The Avast Abuser: Metamorfo Banking Malware Hides By Abusing Avast Executable

medium.com/@chenerlich/the-avast-abuser-metamorfo-banking-malware-hides-by-abusing-avast-executable-ac9b8b392767

Chen Erlich April 9, 2020

Chen

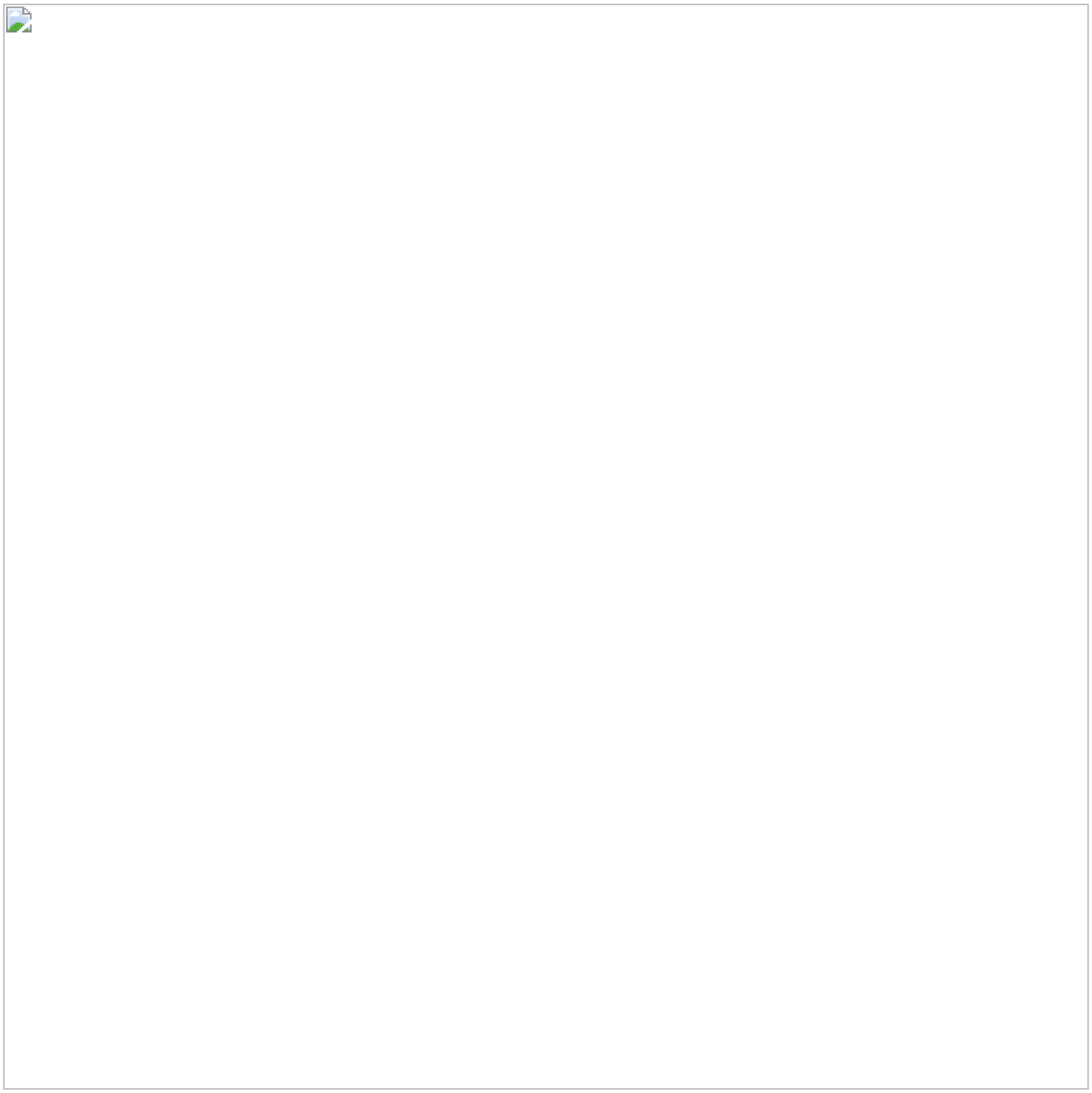
Chen Erlich

Apr 3, 2020

.

