

**SHA256:** eea0083ac04f8bfb3042acd615cabd7be77dee410c8db229b5980379b224e93f

Hancitor is typically delivered through phishing emails containing malicious attachments. These attachments are often weaponized Microsoft Office documents that have macros embedded within them.

Spread through emails, Hancitor tells users to enable macros on the document which allows it to connect to its C2 server to download its second stage loader.

## Analysis

We unpack the zip file and analyze the contents on TrID, DiE, and PeStudio to gather information through an initial investigation on the contents of the binary.

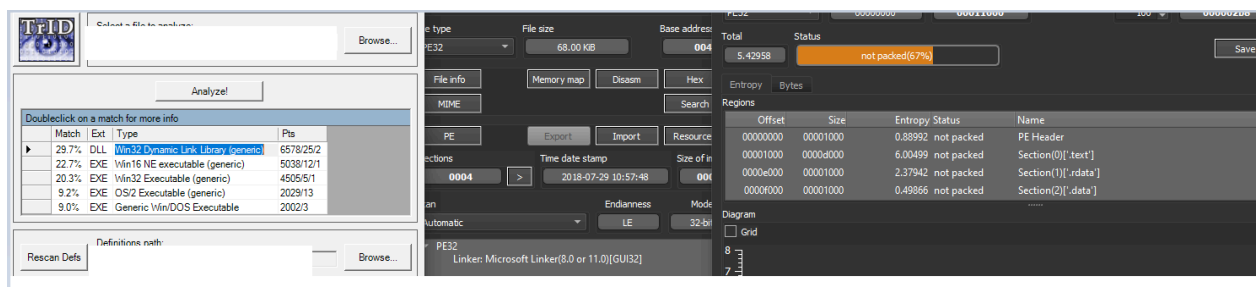


Fig 1. Initial Scan through TrID and DiE.

Looking at the results on Fig 1. It shows that they have detected this as a 32-bit DLL and has a relatively low entropy suggesting that there is a chance that this is neither packer nor encrypted.

indicators (imports > flag)	footprint > sha256	B67CA342FD93FFB3DF03BC426BAB685F07FA685D77D79A05DD6A87401412963B
footprints (count > 12)	size	0xF8 (248 bytes)
virustotal (status > error)	entropy	5.181
dos-header (size > 64 bytes)	file-ratio	0.36 %
dos-stub (size > 248 bytes)	first-bytes-hex	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20...
rich-header (tooling > Visual Studio 2005)	first-bytes-text	.....!...L..!This program cannot be run in DOS ...
file-header (executable > 32-bit)	message	!This program cannot be run in DOS mode.
optional-header (subsystem > GUI)		
directories (count > 3)		
sections (count > 4)		
libraries (KERNEL32.dll)		
imports (flag > 10)		
exports (n/a)		
thread-local-storage (n/a)		
.NET (n/a)		
resources (count > 4)		
strings (count > 1723)		
debug (n/a)		
manifest (n/a)		
version (FileDescription > Java(TM) Platform !		
certificate (n/a)		
overlay (n/a)		

Fig 2. Screenshot of PeStudio Findings.

indicators (imports > flag)	footprint > sha256	CC13635CEA00435B955C8EA52D844D80E2A3CA855F7DB036B2EE74E432DB4752
footprints (count > 12)	location	.rsrsc:0x00010484
virustotal (status > error)	file-type	executable
dos-header (size > 64 bytes)	language	neutral
dos-stub (size > 248 bytes)	code-page	Unicode UTF-16, little endian
rich-header (tooling > Visual Studio 2005)	CompanyName	Sun Microsystems, Inc.
file-header (executable > 32-bit)	FileDescription	Java(TM) Platform SE binary
optional-header (subsystem > GUI)	FileVersion	6.0.200.2
directories (count > 3)	Full Version	1.6.0_20-b02
sections (count > 4)	InternalName	Java(TM) Plug-In Launcher
libraries (KERNEL32.dll)	LegalCopyright	Copyright © 2004
imports (flag > 10)	OriginalFilename	jp2launcher.exe
exports (n/a)	ProductName	Java(TM) Platform SE 6 U20
thread-local-storage (n/a)	ProductVersion	6.0.200.2
.NET (n/a)		
resources (count > 4)		
strings (count > 1723)		
debug (n/a)		
manifest (n/a)		
version (FileDescription > Java(TM) Platform !		
certificate (n/a)		
overlay (n/a)		

