Assignment 8

Adam Livingston

University Of Arizona

CYBV 454 MALWARE THREATS & ANALYSIS

Professor Galde

10 May 2023

LAB 11-1

- Lab11-01.exe: 795f093a536f118fb4c34fcedfa42165 (Figure 1)

Basic properties	
MD5	a9c55bb87a7c5c3c923c4fa12940e719
SHA-1	d971656c6c605a6e2130ab83a38420e655428f94
SHA-256	57d8d248a8741176348b5d12dcf29f34c8f48ede0ca13c30d12e5ba0384
Vhash	054046651d151028z33!z

Figure 1: Virus Total MD5 Hash for file Lab11-01.exe.

Virus Total found 50 of 70 matching security vendor signatures for Lab11-01.exe (Figure 2) and has a compilation timestamp of 2011-11-06 at 18:55:06 UTC (Figure 3).

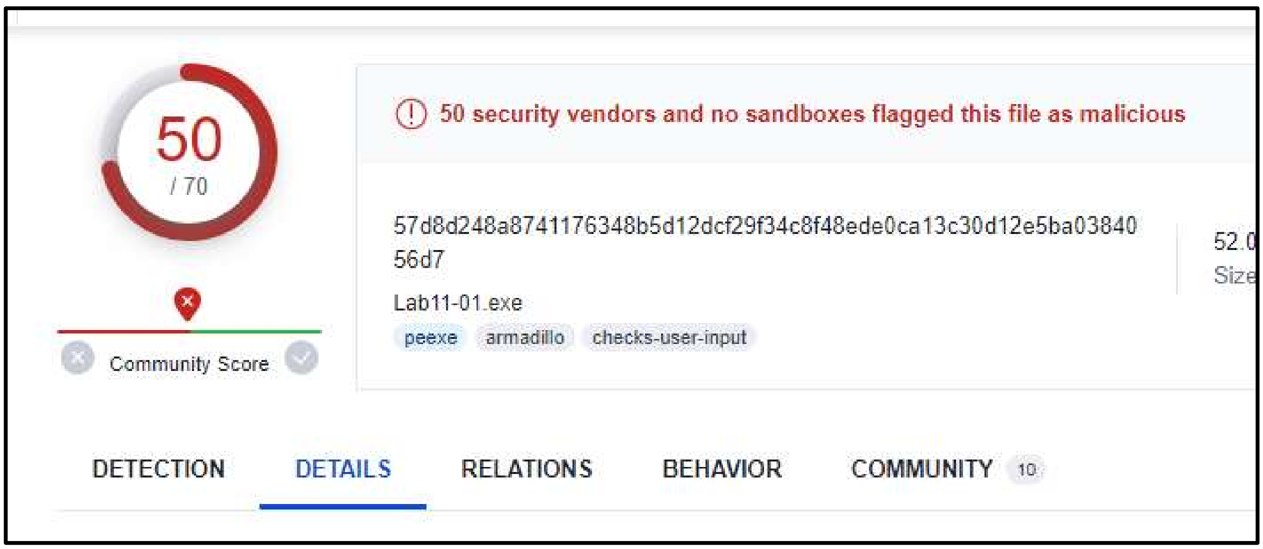


Figure 2: Virus Total Findings for file Lab11-01.exe.

Header	
Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2011-11-06 18:55:06 UTC
Entry Point	5255
Contained Sections	4

Figure 3: Virus Total compilation timestamp for Lab11-01.exe.

The file appears to only import two dynamic linked libraries: kerenel32 and advapi32 (Figure 7). Kernel32.dll indicates that it has the capability to access and modify the core OS functions. Advapi32.dll indicates that core Windows components will be altered, such as the Service Manager and Registry.

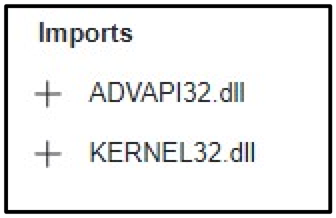


Figure 4: Virus Total imports for Lab11-01.exe.

Virus Total also reports that the file has behaviors of persistence, privilege escalation, defense evasion, and input-capture capabilities (Figures 5 and 6). Based on the advapi32 import and the fact that this malware establishes persistence through an autstart registry key, looking at service creation events and the CurrentControlSet services registry will be important.