10 min read

Source: ensilo.com/BreakingMalware
Twitter: @chen_erlich



SUMMARY

In May 2019, enSilo's Threat Intelligence team observed activity by a cybercrime group, spreading Metamorfo — A Brazilian banking trojan. The variants we discovered abuse an executable digitally signed by Avast, which is one of the most popular AV products in the world for consumers. We were able to connect this activity to a campaign reported by TrendMicro which targeted an executable by a different Anti-Virus vendor, Avira. This further highlights the Modus-Operandi of the group.

This blog post describes in detail one of the variants used in this campaign and highlights unique Tactics, Techniques and Procedures (TTPs) used in this campaign which were not previously disclosed.

TECHNICAL ANALYSIS

In May 2019, enSilo detected a new activity by a Brazilian cybercrime group. Both loader variants and their respective payloads that were analyzed share similar TTPs and code associated with a Brazilian cybercrime group.

Execution Flow Overview

On execution, the MSI downloader starts by checking if it is running in a virtual machine. If not, downloads a zip file, unzips it, deletes itself, establishes persistency and restarts the system.

The zip file contains the following files:
1. **jesus.exe** — Signed AVDump32 — Avast's memory dump utility. Renamed to a random name.
2. **dbghelp.dll** — Malicious file to be side loaded by AJWrDz.exe.

3. **jesus.dmp** — Payload to be loaded by the injected Windows Media Player executable (wmplayer.exe). Later renamed to the same random name as jesus.exe with a .dmp extension.
4. **ssleay64.dll** — Known variant of Metamorfo. Will be loaded and used in the injected wmplayer.exe.
5. **ssleay32.dll** — OpenSSL Shared Library.
6. **borlndmm.dll** — Borland Memory Manager.
7. **libeay32.dll** — OpenSSL Shared Library.

During the rest of the analysis we will refer jesus.exe as "AJWrDz.exe" which is the random name generated in this execution. After the system reboots, the file "AJWrDz.exe" executes, which in turn triggers the side-loading of the malicious (and fake) DLL file "dbghelp.dll". This malicious DLL file injects itself to Windows Media Player process — wmplayer.exe, and reflectively loads the renamed jesus.dmp file, "AJWrDz.dmp".

The following diagram describes the high-level execution flow of the variants in this campaign:

MSI DOWNLOADER

The following is the static characteristics of the Windows Installer (MSI) downloader which starts the infection, this MSI downloader is similar to the one used in the earlier part of the campaign:

File Name: HNR-Not03958576535323.msi

SHA1: F1498E679885389C32FDF5EC39813FE5D4D34F23

Size: 287232 bytes

Creation Time: 2009–12–11 11:47:44

During the time of analysis this variant had very low detection rate in VirusTotal, as can be seen in figure 2:

All MSI files in this campaign have different names, but share unique characteristics:

1. Disguised as “” to look legitimate as seen in figure 3:

2. Created using the “**Advanced Installer**” tool, which is imported in all of them, as shown in figure 4.

3. Contain a **vmdetect.exe** [MD5: 55FFEE241709AE96CF64CB0B9A96F0D7] to avoid detection, as shown in figure 4:

4. Use the . The CustomAction table enables integration of custom code and data into an installation. The source of the code that is executed can be a stream contained within the database, a recently installed file, or an existing executable file. The attackers abused this feature to add the malicious JavaScript/VBS payload as shown in figure 5.

The JavaScript payload

The downloader’s JavaScript payload is obfuscated:

Figure 6: Obfuscated MSI payload
After deobfuscation:

Figure 7: Deobfuscated MSI payload
The samples in this campaign communicate with a URL in the following format:
The ”{random}” part may change between different MSI downloaders. The image2.png file is actually a zip file which is downloaded and extracted to a target folder. In this variant it is

extracted to *%APPDATA%\Macromedia*. Next, the MSI downloader creates in this location a *desktop.txt* file containing the string "NULL".

The purpose of the *desktop.txt* file is to indicate whether the system is infected by the malware. If it exists, the MSI will exit after it will open Adobe's website to explain how to install updates. It does so by executing:

"c:\Windows\System32\cmd.exe /C start /MAX https://helpx.adobe[.]com//acrobat/kb/install-updates-reader-acrobat.html"

Otherwise, it will open legal terms of use page in Adobe's site and continue the payload execution. This following command executes to open the terms of use page:

"*c:\Windows\System32\cmd.exe /C start /MAX https://adobe.ly/2RY5GJR*"

which will redirect to "*https://www.adobe.com//legal/terms.html"*.

