

Dependency Information

Basic static analysis also reveals the external functions imported by this malware sample. PEview 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.

```
Software\HelperSolutions Software
Software\Microsoft\Windows\CurrentVersion\Run
.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

```
.exe;.bat;.reg;.vbs;
Java Security Plugin
%s\%s
javaplugin
Java Security Plugin
%s\%s\%s.exe
Sun Java Security Plugin
Sun Java Security Plugin
Sun Java Security Plugin
Software\Microsoft\Windows\CurrentVersion\Policies\Associations
LowRiskFileTypes
Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0
1806
Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0
1806
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

```
debug.log
|%s:
SecureDll.dll
SecureDll.dll
strokes.log
%s\%s
tmp.log
%s\%s
val1
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

```
Windows 2000
Windows XP
Windows XP Professional x64
Windows Server 2003
Windows Home Server
Windows Server 2003 R2
Windows Vista
Windows Server 2008
Windows Server R2
Windows 7
64 Bit
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.

```
151.248.115.107
```

```
/w19218317418621031041543/gateway.php
```

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

```
response=
page=
&ump=
&ks=
&opt=
&unm=
&cnm=
&view=
&spec=
&query=
&val=
&var=
```

```
download-
update-
checkin:
scanin:
uninstall
```

Figures 12 and 13: URL parameters and bot commands

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

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

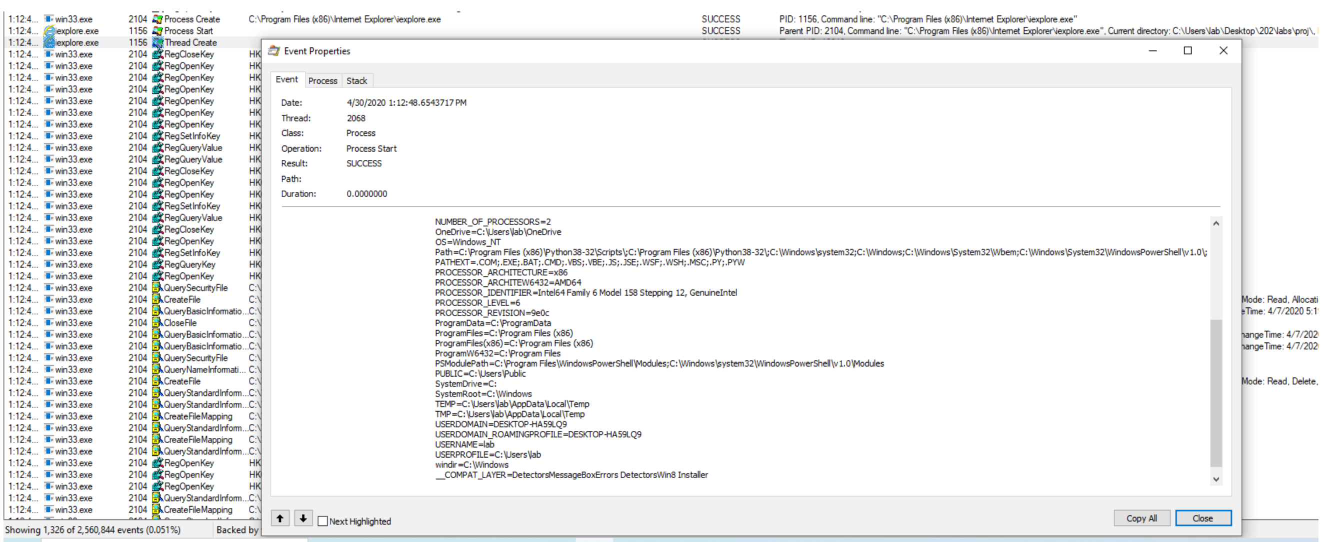
Basic Dynamic

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

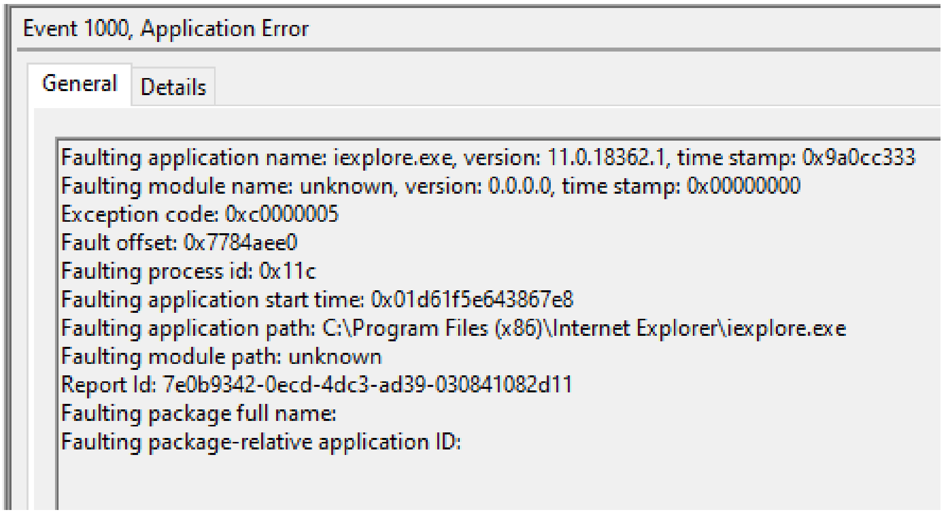


Figure 17: Crash report from Internet Explorer as seen from the Windows Event Viewer

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 an explained in the following series of screenshots.

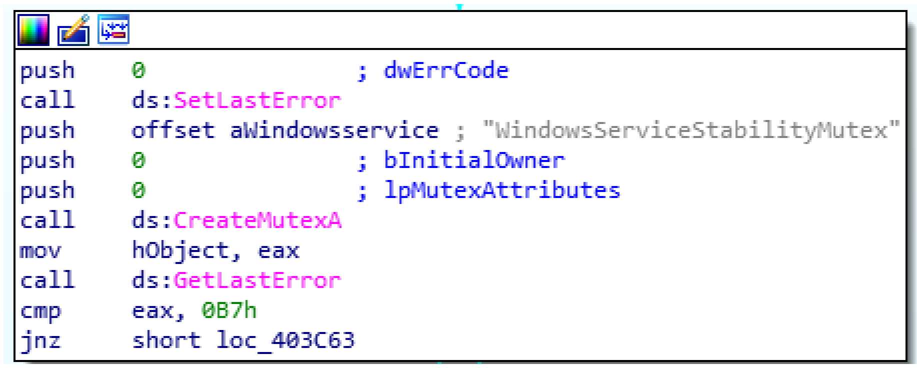


Figure 18: the WindowsServiceStabilityMutex is created as a marker to ensure only one copy of the malware is installed/running at a given time

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.

