Learning From ICEID loader - Including its Steganography Payload Parsing

Published: 2020-08-10 · Archived: 2026-04-05 12:41:55 UTC

This ICEID stager or loader show some interesting way in loading the the ICEID downloader to bypassed some API monitoring tools to unpack its code and to evade forensic tools in memory. I also learned how it tries to parse the png header to decrypt its encrypted payload (Steganography)

So Lets Start!

Loading Runas.exe:

One Interesting code of this IceID stager/loader is that it tries to load the "runas.exe" using LoadLibraryExA. Currrently I don't know what are the other purpose of this loading aside from evading or bypassing emulation tools or engine.

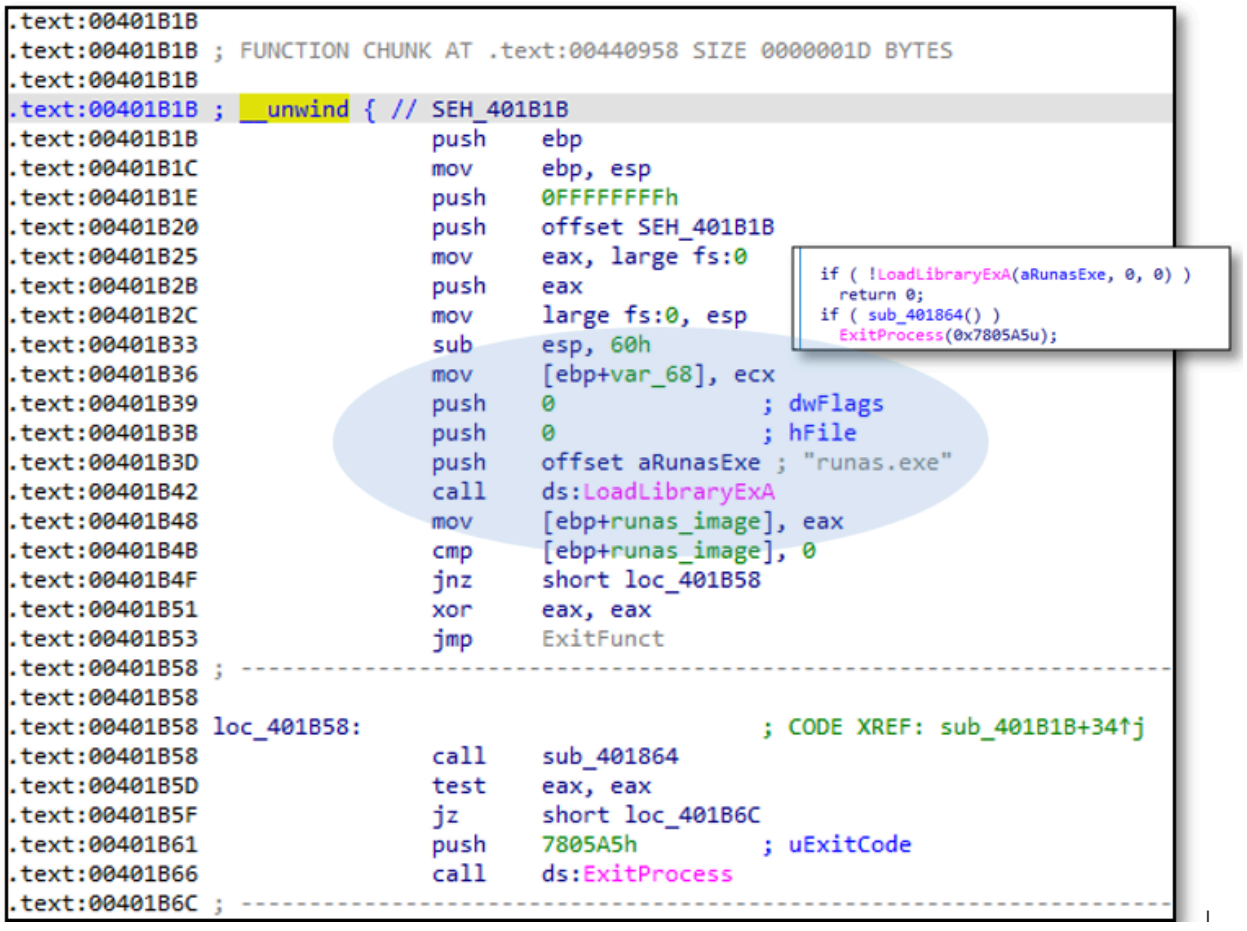


figure 1: loadlibraryExA runas.exe

Decrypting Shellcode Loader and the IceID Downloader:

Next it will decrypt the encrypted shellcode and IceID downloader from its RSRC section. This is done by finding specific resource name, locate its address location and its resource entry size to the file as shown in the figure 2 below.

