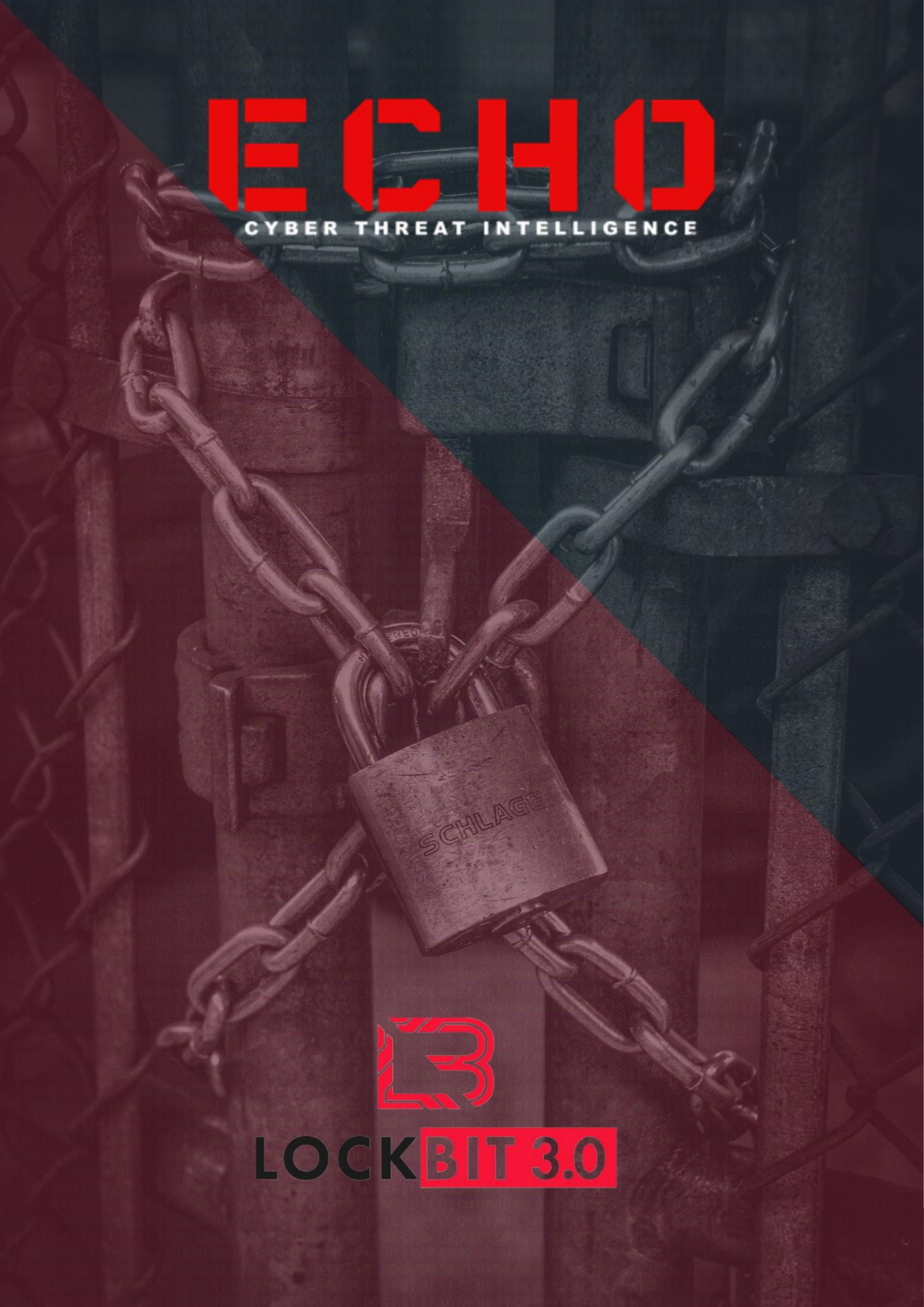


ECHO

CYBER THREAT INTELLIGENCE

LOCKBIT 3.0

Content

Introduction

LockBit 3.0, since its emergence in 2019, has become an exceedingly perilous member within the family of ransomware. As a result, it poses a significant cybersecurity threat to numerous organizations worldwide. LockBit operates by encrypting data on victim systems and subsequently demanding a ransom in exchange for data decryption. However, LockBit 3.0 goes beyond mere data encryption; it also threatens victims with the online publication of their data, thus tarnishing the reputation and credibility of organizations. When deployed on a victim's system, LockBit 3.0 employs highly advanced encryption algorithms, rendering the decryption of data exceedingly challenging and coercing victims into making ransom payments. Ransom payments are usually made using cryptocurrencies, making it nearly impossible to trace the paid ransom.

LockBit 3.0 targets numerous countries worldwide. However, some countries stand out as being more heavily impacted or intensively targeted by this malicious ransomware (CISA,2023). These countries are:

- Russia: LockBit 3.0 frequently targets organizations within Russia, potentially affecting both large and small businesses, government institutions, and individuals.
- United States: Given its status as one of the world's largest economies, the United States is an appealing target for LockBit 3.0. Sectors such as finance, healthcare, manufacturing, and technology are frequently targeted.
- Canada: Canada is another country targeted by LockBit 3.0, and various sectors within Canada may experience the impacts of this ransomware.
- United Kingdom: With one of Europe's largest economies, the United Kingdom represents an attractive target for LockBit 3.0. Sectors such as finance and healthcare are frequent targets.
- Germany: Germany, with its technology, manufacturing, and other sectors, is often heavily targeted by LockBit 3.0.

LockBit 3.0 targets organizations operating across various sectors, and many of these sectors have already experienced the effects of this ransomware (BleepingComputer,2023). The targeted sectors include:

- Healthcare Sector: Healthcare organizations are often targeted by ransomware due to the sensitive patient data they store. LockBit 3.0 aims at hospitals, clinics, and health insurance companies.
- Financial Sector: LockBit 3.0 can cause significant damage to the financial sector by targeting banks, financial consulting firms, and financial institutions.
- Manufacturing Sector: Manufacturing facilities and industrial enterprises are critical in terms of production processes and supply chain management. LockBit 3.0 can lead to production disruptions in the manufacturing sector.
- Technology Sector: Technology companies may become targets of LockBit 3.0 due to the customer information and intellectual property they store. This can significantly impact the reputation and competitiveness of technology firms.
- Other Sectors: LockBit 3.0 can also target organizations in education, retail, energy, and various other sectors.

All these threats collectively make LockBit 3.0 a substantial cybersecurity menace for organizations. It is crucial for organizations to implement robust security measures and develop defense strategies against ransomware. This report provides an in-depth analysis of LockBit 3.0, offering essential information on how organizations can protect themselves against this threat. Taking the appropriate security measures is a critical step in safeguarding data and reputation for organizations.

Attack Chain

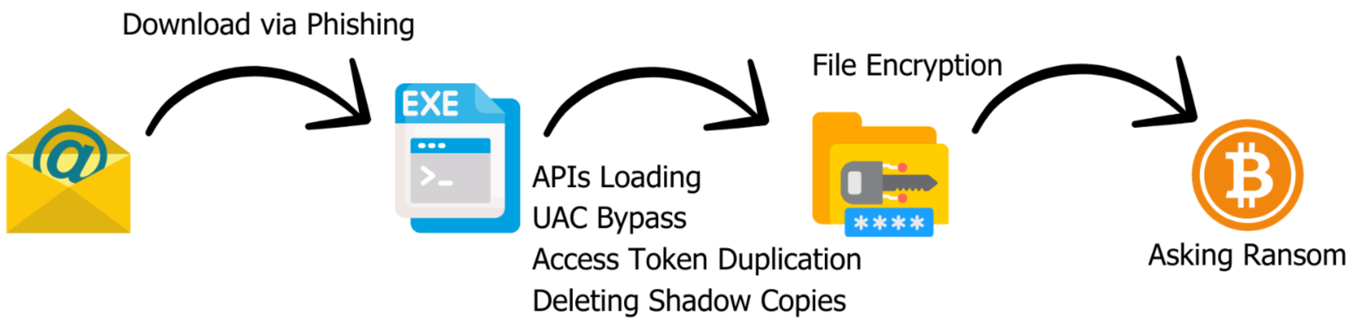


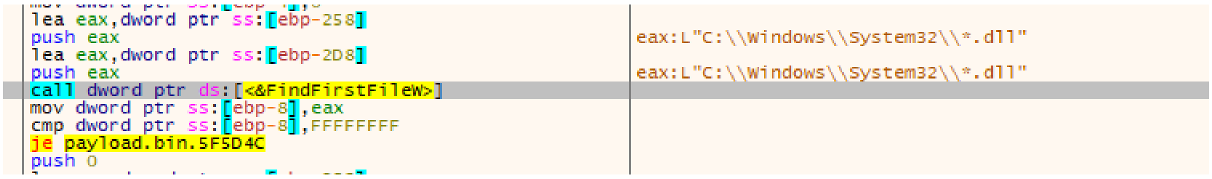
Figure 1 Attack Chain

Technical Analysis

Analysing of Payload.bin

MD5	bbe63d8efc8d8dc7f387b08ee07721ba
SHA256	2e8aaa6338cbf95d8d268559fb8afac64e1c0dfc9ded4bb2de63a9db634e354d
File Type	PE32/EXE

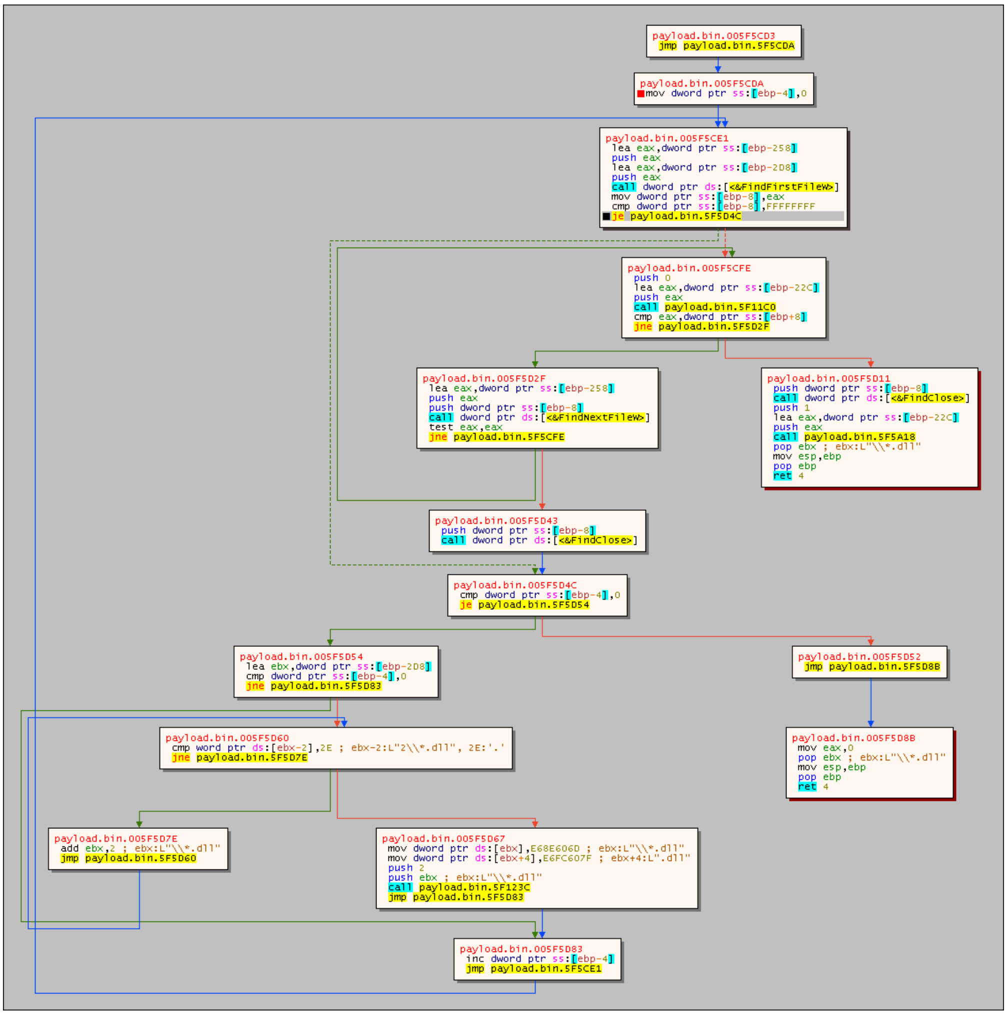
Figure 2 General File Information



```
lea eax,dword ptr ss:[ebp-258]
push eax                                    eax:L"C:\\Windows\\System32\\*.dll"
lea eax,dword ptr ss:[ebp-2D8]
push eax                                    eax:L"C:\\Windows\\System32\\*.dll"
call dword ptr ds:[<&FindFirstFileW>]
mov dword ptr ss:[ebp-8],eax
cmp dword ptr ss:[ebp-8],FFFFFFFF
je payload.bin.5F5D4C
push 0
```

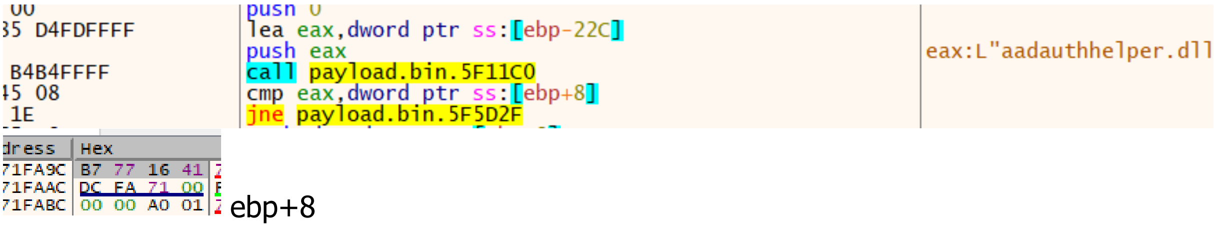
Figure 3 FindFirstFile: C:||Windows||System32||.dll*

It has been observed that the malicious entity sequentially traverses the DLL files within the "**System32**" directory.



```
payload.bin.005F5CD3
 jmp payload.bin.5F5CDA

payload.bin.005F5CDA
 mov dword ptr ss:[ebp-4],0

payload.bin.005F5CE1
 lea eax,dword ptr ss:[ebp-258]
 push eax
 lea eax,dword ptr ss:[ebp-2D8]
 push eax
 call dword ptr ds:[<&FindFirstFileW>]
 mov dword ptr ss:[ebp-8],eax
 cmp dword ptr ss:[ebp-8],FFFFFFFF
 je payload.bin.5F5D4C

payload.bin.005F5CFE
 push 0
 lea eax,dword ptr ss:[ebp-22C]
 push eax
 call payload.bin.5F11C0
 cmp eax,dword ptr ss:[ebp+8]
 jne payload.bin.5F5D2F

payload.bin.005F5D2F
 lea eax,dword ptr ss:[ebp-258]
 push eax
 push dword ptr ss:[ebp-8]
 call dword ptr ds:[<&FindNextFileW>]
 test eax,eax
 jne payload.bin.5F5CFE

payload.bin.005F5D11
 push dword ptr ss:[ebp-8]
 call dword ptr ds:[<&FindClose>]
 push 1
 lea eax,dword ptr ss:[ebp-22C]
 push eax
 call payload.bin.5F5A18
 pop ebx ; ebx:L"\\*.dll"
 mov esp,ebp
 pop ebp
 ret 4

payload.bin.005F5D43
 push dword ptr ss:[ebp-8]
 call dword ptr ds:[<&FindClose>]

payload.bin.005F5D4C
 cmp dword ptr ss:[ebp-4],0
 je payload.bin.5F5D54

payload.bin.005F5D54
 lea ebx,dword ptr ss:[ebp-2D8]
 cmp dword ptr ss:[ebp-4],0
 jne payload.bin.5F5D83

payload.bin.005F5D52
 jmp payload.bin.5F5D8B

payload.bin.005F5D60
 cmp word ptr ds:[ebx-2],2E ; ebx-2:L"2\\*.dll", 2E:'.'
 jne payload.bin.5F5D7E

payload.bin.005F5D8B
 mov eax,0
 pop ebx ; ebx:L"\\*.dll"
 mov esp,ebp
 pop ebp
 ret 4

payload.bin.005F5D7E
 add ebx,2 ; ebx:L"\\*.dll"
 jmp payload.bin.5F5D60

payload.bin.005F5D67
 mov dword ptr ds:[ebx],E68E606D ; ebx:L"\\*.dll"
 mov dword ptr ds:[ebx+4],E6FC607F ; ebx+4:L".dll"
 push 2
 push ebx ; ebx:L"\\*.dll"
 call payload.bin.5F123C
 jmp payload.bin.5F5D83

payload.bin.005F5D83
 inc dword ptr ss:[ebp-4]
 jmp payload.bin.5F5CE1
```