Fig 3. Versions Tab on PEStudio

Analyzing the file through PEStudio shows a few flags such as the low import count. It is also possible to look at the available strings but it does not show much as the data is possibly obfuscated or encrypted.

Running the file on Virustotal shows that this is already a well documented piece of malware and anti-viruses should not have any struggles identifying this.

61 / 74

61/74 security vendors and 2 sandboxes flagged this file as malicious

Reanalyze Similar More

eea0083ac04f8bfb3042acd615cabd7be77dee410c8db229b5980379b224e93f

Size 68.00 KB Last Modification Date 4 minutes ago

EXE

Community Score

peexe runtime-modules malware direct-cpu-clock-access long-sleeps checks-network-adapters detect-debug-environment cve-2016-2569 exploit

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 3

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

Popular threat label trojan.geral/hancitor Threat categories trojan downloader Family labels geral hancitor razy

Security vendors' analysis Do you want to automate checks?

Vendor	Detection	Family
AhnLab-V3	Malware/Win32.Generic.C2909815	TrojanDownloader.Win32/Geral.7cc7d37f
AliCloud	Trojan[downloader].Win/Hancitor.J	Trojan.Agent.Geral
Antiy-AVL	Trojan[Downloader].Win32.Geral	Trojan.Ser.Razy.D2C13
Avast	Win32:Trojan-gen	Win32:Trojan-gen
Avira (no cloud)	TR/AD.DInject.tlctg	Gen:Variant.Ser.Razy.11283
BitDefenderTheta	Gen:NN.ZexaF.36808.eq0@ayX0jLbi	W32.Common.A02C0D0C
CrowdStrike Falcon	Win/malicious_confidence_90% (D)	Malicious.7538ab
Cylance	Unsafe	Malicious (score: 99)

Fig 4. Virustotal Scan

- **Dynamic Analysis:** The next section shows the dynamic analysis on x32dbg and ProcessHacker to monitor if any files are created.

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Si
Software	76B5F660	<kernel32.dll.VirtualAlloc>	Enabled	mov edi,edi	7	
	76B60760	<kernel32.dll.VirtualProtect>	Enabled	mov edi,edi	1	
	76B62370	<kernel32.dll.IsDebuggerPresent>	Enabled	jmp dword ptr ds:[<IsDebuggerPresent>]	0	
	76B72D0F	<kernel32.dll.CreateProcessInternal>	Enabled	mov edi,edi	0	

Fig 5. Breakpoints Set

Following from Fig 5. Only VirtualAlloc and VirtualProtect were hit on the four breakpoints that were set Following this on the debugger will allow us to unpack the second stage shown below.