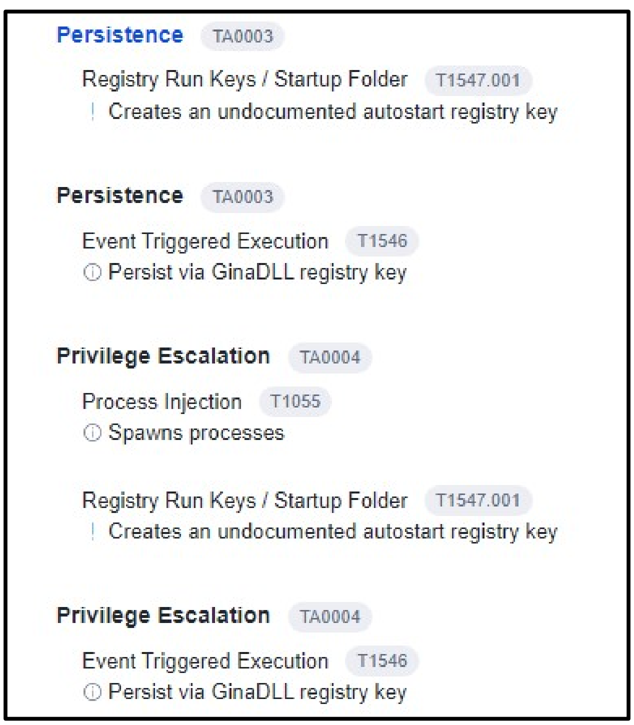


Figure 5: Virus Total behavior for file Lab11-01.exe.

Figure 6: Virus Total behavior for file Lab11-01.exe.

LAB 11-1

LAB 11-1 Question 1

What does the malware drop to disk?

BLUF: msgina32.dll

Within the section .rsc portion of the malware when viewed in PEview, there is a section titled "BINARY TGAD" and has begins with the hex values of "4D 5A" (Figure 7). This is a file signature for a portable executable file according to garykessler.net (Figure 8). This means that Lab11-01.exe contains an additional PE file.

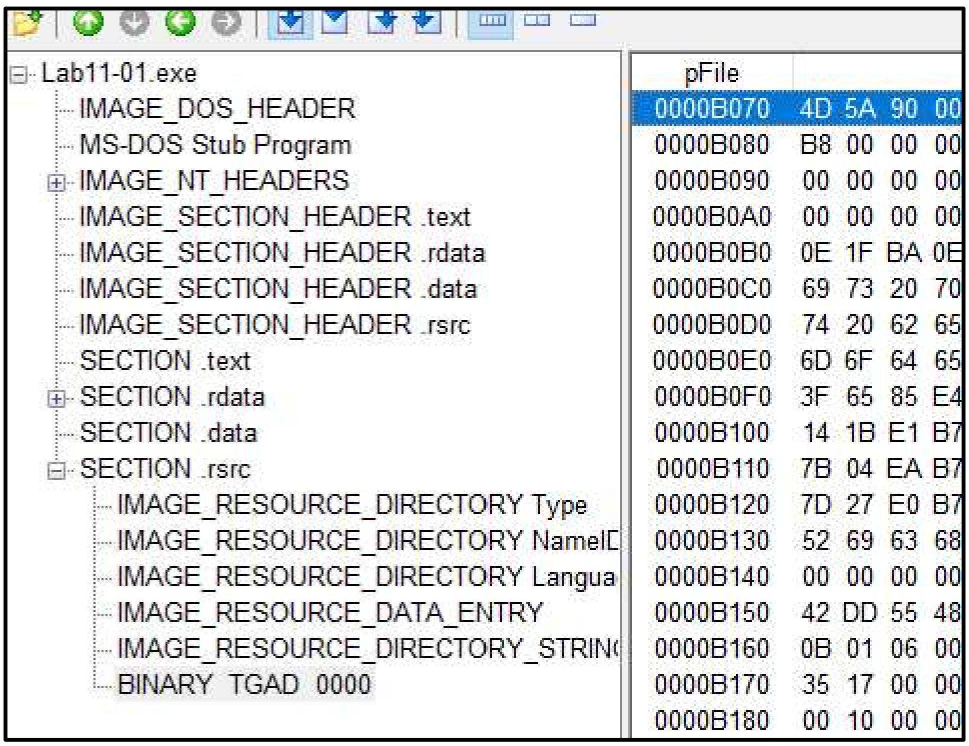
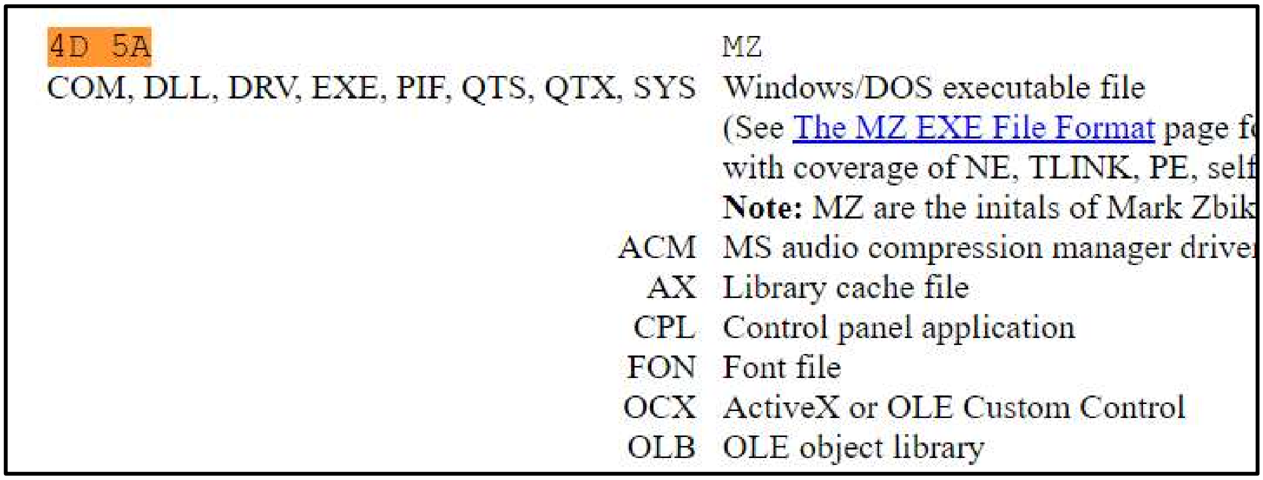