Figure 4 Traversal Algorithm



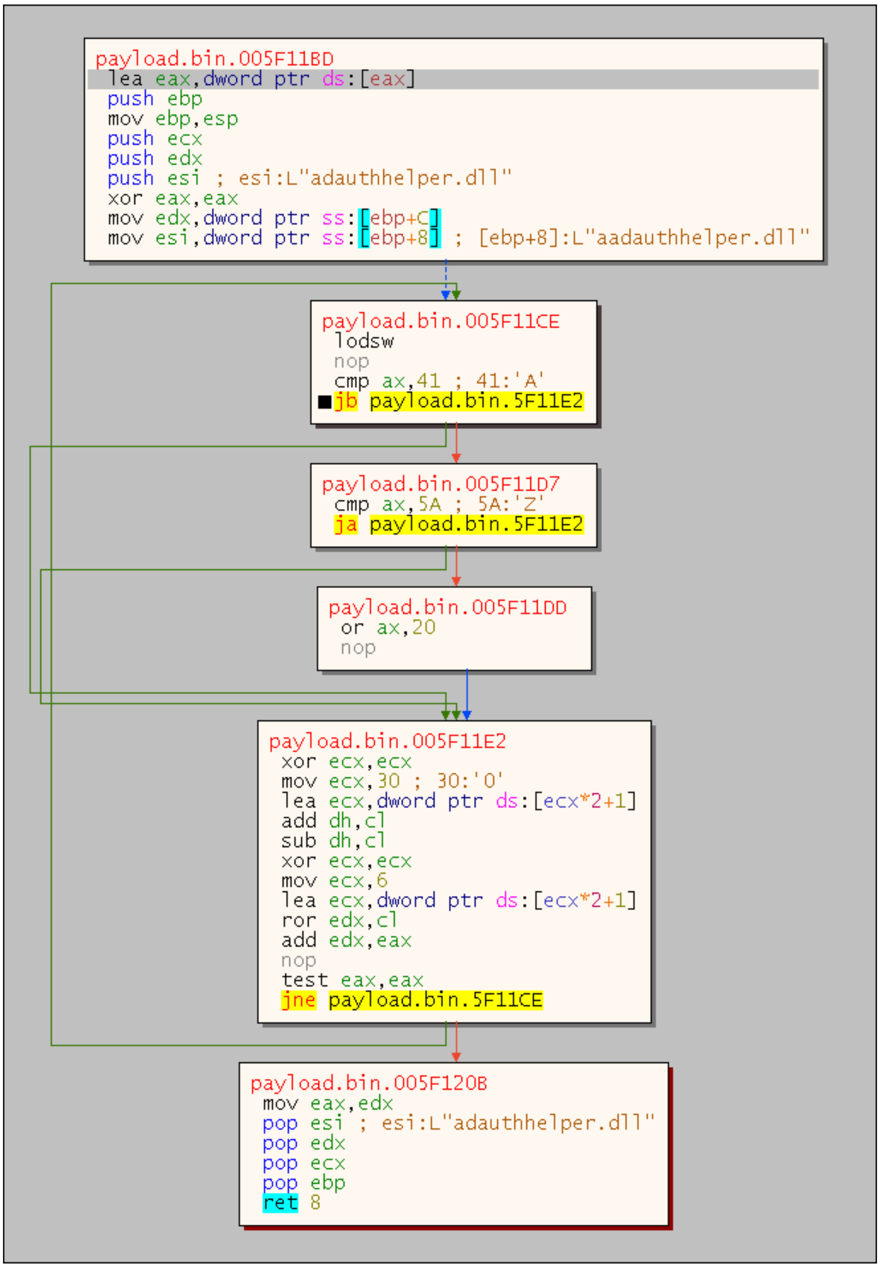
```
00           push 0
35 D4FDFFFF  lea eax,dword ptr ss:[ebp-22C]
             push eax                          eax:L"aadauthhelper.dll
B4B4FFFF     call payload.bin.5F11C0
45 08        cmp eax,dword ptr ss:[ebp+8]
1E           jne payload.bin.5F5D2F
```

```
dress     Hex
71FA9C    B7 77 16 41
71FAAC    DC FA 71 00
71FABC    00 00 A0 01
```

ebp+8

Figure 5 Hash Generating and Comparing

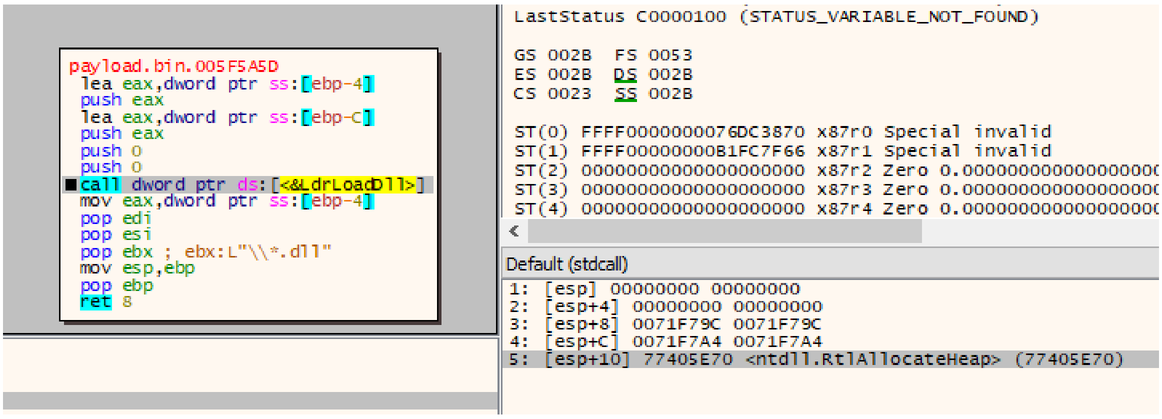
During the traversal process, one notable operation stands out: generating hash for DLL files. Similar to the malicious API Hashing method, it creates hashes of DLL names and then compares them to reach the desired DLL file.



```
payload.bin.005F11BD
 lea eax,dword ptr ds:[eax]
 push ebp
 mov ebp,esp
 push ecx
 push edx
 push esi ; esi:L"adauthhelper.dll"
 xor eax,eax
 mov edx,dword ptr ss:[ebp+C]
 mov esi,dword ptr ss:[ebp+8] ; [ebp+8]:L"aadauthhelper.dll"

payload.bin.005F11CE
 lodsw
 nop
 cmp ax,41 ; 41:'A'
■jb payload.bin.5F11E2

payload.bin.005F11D7
 cmp ax,5A ; 5A:'Z'
 ja payload.bin.5F11E2

payload.bin.005F11DD
 or ax,20
 nop

payload.bin.005F11E2
 xor ecx,ecx
 mov ecx,30 ; 30:'0'
 lea ecx,dword ptr ds:[ecx*2+1]
 add dh,cl
 sub dh,cl
 xor ecx,ecx
 mov ecx,6
 lea ecx,dword ptr ds:[ecx*2+1]
 ror edx,cl
 add edx,eax
 nop
 test eax,eax
 jne payload.bin.5F11CE

payload.bin.005F120B
 mov eax,edx
 pop esi ; esi:L"adauthhelper.dll"
 pop edx
 pop ecx
 pop ebp
 ret 8
```

Figure 6 DLL Name Hashing Algorithm

The hash information for the searched DLL file has been determined to be **"41 16 77 B7"** and it has been identified to correspond to the **"ntdll.dll"** file.



```
payload.bin.005F5A5D
 lea eax,dword ptr ss:[ebp-4]
 push eax
 lea eax,dword ptr ss:[ebp-C]
 push eax
 push 0
 push 0
■call dword ptr ds:[<&LdrLoadDll>]
 mov eax,dword ptr ss:[ebp-4]
 pop edi
 pop esi
 pop ebx ; ebx:L"\\*.dll"
 mov esp,ebp
 pop ebp
 ret 8
```

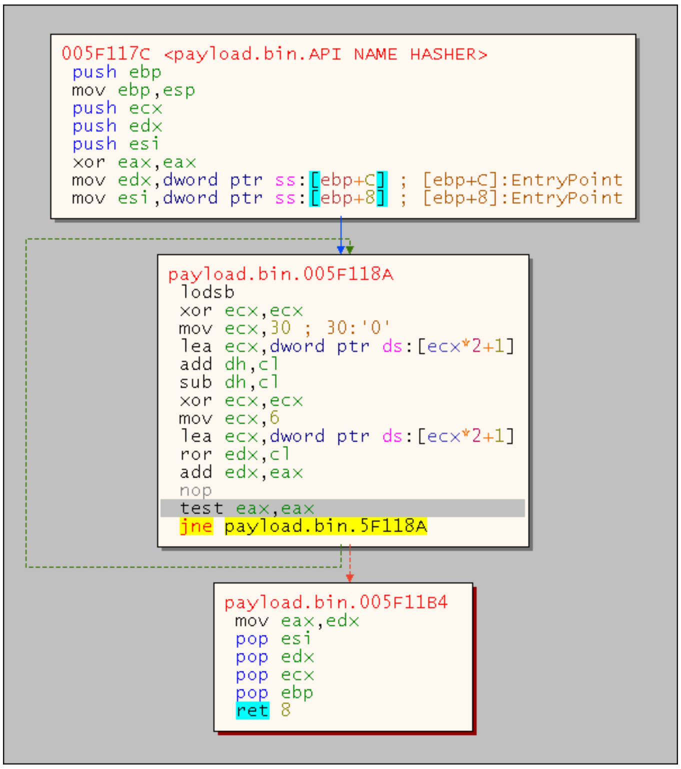
```
LastStatus C0000100 (STATUS_VARIABLE_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000076DC3870 x87r0 Special invalid
ST(1) FFFF00000000B1FC7F66 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000
```

```
Default (stdcall)
1: [esp] 00000000 00000000
2: [esp+4] 00000000 00000000
3: [esp+8] 0071F79C 0071F79C
4: [esp+C] 0071F7A4 0071F7A4
5: [esp+10] 77405E70 <ntdll.RtlAllocateHeap> (77405E70)
```

Figure 7 LdrLoadDll

It is observed that the DLL files are loaded using the **LdrLoadDll** API. It is noted that the malicious actor avoids using the **LoadLibrary** API.

Furthermore, it has been determined that the malicious entity employs the API Hashing technique. Unlike the traditional API Hashing method, it is observed that the **LdrLoadDll** and **LdrGetProcedureAddress** functions are used instead of **LoadLibrary** and **GetProcAddress** functions.

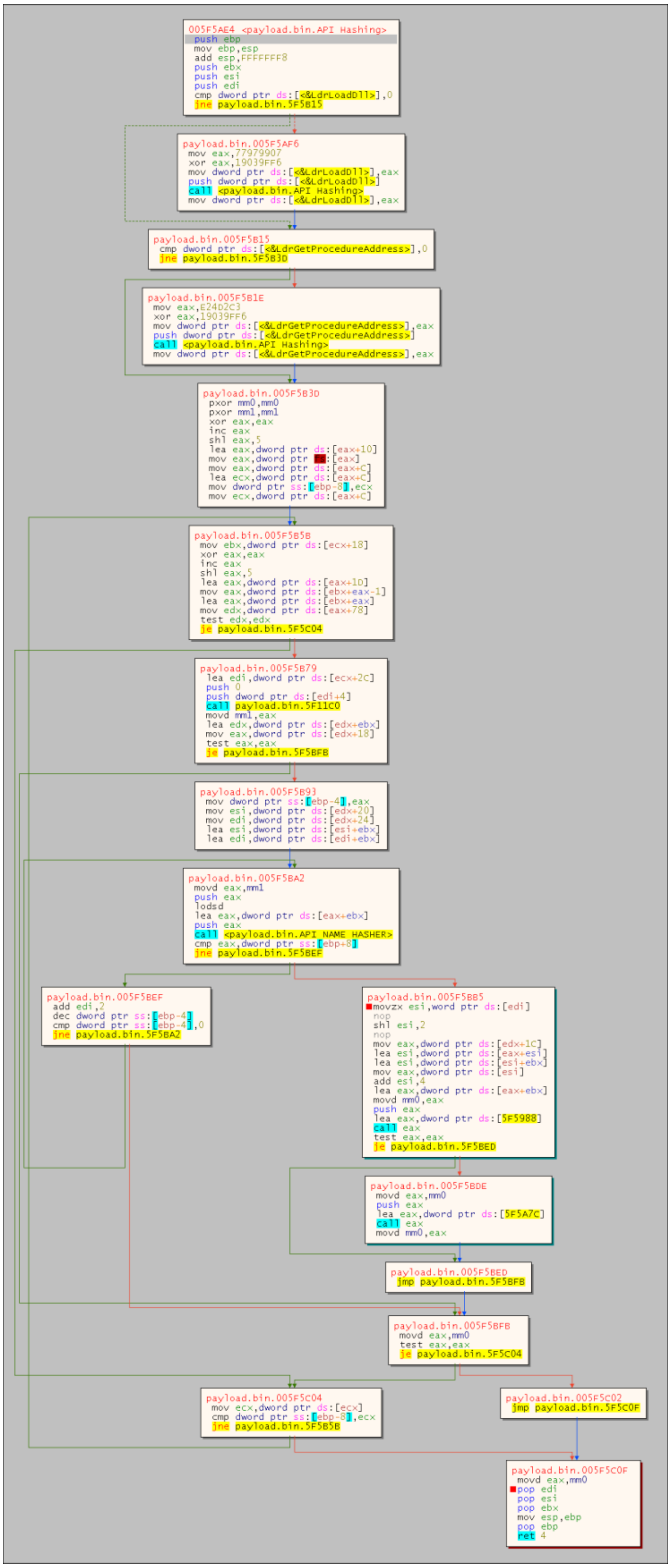


```
005F117C <payload.bin.API NAME HASHER>
 push ebp
 mov ebp,esp
 push ecx
 push edx
 push esi
 xor eax,eax
 mov edx,dword ptr ss:[ebp+C] ; [ebp+C]:EntryPoint
 mov esi,dword ptr ss:[ebp+8] ; [ebp+8]:EntryPoint
```

```
payload.bin.005F118A
 lodsb
 xor ecx,ecx
 mov ecx,30 ; 30:'0'
 lea ecx,dword ptr ds:[ecx*2+1]
 add dh,cl
 sub dh,cl
 xor ecx,ecx
 mov ecx,6
 lea ecx,dword ptr ds:[ecx*2+1]
 ror edx,cl
 add edx,eax
 nop
 test eax,eax
 jne payload.bin.5F118A
```

```
payload.bin.005F11B4
 mov eax,edx
 pop esi
 pop edx
 pop ecx
 pop ebp
 ret 8
```