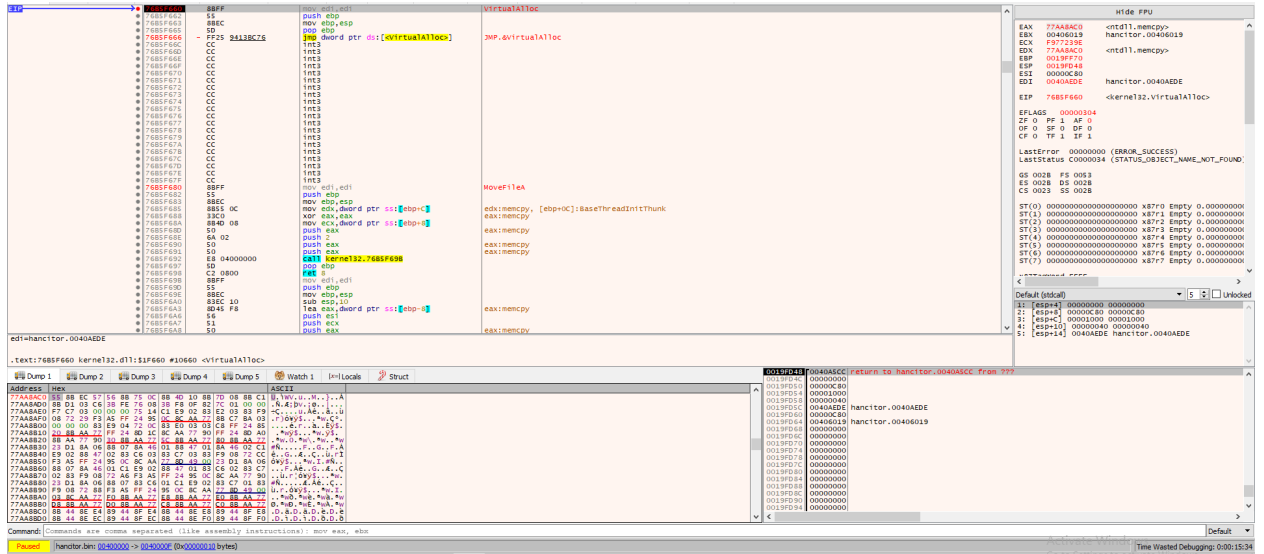


Fig 6. First VirtualAlloc

Hitting the first VirtualAlloc and following EAX to the return shows a bunch of allocated data on Dump1.

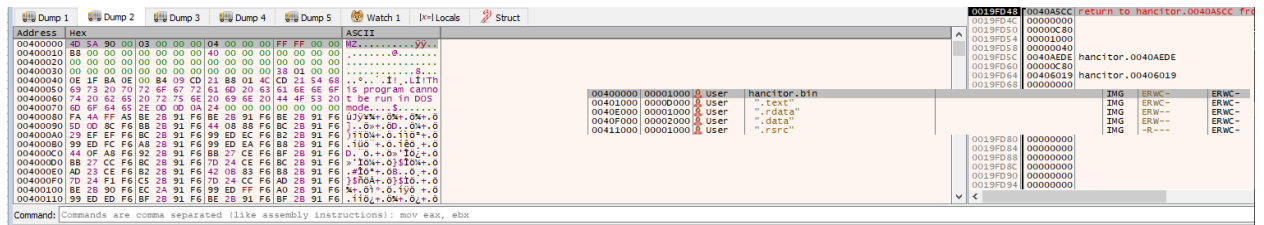


Fig 7. MZ on Dump2

Viewing Dump2 actually shows that it contains an MZ header but following that in memory it is probably just the original entrypoint for Hancitor.

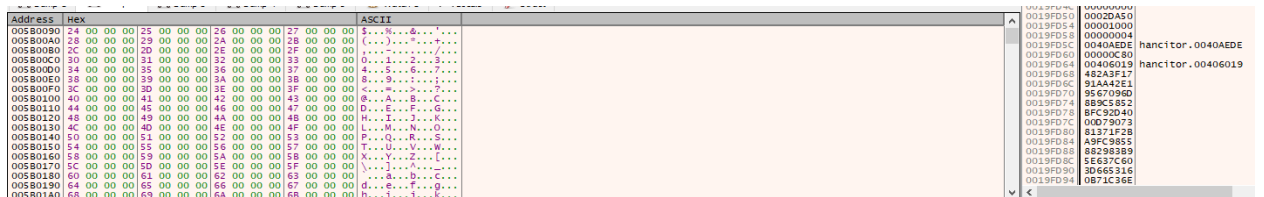


Fig 8. Following through VirtualAlloc Calls.

Continuing on the unpacking process by following VirtualAlloc shows that the data is interestingly unpacked on even rows of data per call. This process is repeated seven times as seen on Fig 5.



Fig 10. Dumping from Memory.