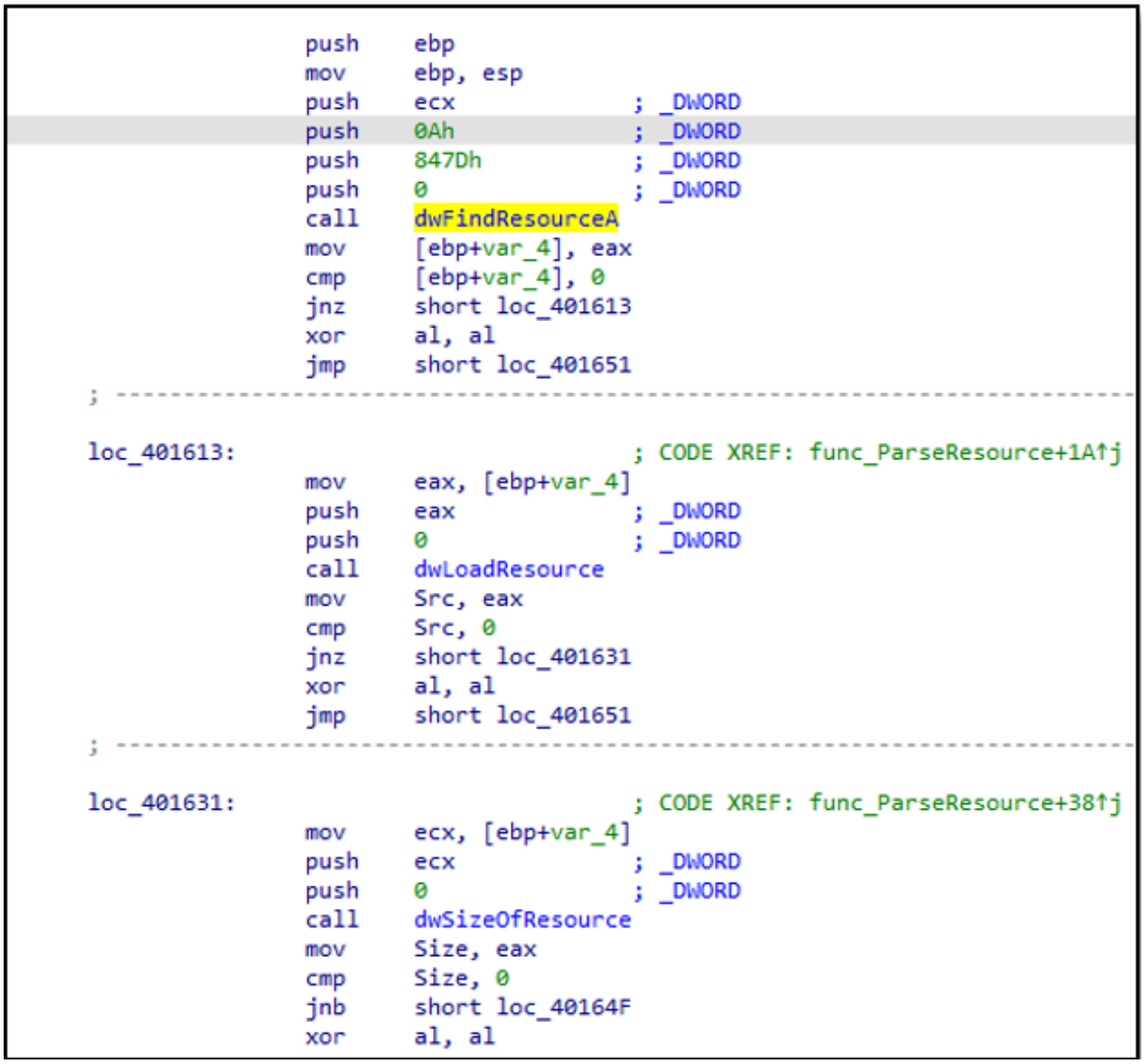
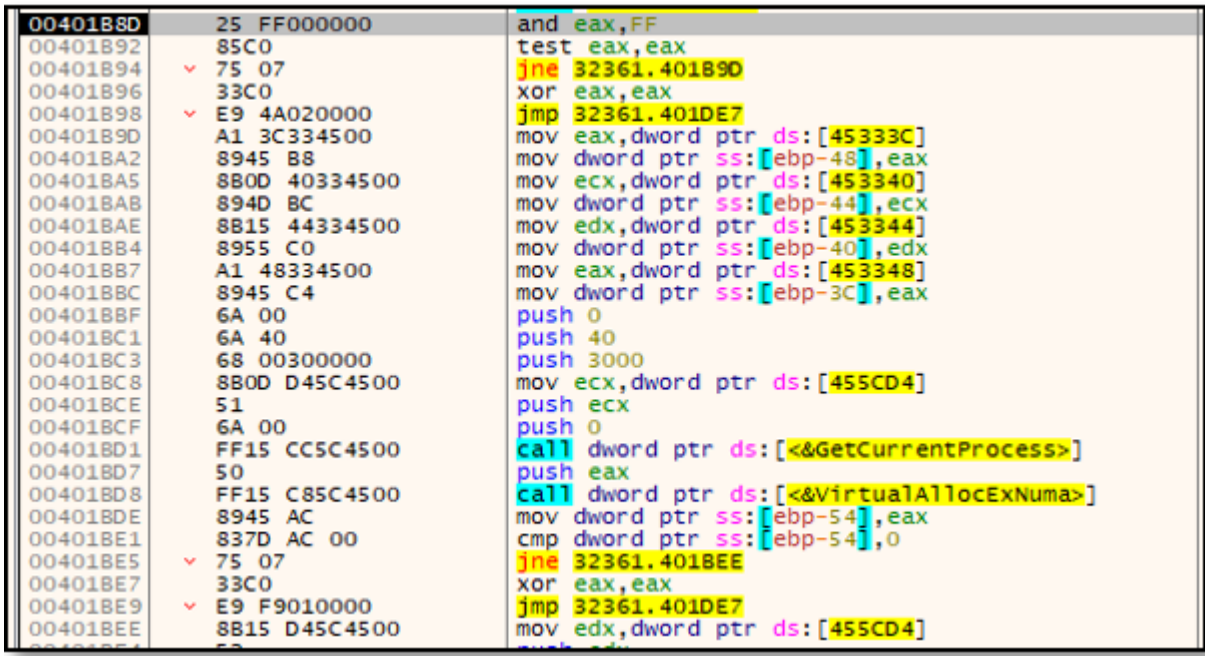