Figure 8 API Name Hashing Algorithm



```
005F5AE4 <payload.bin.API Hashing>
 push ebp
 mov ebp,esp
 add esp,FFFFFFF8
 push ebx
 push esi
 push edi
 cmp dword ptr ds:[<&LdrLoadDll>],0
 jne payload.bin.5F5B15
```

```
payload.bin.005F5AF6
 mov eax,77979907
 xor eax,19039FF6
 mov dword ptr ds:[<&LdrLoadDll>],eax
 push dword ptr ds:[<&LdrLoadDll>]
 call <payload.bin.API Hashing>
 mov dword ptr ds:[<&LdrLoadDll>],eax
```

```
payload.bin.005F5B15
 cmp dword ptr ds:[<&LdrGetProcedureAddress>],0
 jne payload.bin.5F5B3D
```

```
payload.bin.005F5B1E
 mov eax,E24D2C3
 xor eax,19039FF6
 mov dword ptr ds:[<&LdrGetProcedureAddress>],eax
 push dword ptr ds:[<&LdrGetProcedureAddress>]
 call <payload.bin.API Hashing>
 mov dword ptr ds:[<&LdrGetProcedureAddress>],eax
```

```
payload.bin.005F5B3D
 pxor mm0,mm0
 pxor mm1,mm1
 xor eax,eax
 inc eax
 shl eax,5
 lea eax,dword ptr ds:[eax+10]
 mov eax,dword ptr fs:[eax]
 mov eax,dword ptr ds:[eax+C]
 lea ecx,dword ptr ds:[eax+C]
 mov dword ptr ss:[ebp-8],ecx
 mov ecx,dword ptr ds:[eax+C]
```

```
payload.bin.005F5B5B
 mov ebx,dword ptr ds:[ecx+18]
 xor eax,eax
 inc eax
 shl eax,5
 lea eax,dword ptr ds:[eax+1D]
 mov eax,dword ptr ds:[ebx+eax-1]
 lea eax,dword ptr ds:[ebx+eax]
 mov edx,dword ptr ds:[eax+78]
 test edx,edx
 je payload.bin.5F5C04
```

```
payload.bin.005F5B79
 lea edi,dword ptr ds:[ecx+2C]
 push 0
 push dword ptr ds:[edi+4]
 call payload.bin.5F11C0
 movd mm1,eax
 lea edx,dword ptr ds:[edx+ebx]
 mov eax,dword ptr ds:[edx+18]
 test eax,eax
 je payload.bin.5F5BFB
```

```
payload.bin.005F5B93
 mov dword ptr ss:[ebp-4],eax
 mov esi,dword ptr ds:[edx+20]
 mov edi,dword ptr ds:[edx+24]
 lea esi,dword ptr ds:[esi+ebx]
 lea edi,dword ptr ds:[edi+ebx]
```

```
payload.bin.005F5BA2
 movd eax,mm1
 push eax
 lodsd
 lea eax,dword ptr ds:[eax+ebx]
 push eax
 call <payload.bin.API NAME HASHER>
 cmp eax,dword ptr ss:[ebp+8]
 jne payload.bin.5F5BEF
```

```
payload.bin.005F5BEF
 add edi,2
 dec dword ptr ss:[ebp-4]
 cmp dword ptr ss:[ebp-4],0
 jne payload.bin.5F5BA2
```

```
payload.bin.005F5BB5
 movzx esi,word ptr ds:[edi]
 nop
 shl esi,2
 nop
 mov eax,dword ptr ds:[edx+1C]
 lea esi,dword ptr ds:[eax+esi]
 lea esi,dword ptr ds:[esi+ebx]
 mov eax,dword ptr ds:[esi]
 add esi,4
 lea eax,dword ptr ds:[eax+ebx]
 movd mm0,eax
 push eax
 lea eax,dword ptr ds:[5F5988]
 call eax
 test eax,eax
 je payload.bin.5F5BED
```

```
payload.bin.005F5BDE
 movd eax,mm0
 push eax
 lea eax,dword ptr ds:[5F5A7C]
 call eax
 movd mm0,eax
```

```
payload.bin.005F5BED
 jmp payload.bin.5F5BFB
```

```
payload.bin.005F5BFB
 movd eax,mm0
 test eax,eax
 je payload.bin.5F5C04
```

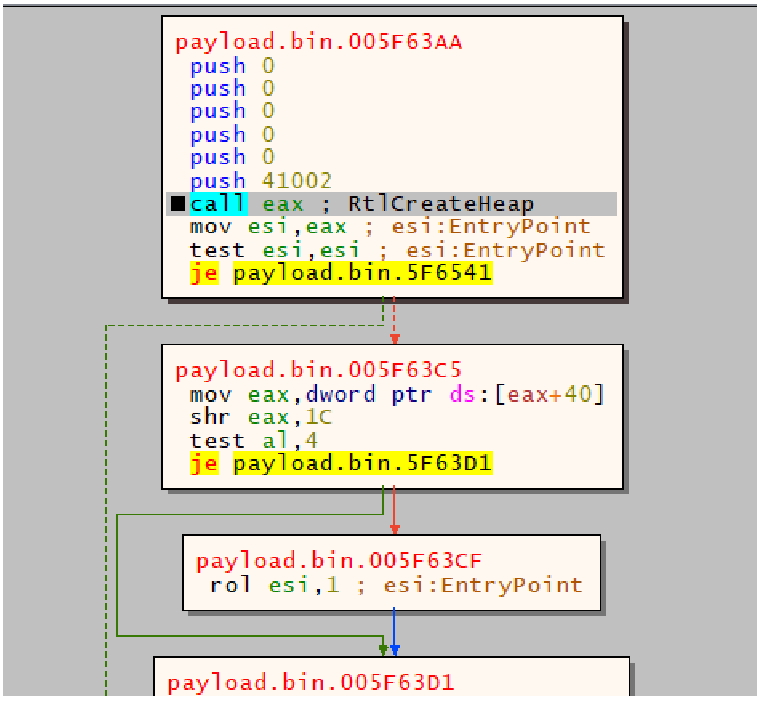
```
payload.bin.005F5C04
 mov ecx,dword ptr ds:[ecx]
 cmp dword ptr ss:[ebp-8],ecx
 jne payload.bin.5F5B5B
```

```
payload.bin.005F5C02
 jmp payload.bin.5F5C0F
```

```
payload.bin.005F5C0F
 movd eax,mm0
 pop edi
 pop esi
 pop ebx
 mov esp,ebp
 pop ebp
 ret 4
```

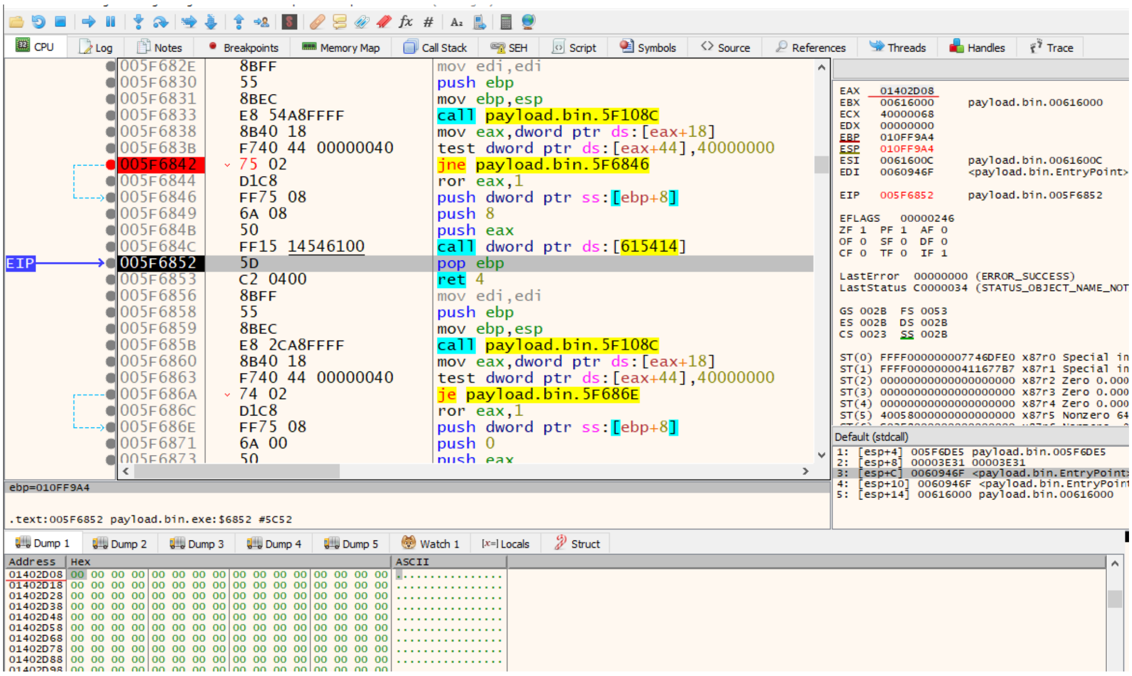
Figure 9 API Hashing Algorithm

It has additionally been identified that the malicious actor employs some anti-debugging techniques. These techniques take advantage of the differences in a heap structure in a debug state compared to its normal state.



```
payload.bin.005F63AA
 push 0
 push 0
 push 0
 push 0
 push 0
 push 41002
 call eax ; RtlCreateHeap
 mov esi,eax ; esi:EntryPoint
 test esi,esi ; esi:EntryPoint
 je payload.bin.5F6541

payload.bin.005F63C5
 mov eax,dword ptr ds:[eax+40]
 shr eax,1C
 test al,4
 je payload.bin.5F63D1

payload.bin.005F63CF
 rol esi,1 ; esi:EntryPoint

payload.bin.005F63D1
```

Figure 10 Anti-Debug: Heap Based



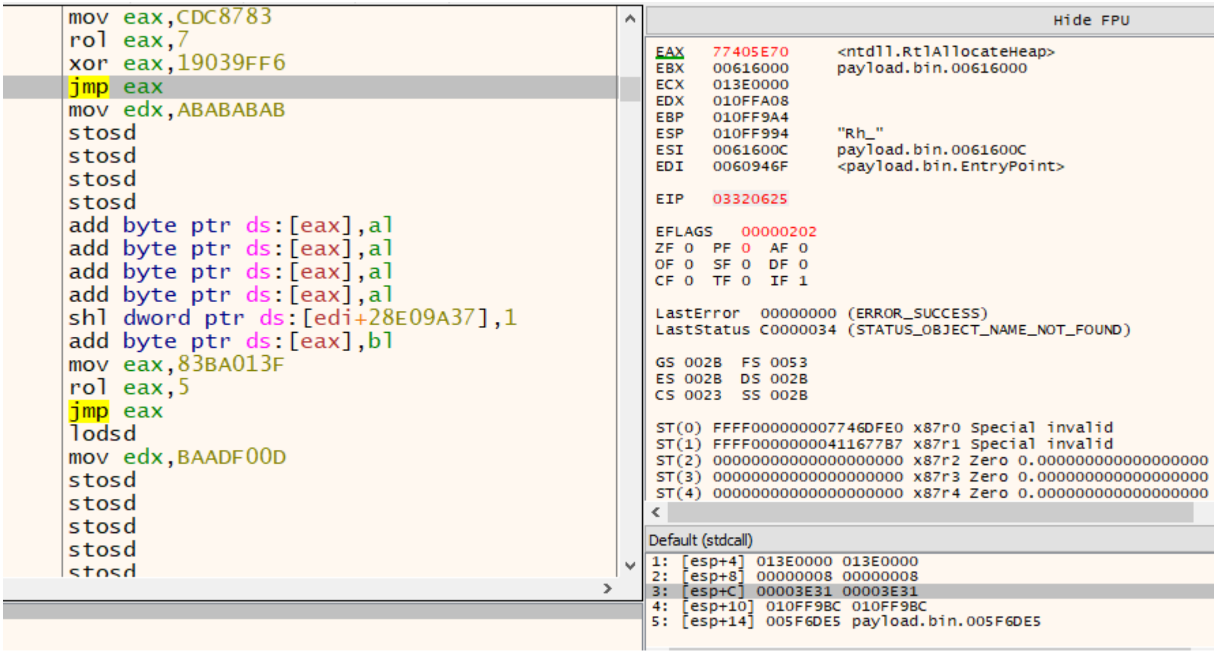
```
005F682E  8BFF              mov edi,edi
005F6830  55                push ebp
005F6831  8BEC              mov ebp,esp
005F6833  E8 54A8FFFF       call payload.bin.5F108C
005F6838  8B40 18           mov eax,dword ptr ds:[eax+18]
005F683B  F740 44 00000040  test dword ptr ds:[eax+44],40000000
005F6842  75 02             jne payload.bin.5F6846
005F6844  D1C8              ror eax,1
005F6846  FF75 08           push dword ptr ss:[ebp+8]
005F6849  6A 08             push 8
005F684B  50                push eax
005F684C  FF15 14546100     call dword ptr ds:[615414]
005F6852  5D                pop ebp
005F6853  C2 0400           ret 4
005F6856  8BFF              mov edi,edi
005F6858  55                push ebp
005F6859  8BEC              mov ebp,esp
005F685B  E8 2CA8FFFF       call payload.bin.5F108C
005F6860  8B40 18           mov eax,dword ptr ds:[eax+18]
005F6863  F740 44 00000040  test dword ptr ds:[eax+44],40000000
005F686A  74 02             je payload.bin.5F686E
005F686C  D1C8              ror eax,1
005F686E  FF75 08           push dword ptr ss:[ebp+8]
005F6871  6A 00             push 0
005F6873  50                push eax

ebp=010FF9A4

.text:005F6852 payload.bin.exe:$6852 #5C52
```

```
EAX   01402D08
EBX   00616000     payload.bin.00616000
ECX   40000068
EDX   00000000
EBP   010FF9A4
ESP   010FF9A4
ESI   0061600C     payload.bin.0061600C
EDI   0060946F     <payload.bin.EntryPoint>

EIP   005F6852     payload.bin.005F6852

EFLAGS   00000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special in
ST(1) FFFF00000000411677B7 x87r1 Special in
ST(2) 00000000000000000000 x87r2 Zero 0.000
ST(3) 00000000000000000000 x87r3 Zero 0.000
ST(4) 00000000000000000000 x87r4 Zero 0.000
ST(5) 40058000000000000000 x87r5 Nonzero 64

1: [esp+4] 005F6DE5 payload.bin.005F6DE5
2: [esp+8] 00003E31 00003E31
3: [esp+C] 0060946F <payload.bin.EntryPoint>
4: [esp+10] 0060946F <payload.bin.EntryPoint
5: [esp+14] 00616000 payload.bin.00616000
```

```
Address   Hex                                              ASCII
01402D08  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D18  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D28  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D38  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D48  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D58  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D68  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D78  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01402D88  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 11 Anti-Debug: Heap Based

Another **heap-based** anti-debugging technique has been identified. Figure 12 contains the code in a patched state.

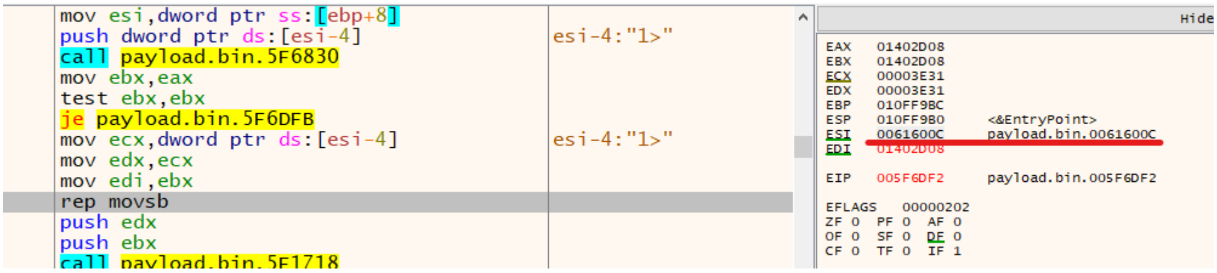


```
mov eax,CDC8783
rol eax,7
xor eax,19039FF6
jmp eax
mov edx,ABABABAB
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
shl dword ptr ds:[edi+28E09A37],1
add byte ptr ds:[eax],bl
mov eax,83BA013F
rol eax,5
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
stosd
stosd
```

```
Hide FPU

EAX   77405E70     <ntdll.RtlAllocateHeap>
EBX   00616000     payload.bin.00616000
ECX   013E0000
EDX   010FFA08
EBP   010FF9A4
ESP   010FF994     "Rh_"
ESI   0061600C     payload.bin.0061600C
EDI   0060946F     <payload.bin.EntryPoint>

EIP   03320625

EFLAGS    00000202
ZF 0  PF 0  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF0000000411677B7 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000

Default (stdcall)
1: [esp+4]  013E0000 013E0000
2: [esp+8]  00000008 00000008
3: [esp+C]  00003E31 00003E31
4: [esp+10] 010FF9BC 010FF9BC
5: [esp+14] 005F6DE5 payload.bin.005F6DE5
```

Figure 12 RtAllocateHeap: 15921 byte

It was observed that after bypassing the anti-debug techniques, **15,921** bytes of space were allocated.

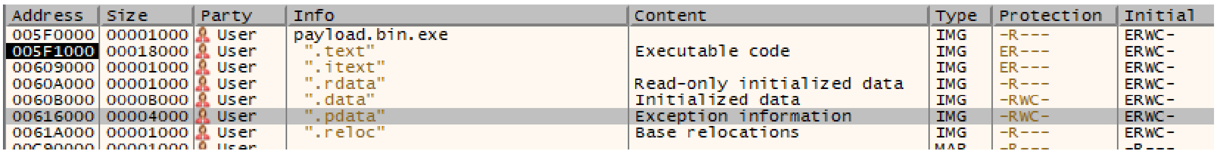