Figure 7: PEview of BINARY TGAD.



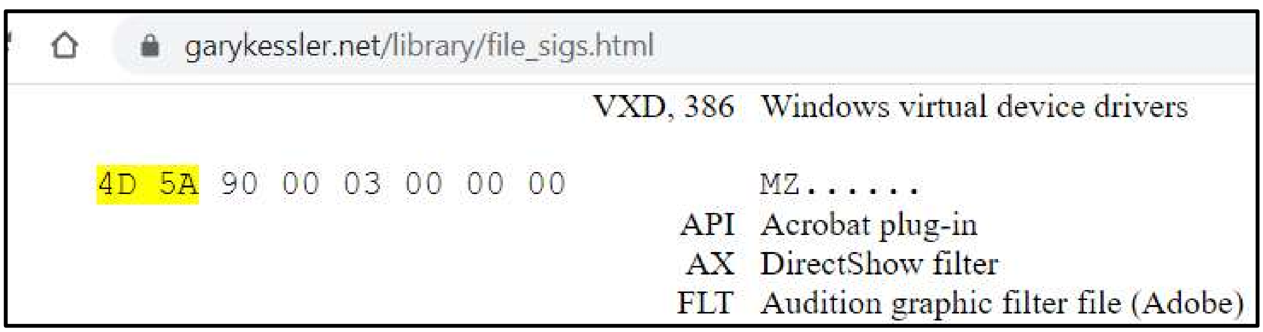
4D 5A	MZ
COM, DLL, DRV, EXE, PIF, QTS, QTX, SYS	Windows/DOS executable file (See The MZ EXE File Format page f with coverage of NE, TLINK, PE, self **Note:** MZ are the initals of Mark Zbik
ACM	MS audio compression manager drive
AX	Library cache file
CPL	Control panel application
FON	Font file
OCX	ActiveX or OLE Custom Control
OLB	OLE object library

Figure 8: 4D 5A shows BINARY TGAD is a PE file.

However, the hex values of the section header extend to 4D 5A 90 00 03 00 00 00 (Figure 9). Referencing Kessler, we see that the PE is a file type of either Acrobat plug-in, DirectShow filter, or Audition graphic filter file for Adobe (Figure 10).

pFile	Raw D
0000B070	4D 5A 90 00 03 00 00 00
0000B080	B8 00 00 00 00 00 00 00

Figure 9: Full file signature header.



garykessler.net/library/file_sigs.html

	VXD, 386	Windows virtual device drivers
4D 5A 90 00 03 00 00 00		MZ......
	API	Acrobat plug-in
	AX	DirectShow filter
	FLT	Audition graphic filter file (Adobe)

Figure 10: Potential specific file types.