figure 2: Finding ResourceEntry

After this It will allocate a Virtual Memory using alternative API of VirtualAlloc which is VirtualAllocExNumA that may evade some unpacking tool that hook common Virtual Allocation API.



```
00401B8D    25 FF000000     and eax,FF
00401B92    85C0            test eax,eax
00401B94    75 07           jne 32361.401B9D
00401B96    33C0            xor eax,eax
00401B98    E9 4A020000     jmp 32361.401DE7
00401B9D    A1 3C334500     mov eax,dword ptr ds:[45333C]
00401BA2    8945 B8         mov dword ptr ss:[ebp-48],eax
00401BA5    8B0D 40334500   mov ecx,dword ptr ds:[453340]
00401BAB    894D BC         mov dword ptr ss:[ebp-44],ecx
00401BAE    8B15 44334500   mov edx,dword ptr ds:[453344]
00401BB4    8955 C0         mov dword ptr ss:[ebp-40],edx
00401BB7    A1 48334500     mov eax,dword ptr ds:[453348]
00401BBC    8945 C4         mov dword ptr ss:[ebp-3C],eax
00401BBF    6A 00           push 0
00401BC1    6A 40           push 40
00401BC3    68 00300000     push 3000
00401BC8    8B0D D45C4500   mov ecx,dword ptr ds:[455CD4]
00401BCE    51              push ecx
00401BCF    6A 00           push 0
00401BD1    FF15 CC5C4500   call dword ptr ds:[<&GetCurrentProcess>]
00401BD7    50              push eax
00401BD8    FF15 C85C4500   call dword ptr ds:[<&VirtualAllocExNuma>]
00401BDE    8945 AC         mov dword ptr ss:[ebp-54],eax
00401BE1    837D AC 00      cmp dword ptr ss:[ebp-54],0
00401BE5    75 07           jne 32361.401BEE
00401BE7    33C0            xor eax,eax
00401BE9    E9 F9010000     jmp 32361.401DE7
00401BEE    8B15 D45C4500   mov edx,dword ptr ds:[455CD4]
```

figure 3: Allocation of Memory

Then it will decrypt the encrypted rsrc data using Microsoft CSP API show in figure below with RC4 algorithm. Interestingly, The common way to decrypt an encrypted blob of data using CSP API is "CryptDecrpyt" butthis malware used "CryptEncrypt" API instead to decrypt the blob.