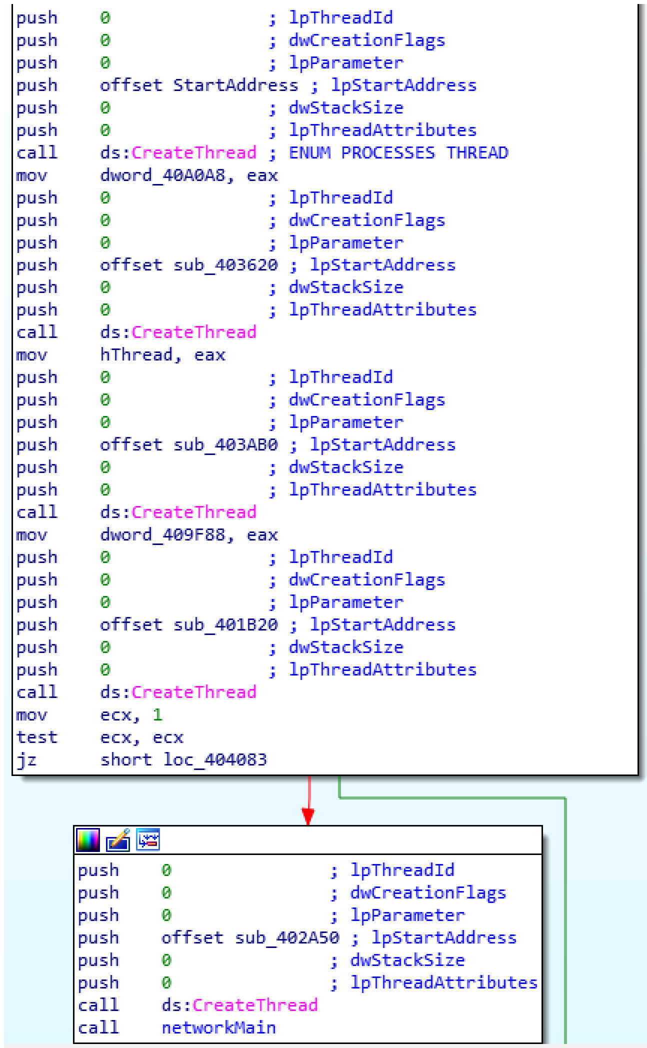


Figure 23: Multiple threads are spawned by the malware, but the most important is the “Enumerate Process” thread. This process will be explained in greater detail later in this section.

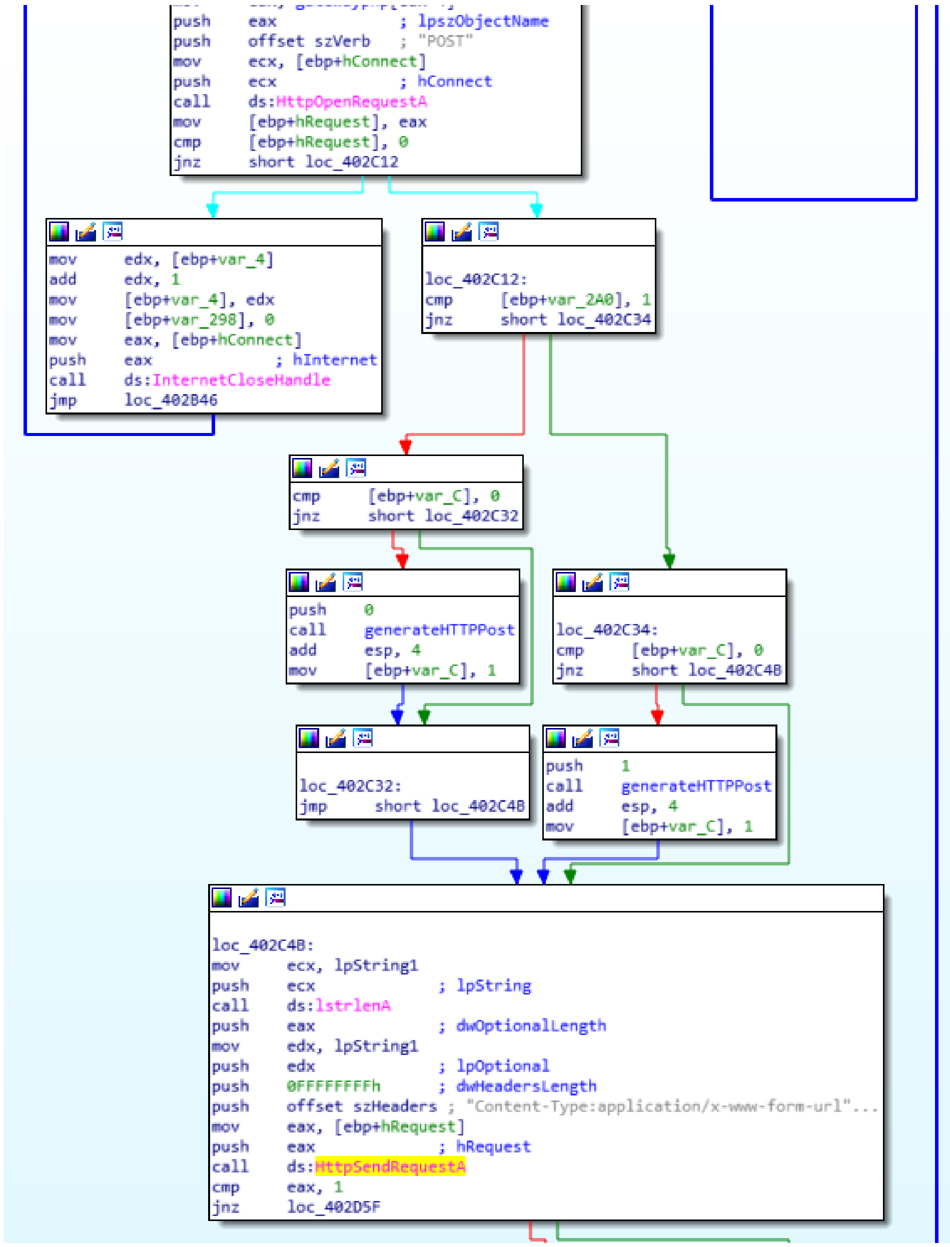


Figure 24: The final step of the main function is to enter the networking loop. This infinite loop is responsible sending and receiving information over HTTP between the client and server. It uses helper functions to generate POST requests and parse the server's response. Commands from the server include "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.

