

Analysis of a Trojan Sample

Friday, February 19, 2021

Matteo Saltini

saltinimatteo98@gmail.com

Practical 1 – Malware Reverse Engineering

Contents

Executive Summary.....	3
Static Analysis.....	3
Basic Identification	3
Packing.....	3
Imports	4
Strings	5
Interesting Functions.....	8
Dynamic Analysis.....	9
Initial Behavior.....	9
Network Behavior.....	10
Registry Keys	11
Files and Processes.....	11
Indicators of Compromise.....	13
Conclusion	13

Executive Summary

This sample is a Trojan that belongs to the Carberp family, a type of malware that steals banking credentials by hooking internet functions and logging keystrokes. The malware is packed in a nontraditional way, it has a significant number of import and strings, but the entropy is close to 8 for the text, data, and rdata sections. The packer used by this malware is called Mystic Compressor. Because the malware imports the following library functions, LoadLibraryA, WSASStartup, GetForegroundWindow, WriteProcessMemory, and ZWQueueApcThread, it is safe to assume that it will call other functions that have not been imported, connect to the, log keystrokes, and inject itself into other processes.

The registry changes are not that significant, all the changes are mostly done to allow Internet Explorer to show content that might otherwise be flagged either as malicious or as phishing scams. The malware drops a copy of itself into the Startup folder to make itself persistent and creates various folders in which to store either downloaded files or stolen information.

The malware will connect to C&C servers, namely fromamericawhichlov.com, hillaryklinton.com, and malborofrientro.com, to download configuration files and upload compressed files, archives, and other miscellaneous files.

Static Analysis

Basic Identification

Analyzing the malware in PEStudio shows multiple multiple important details. The compiler stamp is set to Nov 19, 2008 and the debugger stamp is set to Jun 19, 2011. When examining the debug section, the age, the path, and the guid are all invalid, which shows that the program might have been modified in some way before being released.

property	value
md5	n/a
sha1	n/a
sha256	n/a
age	0
size	58 (bytes)
format	RSDS
debugger-stamp	0x4DFE6ACA (Sun Jun 19 16:31:54 2011)
path	n/a
guid	0000-00-00-00-000000

This malware is a GUI program, even though there are no references in the strings to UI elements such as .jpeg files.

Packing

The malware is packed, but not in a traditional way. If a program were packed through UPX then there would only be two imports, LoadLibrary and GetProcAddress, and there would not be any strings. This is not the case, there are a lot of imports and a lot of strings. In some cases, the signature of the program might be that of the packer. Again, this is not the case for this malware sample, meaning that there is no way to unpack it without dynamic analysis. There are some

indicators in the sections of the program, the .text, .rdata, and .data all present a entropy really close to 8 which is usually an indicator of the program either being packed or obfuscated.

property	value	value	value	value	value
name	.text	.rdata	.data	.rsrc	.reloc
md5	F09D2536AE870ED5B64B955...	020E34F80C467FFB18C01CD...	6B13C848F593A48F5059FAA...	4B56ECC0E70AE95918987F...	B29DF4118B6429018C6F062...
entropy	7.727	7.920	7.862	7.727	5.268
file-ratio (99.35%)	49.51 %	35.50 %	12.70 %	0.65 %	0.98 %
raw-address	0x00000400	0x00013400	0x00020E00	0x00025C00	0x00026000
raw-size (156160 bytes)	0x00013000 (77824 bytes)	0x0000DA00 (55808 bytes)	0x00004E00 (19968 bytes)	0x00000400 (1024 bytes)	0x00000600 (1536 bytes)
virtual-address	0x00401000	0x00414000	0x00422000	0x0044E000	0x0044F000
virtual-size (313856 bytes)	0x00013000 (77824 bytes)	0x0000DA00 (55808 bytes)	0x0002B600 (177664 bytes)	0x00000400 (1024 bytes)	0x00000600 (1536 bytes)
entry-point	0x00011710	-	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040	0x42000040
writable	-	-	x	-	-
executable	x	-	-	-	-