A strings analysis of this section shows a potential file drop of a file named “msgina32.dll” (Figure 11). It also shows that it will write some value into the registry at SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon. This suggests that this registry key will be altered with the path of the dropped msgina32.dll file.

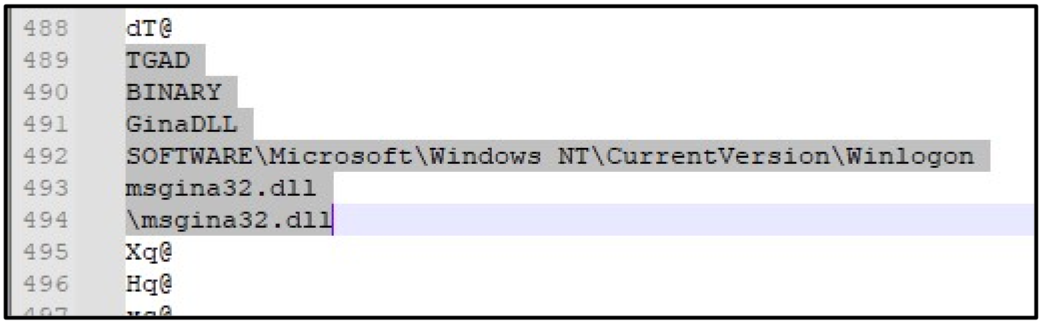


Figure 11: msgina32.dll and registry key.

To test this theory, a dynamic analysis was done to see where any files are dropped as an IDA analysis did not specifically specify the file name as it used pointers instead of a specific file name string when calling CreateFileA. Procmon captured the anticipated file creation of msgina32.dll within the same directory as Lab11-01.exe (Figure 12). This was confirmed as the file now exists within the directory, also shown in Figure 12.

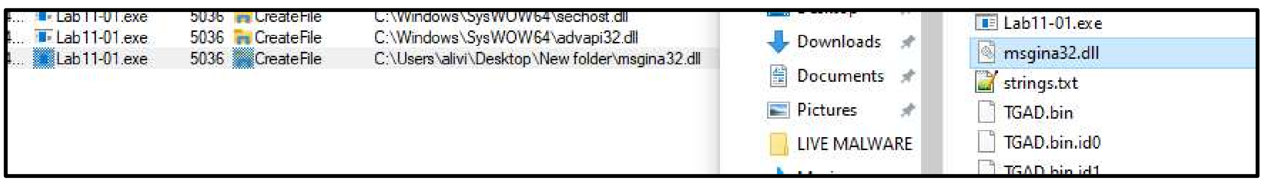


Figure 12: msgina32.dll and registry key.

LAB 11-1 Question 2

How does the malware achieve persistence?

BLUF: Replacing the normal msgina32.dll path in Winlogon with the new one.

After running the malware to find the file drops in Question 1, the registry key identified within the strings analysis of Lab11-01.exe was looked at within regedit. The registry key titled "GinaDLL" was created with data containing the path to the .dll file created (Figure 13). We also see in the procmon capture that Lab11-01.exe successfully set the value of GinaDLL (Figure 14).

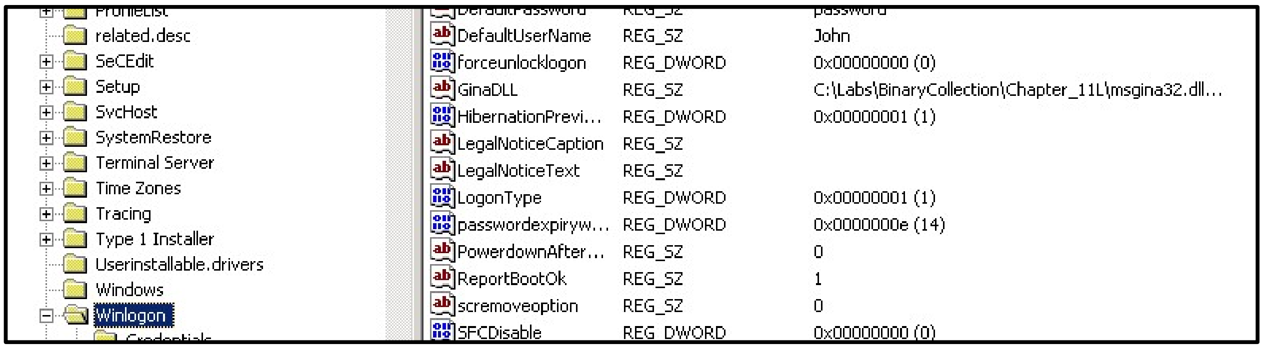


Figure 13: GinaDLL in registry.