```
mov esi,dword ptr ss:[ebp+8]
push dword ptr ds:[esi-4]          esi-4:"1>"
call payload.bin.5F6830
mov ebx,eax
test ebx,ebx
je payload.bin.5F6DFB
mov ecx,dword ptr ds:[esi-4]       esi-4:"1>"
mov edx,ecx
mov edi,ebx
rep movsb
push edx
push ebx
call payload.bin.5F1718
```

```
Hide

EAX   01402D08
EBX   01402D08
ECX   00003E31
EDX   00003E31
EBP   010FF9BC
ESP   010FF9B0     <&EntryPoint>
ESI   0061600C     payload.bin.0061600C
EDI   01402D08

EIP   005F6DF2     payload.bin.005F6DF2

EFLAGS    00000202
ZF 0  PF 0  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1
```

Figure 13 Writing .pdata section

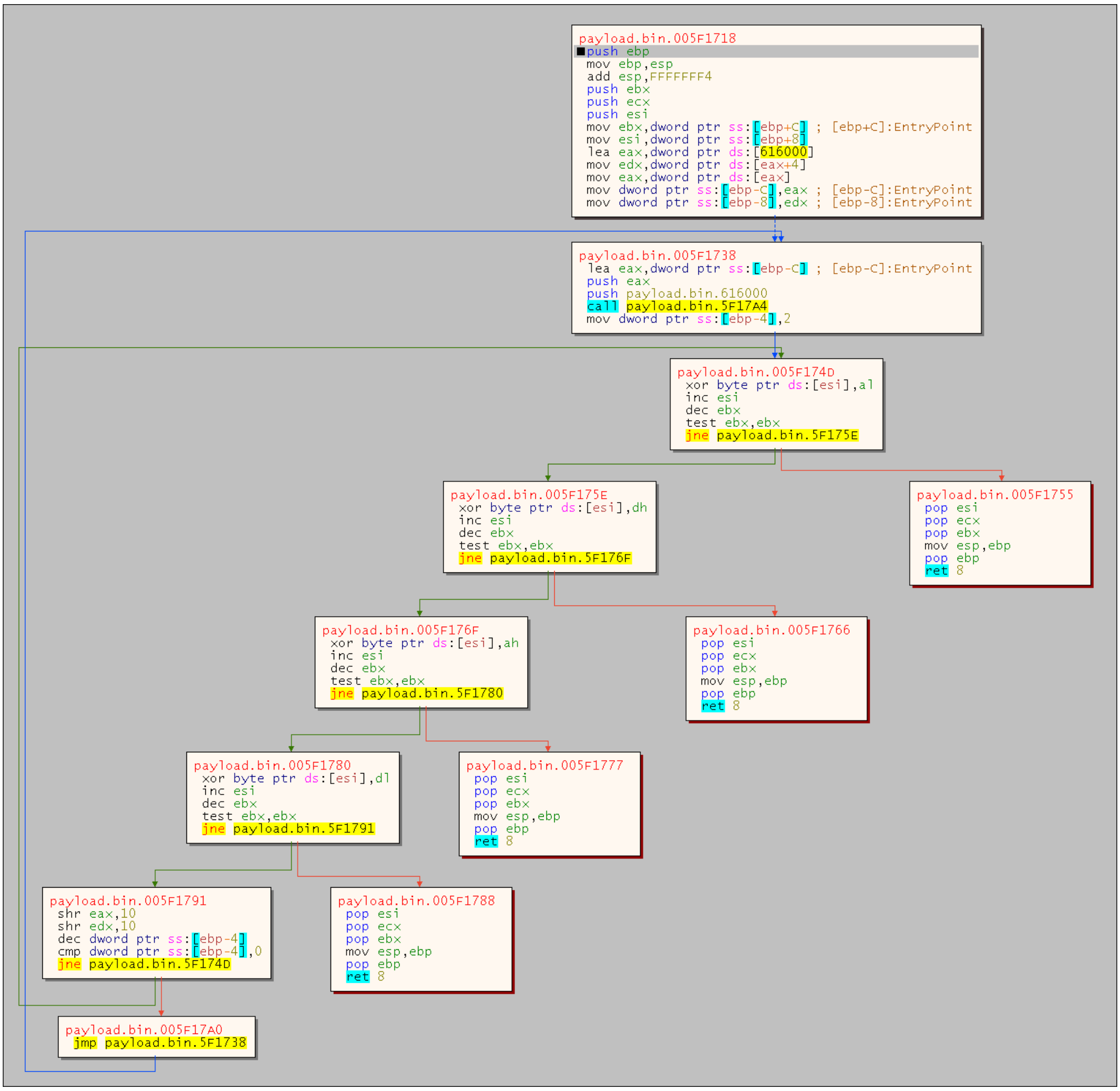
When the data to be written to the allocated space was monitored, it was determined that the starting address of the **.pdata** section is identified.



Address	Size	Party	Info	Content	Type	Protection	Initial
005F0000	00001000	User	payload.bin.exe		IMG	-R---	ERWC-
005F1000	00018000	User	".text"	Executable code	IMG	ER---	ERWC-
00609000	00001000	User	".itext"		IMG	ER---	ERWC-
0060A000	00001000	User	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
0060B000	0000B000	User	".data"	Initialized data	IMG	-RWC-	ERWC-
00616000	00004000	User	".pdata"	Exception information	IMG	-RWC-	ERWC-
0061A000	00001000	User	".reloc"	Base relocations	IMG	-R---	ERWC-
00C90000	00001000	User			MAP	-R---	-R---

Figure 14 .pdata section

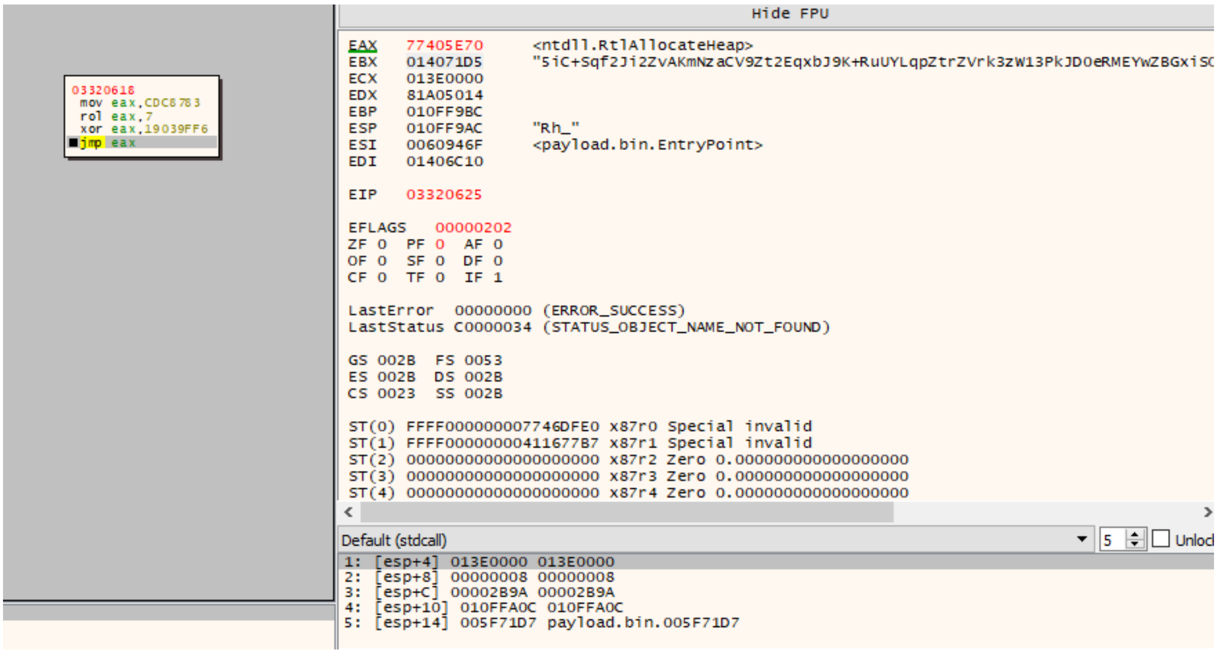
Upon examination of the mentioned section, it is evident that it is encrypted



```
payload.bin.005F1718
■push ebp
 mov ebp,esp
 add esp,FFFFFFF4
 push ebx
 push ecx
 push esi
 mov ebx,dword ptr ss:[ebp+C] ; [ebp+C]:EntryPoint
 mov esi,dword ptr ss:[ebp+8]
 lea eax,dword ptr ds:[616000]
 mov edx,dword ptr ds:[eax+4]
 mov eax,dword ptr ds:[eax]
 mov dword ptr ss:[ebp-C],eax ; [ebp-C]:EntryPoint
 mov dword ptr ss:[ebp-8],edx ; [ebp-8]:EntryPoint

payload.bin.005F1738
 lea eax,dword ptr ss:[ebp-C] ; [ebp-C]:EntryPoint
 push eax
 push payload.bin.616000
 call payload.bin.5F17A4
 mov dword ptr ss:[ebp-4],2

payload.bin.005F174D
 xor byte ptr ds:[esi],al
 inc esi
 dec ebx
 test ebx,ebx
 jne payload.bin.5F175E

payload.bin.005F175E
 xor byte ptr ds:[esi],dh
 inc esi
 dec ebx
 test ebx,ebx
 jne payload.bin.5F176F

payload.bin.005F1755
 pop esi
 pop ecx
 pop ebx
 mov esp,ebp
 pop ebp
 ret 8

payload.bin.005F176F
 xor byte ptr ds:[esi],ah
 inc esi
 dec ebx
 test ebx,ebx
 jne payload.bin.5F1780

payload.bin.005F1766
 pop esi
 pop ecx
 pop ebx
 mov esp,ebp
 pop ebp
 ret 8

payload.bin.005F1780
 xor byte ptr ds:[esi],dl
 inc esi
 dec ebx
 test ebx,ebx
 jne payload.bin.5F1791

payload.bin.005F1777
 pop esi
 pop ecx
 pop ebx
 mov esp,ebp
 pop ebp
 ret 8

payload.bin.005F1791
 shr eax,10
 shr edx,10
 dec dword ptr ss:[ebp-4]
 cmp dword ptr ss:[ebp-4],0
 jne payload.bin.5F174D

payload.bin.005F1788
 pop esi
 pop ecx
 pop ebx
 mov esp,ebp
 pop ebp
 ret 8

payload.bin.005F17A0
 jmp payload.bin.5F1738
```

Figure 15 Decryption of .pdata

The algorithm used to decrypt the data found in the .pdata section is as depicted in Figure 15.

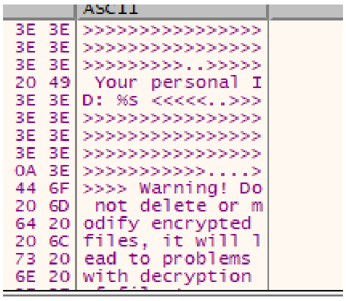


```
03320618
 mov eax,CDC8783
 rol eax,7
 xor eax,19039FF6
■jmp eax
```

```
Hide FPU

EAX    77405E70     <ntdll.RtlAllocateHeap>
EBX    014071D5     "5iC+Sqf2Ji2ZvAKmNzaCV9Zt2EqxbJ9K+RuUYLqpZtrZVrk3zW13PkJD0eRMEYwZBGxiSC
ECX    013E0000
EDX    81A05014
EBP    010FF9BC
ESP    010FF9AC     "Rh_"
ESI    0060946F     <payload.bin.EntryPoint>
EDI    01406C10

EIP    03320625

EFLAGS    00000202
ZF 0   PF 0   AF 0
OF 0   SF 0   DF 0
CF 0   TF 0   IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF00000000411677B7 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000

Default (stdcall)                                   5   □ Unloc
1: [esp+4] 013E0000 013E0000
2: [esp+8] 00000008 00000008
3: [esp+C] 00002B9A 00002B9A
4: [esp+10] 010FFA0C 010FFA0C
5: [esp+14] 005F71D7 payload.bin.005F71D7
```

Figure 16 RtlAllocateHeap

It has been determined that an 11KB space is allocated, and the decrypted data is written into the allocated space.

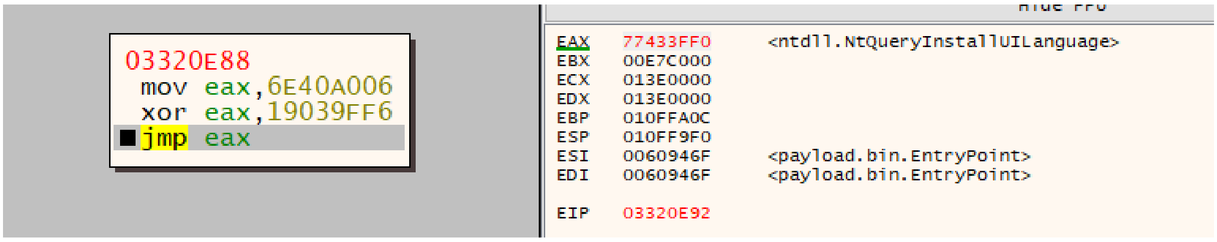


```
        ASCII
 3E 3E >>>>>>>>>>>>>>>>
 3E 3E >>>>>>>>>>>>>>>>
 3E 3E >>>>>>>>>..>>>>>
 20 49  Your personal I
 3E 3E D: %s <<<<<..>>>
 3E 3E >>>>>>>>>>>>>>>>
 3E 3E >>>>>>>>>>>>>>>>
 3E 3E >>>>>>>>>>>>>>>>
 0A 3E >>>>>>>>>>>....>
 44 6F >>>> Warning! Do
 20 6D  not delete or m
 64 20 odify encrypted
 20 6C files, it will l
 73 20 ead to problems
 6E 20 with decryption
```

Figure 17 README.txt Content Decryption

When the decrypted data was examined, it was observed that it is the content of the README.txt file to be created later. It has been determined that the malicious actor generates a unique ID for each computer.
VictimID struct: “BD23223ABCFA78BC”+<randomly_generated_16_character>

The generated VictimID is integrated into the content of the README.txt file.