The main giveaway that the malware was packed though is that when analyzing it statically in Ghidra there does not seem to be any real functionality, given that there are only three functions. After performing some dynamic analysis, as I discuss in the later sections of the report, I was able to identify the packer used in this malware: Mystic Compressor. Here are the changes to the program sections after it is unpacked.

property	value	value	value	value	value
name	.text	.rdata	.data	.rsrc	.reloc
md5	CB6784A7A216700C9FBCA8...	14C49FF76C0B4D5CDBF842...	C7CAAC0BD4673429B8336A770...	4B56ECC0E70AE95918987F...	B29DF4118B6429018C6F062...
entropy	6.040	6.133	5.827	7.727	5.268
file-ratio (99.66%)	26.21 %	18.79 %	53.79 %	0.34 %	0.52 %
raw-address	0x00000400	0x00013400	0x00020E00	0x00047E00	0x00048200
raw-size (295936 bytes)	0x00013000 (77824 bytes)	0x0000DA00 (55808 bytes)	0x00027000 (159744 bytes)	0x00000400 (1024 bytes)	0x00000600 (1536 bytes)
virtual-address	0x01281000	0x01294000	0x012A2000	0x012CE000	0x012CF000
virtual-size (323584 bytes)	0x00013000 (77824 bytes)	0x0000E000 (57344 bytes)	0x0002C000 (180224 bytes)	0x00001000 (4096 bytes)	0x00001000 (4096 bytes)
entry-point	-	0x000196E0	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040	0x42000040
writable	-	-	x	-	-
executable	x	-	-	-	-
shareable	-	-	-	-	-
discardable	-	-	-	-	x
initialized-data	-	x	x	x	x
uninitialized-data	-	-	-	-	-
unreadable	-	-	-	-	-
self-modifying	-	-	-	-	-
virtualized	-	-	-	-	-
file	-	-	executable, offset: 0x00037828, s...	-	-

The only changes can be seen in the .text, .rdata, and .data sections. Each one has a normal entropy level for user code. The size of the .rdata, and .data sections increased. The OEP also changed, it started at address 0x00011710 in the .text section before being unpacked, and it is at address 0x000196E0 after the unpacking was completed. We can also see that there is an executable stored inside the data section. I was unable to dump the executable file and analyze dynamically. The .rsrc and .reloc sections did not change after the program was unpacked.

Imports

Before the program is unpacked PESTudio shows that 6 libraries are being imported, and three of these were marked suspicious: ws2_32.dll, wldap32.dll, wininet.dll. All three dlls are used to provide internet functionality to the program. We can see more dlls being referenced in the strings, such as certcli.dll. In the functions imported we can see Load LibraryA, which is typical for a packed program. There is no reference to GetProcAddress though, and, as I will later discuss, this is because the malware uses a custom subroutine to import library functions by hash. There are numerous imports of network functions, for example WSASStartup, WSAConnect,

WSASocketA, WSAInstallServiceClassA, closesocket, WSARecv, and many more. This leads us to believe that the malware will probably connect to one, or multiple, C&C servers. There are also a lot of references to LDAP functions, which seem to be used to iterate through the LDAP directory and create, modify, and delete files. Given that this malware is known to be a banking credential stealer, and LDAP services are used by many banks around the world, it is safe to assume that this is the reason for the imports. Some other interesting imports are VirtualAllocEx and VirtualAlloc, which are used by the program to load the packer into.

Due to some issues with reconstructing the IAT table when dumping the program with Scylla I was not able to view if there were any additional imports in PESTudio once the malware was unpacked. I was though able to analyze the dumped program statically in Ghidra, and after finding the custom hash import function I was able to see which additional dlls the malware uses. The more interesting ones are crypt32.dll, which is the main dll used in any certificate or encryption operation, and advapi32.dll, and API library that supports registry and security calls.