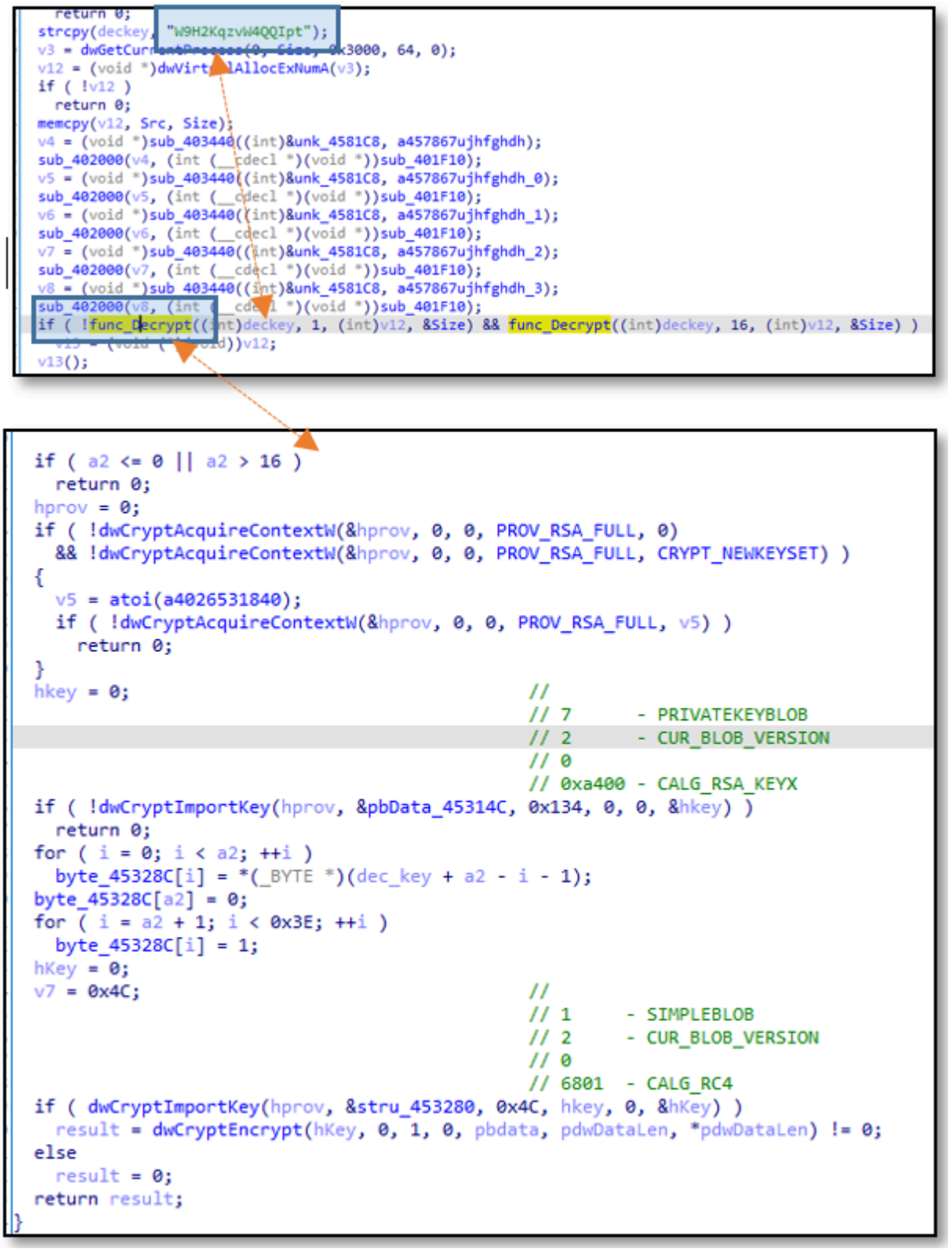


figure 4: the decryption function for encrypted resource section

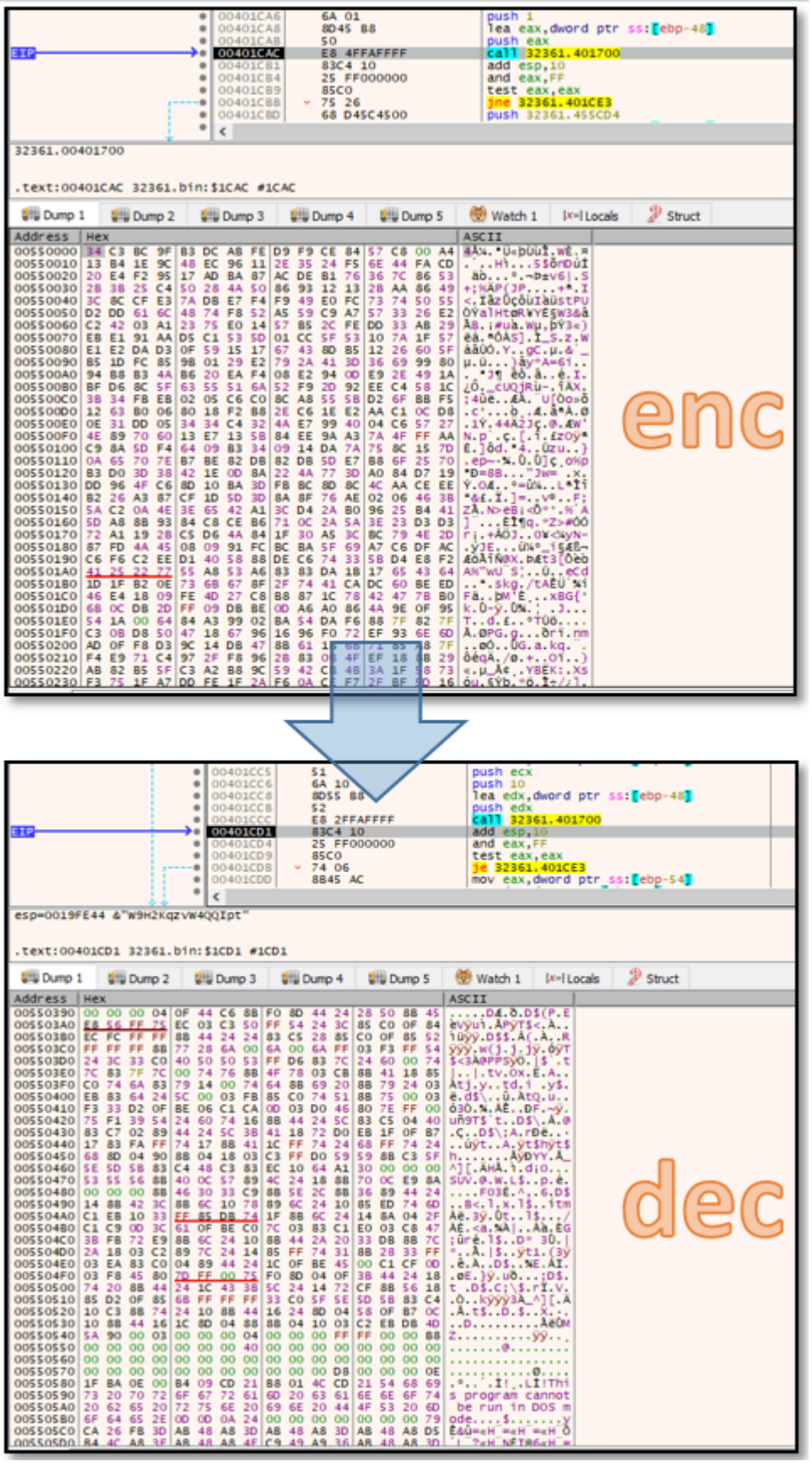


figure 5: decrypted shellcode and ICEID downloader

Processing PNG Payload - Steganography:

This part of the ICEID downloader is really interesting where I learned how it parse the PNG header to look for IDAT PNG header and decrypt it, but first it will check if the commandline of the ICEID downloader process has a arguments "-id=" that contain an int value that would be the name of the downloaded steganography png file that should be place in %tmp% folder.

```
v4 = GetCommandLineA();
proc_arg = StrStrIA(v4, "-id=");
if ( !proc_arg )
  return 0;
payload_name = StrToIntA(proc_arg + 4);
tmp_path = GetTempPathA(0x104u, payload_png_path);
wsprintfA(&payload_png_path[tmp_path], "~%u.tmp", payload_name);
if ( !sub_40143A(payload_png_path, (int)&v12) )
  return 0;
v14 = 4;
v15 = 0;
v13 = &payload_name;
v16 = 0;
v17 = 0;
v7 = func_checkpngPayload(&lpMem);
if ( lpMem )
{
  v10 = lpMem;
  v8 = GetProcessHeap();
  HeapFree(v8, 0, v10);
}
result = 1;
if ( v7 != 1 )
  return 0;
*a1 = v16;
*a2 = v17;
return result;
}
```