```
03320E88
 mov eax,6E40A006
 xor eax,19039FF6
■jmp eax
```

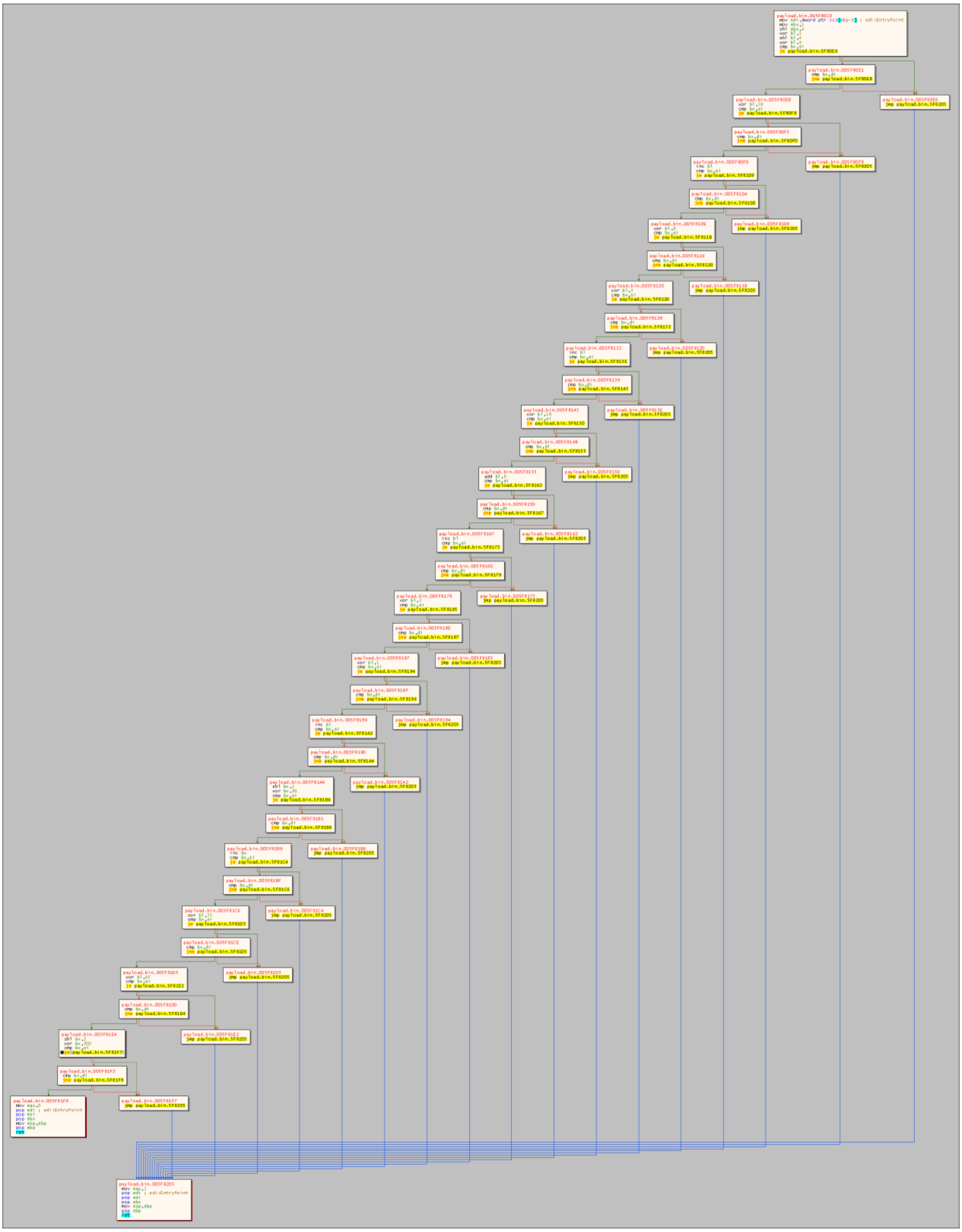
```
Hide FPU

EAX    77433FF0     <ntdll.NtQueryInstallUILanguage>
EBX    00E7C000
ECX    013E0000
EDX    013E0000
EBP    010FFA0C
ESP    010FF9F0
ESI    0060946F     <payload.bin.EntryPoint>
EDI    0060946F     <payload.bin.EntryPoint>

EIP    03320E92
```

Figure 18 NtQueryInstallUILanguage

It was observed that the language information used by the system is obtained.

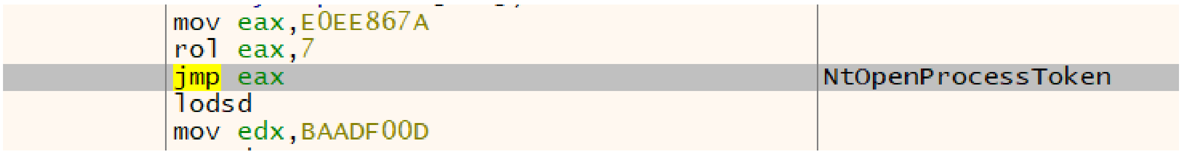
Countries Where LockBit 3.0 Family Does Not Operate:

- Ukraine
- Belarus
- Tajikistan
- Armenia
- Azerbaijan
- Georgia
- Kazakhstan
- Kyrgyzstan
- Turkmenistan
- Uzbekistan
- Tatarstan
- Romania
- Russia
- Moldova
- Saudi Arabia
- Syria



```
payload.bin.005F80C8
 mov edi,dword ptr ss:[ebp-8] ; edi:EntryPoint
 mov ebx,1
 shl ebx,A
 xor bl,1
 shl bl,4
 xor bl,9
 cmp bx,si
 je payload.bin.5F80E6

payload.bin.005F80E1
 cmp bx,di
 jne payload.bin.5F80EB

payload.bin.005F80EB
 xor bl,38
 cmp bx,si
 je payload.bin.5F80F8

payload.bin.005F80E6
 jmp payload.bin.5F8205

payload.bin.005F80F3
 cmp bx,di
 jne payload.bin.5F80FD

payload.bin.005F80FD
 inc bl
 cmp bx,si
 je payload.bin.5F8109

payload.bin.005F80F8
 jmp payload.bin.5F8205

payload.bin.005F8104
 cmp bx,di
 jne payload.bin.5F810E

payload.bin.005F810E
 xor bl,8
 cmp bx,si
 je payload.bin.5F811B

payload.bin.005F8109
 jmp payload.bin.5F8205

payload.bin.005F8116
 cmp bx,di
 jne payload.bin.5F8120

payload.bin.005F8120
 xor bl,3
 cmp bx,si
 je payload.bin.5F812D

payload.bin.005F811B
 jmp payload.bin.5F8205

payload.bin.005F8128
 cmp bx,di
 jne payload.bin.5F8132

payload.bin.005F8132
 inc bl
 cmp bx,si
 je payload.bin.5F813E

payload.bin.005F812D
 jmp payload.bin.5F8205

payload.bin.005F8139
 cmp bx,di
 jne payload.bin.5F8143

payload.bin.005F8143
 xor bl,10
 cmp bx,si
 je payload.bin.5F8150

payload.bin.005F813E
 jmp payload.bin.5F8205

payload.bin.005F814B
 cmp bx,di
 jne payload.bin.5F8155

payload.bin.005F8155
 add bl,8
 cmp bx,si
 je payload.bin.5F8162

payload.bin.005F8150
 jmp payload.bin.5F8205

payload.bin.005F815D
 cmp bx,di
 jne payload.bin.5F8167

payload.bin.005F8167
 inc bl
 cmp bx,si
 je payload.bin.5F8173

payload.bin.005F8162
 jmp payload.bin.5F8205

payload.bin.005F816E
 cmp bx,di
 jne payload.bin.5F8178

payload.bin.005F8178
 xor bl,2
 cmp bx,si
 je payload.bin.5F8185

payload.bin.005F8173
 jmp payload.bin.5F8205

payload.bin.005F8180
 cmp bx,di
 jne payload.bin.5F8187

payload.bin.005F8187
 xor bl,1
 cmp bx,si
 je payload.bin.5F8194

payload.bin.005F8185
 jmp payload.bin.5F8205

payload.bin.005F818F
 cmp bx,di
 jne payload.bin.5F8196

payload.bin.005F8196
 inc bl
 cmp bx,si
 je payload.bin.5F81A2

payload.bin.005F8194
 jmp payload.bin.5F8205

payload.bin.005F819D
 cmp bx,di
 jne payload.bin.5F81A4

payload.bin.005F81A4
 shl bx,1
 xor bx,90
 cmp bx,si
 je payload.bin.5F81B6

payload.bin.005F81A2
 jmp payload.bin.5F8205

payload.bin.005F81B1
 cmp bx,di
 jne payload.bin.5F81B8

payload.bin.005F81B8
 inc bx
 cmp bx,si
 je payload.bin.5F81C4

payload.bin.005F81B6
 jmp payload.bin.5F8205

payload.bin.005F81BF
 cmp bx,di
 jne payload.bin.5F81C6

payload.bin.005F81C6
 xor bl,35
 cmp bx,si
 je payload.bin.5F81D3

payload.bin.005F81C4
 jmp payload.bin.5F8205

payload.bin.005F81CE
 cmp bx,di
 jne payload.bin.5F81D5

payload.bin.005F81D5
 xor bl,6F
 cmp bx,si
 je payload.bin.5F81E2

payload.bin.005F81D3
 jmp payload.bin.5F8205

payload.bin.005F81DD
 cmp bx,di
 jne payload.bin.5F81E4

payload.bin.005F81E4
 shl bx,2
 xor bx,90D
 cmp bx,si
 je payload.bin.5F81F7

payload.bin.005F81E2
 jmp payload.bin.5F8205

payload.bin.005F81F2
 cmp bx,di
 jne payload.bin.5F81F9

payload.bin.005F81F9
 mov eax,0
 pop edi ; edi:EntryPoint
 pop esi
 pop ebx
 mov esp,ebp
 pop ebp
 ret

payload.bin.005F81F7
 jmp payload.bin.5F8205

payload.bin.005F8205
 mov eax,1
 pop edi ; edi:EntryPoint
 pop esi
 pop ebx
 mov esp,ebp
 pop ebp
 ret
```

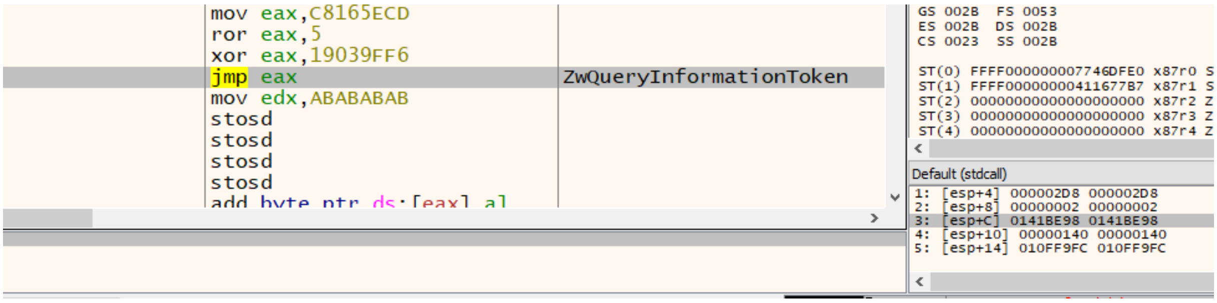
Figure 19 Country Checking



```
mov eax,E0EE867A
rol eax,7
jmp eax                                    NtOpenProcessToken
lodsd
mov edx,BAADF00D
```

Figure 20 NtOpenProcessToken

It has been determined that the Access token handle specific to the process is obtained.



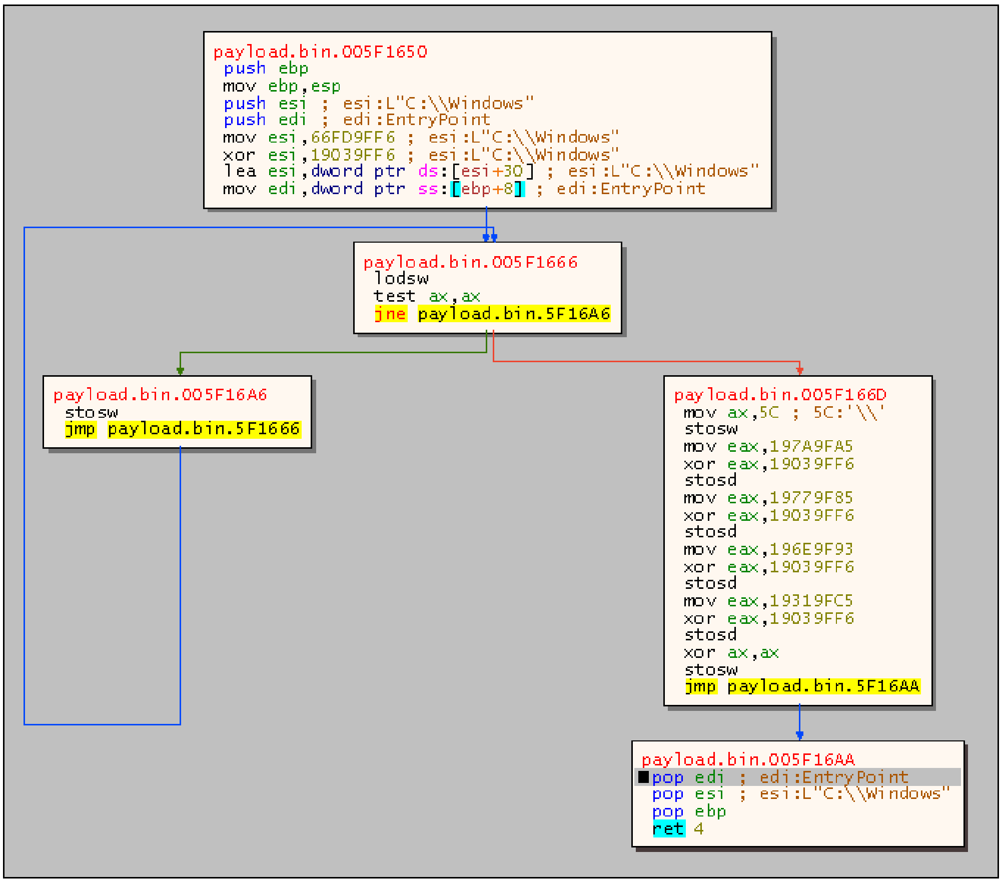
```
mov eax,C8165ECD
ror eax,5
xor eax,19039FF6
jmp eax                    ZwQueryInformationToken
mov edx,ABABABAB
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
```

```
GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 S
ST(1) FFFF00000000411677B7 x87r1 S
ST(2) 00000000000000000000 x87r2 Z
ST(3) 00000000000000000000 x87r3 Z
ST(4) 00000000000000000000 x87r4 Z
```

```
Default (stdcall)
1: [esp+4] 000002D8 000002D8
2: [esp+8] 00000002 00000002
3: [esp+C] 0141BE98 0141BE98
4: [esp+10] 00000140 00000140
5: [esp+14] 010FF9FC 010FF9FC
```

Figure 21 ZwqueryInformationToken

It has been observed that the user group information is retrieved using the Access token structure in which the process is running.



```
payload.bin.005F1650
 push ebp
 mov ebp,esp
 push esi  ; esi:L"C:\\Windows"
 push edi  ; edi:EntryPoint
 mov esi,66FD9FF6 ; esi:L"C:\\Windows"
 xor esi,19039FF6 ; esi:L"C:\\Windows"
 lea esi,dword ptr ds:[esi+30] ; esi:L"C:\\Windows"
 mov edi,dword ptr ss:[ebp+8] ; edi:EntryPoint
```

```
payload.bin.005F1666
 lodsw
 test ax,ax
 jne payload.bin.5F16A6
```

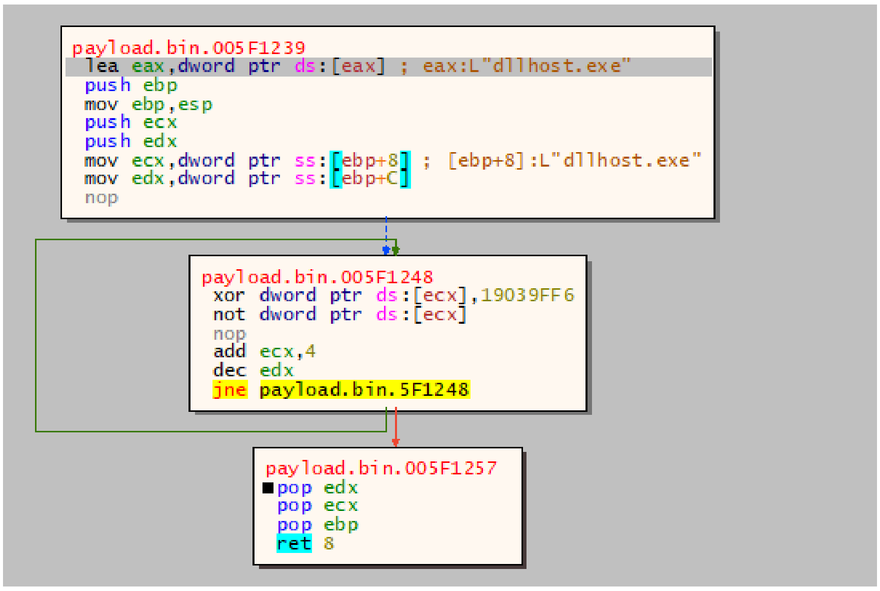
```
payload.bin.005F16A6
 stosw
 jmp payload.bin.5F1666
```

```
payload.bin.005F166D
 mov ax,5C ; 5C:'\\'
 stosw
 mov eax,197A9FA5
 xor eax,19039FF6
 stosd
 mov eax,19779F85
 xor eax,19039FF6
 stosd
 mov eax,196E9F93
 xor eax,19039FF6
 stosd
 mov eax,19319FC5
 xor eax,19039FF6
 stosd
 xor ax,ax
 stosw
 jmp payload.bin.5F16AA
```