Strings

Before the program gets unpacked there are not many interesting strings. Out of all the functions strings there are three interesting ones which are used for file compression, LZInit, LZRead, and LZClose. It is possible that the malware will store the data that it stole in a compressed folder and then send it to one of its C&C servers.

After the program is unpacked there are many suspicious strings that hint towards some of the functionality of the malware.

RapportMgmtService.exe is an executable which is part of the Rapport Management Service, which is used to protect communications between enterprises, especially banks. (<https://www.file.net/process/rapportmgmtservice.exe.html>). Additionally, cyberterm.exe is an executable for an unknown Russian payment related tool, and Web Money is referenced multiple times. WebMoney is a Russian online payment settlement application. A possibility might be that after backing credentials are stolen it will wire money to a WebMoney account.

RapportUtil.dll

.\patching_sentry\periodic_checkpoint.cpp

RapportMgmtService.exe

CyberTerm.mdb

cyberterm.exe

cyberterm.mdb

[WebMoney](#)
[WebMoney Keeper Classic](#)
[WebMoney Keeper](#)
[WMID](#)
[SysTabControl32](#)
[after WEBMoney!!!](#)
[SysListView32](#)
[jsif](#)
[OpenEvent...](#)
[pCloseHandle](#)
[before WEBMoney!!!](#)
[SendBalanseToLog](#)
[comment](#)
[0xF8083E3B](#)
[hash](#)
[SendBalanseToLog](#)

There are multiple references to strings typical of a keylogger, such as the “start thread for keylogging”, and the strings representing keys that are not represented in ASCII. There are also some direct references to functions typical of keyloggers, such as `GetForegroundWindow`, and `GetAsyncKeyState`, which might be imported at runtime. This is probably one of the ways that the malware will steal credentials.

[SBER KEYLOGGER](#)

[create Event Failed](#)
[start thraed for keylogging SendLoadedThred](#)
[DELETE](#)
[\[backspace down\]](#)
[\[tab down\]](#)
[\[enter down\]](#)
[{Enter}](#)
[Left](#)
[Right](#)
[Down](#)
[{Enter}](#)
[... ..](#)
[{BackSpace}](#)
[{#%0x}](#)
[{Click}](#)
[{RClick}](#)
[%s \(x= %d;y= %d\)](#)

The malware has also some internet functionality. There are multiple strings referencing URL requests, parameters, and queries, such as POST, GET, DELETE, User-Agent, and there are also domain names and IP addresses.

```

GET /stat?uptime= %d&downlink= %d&uplink= %d&id= %s&statpass= %s&comment= %s HTTP/1.0\r\n\r\n
DELETE
CONNECT
DELETE
CONNECT

POST %s HTTP/1.1\r\nHost: %s\r\nUser-Agent: %s\r\nAccept: text/html\r\nConnection: Close\r\nContent-Type: applica...
GET %s HTTP/1.0\r\nHost: %s\r\nUser-Agent: %s\r\nConnection: close\r\n\r\n
POST /get/scr.html HTTP/1.0\r\nHost: %s\r\nUser-Agent: %s\r\nConnection: close\r\nContent-Length: %d\r\nContent-...
POST /get/cab.html HTTP/1.0\r\nHost: %s\r\nUser-Agent: %s\r\nConnection: close\r\nContent-Length: %d\r\nContent-...
User-Agent

http://ibanksystemdwersfssnk.com/boffl.php?uid=
http://94.240.148.127/rt.jar
http://system-ibank2.com/s.dll

cookies.sqlite
Cookies\index.dat
*.txt
Cookies\index.dat
\Cookies\index.dat

Url: %s\r\nLogin: %s\r\nPassword: %s\r\nUserAgent: %s\r\n