After finding the MZ file, we dump the binary to a file from memory through x32dbg or Process Hacker. However, attempting to open the binary on IDA comes up with issues and it looks like it is necessary to fix the import address table and to rebase the binary on PE-Bear first.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
3510		0	FALSE	45800068	FC7D83FC	6A207500	30006840	45800000
3524		0	FALSE	6A30F0	51084D8B	407815FF	45800068	FC558BFC
3538		0	FALSE	83F45589	7500FC7D	86E905	45800000	41E950F0
354C		0	FALSE	83FFFD0E	4B8904C4	F87D83F8	E8027500	F44D8B9F
3560		0	FALSE	F8558B51	10458B52	C4D8B50	58E851	C4830000
3574		0	FALSE	74C0B510	147D8324	8B087400	45081455	830289F4
3588		0	FALSE	7400187D	EC4D8B0E	3F4558B	458B2851	EB108918
359C		0	FALSE	6A2DEB02	F04D8B00	F8558B51	FC458B52	84D8B50
35B0		0	FALSE	9415FF31	83006840	E80275C0	E845C70F	1
35C4		0	FALSE	809FD213	FFFFF72D	F87D83	45800C74	F82850F8
35D8		0	FALSE	83FFFD0D	7D8304C4	1D7400FC	E87D83	681775
35EC		0	FALSE	8B000080	8B51F04D	8B52FC55	FF500845	68407C15
3600		0	FALSE	E8458B00	C35DE58B	CCCCCCCC	CCCCCCCC	83EC8B55
3614		0	FALSE	458B08EC	FC458B08	8BFC4D8B	51030855	F8558B3C

Fig 11. Broken IAT

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	2E00	1000	2CF0	60000020	0	0	0
.data	3200	A00	4000	984	40000040	0	0	0
.data	3C00	2200	5000	22AC	C0000040	0	0	0
.reloc	5E00	200	8000	1D0	42000040	0	0	0

Fig 12. Fixing the IAT

To fix the import address table; the Raw address is initially changed to match the virtual address and subtracting the values of the raw address and applying the value to raw size. Afterwards, match the virtual size with the raw size and try to fill in the memory for the last slot in the raw size. The image base in the Optional Hdr tab may also be needed to match the address from where we dumped.