Figure 14: GinaDLL value captured by procmon.

To detail this process, additional static analysis was done. We see in Lab11-01.exe within IDA that there is a call to RegSetValueA after a push of the GinaDLL string within the variable ValueName (Figure 15). Since this action occurs at 0x401047, the file was opened in OllyDbg and a breakpoint was set at that address.

```
.text:00401032
.text:00401032     loc_401032:
.text:00401032 008 mov     ecx, [ebp+cbData]
.text:00401035 008 push    ecx              ; cbData
.text:00401036 00C mov     edx, [ebp+lpData]
.text:00401039 00C push    edx              ; lpData
.text:0040103A 010 push    1                ; dwType
.text:0040103C 014 push    0                ; Reserved
.text:0040103E 018 push    offset ValueName ; "GinaDLL"
.text:00401043 01C mov     eax, [ebp+phkResult]
.text:00401046 01C push    eax              ; hKey
.text:00401047 020 call    ds:RegSetValueExA ; Indirect Call Near Procedure
.text:0040104D 008 test    eax, eax         ; Logical Compare
.text:0040104F 008 jz      short loc_401062 ; Jump if Zero (ZF=1)
```

Figure 15: IDA showing the registry value being set.

We see in Olly that the value name GinaDLL is indeed pushed onto the stack in addition to EDX containing the path to the newly-created and malicious msgina32.dll, confirming this is where the value is set in the registry (Figure 16).

```
BufSize                        Registers (FPU)
                               EAX 00000118
Buffer                         ECX 00000104
ValueType = REG_SZ             EDX 0019FE18 ASCII "C:\Users\alivi\Desktop\New folder\msgina32.d
Reserved = 0                   EBX 0000000E
ValueName = "GinaDLL"          ESP 0019FDDC
                               EBP 0019FDF8
hKey                           ESI 004080B2 Lab11-01.004080B2
RegSetValueExA                 EDI 0019FE47

                               EIP 00401047 Lab11-01.00401047

                               C 0  ES 002B 32bit 0(FFFFFFFF)
hObject                        P 1  CS 0023 32bit 0(FFFFFFFF)
CloseHandle                    A 0  SS 002B 32bit 0(FFFFFFFF)
                               Z 1  DS 002B 32bit 0(FFFFFFFF)
```

Figure 16: OllyDbg showing call to RegSetValueExA.

GinaDLL is used by the Winlogon service to provide a user interface for the user to log on and authenticate their credentials. Since the path is now set to the newly-created file within the same directory as Lab11-01.exe, this is how the malware established persistence. Instead of the regular GinaDLL used by Winlogon, it is now the one that was created.

LAB 11-1 Question 3

How does the malware steal user credentials?

Since it was identified that GinaDLL is used to provide a user interface for entering in user credentials, it can be hypothesized that the newly-created file intercepts them. This is a well-known exploit used by attackers and is known as GINA Interception (article here).

Strings was run on the newly-created msgina32.dll to begin confirming our suspicions. We see that there is a string to MSGina.dll as well as a format string for a user to input, or for the malware to capture, a username and password (Figure 17). The strings also captured a lot of functions with the prefix "Wlx". There is also a reference to msutil32.sys, which is a file unknown to be native to Windows.

```
WlxShutdown
WlxStartApplication
WlxWkstaLockedSAS
GinaDLL
Software\Microsoft\Windows NT\CurrentVersion\Winlogon
MSGina.dll
UN %s DM %s PW %s OLD %s
WlxLoggedOutSAS
ErrorCode:%d ErrorMessage:%s.
%s %s - %s
msutil32.sys
```

Figure 17: msgina32.dll strings.