```

We can also see that Registry Keys will be modified. I will go more in detail on these changes in the dynamic analysis, but these are some the keys that might be changed by the malware.

```

Software\Microsoft\Internet Explorer\Privacy
Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\
SOFTWARE\JavaSoft\Java Runtime Environment
CurrentVersion
SOFTWARE\JavaSoft\Java Runtime Environment
CurrentVersion
SOFTWARE\JavaSoft\Java Runtime Environment
CurrentVersion
Software\FTC\SKSB\KeyStorage
SOFTWARE\SBRF
SOFTWARE\SBRF
Software\Microsoft\Internet Explorer\Main
Software\Microsoft\Internet Explorer\PhishingFilter
Software\Microsoft\Internet Explorer\PhishingFilter
SOFTWARE\Microsoft\Windows NT\CurrentVersion

```

There are two references to cert.pfx and secret.key, which might be an indicator that the malware will add its own certificate or somehow modify the certificate store for the internet browser, and that some sort of encryption routine is being used in the program.

Interesting Functions

The following analysis was mostly based on static analysis performed in Ghidra after the program was unpacked and then dumped with Scylla. Only the main function and the injection subroutines were observed in a debugger.

The most recurring function is a custom subroutine that imports processes by hash. The parameter are the hash of the desired function and an integer which is used in a switch case to determine what dll should be loaded. The function then returns a pointer to the library call that corresponds to the hash. This function was probably implemented for some sort of AV evasion. Most suspicious programs import both LoadLibrary and GetProcAddress, so by calling only one of the two it might seem less suspicious.

The second most referenced function is an inline hooking function. It takes as parameters the hash of the library function to be hooked, an integer used for the previously described hash import function and a pointer to a function that hooks the library function. This malware hooks the following library functions: CreateFile, InternetWriteFile, PFXImpCertStore, DialogBoxParamW, PR_OpenTCPSocket, CloseConnection, Write, Read, ShowWindow, SetWindowPosition, Connect, CreateFile, TranslateMessage, InternetConnectA, InternetConnectW, HTTPSendRequestA, HTTPSendRequestW, HTTPSendRequestExA, HTTPSendRequestExW, InternetReadFile, InternetReadFileExA, InternetReadFileExW, InternetQueryDataAvailable, InternetCloseHandle, HTTPOpenRequestA, HTTPOpenRequestW, HTTPQueryInfoA, HTTPQueryInfoW, ShowWindow, WaveOutWrite, LoadLibraryW, DispatchMessageA, DispatchMessageW, SetFocus, SetWindowTextW. The hooks for all these functions are then passed into a function that probably parses the various parameters and then fulfills the requests.

There are two injection subroutines called by the main function for a total of three times. Both start a process and then try and inject a part of memory into it. The first subroutine will first try and start the svchost.exe process and then inject into it by using the functions ZwQueueApcThread and ZwResumeThread. If this injection fails it will start an iexplorer.exe process and try and use the same library functions to inject memory into the newly created process. If both methods fail, then it will again try to start and then inject svchost.exe or iexplorer.exe. This time though the injection mechanism is different, it uses the library functions writeProcessMemory, setContextThread, and ZWResumeThread.

The other injection subroutine is similar, but instead of svchost.exe or iexplorer.exe it will start explorer.exe and inject itself into it by using ZWQueueApcThread and ZwResumeThread without trying the writeProcessMemory function.

