Malware analysis “sample2.exe”

Filippo Contro VR437055
Michele Martini VR437056

9 Gennaio 2020

Summary

Simple calculator app

But under the hood...

Summary

- Post requests
- Security center deactivation
- Infection

Summary

- Post requests
- Security center deactivation
- Infection $\longrightarrow$ Polymorphic malware

Static analysis

Static analysis

First look

Static analysis

First look

High entropy

Static analysis

First look

High entropy *longrightarrow* Obfuscation or Packing?

Static analysis

First look

High entropy *longrightarrow* Obfuscation or Packing?

Let's look at the sections

Sections

The first three sections are OK! But **.vmp0** NO!

Sections

- High entropy
- Big portion of code (87%)
- Both writable and executable

Sections

- High entropy
- Big portion of code (87%)
- Both writable and executable $\longrightarrow$ Very suspicious

Obfuscation

The presence of the 3 main sections (text, data, resources) suggests the absence of packing.

Obfuscation

The presence of the 3 main sections (text, data, resources) suggests the absence of packing.

PeID confirms our supposition.

Obfuscation

The presence of the 3 main sections (text, data, resources) suggests the absence of packing.

PeID confirms our supposition. The name **vmp0** is given by **VMProtect**.

Imports

- getModuleHandleA
- loadLibraryA
- getProcAddressA
- getStartupInfoA
- getCommandLineW

Imports

- getModuleHandleA
- loadLibraryA
- getProcAddressA
- getStartupInfoA
- getCommandLineW

Other dealing with **Threads**, **Clipboard**, **System**

Other

- **Version**: Legit information, but no date
- **Strings**: Thousands of crypted strings
- **Certificate**: it is missing

Dynamic analysis

Dynamic analysis

We used 3 tools:

- **regshot**, to detect files and registers alterations between a time lapse;
- **procmon**, to log system functions called by the malware;
- **fakenet**, to track internet traffic in a simulated network.

Dynamic analysis

In order to get consistent results we followed this schedule:

1. launch Fakenet;
2. launch and setup Procmon;
3. launch Regshot, setup path and run of its first shot;
4. start Procmon analysis and launch of the malware;
5. interaction with calculator by the GUI;
6. stop Procmon tracking
7. second Regshot shot;
8. stop Fakenet;

First run

We ran the malware without any tools.

After a little bit the **Windows Security Center** was deactivated.

First run

We ran the malware without any tools.

After a little bit the **Windows Security Center** was deactivated.

It could not be restarted.

First run

We ran the malware without any tools.

After a little bit the **Windows Security Center** was deactivated.

It could not be restarted.

The malware needs time to perform those actions.

Fakenet

With fakenet we registered many POST requests.

More than 900, to 300 different destinations.

Fakenet

With fakenet we registered many POST requests.

More than 900, to 300 different destinations.

Every request contains:

- Request type
- Destination URL
- Protocol
- User agent

Fakenet

With fakenet we registered many POST requests.

More than 900, to 300 different destinations.

Every request contains:

- Request type
- Destination URL
- Protocol
- User agent

Here's an example:

```
User-Agent:  Mozilla/4.0
(compatible; MSIE 28;...
```

Fakenet

With fakenet we registered many POST requests.

More than 900, to 300 different destinations.

Every request contains:

- Request type
- Destination URL
- Protocol
- User agent

Here's an example:

```
User-Agent:  Mozilla/4.0
(compatible; MSIE 28;...
```

There are not imported libraries to send HTTP requests

Fakenet

With fakenet we registered many POST requests.

More than 900, to 300 different destinations.

Every request contains:

- Request type
- Destination URL
- Protocol
- User agent

Here's an example:

```
User-Agent:  Mozilla/4.0
(compatible; MSIE 28;...
```

There are not imported libraries to send HTTP requests

$\longrightarrow$ *dynamically imported*

Procmon

We used procmon to keep track of every action made by the malware
Dividing them in 3 categories:

- DLL
- Registry
- Files

DLL

We measured 46 different dll files loaded with the *LoadImage* primitive.

Among them the most interesting are:

- **cryptbase - crypt32**: to handle cryptography
- **ws2_32**: to manage web socket

Registry

We saw many open-read-close actions on many system registers, but only a few write

Registry

We saw many open-read-close actions on many system registers, but only a few write

The only keys modified were:

- Language list, which has no interesting effects
- Windows internet zones set to 0 which means *Allow anything* for each network type

Files

We detect the infection of other files watching the “WriteFile” operations and the amount of bytes written.