We see in IDA Pro in DLLMain of msgina32.dll that the malware calls a few important external functions (Figure 18). We see a call to lstrcatW after "\\MSGina" and the return string from GetSystemDirectoryW is pushed onto the stack. The function lstrcatW takes the string in ECX as the as the destination with "MSGina" as the source and will concatenate them together. This is likely the destination directory for the MSGina.dll payload. Then LoadLibraryW is called with the same file name pushed as the argument, storing it into the hLibModule variable (on Windows

XP, IDA identified it as hModule). LoadLibraryW is used to load a .dll (in this case, MSGina.dll) into a process' address space, with the handle to that address space being the return value. Therefore, any time that hLibModule is referenced within the code will allow the process to access MSGina.dll. MSGina.dll is also a windows-native .dll with similar functionality. Therefore, since we already identified that msgina32.dll is intercepting user login credentials, it most likely needs access to the legitimate functions that MSGina.dll provides.

```
.text:1000105F 208 push    esi
.text:10001060 20C mov     esi, [esp+20Ch+hLibModule]
.text:10001067 20C push    esi              ; hLibModule
.text:10001068 210 call    ds:DisableThreadLibraryCalls ; Indirect Call Near Procedure
.text:1000106E 20C lea     eax, [esp+20Ch+Buffer] ; Load Effective Address
.text:10001072 20C push    104h             ; uSize
.text:10001077 210 push    eax              ; lpBuffer
.text:10001078 214 mov     hModule, esi
.text:1000107E 214 call    ds:GetSystemDirectoryW ; Indirect Call Near Procedure
.text:10001084 20C lea     ecx, [esp+20Ch+Buffer] ; Load Effective Address
.text:10001088 20C push    offset String2   ; "\\MSGina"
.text:1000108D 210 push    ecx              ; lpString1
.text:1000108E 214 call    ds:lstrcatW      ; Indirect Call Near Procedure
.text:10001094 20C lea     edx, [esp+20Ch+Buffer] ; Load Effective Address
.text:10001098 20C push    edx              ; lpLibFileName
.text:10001099 210 call    ds:LoadLibraryW  ; Indirect Call Near Procedure
.text:1000109F 20C xor     ecx, ecx         ; Logical Exclusive OR
.text:100010A1 20C mov     hLibModule, eax
.text:100010A6 20C test    eax, eax         ; Logical Compare
.text:100010A8 20C setnz   cl               ; Set Byte if Not Zero (ZF=0)
.text:100010AB 20C mov     eax, ecx
.text:100010AD 20C pop     esi
.text:100010AE 208 add     esp, 208h        ; Add
.text:100010B4 000 retn    0Ch              ; Return Near from Procedure
```

Figure 18: msgina32.dll DLLMain.

Because we know that this malware is GINA Interception, there are two export functions that are critical to analyze: WlxLoggedOnSAS and WlxLoggedOutSAS. WlxLoggedOnSAS immediately pushes an offset and calls sub_10001000, not calling the true WlxLoggedOnSAS function (Figure 19). The subroutine simply has code that calls GetProcAddress for MSGina.dll, returns it, and then WlxLoggedOnSAS jumps to the address.

```
text:10001350      pReserved= dword ptr  0Ch
text:10001350
text:10001350 000 push    offset aWlxloggedonsas_0 ; "WlxLoggedOnSAS"
text:10001355 004 call    sub_10001000     ; Call Procedure
text:1000135A 000 jmp     eax              ; Indirect Near Jump
text:1000135A      WlxLoggedOnSAS endp
text:1000135A
```

Figure 19: WlxLoggedOnSAS simply calls sub_10001000.

WlxLoggedOutSAS is called when the user logs out of the system. Once again, it resolves the address to MSGina.dll. However, instead of returning to the MSGina.dll address as seen previously, it continues. It pushes the previously-identified format string and calls sub_10001570 (Figure 20).

```
.text:100014EF 008 mov     ecx, [esi+0Ch]
.text:100014F2 008 mov     edx, [esi+8]
.text:100014F5 008 push    ecx
.text:100014F6 00C mov     ecx, [esi+4]
.text:100014F9 00C push    edx
.text:100014FA 010 push    ecx
.text:100014FB 014 push    eax             ; Args
.text:100014FC 018 push    offset aUnSDmSPwSOldS ; "UN %s DM %s PW %s OLD %s"
.text:10001501 01C push    0               ; dwMessageId
.text:10001503 020 call    sub_10001570    ; Call Procedure
.text:10001508 020 add     esp, 18h        ; Add
```

Figure 20: WlxLoggedOnSAS simply calls sub_10001000.

Figure 21 shows sub_10001570 and what appears to be the credential-logging aspect of the malware. First, it opens the file msutil32.sys which is not a native Windows file. Since Winlogon resides in the \System32 folder, and Winlogon uses msgina32.dll, that is most likely where msutil32.sys will reside. Additionally, there is a format string that is pushed at 0x100015D6 followed by the file name prior to calling fwprintf. Therefore, we can conclude that msutil32.sys is storing the user’s login credentials.