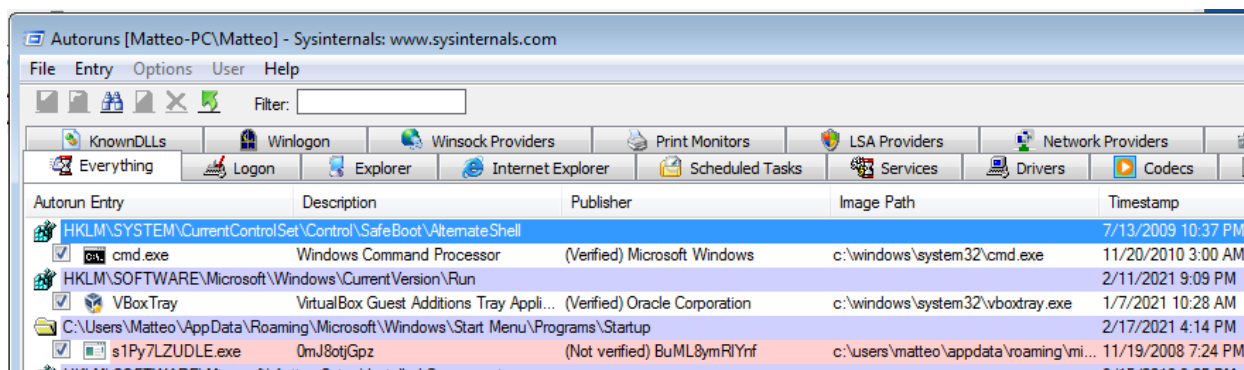
Finally, the main function of the program is quite straightforward. It creates a {random-name} folder in the system directory, it will load all the necessary dlls, then it will create the {random-name}.exe in the C:\Users\Matteo\AppData\Roaming\Microsoft\Windows\Programs\Startup directory. After this it will try and close op_mon.exe, a process that provides system tray access to the various Outpost security applications. It will then run the three different injection subroutines. The final injection into explorer.exe will terminate the program and delete the Practical1.exe file before main can return.

Dynamic Analysis

Initial Behavior

When the malware is first executed it does not seem to do much. After spawning an svchost.exe process it terminates and deletes itself from the directory where it is stored. Because of this svchost.exe will be detached from any process tree, which is suspicious. By debugging the malware in x32dbg I was able to see more suspicious behavior. The first thing the program does is try to delete a directory and a file that do not exist, and then load a section of the .text section into virtual memory. Using an XOR multibyte encoding scheme it decodes said memory. The key to this encoding scheme is the value 0x273d81cb. The decoded memory is a packer, Mystic Compressor, that, once called, unpacks the malware. The packer also modifies the return address, so once it is finished unpacking, it returns to the new OEP, where we have the main function of the malware. At this point I used Scylla to dump the program so that I could later analyze it statically both in PE Studio and in Ghidra. Because the import table was missing in the dumped file, I was at first limited in analyzing the various functions that were called in the program. After realizing that the malware implemented a custom hash import function, I was able to further understand how the malware functioned.

Once the malware enters the main function it creates a folder with a random name in the C:\ directory, which will be later used to store configuration files, .inf and .dat. It also drops a copy of itself in the C:\Users\Matteo\AppData\Roaming\Microsoft\Windows\Programs\Startup, thus achieving persistence, as is shown in the autoruns screenshot.



The malware will then sequentially spawn three processes, svchost.exe twice and explorer.exe, and then inject different sections of code into each one. The injection is done using two subroutines, one will create the desired process and then call the library functions ZwQueueApcThread and ZwResumeThread. In the case this subroutine fails the malware will instead write directly to the process memory after creating it by calling the two library functions writeProcessMemory and setContextThread. Also, in the case that the malware is not able to inject itself into svchost.exe it will try to do so into iexplorer.exe. The first svchost.exe process is persistent, and it contains the various hooks to the internet functions previously described. The second svchost.exe process is used to check whether ole32.dll is present in the system and if it is not it will drop a copy in the system. ole32.dll provides access to the COM library, it is safe to assume that it will have some COM functionality. I was not able to identify, neither through static nor dynamic analysis, where the malware uses a COM function call. The injected explorer.exe

process will terminate the Practical1.exe process and then delete the file. Each injected process has a lot more functionality that I was not able to identify.

Network Behavior

When first executed the malware will make a DNS query and try to connect to its C&C servers. If fakeDNS is not accepting any connections, then the malware will try and contact three DNS servers on repeat until it gets a response. Following is a screenshot of Wireshark showing the malware performing the DNS queries for the domains fromamericawhichlov.com, malborofrientro.com, and hillaryklinton.com.