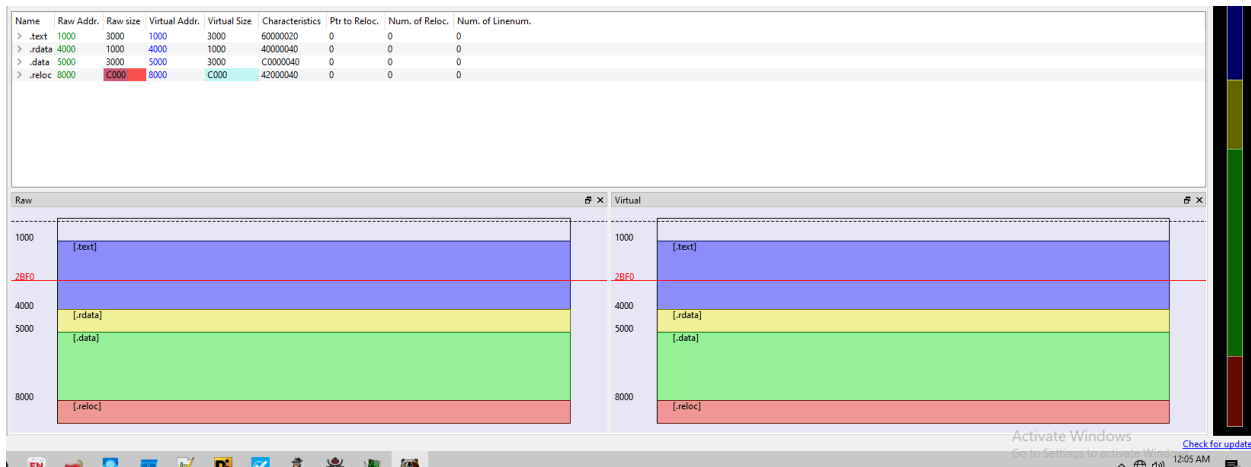


Fig 13. Fixed IAT

Disasm: .text    General    Strings    DOS Hdr    Rich Hdr    File Hdr    Optional Hdr    Section Hdrs    Imports    BaseReloc								
Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk
4310	WININET.dll	10	FALSE	4494	0	0	4592	40E4
4324	IPHLPAPI.DLL	1	FALSE	43E0	0	0	45B6	4030
4338	PSAPI.DLL	2	FALSE	4480	0	0	45F0	40D0
434C	ntdll.dll	1	FALSE	44C0	0	0	4610	4110
4360	KERNEL32.dll	37	FALSE	43E8	0	0	487A	4038
4374	USER32.dll	1	FALSE	448C	0	0	4894	40DC
4388	ADVAPI32.dll	11	FALSE	43B0	0	0	4976	4000