```
payload.bin.005F16AA
 pop edi  ; edi:EntryPoint
 pop esi  ; esi:L"C:\\Windows"
 pop ebp
 ret 4
```

Figure 22 Creation System32 Path

In Figure 22, the expression **"C:\Windows\System32"** is being constructed. It is observed that XOR method is used to evade security products.



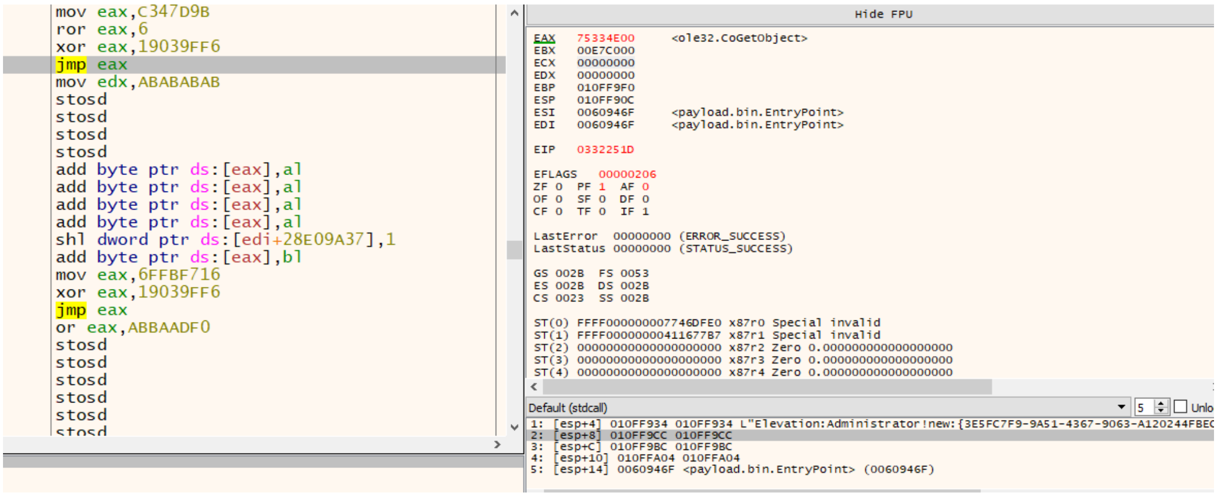
```
payload.bin.005F1239
 lea eax,dword ptr ds:[eax] ; eax:L"dllhost.exe"
 push ebp
 mov ebp,esp
 push ecx
 push edx
 mov ecx,dword ptr ss:[ebp+8] ; [ebp+8]:L"dllhost.exe"
 mov edx,dword ptr ss:[ebp+C]
 nop

payload.bin.005F1248
 xor dword ptr ds:[ecx],19039FF6
 not dword ptr ds:[ecx]
 nop
 add ecx,4
 dec edx
 jne payload.bin.5F1248

payload.bin.005F1257
 pop edx
 pop ecx
 pop ebp
 ret 8
```

Figure 23 Decryption Algorithm

The expressions decrypted with the algorithm shown in Figure 23 are as follows:

- dllhost.exe
- Elevation:Administrator!new:{{3E5FC7F9-9A51-4367-9063-A120244FBEC7}}

```
DWORD* decryption_function(DWORD *arry,size_t size) {

       for (int i = 0; i < size; i++) {

              arry[i] = arry[i] ^ 0x19039ff6;
              arry[i] = ~(arry[i]);

              std::cout << std::hex<< arry[i]<<" ";

       }
       return arry;
}
```



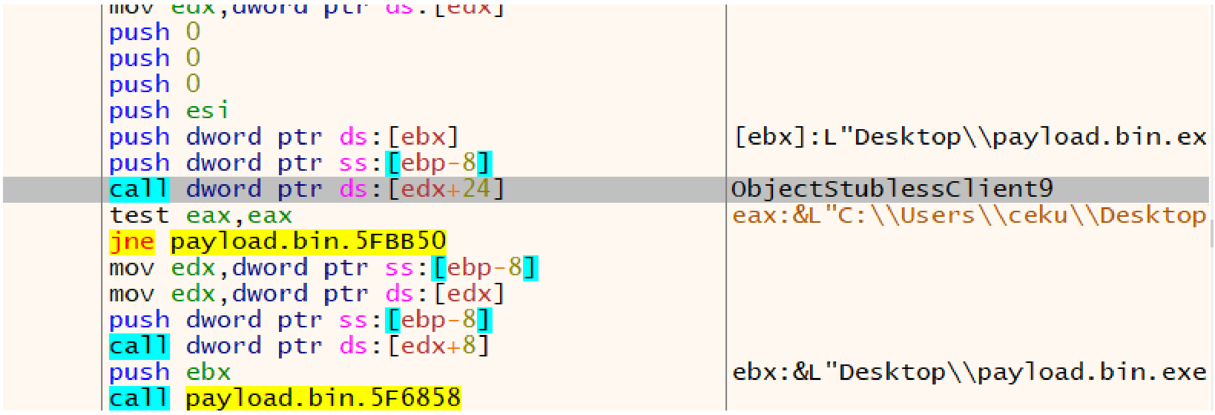
```
mov eax,C347D9B
ror eax,6
xor eax,19039FF6
jmp eax
mov edx,ABABABAB
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
shl dword ptr ds:[edi+28E09A37],1
add byte ptr ds:[eax],bl
mov eax,6FFBF716
xor eax,19039FF6
jmp eax
or eax,ABBAADF0
stosd
stosd
stosd
stosd
stosd
stosd
```

```
EAX    75334E00     <ole32.CoGetObject>
EBX    00E7C000
ECX    00000000
EDX    00000000
EBP    010FF9F0
ESP    010FF90C
ESI    0060946F     <payload.bin.EntryPoint>
EDI    0060946F     <payload.bin.EntryPoint>

EIP    0332251D

EFLAGS    00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF00000000411677B7 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000
```

Default (stdcall) 5 □ Unlo

```
1: [esp+4] 010FF934 010FF934 L"Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC
2: [esp+8] 010FF9CC 010FF9CC
3: [esp+C] 010FF9BC 010FF9BC
4: [esp+10] 010FFA04 010FFA04
5: [esp+14] 0060946F <payload.bin.EntryPoint> (0060946F)
```

Figure 24 CoGetObject



```
mov edx,dword ptr ds:[edx]
push 0
push 0
push 0
push esi
push dword ptr ds:[ebx]              [ebx]:L"Desktop\\payload.bin.ex
push dword ptr ss:[ebp-8]
call dword ptr ds:[edx+24]           ObjectStublessClient9
test eax,eax                         eax:&L"C:\\Users\\ceku\\Desktop
jne payload.bin.5FBB50
mov edx,dword ptr ss:[ebp-8]
mov edx,dword ptr ds:[edx]
push dword ptr ss:[ebp-8]
call dword ptr ds:[edx+8]
push ebx                             ebx:&L"Desktop\\payload.bin.exe
call payload.bin.5F6858
```

Figure 25 ObjectStublessClient9

It has been determined that it operates as a child process under dllhost.exe by bypassing UAC.

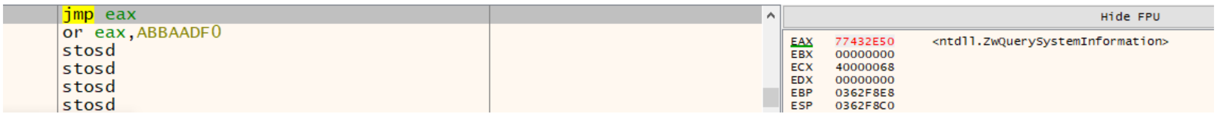
It has been observed that a portion of the previously decrypted data is hashed using the MD5 method.

The data hashed with MD5 is as follows:

```
3a2223bd  bc78fabc  bb04ea7f  3286dcc7
0860c7ff  c03edf06  3e570a04  9c55aaee
6a051e98  96cd73c8  d17595fc  a1ad958b
fa52e8cc  8b65411c  587767a2  fd5a5db2
809964a8  0cf2a551  be0e3392  1b07e687
ec4c1f53  605e4e11  293dbfd1  5540bb91
b186938a  9c496dae  d13d64ea  a6577138
4b9adcb3  c985d873  11100549  5892daf0
```

Following another decryption process, the resulting expression is as follows:
"{{%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X%02X}}"

The MD5 hash is integrated into the specified format in the above text.

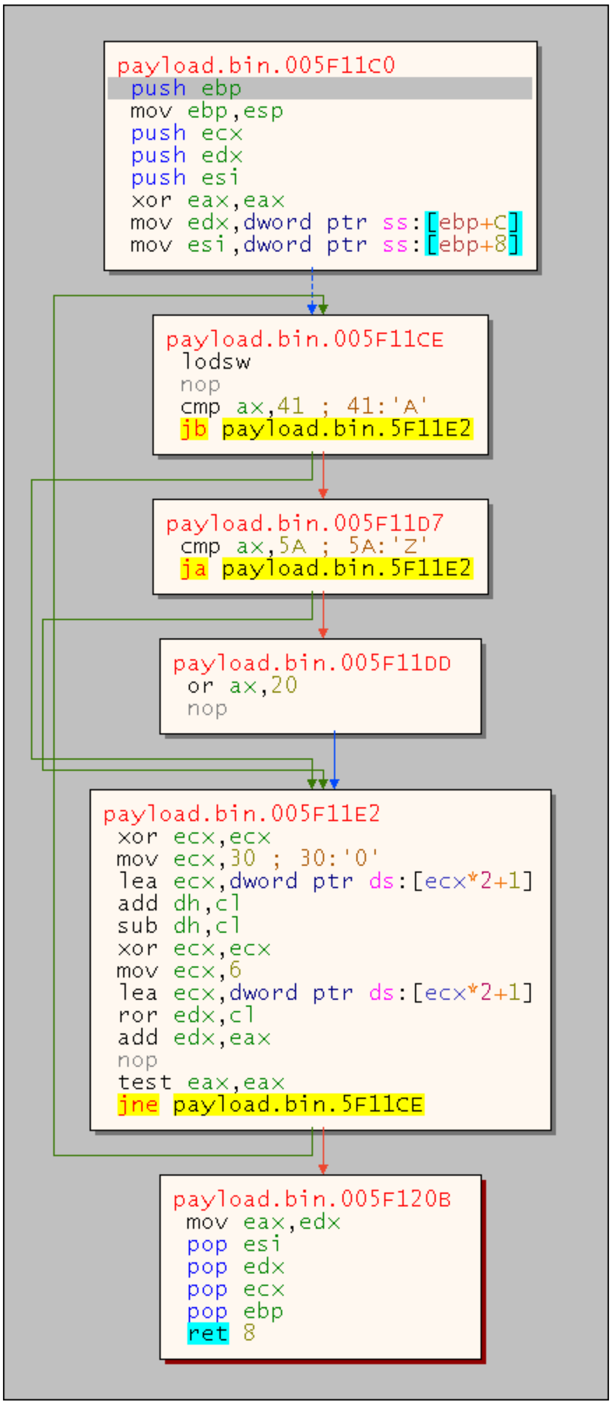


```
jmp eax
or eax,ABBAADF0
stosd
stosd
stosd
stosd
```

```
                                   Hide FPU
EAX   77432E50     <ntdll.ZwQuerySystemInformation>
EBX   00000000
ECX   40000068
EDX   00000000
EBP   0362F8E8
ESP   0362F8C0
```

Figure 26 ZwQuerySystemInformation

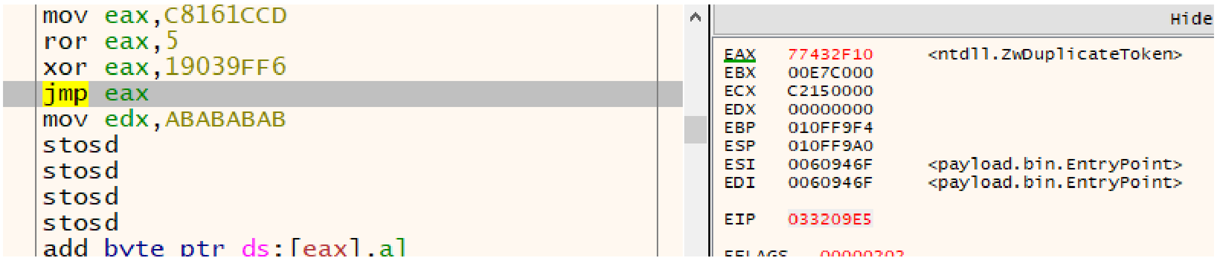
It has been determined that information about the processes running on the system is retrieved



```
payload.bin.005F11C0
 push ebp
 mov ebp,esp
 push ecx
 push edx
 push esi
 xor eax,eax
 mov edx,dword ptr ss:[ebp+C]
 mov esi,dword ptr ss:[ebp+8]

payload.bin.005F11CE
 lodsw
 nop
 cmp ax,41 ; 41:'A'
 jb payload.bin.5F11E2

payload.bin.005F11D7
 cmp ax,5A ; 5A:'Z'
 ja payload.bin.5F11E2

payload.bin.005F11DD
 or ax,20
 nop

payload.bin.005F11E2
 xor ecx,ecx
 mov ecx,30 ; 30:'0'
 lea ecx,dword ptr ds:[ecx*2+1]
 add dh,cl
 sub dh,cl
 xor ecx,ecx
 mov ecx,6
 lea ecx,dword ptr ds:[ecx*2+1]
 ror edx,cl
 add edx,eax
 nop
 test eax,eax
 jne payload.bin.5F11CE

payload.bin.005F120B
 mov eax,edx
 pop esi
 pop edx
 pop ecx
 pop ebp
 ret 8
```

Figure 27 Process Name Hashing Algorithm

It has been observed that, similar to navigating through DLL files, hashes are generated for process names and compared with the hash of a desired process name. It was determined that the target process name is **"explorer.exe"**.

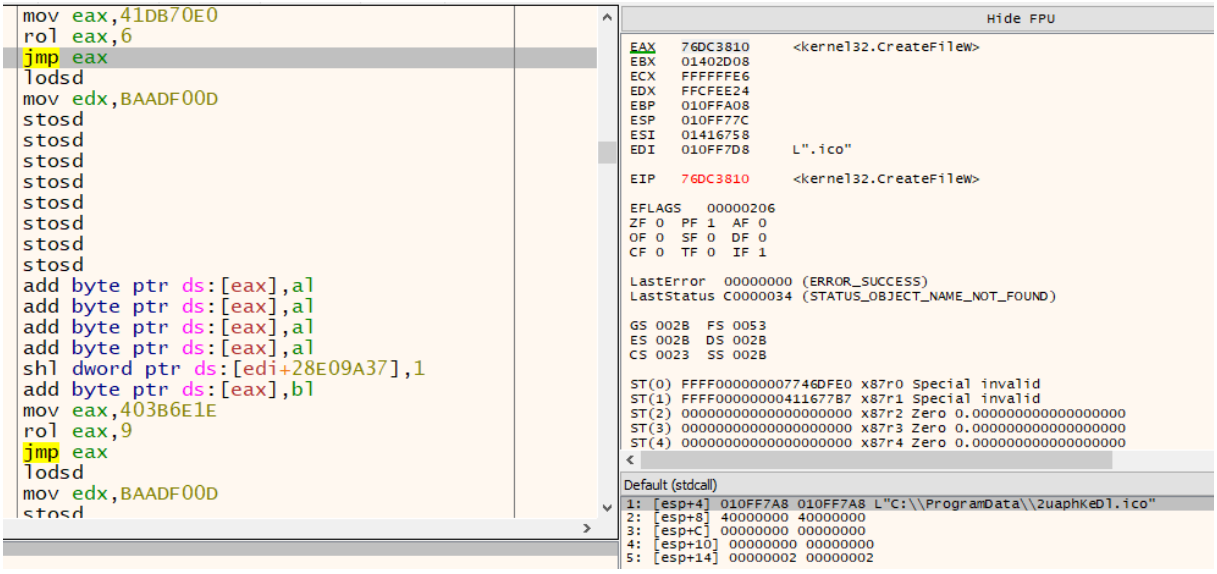


```
mov eax,C8161CCD
ror eax,5
xor eax,19039FF6
jmp eax
mov edx,ABABABAB
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
```

```
EAX   77432F10     <ntdll.ZwDuplicateToken>
EBX   00E7C000
ECX   C2150000
EDX   00000000
EBP   010FF9F4
ESP   010FF9A0
ESI   0060946F     <payload.bin.EntryPoint>
EDI   0060946F     <payload.bin.EntryPoint>