1408	1269.3356974...	192.168.245.128	192.168.245.133	DNS	83 Standard query 0x8489 A fromamericawhichlov.com
1409	1269.3358033...	192.168.245.133	192.168.245.128	ICMP	111 Destination unreachable (Port unreachable)
1410	1273.4150890...	192.168.245.128	192.168.245.133	DNS	79 Standard query 0xf82c A malborofrientro.com
1411	1273.4151184...	192.168.245.133	192.168.245.128	ICMP	107 Destination unreachable (Port unreachable)
1412	1274.4142654...	192.168.245.128	192.168.245.133	DNS	79 Standard query 0xf82c A malborofrientro.com
1413	1274.4142950...	192.168.245.133	192.168.245.128	ICMP	107 Destination unreachable (Port unreachable)
1414	1275.4144945...	192.168.245.128	192.168.245.133	DNS	79 Standard query 0xf82c A malborofrientro.com
1415	1275.4145231...	192.168.245.133	192.168.245.128	ICMP	107 Destination unreachable (Port unreachable)
1416	1277.4145433...	192.168.245.128	192.168.245.133	DNS	79 Standard query 0xf82c A malborofrientro.com
1417	1277.4145745...	192.168.245.133	192.168.245.128	ICMP	107 Destination unreachable (Port unreachable)
1418	1281.4142108...	192.168.245.128	192.168.245.133	DNS	79 Standard query 0xf82c A malborofrientro.com
1419	1281.4142408...	192.168.245.133	192.168.245.128	ICMP	107 Destination unreachable (Port unreachable)
1420	1285.4147960...	192.168.245.128	192.168.245.133	DNS	78 Standard query 0xbb24 A hillaryklinton.com
1421	1285.4148256...	192.168.245.133	192.168.245.128	ICMP	106 Destination unreachable (Port unreachable)
1422	1286.4148422...	192.168.245.128	192.168.245.133	DNS	78 Standard query 0xbb24 A hillaryklinton.com
1423	1286.4148719...	192.168.245.133	192.168.245.128	ICMP	106 Destination unreachable (Port unreachable)
1424	1287.4145880...	192.168.245.128	192.168.245.133	DNS	78 Standard query 0xbb24 A hillaryklinton.com

If the DNS query is successful the malware will download the files klpclsdt.dat and wndsksi.inf, described in the following section, and it will try and upload different files to the server through POST requests.

26	123.049039803	192.168.245.127	192.168.245.133	HTTP	116 POST /aaaaaaanfgqyycimde.rar HTTP/1.1 (...)
29	123.070316737	192.168.245.133	192.168.245.127	HTTP	312 HTTP/1.1 200 OK (text/html)
45	123.724622332	192.168.245.127	192.168.245.133	HTTP	101 POST /m.inc HTTP/1.1 (application/x-www...
48	123.744403299	192.168.245.133	192.168.245.127	HTTP	312 HTTP/1.1 200 OK (text/html)
64	123.857986304	192.168.245.127	192.168.245.133	HTTP	1110 POST /jdnwesrzju.7z HTTP/1.1 (applicati...

The following screenshot shows the form of the request itself. As we can see the value in the form is base64 encoded. After decoding it the string is still not legible. Given that the malware imports encryption dlls, and there are references to a secret.key file, it is safe to assume that the value is further encrypted. The encryption routine used though is unknown.