Note that both URLs are with the Brazilian 2-letter abbreviation, suggesting the victims' origin.

The files are extracted to a newly created folder with a randomized name under the same path, and the zip file is then deleted. The "AJWrDz.exe" executable path is written to the registry Run key "*HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*" to achieve persistency. As a final step the system is restarted to trigger its execution.

THE AVAST ABUSE

After the system restart, the "AJWrDz.exe" file executes. Its static characteristics are:

File Name: AJWrDz.exe (renamed from jesus.exe)

SHA1: 2A1A5D7C85560924EDC434A1D2F23ED3445D86F4

Size: 814296 bytes

Creation Time: 2018–10–08 13:07:15

This is a legitimate file, **AVDump32.exe,** digitally signed by **"AVAST Software**" as shown in figure 8:

AvDump32.exe legitimate use is to create *.dmp files of Avast processes in case there is an unhandled exception. When Avast is installed legitimately on a system the file is located in Its original location: C:\Program Files\AVAST Software\Avast.

Figure 9 suggests that this file was submitted to VirusTotal as “jesus.exe”, which is the name of the file in the downloaded zip, before it’s being renamed in the MSI payload:

AvDump32.exe is abused by the Metamorfo to side-load the “dbghelp.dll” by leveraging the DLL search order. Note that this is a common issue which makes it possible to leverage the DLL side-loading attack, often referred to as DLL Hijacking. Figure 10 shows the DLL files imported by this executable:

You can find another example of abusing the DLL search order in one of our previous blog posts.

The side loaded “dbghelp.dll” is a malicious file written in Delphi and compiled using the Embarcadero Delphi IDE with the following characteristics:

File Name: dbghelp.dll

SHA1: 08823578841AEED044EAD81ED6DB16DD95B6FF4B

Size: 5595136 bytes

Creation Time: 2019–04–27 22:17:17

After being side-loaded by AvDump32.exe, the DLL’s execution starts with the following steps:

1. Resolves WINAPI functions
2. Hides its GUI using WINAPI call
3. Compares if the DLL is being ran by wmplayer. More on this later.

Next, the DLL file creates the mutex — *[7F4HRE-375E-AEF3-BE9A-OBJT389F53]* and writes to *HKCU\Software\index (*as shown in figure 11) the name of the running process which is later used to know the name of the .dmp file that should be loaded. Finally, it injects itself to Windows Media Player executable — **wmplayer.exe**.

The process wmplayer.exe is a rather strange victim for injection given that various Windows distributions don’t come with Windows Media Player installed by default, so it can only be implied that this software is probably more common in the victim’s origin, and that it probably targets home users.

Metamorfo uses a DLL injection technique with a twist. Instead of getting a handle to the victim process using *OpenProcess,* which relies on having a running process, the injection uses *CreateProcess* with CREATE_SUSPENDED flag. Then it creates a remote thread which loads the malicious DLL and executes it. The process’ main thread is never resumed and thus only the malware code executes. Figure 12 shows the *CreateProcess* call:

The injection flow is as follows:

INJECTED PAYLOAD

Upon injection, the DLL validates that it runs under the wmplayer.exe process by checking the process name and goes on to execute its malicious activity. It creates a second mutex — *One-InstanceJes*, resolves more WINAPI functions, checks for the registry “index” key (which was previously written)the execution location and for the “ADWrDz.dmp” file. If this file exists, it extracts it in-memory using *RtlDecompressFragment* and reflectively loads it.

No DEP

dbghelp.dll is incompatible with DEP (Data Exception Prevention), as shown in Figure 14. Thus, when it loads the operating system will disable DEP for the injected wmplayer.exe process. This means that code can be executed from memory regions that are not marked as executable in the context of this process.

Metamorpho uses this to execute the reflectively loaded payload from a non-executable region. This makes the payload harder to detect by memory forensics toolkits and security products which many times look specifically for executable memory.