EIP   033209E5

EFLAGS   00000202
```

Figure 28 DuplicateToken

It was determined that the process access token information for the **"explorer.exe"** process was copied



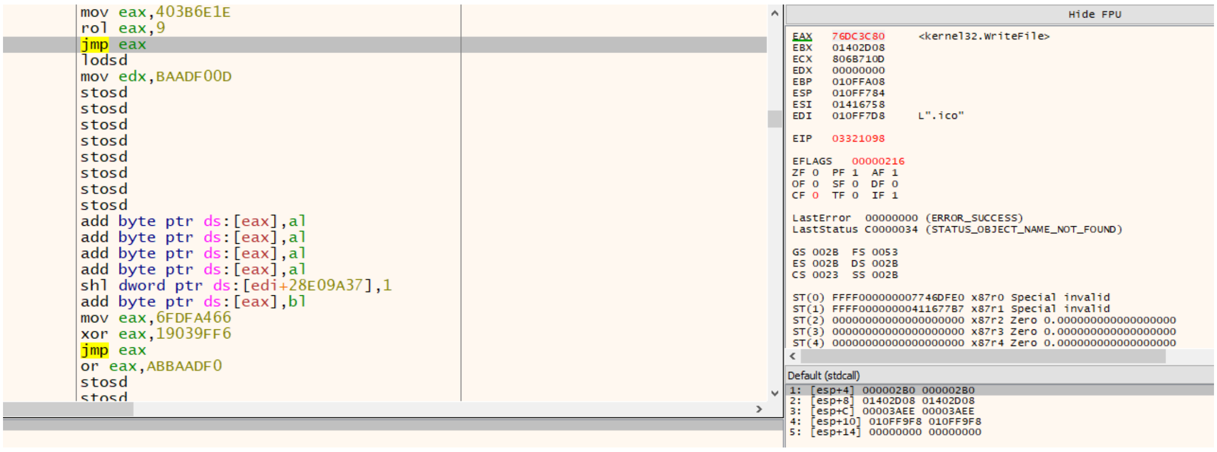
```
mov eax,41DB70E0
rol eax,6
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
stosd
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
shl dword ptr ds:[edi+28E09A37],1
add byte ptr ds:[eax],bl
mov eax,403B6E1E
rol eax,9
jmp eax
lodsd
mov edx,BAADF00D
stosd
```

```
EAX   76DC3810     <kernel32.CreateFileW>
EBX   01402D08
ECX   FFFFFFE6
EDX   FFCFEE24
EBP   010FFA08
ESP   010FF77C
ESI   01416758
EDI   010FF7D8     L".ico"

EIP   76DC3810     <kernel32.CreateFileW>

EFLAGS   00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF0000000041167787 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000
```

```
Default (stdcall)
1: [esp+4] 010FF7A8 010FF7A8 L"C:\\ProgramData\\2uaphKeDl.ico"
2: [esp+8] 40000000 40000000
3: [esp+C] 00000000 00000000
4: [esp+10] 00000000 00000000
5: [esp+14] 00000002 00000002
```

Figure 29 CreateFile: Creation LockBit Icon File

It was determined that the **"C:\\ProgramData\\2uaphKeDl.ico"** file was created.



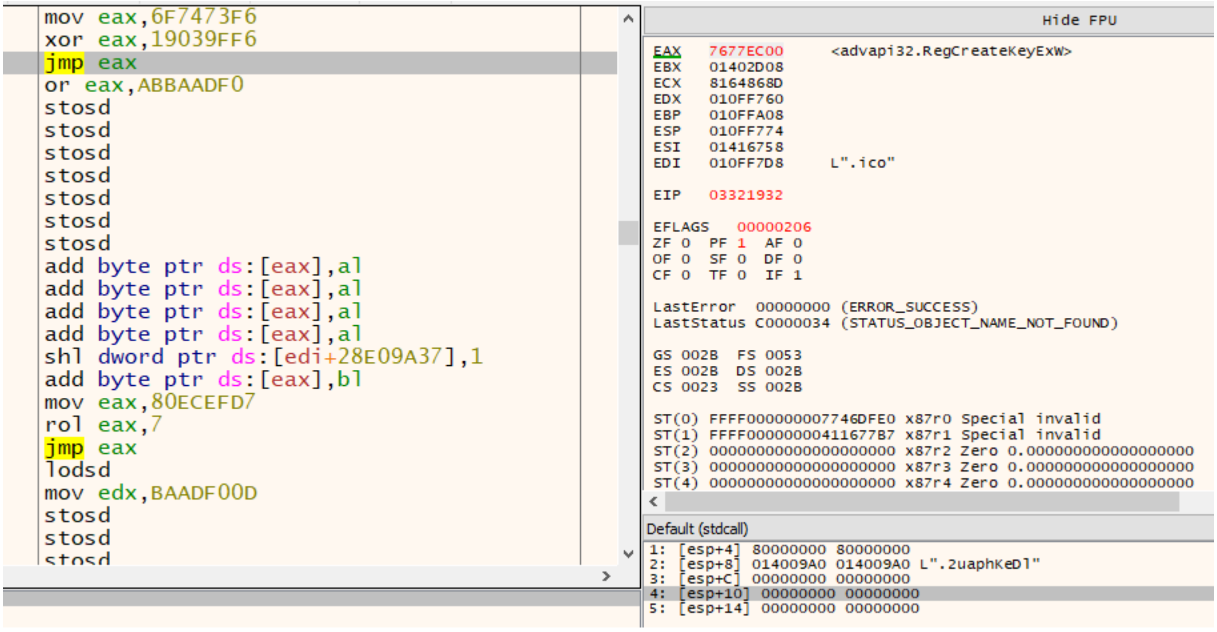
```
mov eax,403B6E1E
rol eax,9
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
stosd
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
shl dword ptr ds:[edi+28E09A37],1
add byte ptr ds:[eax],bl
mov eax,6FDFA466
xor eax,19039FF6
jmp eax
or eax,ABBAADF0
stosd
stosd
```

```
EAX   76DC3C80     <kernel32.WriteFile>
EBX   01402D08
ECX   806B710D
EDX   00000000
EBP   010FFA08
ESP   010FF784
ESI   01416758
EDI   010FF7D8     L".ico"

EIP   03321098

EFLAGS   00000216
ZF 0  PF 1  AF 1
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF00000000411677B7 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000
```

```
Default (stdcall)
1: [esp+4] 000002B0 000002B0
2: [esp+8] 01402D08 01402D08
3: [esp+C] 00003AEE 00003AEE
4: [esp+10] 010FF9F8 010FF9F8
5: [esp+14] 00000000 00000000
```

Figure 30 WriteFile: LockBit Icon

It was determined that the content of the famous LockBit file icon was being written.

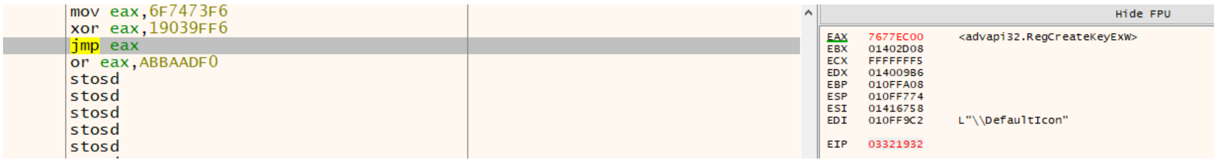


```
mov eax,6F7473F6
xor eax,19039FF6
jmp eax
or eax,ABBAADF0
stosd
stosd
stosd
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
shl dword ptr ds:[edi+28E09A37],1
add byte ptr ds:[eax],bl
mov eax,80ECEFD7
rol eax,7
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
```

```
EAX    7677EC00     <advapi32.RegCreateKeyExW>
EBX    01402D08
ECX    8164868D
EDX    010FF760
EBP    010FFA08
ESP    010FF774
ESI    01416758
EDI    010FF7D8     L".ico"

EIP    03321932

EFLAGS   00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) FFFF000000007746DFE0 x87r0 Special invalid
ST(1) FFFF0000000411677B7 x87r1 Special invalid
ST(2) 00000000000000000000 x87r2 Zero 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Zero 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000

Default (stdcall)
1: [esp+4] 80000000 80000000
2: [esp+8] 014009A0 014009A0 L".2uaphKeDl"
3: [esp+C] 00000000 00000000
4: [esp+10] 00000000 00000000
5: [esp+14] 00000000 00000000
```

Figure 31 RegCreateKeyExW

It was determined that a subkey named **".2uaphKeDl"** was created under the **"HKEY_CLASSES_ROOT"** key.

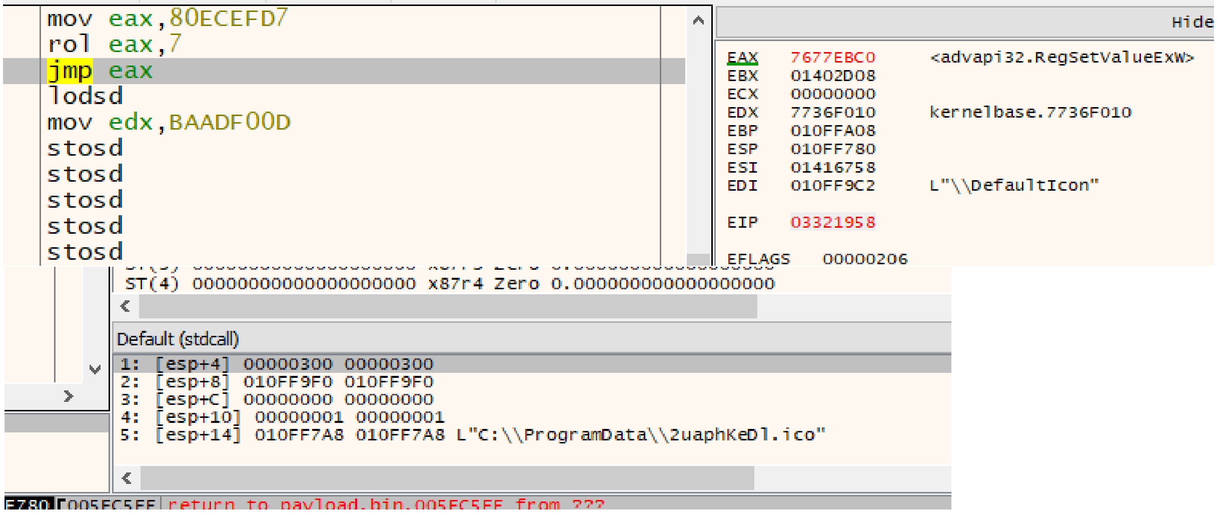


```
mov eax,6F7473F6
xor eax,19039FF6
jmp eax
or eax,ABBAADF0
stosd
stosd
stosd
stosd
stosd
```

```
EAX    7677EC00     <advapi32.RegCreateKeyExW>
EBX    01402D08
ECX    FFFFFFF5
EDX    014009B6
EBP    010FFA08
ESP    010FF774
ESI    01416758
EDI    010FF9C2     L"\\DefaultIcon"

EIP    03321932
```

Figure 32 RegCreateKeyExW: DefaultIcon

It was determined that a subkey named **"DefaultIcon"** was also opened under the **".2uaphKeDl"** subkey**.**



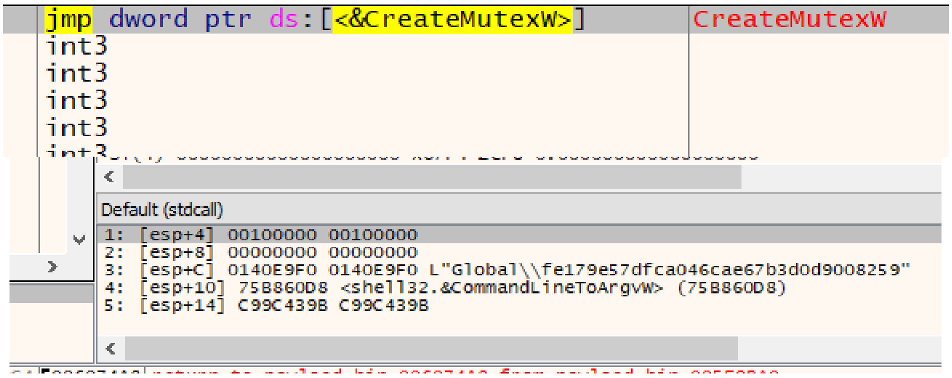
```
mov eax,80ECEFD7
rol eax,7
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
stosd
stosd
```

```
EAX    7677EBC0     <advapi32.RegSetValueExW>
EBX    01402D08
ECX    00000000
EDX    7736F010     kernelbase.7736F010
EBP    010FFA08
ESP    010FF780
ESI    01416758
EDI    010FF9C2     L"\\DefaultIcon"

EIP    03321958

EFLAGS   00000206
```

```
ST(4) 00000000000000000000 x87r4 Zero 0.000000000000000000

Default (stdcall)
1: [esp+4] 00000300 00000300
2: [esp+8] 010FF9F0 010FF9F0
3: [esp+C] 00000000 00000000
4: [esp+10] 00000001 00000001
5: [esp+14] 010FF7A8 010FF7A8 L"C:\\ProgramData\\2uaphKeDl.ico"

F780 [005FC5FF return to payload.bin.005FC5FF from ???
```

Figure 33 RegSetValueExW: Setting Icon File Path

The value contained in the created subkey specifies the directory of the created icon file.

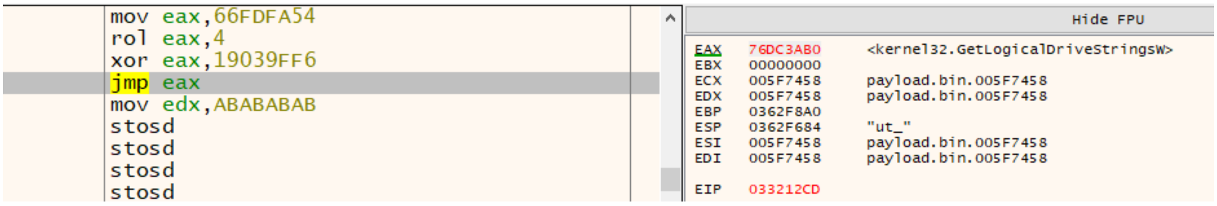


```
jmp dword ptr ds:[<&CreateMutexW>]        CreateMutexW
int3
int3
int3
int3
```

```
Default (stdcall)
1: [esp+4] 00100000 00100000
2: [esp+8] 00000000 00000000
3: [esp+C] 0140E9F0 0140E9F0 L"Global\\fe179e57dfca046cae67b3d0d9008259"
4: [esp+10] 75B860D8 <shell32.&CommandLineToArgvW> (75B860D8)
5: [esp+14] C99C439B C99C439B
```

Figure 34 CreateMutex

It was determined that a Mutex named **"Global\\fe179e57dfca046cae67b3d0d9008259"** was created.

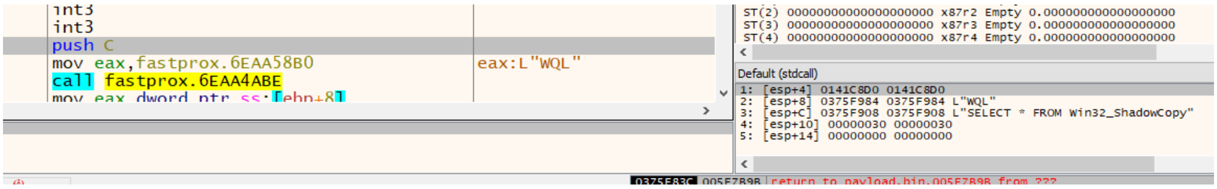


```
mov eax,66FDFA54
rol eax,4
xor eax,19039FF6
jmp eax
mov edx,ABABABAB
stosd
stosd
stosd
stosd
```

```
Hide FPU
EAX   76DC3AB0     <kernel32.GetLogicalDriveStringsW>
EBX   00000000
ECX   005F7458     payload.bin.005F7458
EDX   005F7458     payload.bin.005F7458
EBP   0362F8A0
ESP   0362F684     "ut_"
ESI   005F7458     payload.bin.005F7458
EDI   005F7458     payload.bin.005F7458

EIP   033212CD
```

Figure 35 GetLogicalDriveStringsW

Directory information of the drivers on the device is being retrieved. The retrieved drivers are being checked for whether they are a storage unit, such as a hard disk.