21	1.034939080	192.168.245.127	192.168.245.133	HTTP	124 POST /yaaaaaagswskyysu.tpl HTTP/1.1 (application/x-www-form-urlencoded...)
22	1.034943268	192.168.245.133	192.168.245.127	TCP	54 80 → 49161 [ACK] Seq=1 Ack=407 Win=64128 Len=0
23	1.051798083	192.168.245.133	192.168.245.127	TCP	204 80 → 49161 [PSH, ACK] Seq=1 Ack=407 Win=64128 Len=150 [TCP segment o...]
24	1.054401195	192.168.245.133	192.168.245.127	HTTP	312 HTTP/1.1 200 OK (text/html)
25	1.054552599	192.168.245.127	192.168.245.133	TCP	60 49161 → 80 [ACK] Seq=407 Ack=410 Win=65280 Len=0
26	1.063322018	192.168.245.127	192.168.245.133	TCP	60 49161 → 80 [FIN, ACK] Seq=407 Ack=410 Win=65280 Len=0
27	1.063342096	192.168.245.133	192.168.245.127	TCP	54 80 → 49161 [ACK] Seq=410 Ack=408 Win=64128 Len=0
28	1.210287864	192.168.245.127	192.168.245.133	TCP	66 49162 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; ...					
Host: fromamericawhichlov.com\r\n					
Connection: Close\r\n					
Content-Type: application/x-www-form-urlencoded\r\n					
Content-Length: 70\r\n					
\r\n					
[Full request URI: http://fromamericawhichlov.com/yaaaaaagswskyysu.tpl]					
[HTTP request 1/1]					
[Response in frame: 24]					
File Data: 70 bytes					
HTML Form URL Encoded: application/x-www-form-urlencoded					
Form item: "djeiu" = "FCfy+54GH6et5PI/71cUrzLC30+LagEZT2L/Rt02n2VkmIoYh+G="					
Key: djeiu					
Value: FCfy+54GH6et5PI/71cUrzLC30+LagEZT2L/Rt02n2VkmIoYh+G=					

Registry Keys

The malware creates 1 unique key, HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\GDIPlus, and adds the value FontCachePath, that stores the path of the GDI Font .dat file. GDI+ is a C API that allows the use of graphics and formatted text on the video display. The remaining keys that the malware modifies are mostly related to Internet Explorer.

```
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Internet Explorer\Main\TabProcGrowth: "0"
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Internet Explorer\PhishingFilter\ShownVerifyBalloon: 0x00000002
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Internet Explorer\PhishingFilter\Enabled: 0x00000001
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1609: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1406: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1\1609: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1\1406: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4\1609: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4\1406: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1406: 0x00000003
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1406: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1609: 0x00000001
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1609: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1406: 0x00000003
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1406: 0x00000000
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1609: 0x00000001
HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1609: 0x00000000
```

The Internet Settings\Zones key represent all the different security zones for Internet Explorer:

0->My Computer

1->Local Intranet Zone

2->Trusted sites Zone

3->Internet Zone

4->Restricted Sites Zone

For each of these keys the malware is modifying the values of the subkeys 1406 and 1609. 1406 represents the ability to access data sources across domains, and 1609 is the ability to display mixed content. These are three values that these registry keys can be set to:

0->Enabled

1->Prompt

3->Disable

In the dumped disassembled PE file there seems to be a call to setRegKeyValue for a registry key that does not get modified on my VM. This key is Software\Microsoft\Internet Explorer\Privacy\CleanCookies. It is probably set to force the user to have to login again on every website so that the malware can steal the credentials for every account the user has.

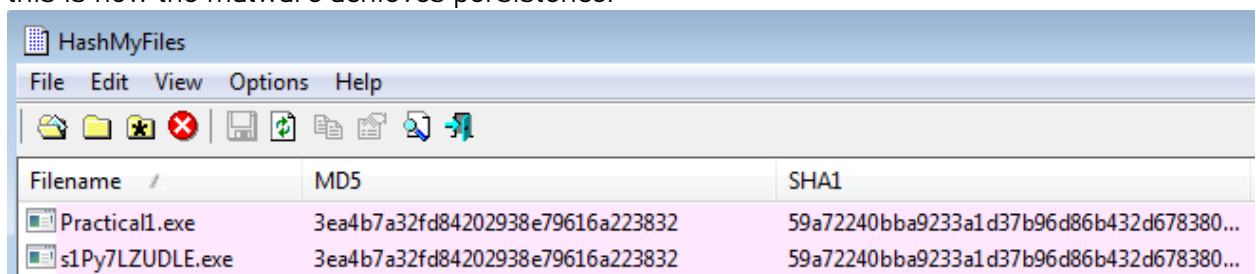
Files and Processes

Following is a screenshot of the hashes for Practical1.exe that I will use to compare files to any file or executable that might be dropped by the program.