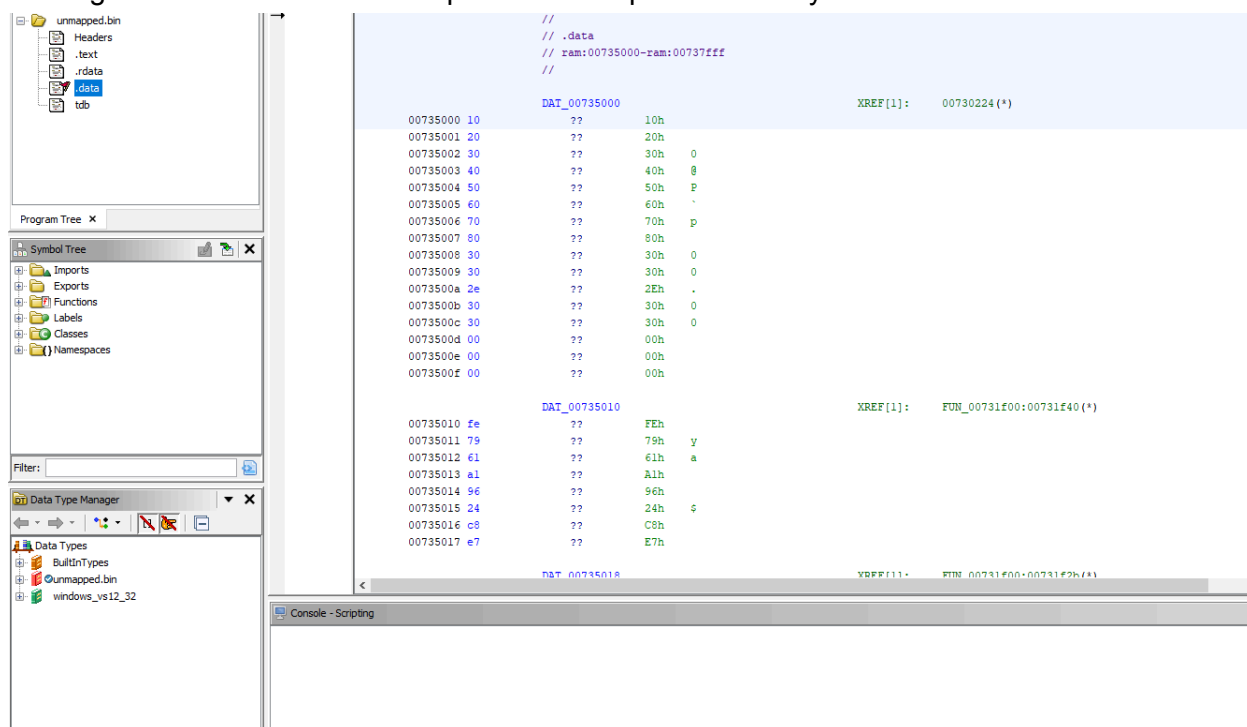
  

ADVAPI32.dll [ 11 entries ]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
4000	CryptDestroyHa...	-	4962	76AAF970	-	B6
4004	CryptHashData	-	4952	76AAF560	-	C8
4008	CryptCreateHash	-	4940	76AAF440	-	B3
400C	CryptDecrypt	-	4930	76AB4A80	-	B4
4010	CryptDestroyKey	-	491E	76AAF8D0	-	B7
4014	CryptDeriveKey	-	490C	76AC0F80	-	B5
4018	CryptReleaseCo...	-	48F6	76AAF9F0	-	CB
401C	CryptAcquireC...	-	48DE	76AB0060	-	B0
4020	LookupAccoun...	-	48CA	76AD8EC0	-	190
4024	GetTokenInfor...	-	48B4	76AAEB00	-	15A
4028	OpenProcessTo...	-	48A0	76AAEF60	-	1F7

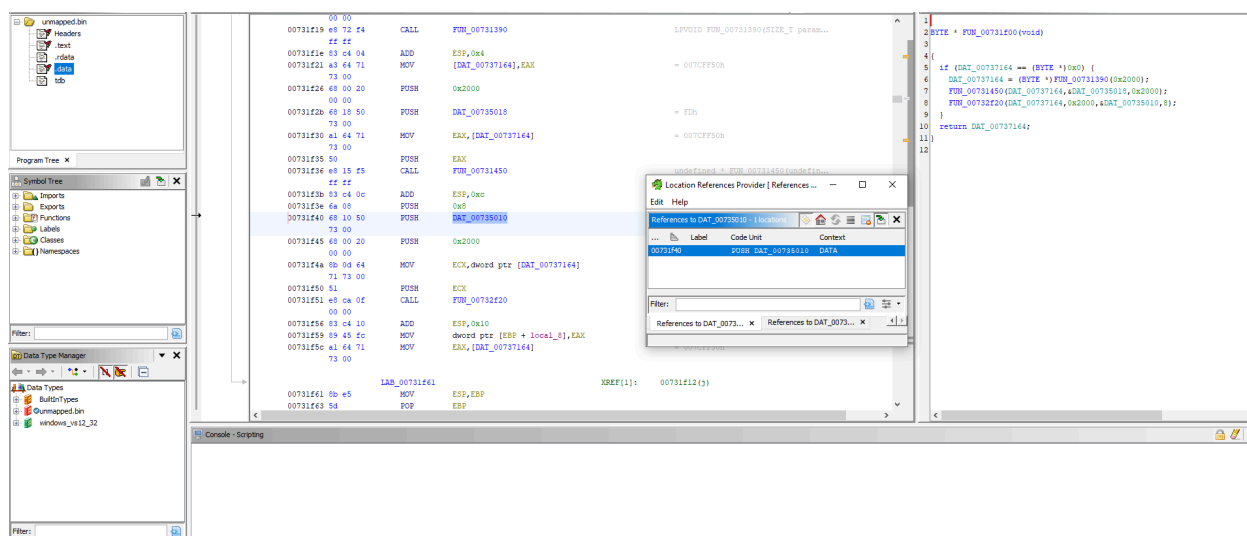
Fig 14. Fixed Imports Header

After this process is done. We then dump the unmapped file and open the unmapped file for the next stages of the analysis.

Viewing the data section we see upfront two suspicious data symbols.



Follow the references of these data



Can also see that it pushes a 0x8

Following the function that called the data leads us to an the malwars encryption routine



```

1
2 BYTE * FUN_00731f00(void)
3
4 {
5     if (DAT_00737164 == (BYTE *)0x0) {
6         DAT_00737164 = (BYTE *)FUN_00731390(0x2000);
7         FUN_00731450(DAT_00737164, &DAT_00735018, 0x2000);
8         FUN_00732f20(DAT_00737164, 0x2000, &DAT_00735010, 8);
9     }
10    return DAT_00737164;
11 }
12