```
int3
int3
push C
mov eax,fastprox.6EAA58B0          eax:L"WQL"
call fastprox.6EAA4ABE
```

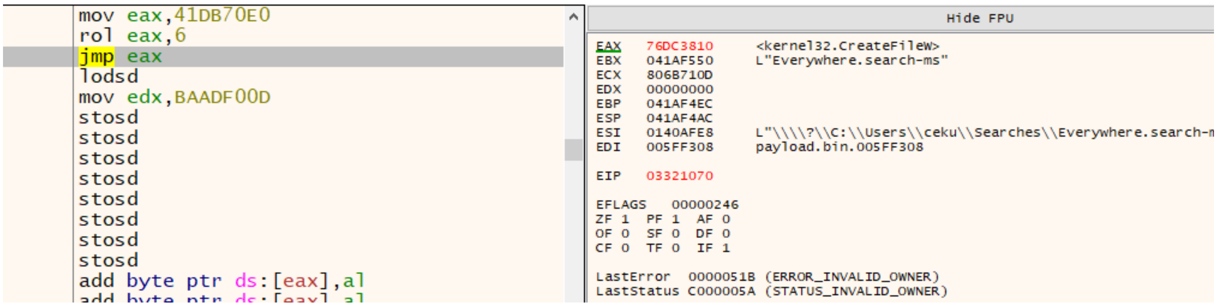
```
ST(2) 00000000000000000000 x87r2 Empty 0.000000000000000000
ST(3) 00000000000000000000 x87r3 Empty 0.000000000000000000
ST(4) 00000000000000000000 x87r4 Empty 0.000000000000000000

Default (stdcall)
1: [esp+4] 0141C8D0 0141C8D0
2: [esp+8] 0375F984 0375F984 L"WQL"
3: [esp+C] 0375F908 0375F908 L"SELECT * FROM Win32_ShadowCopy"
4: [esp+10] 00000030 00000030
5: [esp+14] 00000000 00000000
```

Figure 36 Delete Shadow Copies

It was determined that a WMI query is executed to collect Shadow copies.

Subsequently, it encrypts files by traversing directories, especially those under the **"C:\"** directory.

Unlike traditional ransomware, it has been observed that it traverses directories and encrypts files using a single thread. The main reasons for its speed despite using a single thread are as follows: checking the importance of the traversed directories and using custom encryption functions instead of ready-made functions while encrypting files.

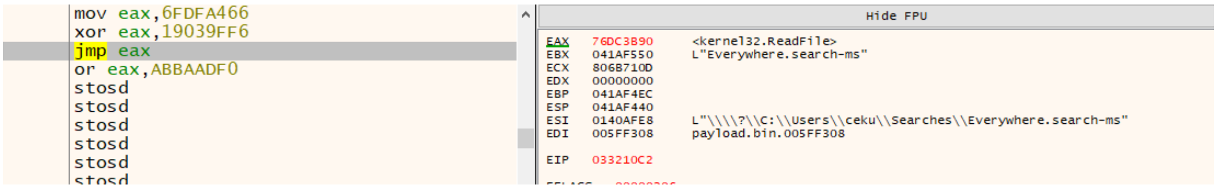


```
mov eax,41DB70E0
rol eax,6
jmp eax
lodsd
mov edx,BAADF00D
stosd
stosd
stosd
stosd
stosd
stosd
stosd
stosd
add byte ptr ds:[eax],al
add byte ptr ds:[eax],al
```

```
Hide FPU

EAX   76DC3810     <kernel32.CreateFileW>
EBX   041AF550     L"Everywhere.search-ms"
ECX   806B710D
EDX   00000000
EBP   041AF4EC
ESP   041AF4AC
ESI   0140AFE8     L"\\\\?\\C:\\Users\\ceku\\Searches\\Everywhere.search-m
EDI   005FF308     payload.bin.005FF308

EIP   03321070

EFLAGS   00000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  0000051B (ERROR_INVALID_OWNER)
LastStatus C000005A (STATUS_INVALID_OWNER)
```

Figure 37 Opening File that will Encrypt

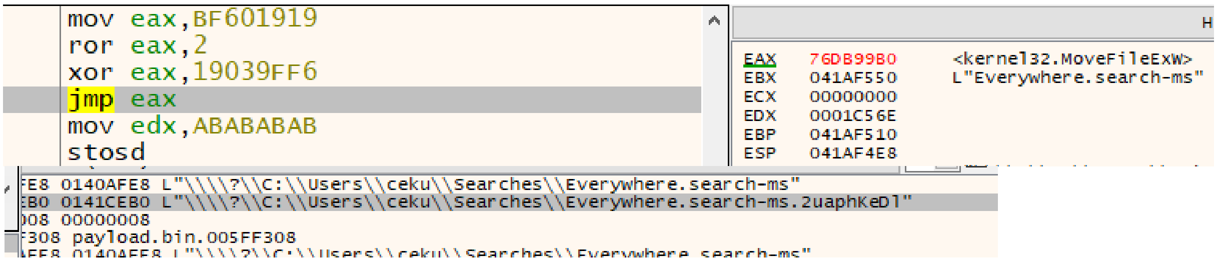
The opening of the detected file,



```
mov eax,6FDFA466
xor eax,19039FF6
jmp eax
or eax,ABBAADF0
stosd
stosd
stosd
stosd
stosd
stosd
```

```
Hide FPU

EAX   76DC3B90     <kernel32.ReadFile>
EBX   041AF550     L"Everywhere.search-ms"
ECX   806B710D
EDX   00000000
EBP   041AF4EC
ESP   041AF440
ESI   0140AFE8     L"\\\\?\\C:\\Users\\ceku\\Searches\\Everywhere.search-ms"
EDI   005FF308     payload.bin.005FF308

EIP   033210C2
```

The reading,



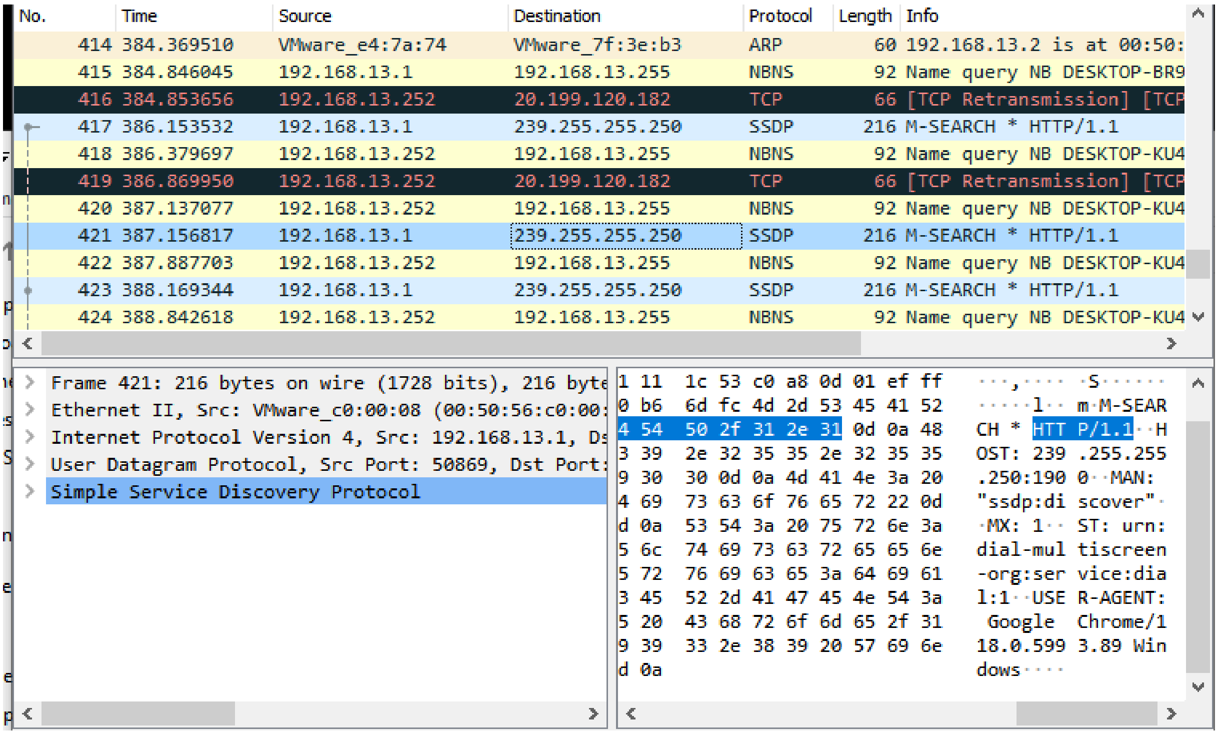
```
mov eax,BF601919
ror eax,2
xor eax,19039FF6
jmp eax
mov edx,ABABABAB
stosd
```

```
EAX   76DB99B0     <kernel32.MoveFileExW>
EBX   041AF550     L"Everywhere.search-ms"
ECX   00000000
EDX   0001C56E
EBP   041AF510
ESP   041AF4E8
```

```
FE8 0140AFE8 L"\\\\?\\C:\\Users\\ceku\\Searches\\Everywhere.search-ms"
EB0 0141CEB0 L"\\\\?\\C:\\Users\\ceku\\Searches\\Everywhere.search-ms.2uaphKeDl"
008 00000008
F308 payload.bin.005FF308
FE8 0140AFE8 L"\\\\?\\C:\\Users\\ceku\\Searches\\Everywhere.search-ms"
```

Figure 38 MoveFileExW

A copy is created with a different extension. Then, the newly created file is opened, and its content is encrypted and overwritten on the same file again.



```
No.  Time        Source           Destination        Protocol Length Info
 414 384.369510 VMware_e4:7a:74  VMware_7f:3e:b3    ARP          60 192.168.13.2 is at 00:50:
 415 384.846045 192.168.13.1     192.168.13.255     NBNS         92 Name query NB DESKTOP-BR9
 416 384.853656 192.168.13.252   20.199.120.182     TCP          66 [TCP Retransmission] [TCP
 417 386.153532 192.168.13.1     239.255.255.250    SSDP        216 M-SEARCH * HTTP/1.1
 418 386.379697 192.168.13.252   192.168.13.255     NBNS         92 Name query NB DESKTOP-KU4
 419 386.869950 192.168.13.252   20.199.120.182     TCP          66 [TCP Retransmission] [TCP
 420 387.137077 192.168.13.252   192.168.13.255     NBNS         92 Name query NB DESKTOP-KU4
 421 387.156817 192.168.13.1     239.255.255.250    SSDP        216 M-SEARCH * HTTP/1.1
 422 387.887703 192.168.13.252   192.168.13.255     NBNS         92 Name query NB DESKTOP-KU4
 423 388.169344 192.168.13.1     239.255.255.250    SSDP        216 M-SEARCH * HTTP/1.1
 424 388.842618 192.168.13.252   192.168.13.255     NBNS         92 Name query NB DESKTOP-KU4
```

```
> Frame 421: 216 bytes on wire (1728 bits), 216 byte
> Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:
> Internet Protocol Version 4, Src: 192.168.13.1, Ds
> User Datagram Protocol, Src Port: 50869, Dst Port:
> Simple Service Discovery Protocol
```

```
1 11  1c 53 c0 a8 0d 01 ef ff   ···,···· ·S······
0 b6  6d fc 4d 2d 53 45 41 52   ·····l·· m·M-SEAR
4 54  50 2f 31 2e 31 0d 0a 48   CH * HTT P/1.1··H
3 39  2e 32 35 35 2e 32 35 35   OST: 239 .255.255
9 30  30 0d 0a 4d 41 4e 3a 20   .250:190 0··MAN:
4 69  73 63 6f 76 65 72 22 0d   "ssdp:di scover"·
d 0a  53 54 3a 20 75 72 6e 3a   ·MX: 1·· ST: urn:
5 6c  74 69 73 63 72 65 65 6e   dial-mul tiscreen
5 72  76 69 63 65 3a 64 69 61   -org:ser vice:dia
3 45  52 2d 41 47 45 4e 54 3a   l:1··USE R-AGENT:
5 20  43 68 72 6f 6d 65 2f 31    Google  Chrome/1
9 39  33 2e 38 39 20 57 69 6e   18.0.599 3.89 Win
d 0a                            dows····
```

As a result of the investigations, some IP information that could be associated with the malicious software has been identified. These are:

- 239.255.255.250
- 224.0.0.252

Rules

YARA

```
rule LockBit_3_0{

meta:
    date = "2023-10-26"
    description = "Detects LockBit 3.0"
    author = "Bilal BAKARTEPE - EchoCTI Malware Team"
    hash = "bbe63d8efc8d8dc7f387b08ee07721ba"
    verdict = "dangerous"
    platform = "windows"

strings:
    $hash1={2D D8 63 77} //ntdll RtlAllocateHeap
    $hash2={54 31 19 c3} //FindFirstFile
    $hash3={23 56 69 4e} //FindNextFile
    $hash4={8a a5 43 61} //FindClose
    $hash5={f6 9f 03 19} //MD4Init

    $xorkey={f6 9f 03 19} //xor key for hashed API's

    $opc1={55 8B EC 51 52 56 33 C0 8B 55 0C 8B 75 08 AC 33 C9 B9 30 00 00 00 8D 0C 4D 01 00 00 00 02
F1 2A F1 33 C9 B9 06 00 00 00 8D 0C 4D 01 00 00 00 D3 CA 03 D0 90 85 C0 75 D6 8B C2 5E 5A 59 5D} //API
name hasher algorithm
    $opc2={55 8B EC 56 57 BE F6 9F FD 66 81 F6 F6 9F 03 19 8D 76 30 8B 7D 08 66 AD 66 85 C0 75 39 66
B8 5C 00  66 AB B8 A5 9F 7A 19 35 F6 9F 03 19 AB B8 85 9F 77 19  35 F6 9F 03 19 AB  B8 93 9F 6E 19  35
F6 9F 03 19  AB B8 C5 9F 31 19  35 F6 9F 03 19 AB 66 33 C0 66 AB EB 04 66 AB EB BC 5F 5E 5D C2 04 00}
//deobfuscating "C:\\windows\\system32" string
    $opc3={C7 03 55 60 D6 E6 C7 43 04 27 60 98 E6 C7 43 08 65 60 90 E6 C7 43 0C 09 60 FC
E6}//deobfuscating "*.dll" string
    $opc4={55 8B EC 51 52 8B 4D 08 8B 55 0C 90 81 31 F6 9F 03 19 F7 11 90 83 C1 04 4A 75 F1 5A 59 5D}
//deobfuscating "*.dll" string together
    $opc5={66 83 F8 41 72 0B 66 83 F8 5A 77 05 66 83 C8 20  90 33 C9 B9 30 00 00 00 8D 0C 4D 01 00 00
00 02 F1 2A F1 33 C9 B9 06 00 00 00 8D 0C 4D 01 00 00 00 D3 CA 03 D0 90 85 C0 75 C3} //Dll name
hashing
    $opc6={8B 40 18 F7 40 44 00 00 00 40 74 02 D1 C8}//Heap-based Anti-debug
    $opc7={B9 5D 34 A8 B2 81 F1 F6 9F 03 19 39 48 10 74 01 AB C6 00 B8}//Heap-based Anti-debug

condition:
    any of ($opc*) or (any of ($hash*)and $xorkey)
}
```

SIGMA – 1

```
title: LockBit 3.0 Registry Operation
status: experimental
description: Detects LockBit 3.0 icon file definition
author: Bilal BAKARTEPE
date: 2023/10/26
logsource:
  category: registry_set
  product: windows
detection:
  selection:
    CommandLine|contains|all:
    - HKEY_CLASSES_ROOT
    - .2uaphKeDl
    TargetObject|endswith: reg.exe
  condition: selection
falsepositives:
- Unknown
level: high
```

SIGMA - 2

```
title: Win32_ShadowCopy Query Alert
description: Detects a query for Win32_ShadowCopy class in WMI.
author: Bilal BAKARTEPE
date: 2023-10-26
logsource:
  product: windows
  service: security
detection:
  selection:
    EventID: 10  # Event ID for WMI Queries (Adjust this if needed)
    Query: "*FROM Win32_ShadowCopy*"
  condition: selection
level: high
tags:
  - wmi
  - windows
  - alert
falsepositives:
  - Legitimate use of WMI for querying Win32_ShadowCopy
fields:
  - Query
  - EventID
  - ComputerName
  - User
  - ProcessName
  - ParentProcessName
  - ParentProcessID
  - CommandLine
```

MITRE ATT&CK Table

Tactic	ID	Technic Name
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control
Privilege Escalation	T1134	Access Token Manipulation
Discovery	T1083	File and Directory Discovery
Discovery	T1069.002	Permission Groups Discovery: Domain Groups
Discovery	T1082	System Information Discovery
Execution	T1047	Windows Management Instrumentation

ECHO

CYBER THREAT INTELLIGENCE