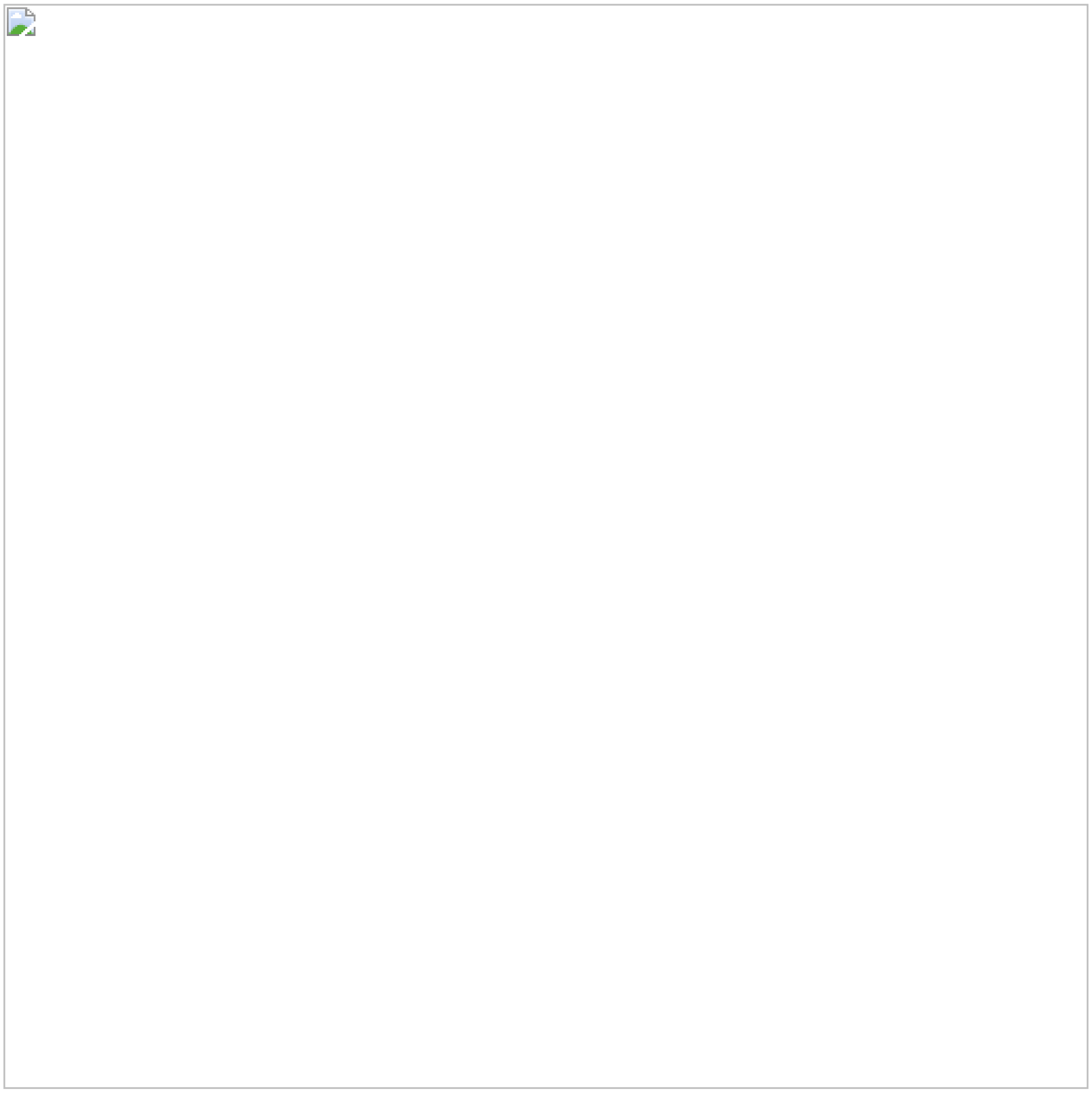
Leveraging CreateTimerQueueTimer

Once “ADWrDz.dmp” is loaded into memory Metamorfo leverages the *CreateTimerQueueTimer* WINAPI call to execute it (as shown in figure 14).

CreateTimerQueueTimer is a WINAPI that creates a queue for timers. These timer objects allow the selection of a callback function at a specified time. The original function of the API is to be part of the process chain by creating a timer routine, but here, the callback function of the API is the entry point of the malware’s actual payload.

The use of *CreateTimerQueueTimer* makes detection harder since the payload will not run in the remote thread context.

This kind of technique was previously used in malware variants such as Emotet and Hancitor.



Throughout the “ADWrDz.dmp” execution, it outputs debug comments in Portuguese as shown in figure 16:

Entering the *CreateTimerQueueTimer* callback, Metamorfo creates another mutex — *libea54*, and starts checking for the existence of directories and files relevant for its execution. Since there are multiple variations of Metamorfo in this campaign, the attackers used different locations in the file system to drop their files, see IOCs section.