figure 6: checking the process commandline and decrytion function

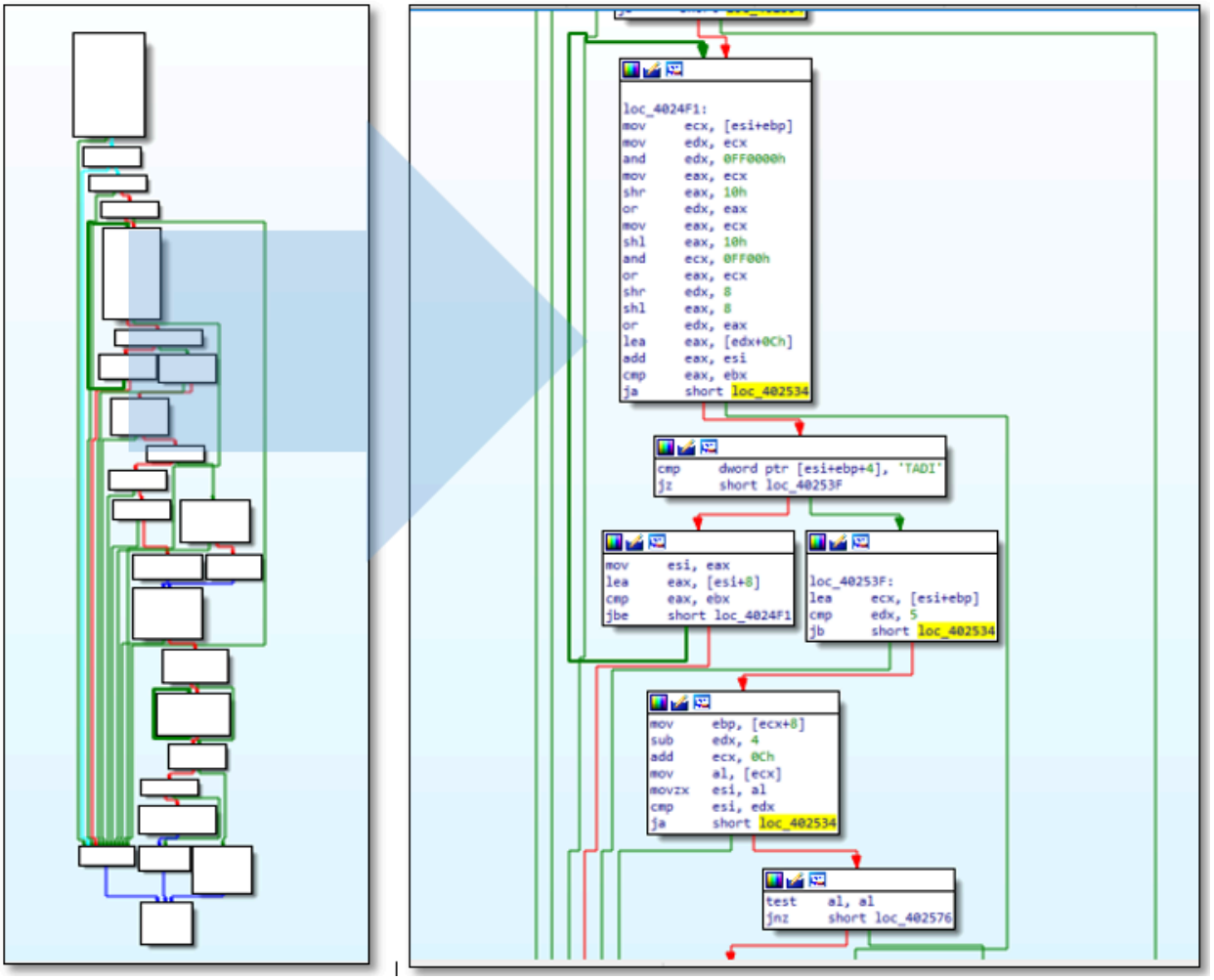


figure 7: the parsing of PNG payload

after having the IDAT header position it will parse the rc4 key below it and the encrypted data to decrypt it using RC4 decryption algorithm.