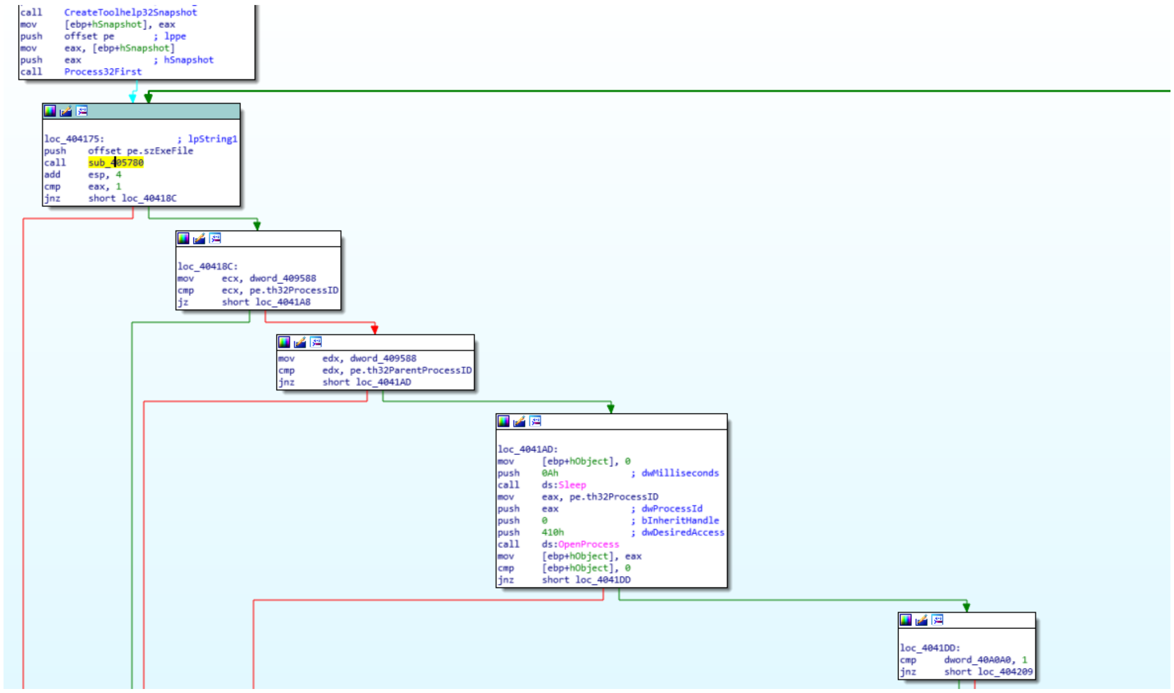


Figure 25: A small subsection of the enumProcesses algorithm that shows the CreateToolhelp32Snapshot function being used to begin analyzing process memory for magstripe card data.

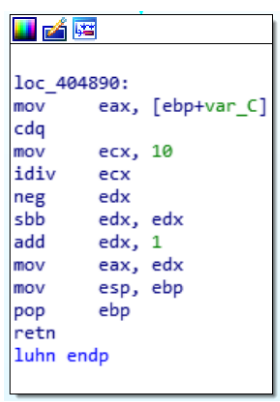


Figure 26: The return statement of the Luhn algorithm function which is used to determine whether the check digit of a card number is valid.

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.