We obtained the sequence of actions that the malware implement to infect other files

Files

We detect the infection of other files watching the "WriteFile" operations and the amount of bytes written.

We obtained the sequence of actions that the malware implement to infect other files

- Read the *.exe* victim file.
- Write of the content plus the infected part in a *.vir* file with the same name.
- Copy of the content of the *.vir* file to the *.exe* one changing the EOF location.
- Set of fake information on the executable such as creation and last access time.
- Delete the *.vir* file.

Files

The infected files were many, and in different location.

They were mainly common executables, run frequently by the average user.

Files

The infected files were many, and in different location.

They were mainly common executables, run frequently by the average user.

The main ones were:

- Windows Media Player
- Internet Explorer
- Windows Defender
- Windows Mail
- Windows Photo Viewer
- and more...

We inspect the infected files with pestudio and we found a new section called **.vmp0**

Regshot

With regshot we had a confirmation of all the actions tracked with procmon.

The fact that caught our attention was the registry change related to the Windows Security Center.

$\texttt{HKLM}\backslash\texttt{System}\backslash\texttt{CurrentControlSet}\backslash\texttt{services}\backslash\texttt{wscsvc}\backslash\texttt{Start} = 4$

The value 4 means disabled.

Regshot

With regshot we had a confirmation of all the actions tracked with procmon.

The fact that caught our attention was the registry change related to the Windows Security Center.

`HKLM\System\CurrentControlSet\services\wscsvc\Start` = 4

The value 4 means disabled.

The weird thing is that this value change has not been made by the malware.

We discovered that the value was changed by **services.exe**

Reverse engineering

Reverse engineering

The reverse engineering was divided in 2 phases:

- Code rebuilding
- Debugging

Code rebuilding

We explored the cfg of the start funcion created by IDA, and we built a pseudo code for the first part, which deals with the decryption of the obfuscated zone.

Code rebuilding

We explored the cfg of the start funcion created by IDA, and we built a pseudo code for the first part, which deals with the decryption of the obfuscated zone.

The *.vmp0* section is decrypted through a cycle that perform an arithmetic xor of the code with a certain key.

The cycle is repetead 7 times, but during the last one the key is incremented by one.

The key is $0x58$.

Debugging

To debug the code we used Ollydbg alongside procmon, executing instructions one by one, stepping over the function calls and keeping track of the actions performed.

We had 2 main target:

- Detect the infection function
- Detect the deactivation of the security center

Debugging

Eventually we achieved a procedure to debug the infection function:

1. breakpoint in 0101273*A*; then after the *RET* the malware enters the obfuscated section.
2. breakpoint in 010*AAB*30; then there is the creation of the second thread which is the analyzed one.
3. breakpoint in 010*BD*0*A*9 which is the begin of the target function

Debugging

Eventually we achieved a procedure to debug the infection function:

1. breakpoint in 0101273*A*; then after the *RET* the malware enters the obfuscated section.
2. breakpoint in 010*AAB*30; then there is the creation of the second thread which is the analyzed one.
3. breakpoint in 010*BD*0*A*9 which is the begin of the target function

In particular we discovered that the thread calls FUN_010ACABF, which then calls FUN_0109F059, which then calls iteratively FUN_010B0CA this last one contains FUN_010BD0A9 which is the target function that performs the malicious actions.

Infection

The infection function is FUN_10BD636.

It has only the target name as parameter.

Infection

The infection function is FUN_10BD636.

It has only the target name as parameter.

This procedure is huge, with too much code to reverse; it even contains a recursive call inside.

Security center dectivation

We set the procmon filters to monitor “services.exe”, and in particular the “RegSetValue” operation.

Security center dectivation

We set the procmon filters to monitor “services.exe”, and in particular the “RegSetValue” operation.

The malicious action is done at $010B10C7$. The third time that this instruction is executed the security center is deactivated

This action is a call to **StartServiceA** from **ADVAPI32.dll**.

Security center dectivation

We set the procmon filters to monitor “services.exe”, and in particular the “RegSetValue” operation.

The malicious action is done at $010B10C7$. The third time that this instruction is executed the security center is deactivated

This action is a call to **StartServiceA** from **ADVAPI32.dll**.

Reading the documentation we discovered that this function needs priviledge to close services

Security center dectivation

We set the procmon filters to monitor “services.exe”, and in particular the “RegSetValue” operation.

The malicious action is done at $010B10C7$. The third time that this instruction is executed the security center is deactivated

This action is a call to **StartServiceA** from **ADVAPI32.dll**.

Reading the documentation we discovered that this function needs priviledge to close services

$\longrightarrow$ *probable priviledge escalation*

Conclusioni