Next, Metamorfo checks for the existence of “mreb.xml” and “mreboot”, as can be seen in figure 17. These artifacts were not available to us and we couldn’t verify their purpose.

r

If they aren’t found, it creates another mutex by the name — *[7F4HRE-375E-AEF3-BE9A-OBJT389F53].* Then, it checks internet connection by trying to resolve “*goole.com*” (Misspelled) address. If internet connection is available,

it sends a GET request to “*https://www.localizaip.com[.]br/api/iplocation.php”* to retrieve geo data.

Metamorfo’s C&C communication is encrypted using the dropped OpenSSL libraries libeay32.dll and ssleay32.dll.

Based on the gathered data, if the victim is not from Brazil or Portugal it will print the following output and send the collected data to the C&C “*https://x1-lb12.internal[.]gocache.me”,* which resided in Brazil, and finish.

a

If the victim is from Brazil or Portugal, it will start monitoring running applications in the system using a message loop:

The ssleay64.dll payload

If the malware identifies a file named “mreb.xml” or a folder named “mreboot”, it loads a malicious “ssleay64.dll”, also written in Delphi, compiled by Borland Delphi, which has the following characteristics:

File Name: ssleay64.dll

SHA1: F5E63580710E8FA884377A746FC822E5

Size: 1445888 bytes

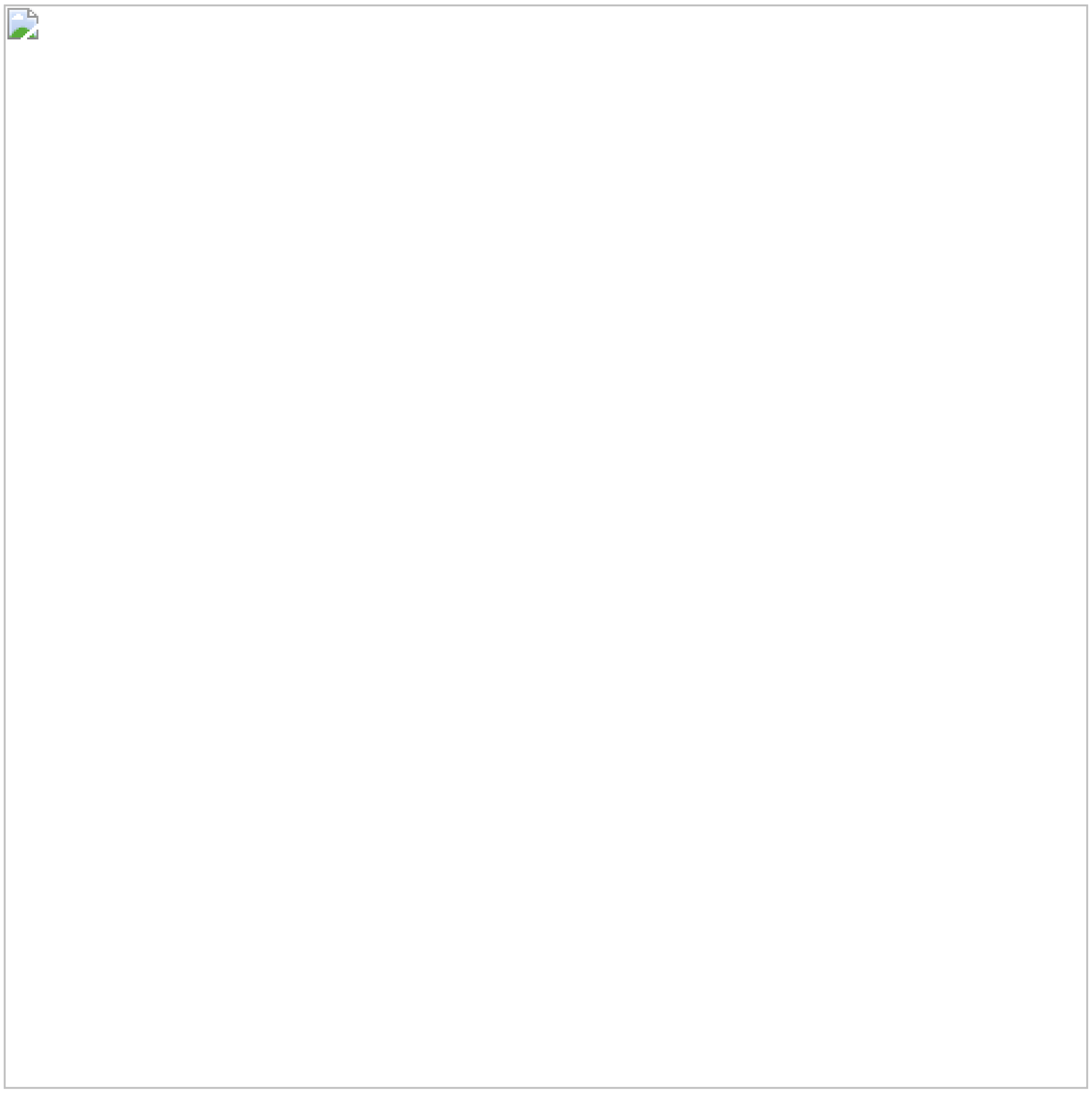
Creation Time: 2019–04–08 12:15:27

The DLL holds various resources. Some of them are encrypted and will be used as payloads to steal victim’s data, while others are cursor related resources.

Like samples from previous campaigns Metamorfo can display fake forms on targeted banking sites and steal credentials from the victims. On previous campaigns Metamorfo used Windows Update to hide its malicious activity. Similarly, in this campaign Metamorfo uses a fake “Blue Screen” window. It does so after disabling the taskbar as can be seen in Figures 20 and 21:

Evading Banking Protection & Anti-Fraud Products

Metamorfo also makes efforts to evade banking protection and anti-fraud products by setting a hook on *LoadLibraryW* function and checking which DLL is loaded, the trampoline can be seen in Figure 22:



With the help of this trampoline, for every *LoadLibraryW* call the attackers will check if the DLL to be loaded contains one of the following anti-fraud and banking protection strings:

- Gbpinj
- Scpbrad
- Scpad
- Trusteer
- Warsaw
- Gblplugin
- Ipsbho
- Hook

If one of them is matched, the DLL *LoadLibraryW* call is trying to load wouldn't load.

Figure 23 shows a few searched strings:

IOCS

Hashes:

- MSI — F1498E679885389C32FDF5EC39813FE5D4D34F23
- Other related samples
- AvDump32.exe — 2A1A5D7C85560924EDC434A1D2F23ED3445D86F4
- Dbghelp.dll — 08823578841AEED044EAD81ED6DB16DD95B6FF4B
- Other related samples
- Ssleay64.dll -
- F5E63580710E8FA884377A746FC822E5

- C00BF102482C61E4CAB3C6B6666697779092FADC
- 6242CC3009A96F97AB9586C970DB26EDE5512F9A
- 03A5BEF2B9DE1DF5C19C9F4D2AEC6F780F4749D0
- C15154D7323EA0C7A40912C799599DACCEB4E7CE

URLs:

https://s3-eu-west-1[.]amazonaws.com/disenyrt3/image2.png

https://s3-eu-west-1[.]amazonaws.com/sharknadorki/image2.png

https://s3-eu-west-1[.]amazonaws.com/jasonrwk5wg/image2.png

https://s3-eu-west-1[.]amazonaws.com/frezaaaewrwty/image2.png

https://s3-eu-west-1[.]amazonaws.com/cadeaadl54t4gw4/image2.png

https://s3-eu-west-1[.]amazonaws.com/sharknadorki/image2.png
https://s3-eu-west-1[.]amazonaws.com/jooosan/image2.png
https://s3-eu-west-1[.]amazonaws.com/shhakkr/image2.png

www.goole[.]com

https://www.localizaip.com[.]br/api/iplocation.php

mrs04s09-in-f206.1e100[.]net

lhr25s13-in-f78.1e100[.]net

dub08s01-in-f14.1e100[.]net

lhr25s11-in-f46.1e100[.]net

Files:

- %APPDATA%\Macromedia
- %APPDATA%\Macromedia\desktop.txt
- %APPDATA%\TeamViewer
- %APPDATA%\TeamViewer\desktop.txt
- %APPDATA%\DMCache
- %APPDATA%\DMCache\desktop.txt
- %APPDATA%\AnyDesk
- %APPDATA%\AnyDesk\desktop.txt

Registry:

HKCU\Software\index

Mutexes:

- [7F4HRE-375E-AEF3-BE9A-OBJT389F53]
- libea54
- One-InstanceJes

Thanks for reading. Follow me on Twitter for more posts like this one.

Chen Erlich

The latest Tweets from Chen Erlich (@chen_erlich). Security Researcher. Opinions are on my own. #MalwareResearch…

twitter.com