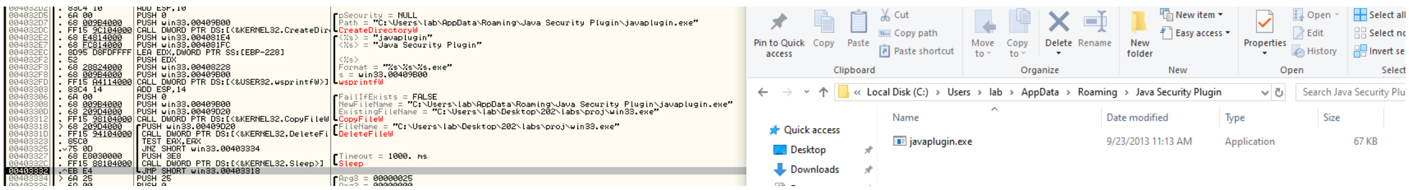
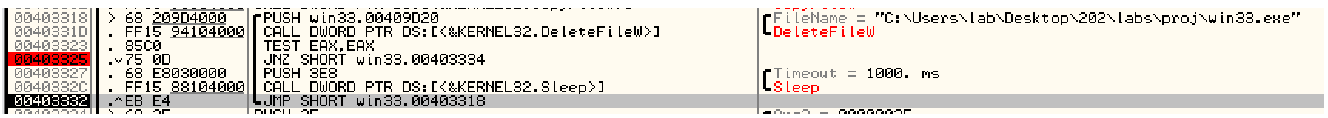
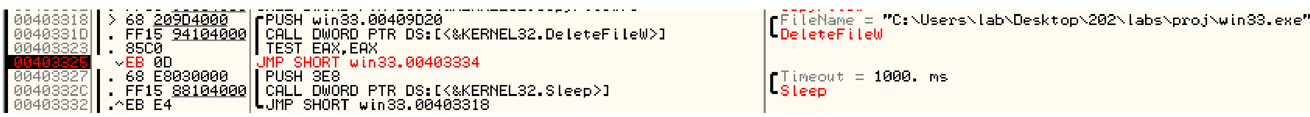


Figure 27: A side-by side image that shows the malware after copying itself into the user's AppData directory