property	value
md5	3EA4B7A32FD84202938E79616A223832
sha1	59A722408BA9233A1D37B96D86B432D678380E38
sha256	A67A1CA66F666EABEF466BD6BEBA25867FD67BA697C1C7C02CDE2C51E4E8289D

The malware will always drop the following files and folders in the same place:

- {random-name} folder in the system directory
- {random-name}.exe is a copy of the program that is stored in the directory C:\Users\Matteo\AppData\Roaming\Microsoft\Windows\Programs\Startup. Following is a screenshot of HashMyFiles that shows that this executable has the same hashes as Practical1.exe, proving that it is an exact copy. As mentioned previously in the report this is how the malware achieves persistence.



Filename	MD5	SHA1
Practical1.exe	3ea4b7a32fd84202938e79616a223832	59a72240bba9233a1d37b96d86b432d678380...
s1Py7LZUDLE.exe	3ea4b7a32fd84202938e79616a223832	59a72240bba9233a1d37b96d86b432d678380...

- MicroST folder is dropped in the C:\Users\Matteo\AppData\Roaming directory. In both the NDG test environment and on the virtual machine on my local machine the malware never dropped any files into this folder.
- Inside the {random-name} folder in the system directory the malware will create two files: klpclsdt.dat and wndsksi.inf.

property	value
md5	2781E6150CB5427EAE0882351AF1DE2D
sha1	5D9C72F2D39BB2E2E0FB646F1DE113D88ECE0FC4
sha256	E0CB13BC0810E02793DFC98230876B8AF72BC36F7A3AE6E97312DCB003855AF0

Screenshot of wndsksi.inf hashes

property	value
md5	E8A9EEC432EF3FCD30AEA8523A9C347F
sha1	04A0636BCC59A7620C868B445EC13227EADADC92
sha256	43188AB56155CB3DBE2AB667E591F6ADC23858E714A90C0EFB39371475D3B3DC

Screenshot of klpclsdt.dat hashes.

The names for the dropped folder and the dropped executable are not entirely random, they remain consistent on the same host, regardless of how many times the malware is run. Presumably, they might be based on some type of local configuration, for example the host name. This was tested both on a local VM and on a VM inside NDG. The names are different across VMs, but they remain the same inside a single machine.

The malware also starts three different processes:

- svchost.exe
- svchost.exe
- explorer.exe

As was discussed previously the first the svchost.exe process is the only one that is persistent, the other two will run and then terminate soon after they are started.

Indicators of Compromise

Following are the host indicators of compromise:

- Presence of {random-name} folder in the system directory
- Presence of MicroST folder in the C:\Users\Matteo\AppData\Roaming directory
- Presence of {random-name}.exe file in the C:\Users\Matteo\AppData\Roaming\Microsoft\Windows\Programs\Startup
- Modification of the following registry keys:
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Internet Explorer\PhishingFilter
 - HKU\S-1-5-21-615309556-1163276436-1264706328-1001\Software\Microsoft\Internet Explorer\Main\TabProcGrowth
- Detached svchost.exe process

Following are the network indicators of compromise:

- DNS queries to all of the following domains, and HTTP POST requests to one of them:
 - fromamericawhichlov.com
 - hillaryklinton.com
 - malborofrientro.com
- HTTP requests to the following websites:
 - <http://ibankssystemdwersfssnk.com/boffl.php?uid=>
 - http://94.2499.148.127/rt_jar
 - <http://system-ibank2.com/s.dll>

Conclusion

The malware shows most of the signs that are necessary to identify it as part of the Carberp family. It hooks internet functions, all of which are listed in the interesting functions section, it has keylogging functionalities to possibly steal credentials, it downloads configuration files from C&C servers, it makes itself persistent in a manner that does not require admin privileges, and finally it references executables, strings, and http links that are all banking related.