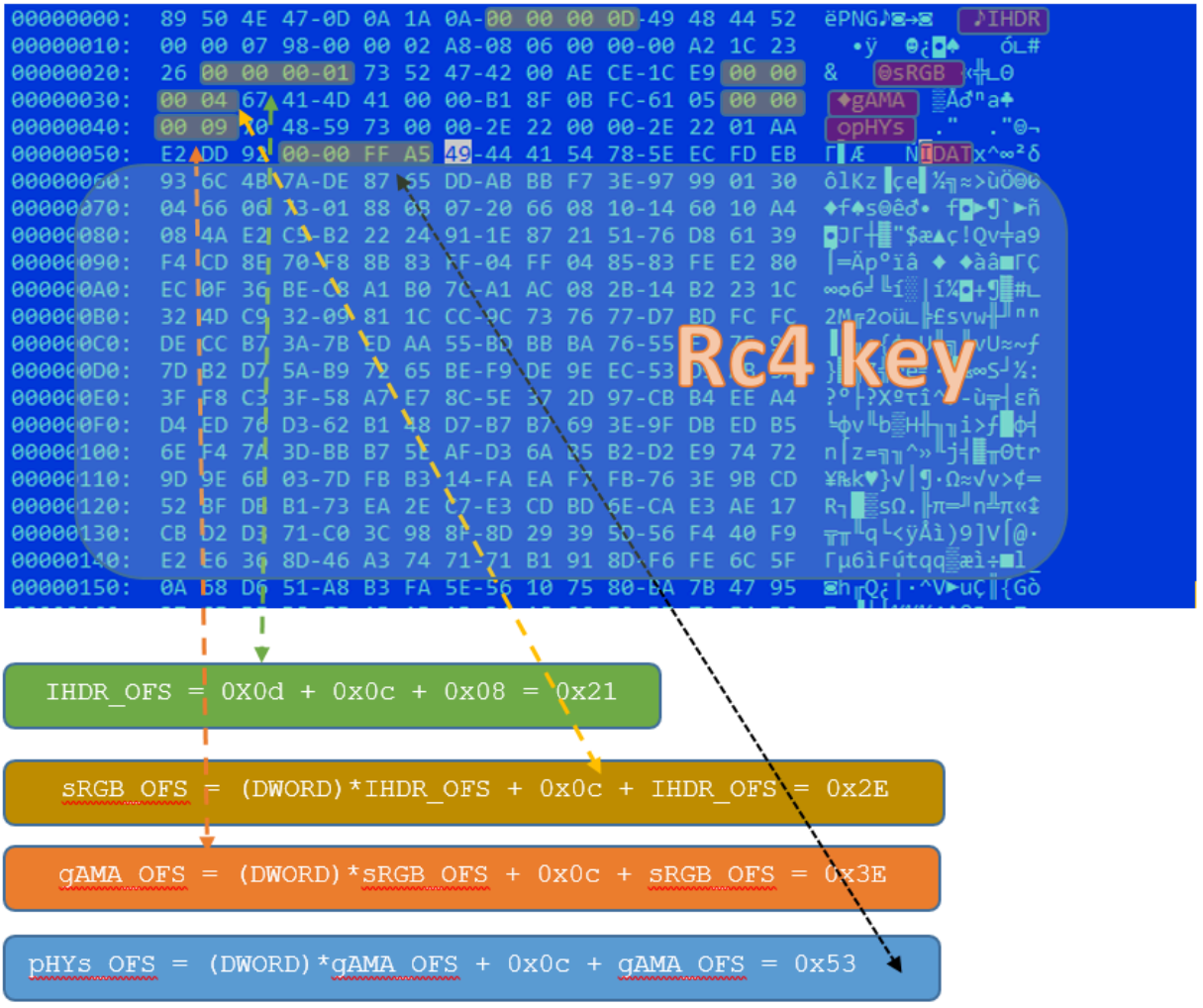


figure 8: parsing png header payload

ANTI-Memory Forensic:

Also I notice that upon loading the ICEID downloader to the memory to execute it, the loader removed the DOS header as a common anti-forensic technique.