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"
HKU\.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.

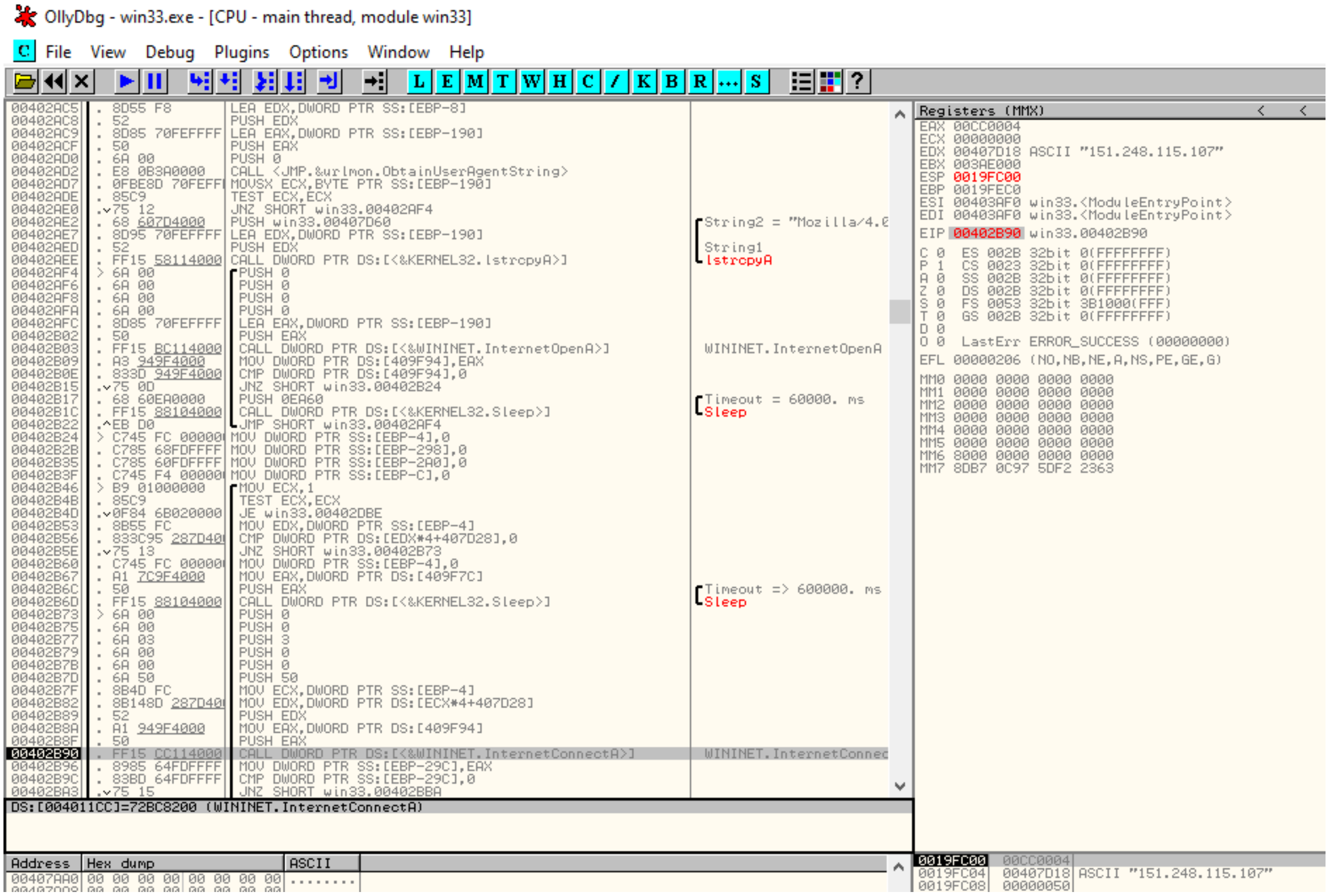
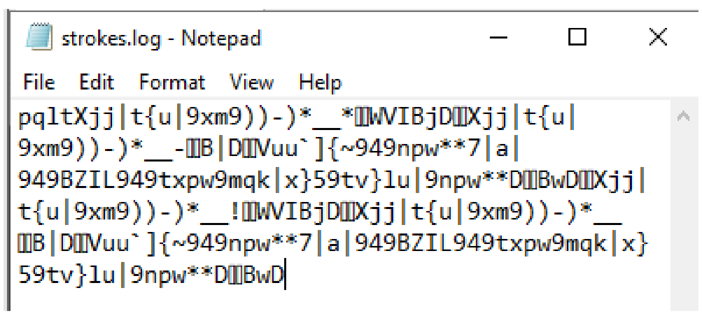