```

Notable parameters in the encryption calls are 0x8004 in CyptHashData, 0x6801, and 0x280011 on CryptDeriveKey. Where 0x8004 is the ALG\_ID(from Wincrypt.h) for SHA1 and 0x6801 indicates that it will be using RC4. The parameter 0x280011 is interesting because Windows uses this to store two variables that it will use based from the most and least significant bit. We are more interested in the most significant word 0x0028h because this will determine the byte size which can be computed by converting 28h to decimal then dividing by 8 which gives us 40/8 or 5 bytes.

```

local_14 = 0;
BVar1 = CryptAcquireContextA(&local_c, (LPCSTR)0x0, (LPCSTR)0x0, 1, 0xf0000000);
if (((BVar1 != 0) && (BVar1 = CryptCreateHash(local_c, 0x8004, 0, 0, &local_8), BVar1 != 0)) &&
    (BVar1 = CryptHashData(local_8, param_3, param_4, 0), BVar1 != 0)) &&
    ((BVar1 = CryptDeriveKey(local_c, 0x6801, local_8, 0x280011, &local_10), BVar1 != 0) &&
    (BVar1 = CryptDecrypt(local_10, 0, 1, 0, param_1, &param_2), BVar1 != 0))) {
    local_14 = param_2;
}
if (local_8 != 0) {

```

Decrypting the config on cyberchef

Copy the suspected key and the suspected encrypted configuration.

0073500f	00	??	00h	
		<b>DAT_00735010</b>		XREF[1]: FUN_007:
00735010	fe	??	FEh	
00735011	79	??	79h	y
00735012	61	??	61h	a
00735013	a1	??	A1h	
00735014	96	??	96h	
00735015	24	??	24h	\$
00735016	c8	??	C8h	
00735017	e7	??	E7h	
		<b>DAT_00735018</b>		XREF[1]: FUN_007:
00735018	fd	??	FDh	
00735019	52	??	52h	R
0073501a	8a	??	8Ah	
0073501b	ac	??	ACh	
0073501c	0a	??	0Ah	
0073501d	7f	??	7Fh	
0073501e	7f	??	7Fh	
0073501f	7c	??	7Ch	

Convert from hex then sha1 which gives us 433b7ee7e6740376e8bb4f75ab235f900d2390f7.  
We will only take 5 bytes from this as our decryption key as set on the encryption routine which leaves us with 433b7ee7e6 (two characters per byte).

Recipe

From Hex

Delimiter  
Auto

SHA1

Rounds  
80

Input

fe7961a19624c8e7

Output

433b7ee7e6740376e8bb4f75ab235f900d2390f7

STEP

BAKE!

Auto Bake

Activate Windows

Go to Settings to activate Windows.

We then input the encrypted configuration as a hex input on the RC4 recipe on cyberchef and put our key back as hex on the same recipe which gives us the decrypted configuration.

The screenshot shows the CyberChef web application with the RC4 recipe selected. The interface is divided into three main sections: Recipe, Input, and Output.

**Recipe Section:**

- RC4:** The selected recipe.
- Passphrase:** 433b7ee7e6
- Input format:** Hex
- Output format:** Latin1

**Input Section:**

A large text area containing a long hexadecimal string, which is the encrypted configuration. The string starts with 'da7b5877e6619786d0d2583d8d71828b7b08c38d0877cc78ac4405ae6b88e699483eaa8f3eb451be04bb53b4d438a326d4172105fab35d3' and ends with '622b5b544f84cfdca2c9a2391dde2079fb8'.

**Output Section:**

The output is displayed in a yellow-highlighted text area. It starts with '13ktb12' followed by a long string of 'N' characters. Below this, there are several lines of URLs, including 'http://sindintwatold.com/4/forum.php', 'http://hegenbofor.ru/4/forum.php', and 'http://justhardogot.ru/4/forum.php'. The output ends with another long string of 'N' characters.

**Footer:**

The bottom of the interface shows a 'STEP' button, a 'BAKE!' button with a chef icon, and an 'Auto Bake' checkbox.