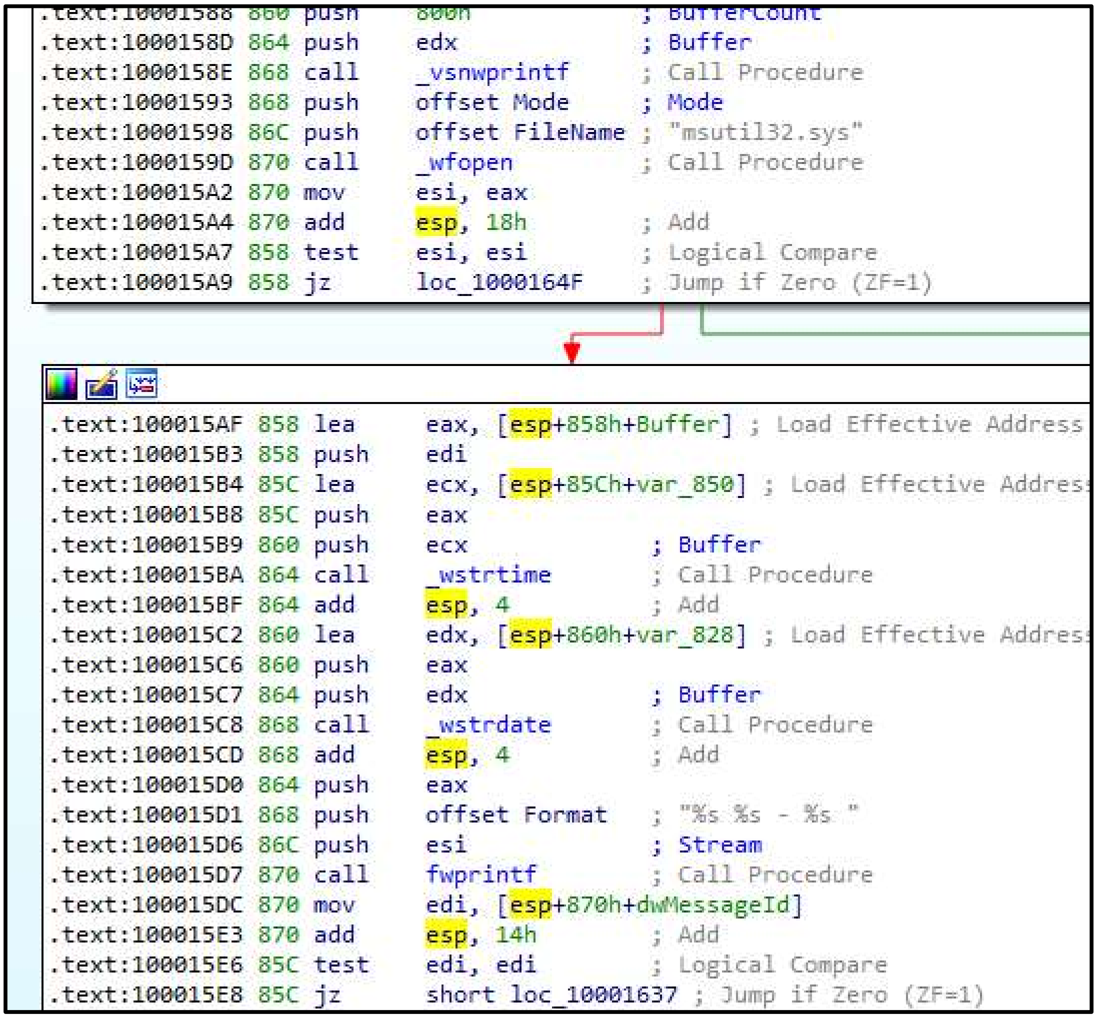


Figure 21: Capturing login credentials.

To dynamically analyze this, the VM needs to restart and the user needs to log back in. There is not a way that is known to this analyst how to force the malware behavior without going through these steps. First, the \system32 folder was checked prior to logging out and was confirmed that msutil32.sys was not present. This was expected due to the static analysis showing the creation of the file when the user logs out. However, after logging back in, msutil32.sys was present in \System32 (Figure 22). Opening the .sys file with Notepad++, we confirm that the login credentials were captured (Figure 23).