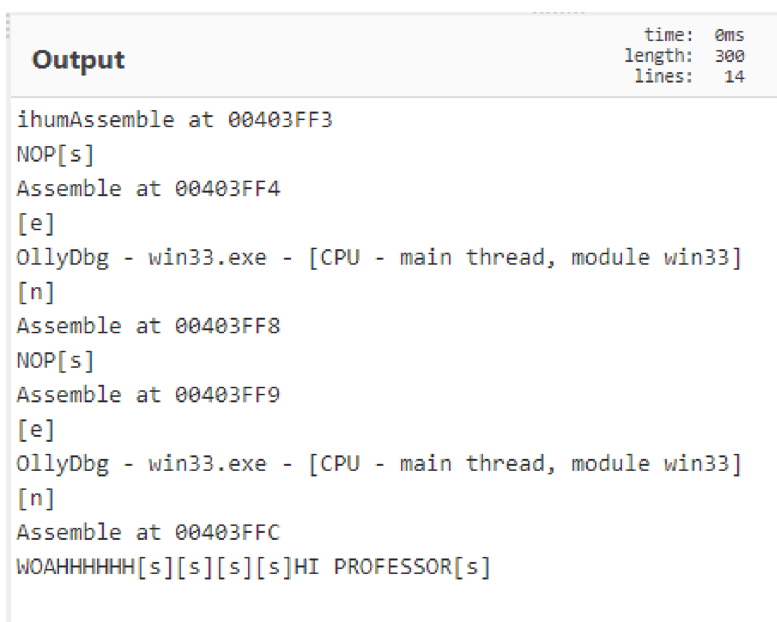
Figure 31: Once the malware enters the network loop, it calls the InternetConnectA function with the IP address discovered during basic static analysis, confirming the previous hypothesis. It is seen using the user agent string which was also found during basic static analysis.

Figure 32: The HTTP request being built to connect to the attacker's server

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.