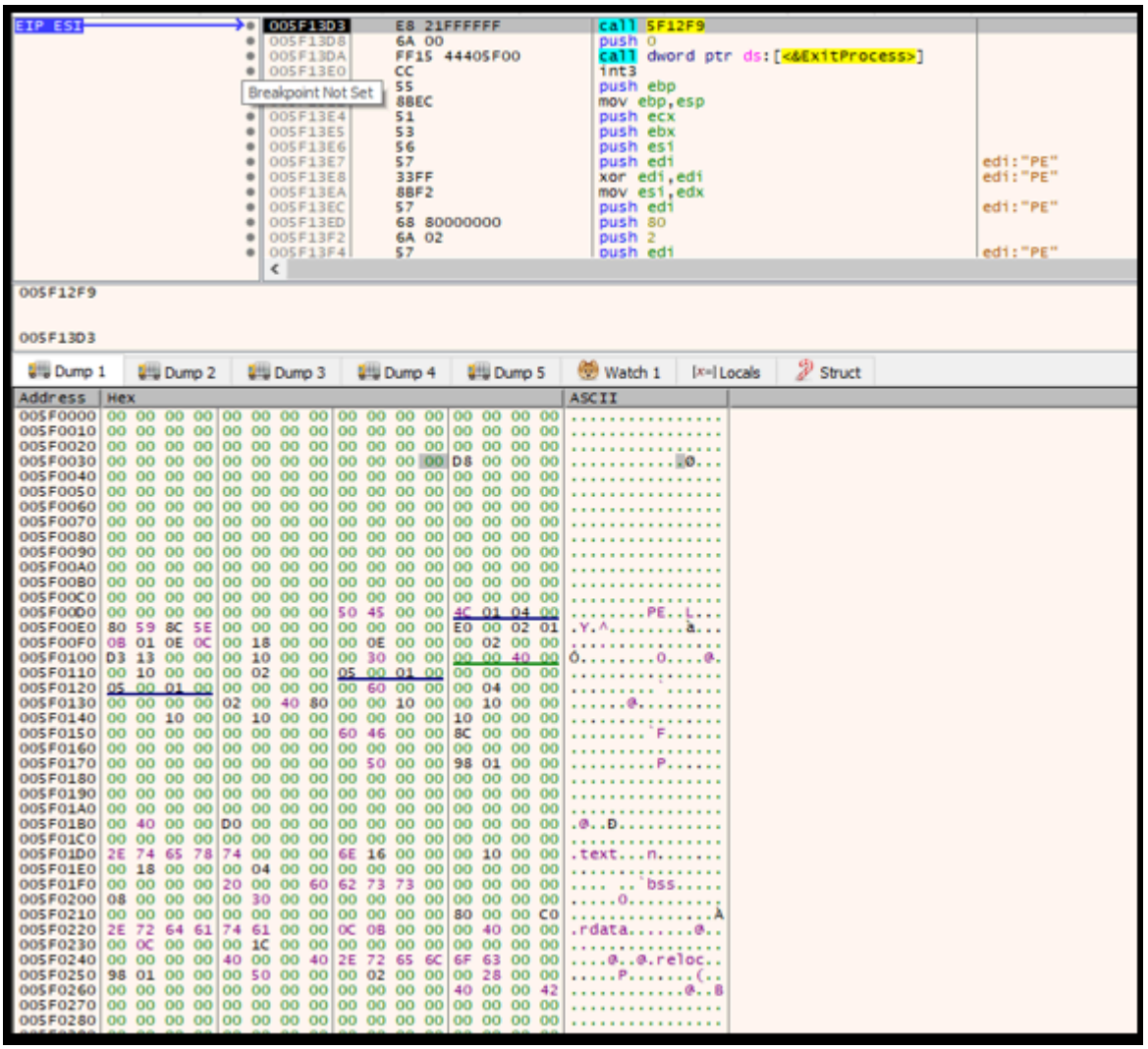


figure 9: Anti Memory Forensic Technique

IOC:

Loader:

https://app.any.run/tasks/b4beb108-60c8-4ae5-8f7b-4f21ffa5da7a/

SHA1: 56be44a912e2677e98cbce0c42a8344a7de34ea1

MD5: bd57f946b9294c90772e57e20247d1eb

SHA256: 81801711abd4b24eb39be359ce18a54600f3a362c033a38c01881c941f8743b4

ICEID downloader:

Sha1: e8a1f8e06b332cece343718d80ad942b1466c07b

MD5: 5c2766313ce3ce8d3321c81f347d2813

Sha256: 295dc254c4d168ab935e84b229746586ce69028f39f0612f6a900b7a01bae9e5

Strings:

8476:GetNativeSystemInfo
8496:ZwQuerySystemInformation
8524:NTDLL.DLL
8544:0123456789ABCDEF
8576:RtlGetVersion
8688:GetAdaptersInfo
8704:IPHLPAPI.DLL
8776:url("
8788:src="
9146:LookupAccountNameW
9168:GetUserNameA
9182:ADVAPI32.dll
9198:StrStrIA
9210:StrToIntA
9222:StrChrA
9230:SHLWAPI.dll
9244:GetModuleFileNameA
9266:HeapFree
9278:WaitForSingleObject
9300:GetCommandLineA
9318:Sleep
9326:GetTempPathA
9342:LoadLibraryA
9358:GetProcAddress
9376:ExitProcess
9390:GetProcessHeap
9408:GetTickCount
9424:ReadFile
9436:WriteFile
9448:CreateFileA
9462:CloseHandle
9476:HeapAlloc
9488:GetFileSize
9502:lstrlenA
9514:HeapReAlloc
9528:GetComputerNameExW
9550:GetTickCount64
9568:GetLastError
9584:SwitchToThread
9602:GetComputerNameExA
9622:KERNEL32.dll

9638:wsprintfA
9650:wsprintfW
9660:USER32.dll
9674:WinHttpQueryDataAvailable
9702:WinHttpConnect
9720:WinHttpSetStatusCallback
9748:WinHttpSendRequest
9770:WinHttpCloseHandle
9792:WinHttpSetOption
9812:WinHttpOpenRequest
9834:WinHttpReadData
9852:WinHttpQueryHeaders
9874:WinHttpOpen
9888:WinHttpReceiveResponse
9914:WinHttpQueryOption
9936:WinHttpAddRequestHeaders
9962:WINHTTP.dll
9976:memset
9984:MSVCRT.dll
10752:dave

Closing:

In this blog post I learned new way to allocate Virtual memory, LoadLibraryExA for executable and last parsing png header. I hope I share something. :)