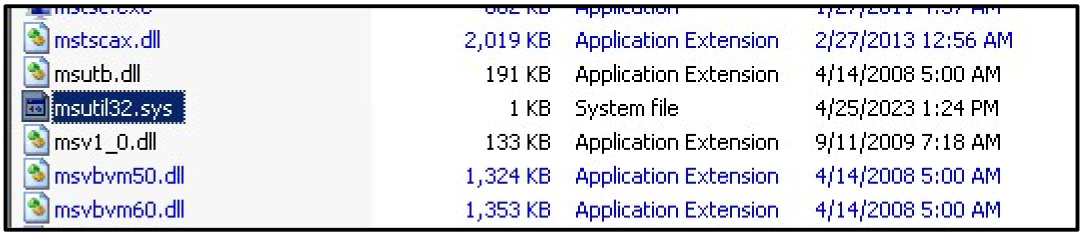


Figure 22: msutil32.sys exists after logging off and back in.

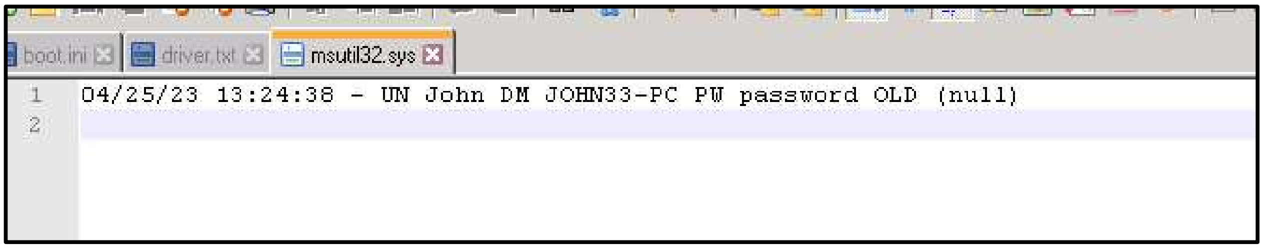


Figure 23: Captured login credentials.

LAB 11-1 Question 4

What does the malware do with stolen credentials?

As stated in Question 3, specifically in Figures 22 and 23, the malware stores the stolen credentials in a file named msutil32.sys. This file is created after the user logs out. Then, when a user logs back in, it stores the credentials that they enter into msutil32.sys. This is done by using legitimate windows processes and dynamic linked libraries, specifically Winlogon and msgina32 and msgina.dll. This malware inserted itself into the logon process to give the user the full login functionality, but their credentials were captured into a file that appears legitimate.

Although the credentials were placed into a file, there wasn't any indication that the credentials were exfiltrated over a c2 channel. There were not references to domains nor any unrecognized network captured. Therefore, it is unlikely that any unauthorized user will actually see and use them. However, if the user of the machine allows the malware owner to access the machine physically or remotely, then the credentials can be exfiltrated.

LAB 11-1 Question 5

How can you use this malware to get user credentials from your test environment?

As stated in Questions 3 and 4 and shown in Figures 22 and 23, one can get credentials from the test environment after rebooting the machine. This is because msutil32.sys is created after the user logs off. An additional test was done to see if it captured invalid credentials, but found that it does not. The malware only captures credentials that are valid during the login process and stores them with the rest of the credentials that it captured (Figure 24).

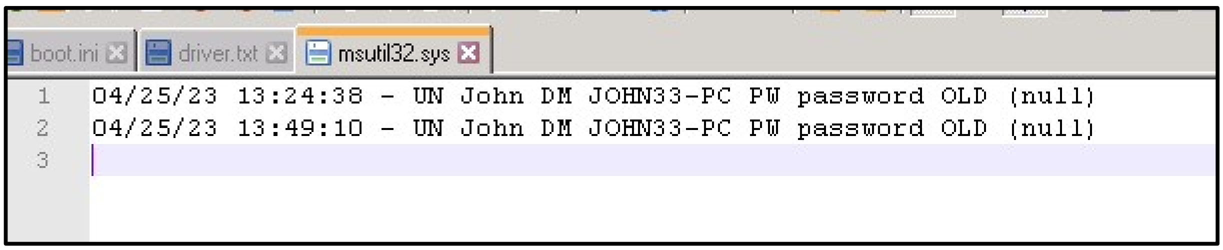


Figure 24: Captured login credentials for first and second login.

The book states that this malware has the functionality to capture credentials across multiple user accounts. But this could not be tested on the provided VM due to only one username being accessible to the analyst.