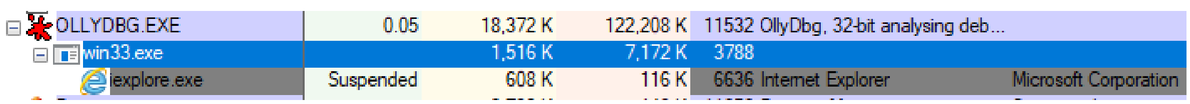
Figures 33 and 34: Encrypted data in the strokes file followed by the decrypted data

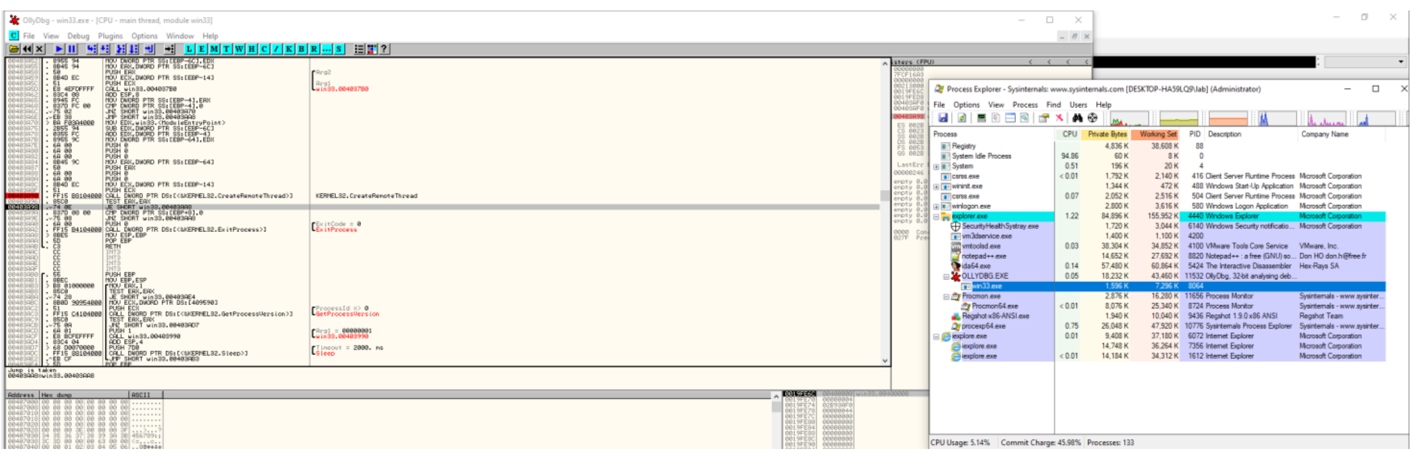
Challenges

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.

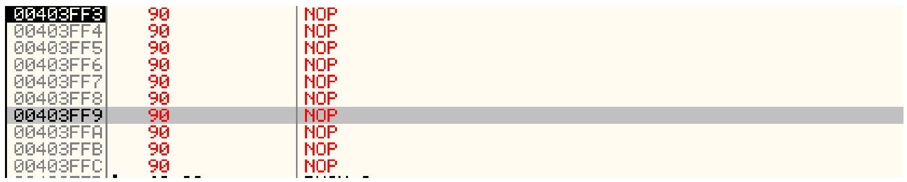


Figure 37: Replacing the function call and surrounding area with NOPs stops this issue

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.

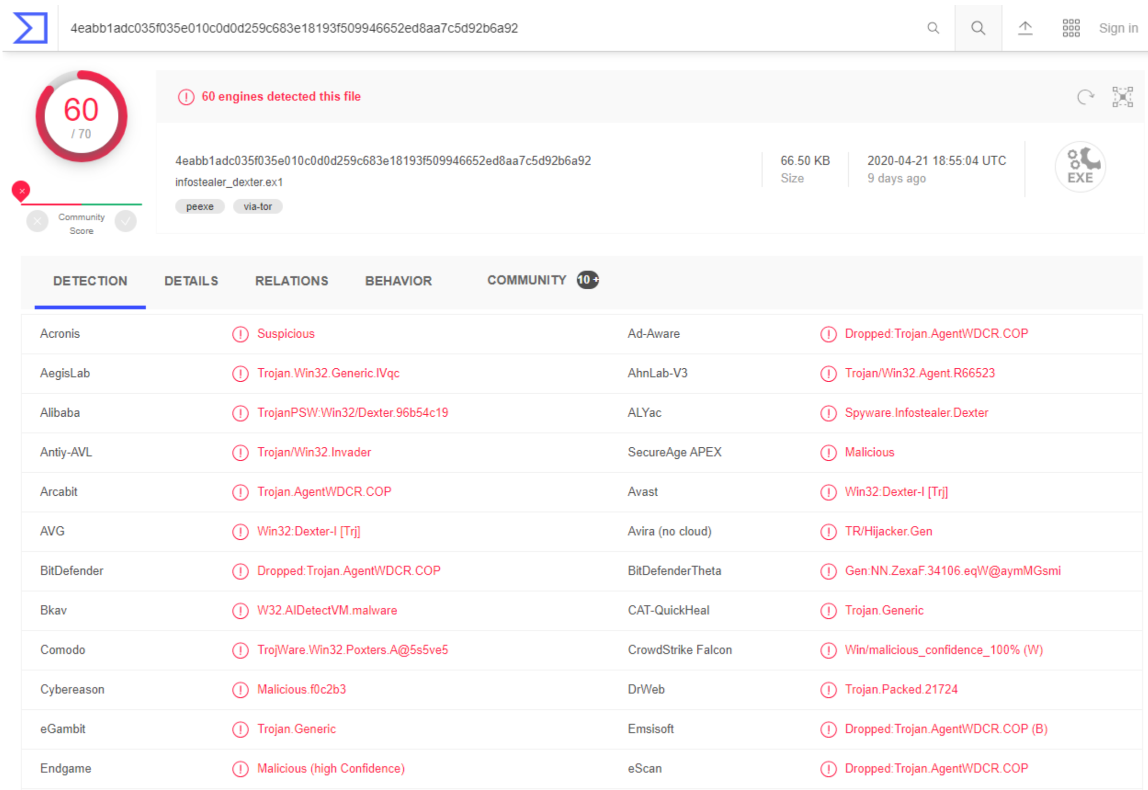


Figure 38: VirusTotal detection rate of 60/70 for the hash

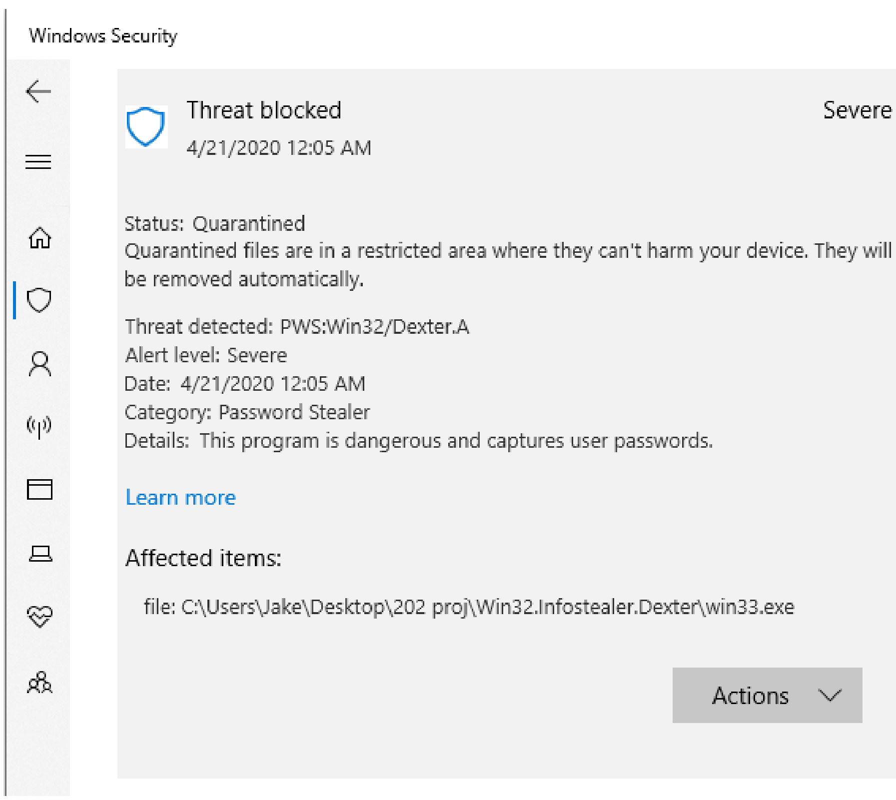


Figure 39: Windows Defender quarantined the file while also recognizing it as Win32/Dexter.A