




Writeup

Descripción del reto:

Nuestros investigadores de threat intel detectaron un ciber ataque en Latino America y aunque sus TTPs indicaban ser del ransomware SEXI Aparicio un archivo extra que no corresponde a los atacantes originales, este archivo se llama numbers.txt y una nota de rescata que comienza con "Arriba TEPITO". Igual fue localizado el repositorio de los atacantes y lograron recuperar su código fuente que esperamos te sea útil. Cuando aparecen atacantes que imitan a otro suelen cometer errores en su implementación, ¿será que esta imitación echa en Tepito cumpla ese patrón? Necesitamos recuperar la información contenida en el archivo flag.txt.extEsxi Adjuntamos zip, password: infected NOTA: Te adjuntamos todo el proyecto (codigo_esxi.zip) que deberas abrir con visual studio community 2022, ya que tendrás que recompilarlo para modificarlo y resolver el reto.

En el reto nos dan 3 archivos, un zip que contiene el código fuente del ransomware, una seria de números que nos dicen que pertenecen a los números generados por

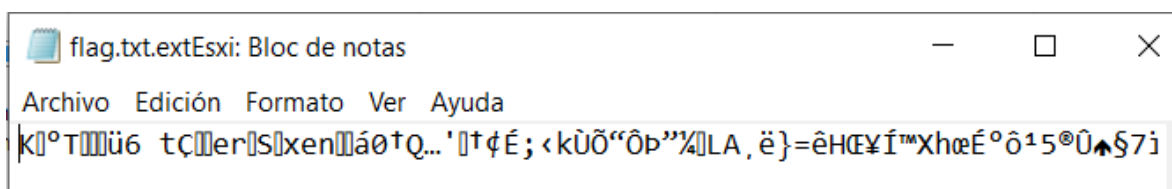
Por el reto obtenemos 3 archivos, código_esxi.zip, flag.txt.extEsxi y numbers.txt;

 codigo_esxi.zip	31/07/2024 01:23 a. m.	Carpeta comprimida (e...	34 KB
 flag.txt.extEsxi	30/07/2024 11:51 p. m.	Archivo EXTESXI	1 KB
 numbers.txt	31/07/2024 12:01 a. m.	Archivo TXT	3 KB

En el archivo nubers.txt existen 200 números de 4 bytes hexadecimales que no tenemos claro a que podrían pertenecer

```
numbers.txt
0xa4d9ec9,0xb1261369,0xec93d5b9,0xc24936d1,0xd5244765,0xde92ffbf,0xdb49a3d2,0x6da4d1e9,0x82d2b4f9,0xf5698671,0xceb41
f35,0xd35ad397,0xddadb5c6,0x6ed6dae3,0x836bb17c,0x41b5d8be,0x20daec5f,0xa46daa22,0x5236d511,0x9d1bb685,0xfa8d074f,0x
c9465faa,0x64a32fd5,0x86514be7,0xf72879fe,0x7b943c9f,0x89cac272,0x44e56139,0x96726c91,0xff39ea45,0xc9c292f,0xd1cec8
9a,0x68e7644d,0x80736e2b,0xf4396b18,0x7a1cb58c,0x3d0e5ac6,0x1e872d63,0xbb434abc,0x5da1a55e,0x2ed0d2af,0xa368b55a,0x5
1b4aad,0x9cdfaf15b,0xfa6da4a0,0x7d36d250,0x3e9b6928,0x1f4db494,0xfa6da4a,0x7d36d25,0xb7e96a9f,0xe9ff46942,0x77fa34a1,
0x8ffdc65d,0xf3fe3f23,0xcdfffc39c,0x66ffe1ce,0x337ff0e7,0xadbf247e,0x56df923f,0x9f6f1512,0x4fb78a89,0x93db1949,0xfded
50a9,0xcaf67459,0xd17be621,0xdcdbd2f1d,0xda5e4b83,0xd92ff9cc,0x6c97fce6,0x364bfe73,0xaf252334,0x5792919a,0x2bc948cd,0
xa1e4786b,0xe4f2e038,0x7279701c,0x393cb80e,0x1c9e5c07,0xba4ff20e,0x5d27f907,0x9a93208e,0x4d499047,0x92a4142e,0x49520
a17,0x90a9d906,0x4854ec83,0x902aaa4c,0x48155526,0x240aaa93,0xa6058944,0x5302c4a2,0x29816251,0xa0c06d25,0xe460ea9f,0x
c630a942,0x631854a1,0x858cf65d,0xf6c6a723,0xcfc638f9c,0x67b1c7ce,0x33d8e3e7,0xadecadfe,0x5f6f56ff,0x9f7bf772,0x4fbdff
b9,0x93de21d1,0xfdefcce5,0xcaf73a7f,0xd17b4132,0x68bda099,0x805e0c41,0xf42fda2d,0xce17311b,0xd30b4480,0x6985a240,0x3
4c2d120,0x1a616890,0xd30b448,0x6985a24,0x34c2d12,0x1a61689,0xb4d3d749,0xee6937a9,0xc3447d09,0xd59aff1e,0xdceda3fd,0
xdb60df3,0xd9b3da4,0x6cd9ed7a,0x366cf6bd,0xaf36a753,0xe30b8fa4,0x71cdc7d2,0x38e6e3e9,0xa873adf9,0xc300af1,0xc41c59
75,0xd60ef0b7,0xf07a456,0xf83d22b,0x83c13518,0x41e09a8c,0x20f04d46,0x07826a3,0xb3ccf5c,0x5e1e67ae,0x2f0f33d7,0xa
38745e6,0x51c3a2f3,0x9ce10d74,0x4e7086ba,0x2738435d,0xa79cfda3,0xe7cea2dc,0x73e7516e,0x39f3a8b7,0xa8f90856,0x547c842
b,0x9e3e9e18,0x41f4f0c,0x278fa786,0x13c7d3c3,0xbde335ec,0x5ef19af6,0x2f78cd7b,0xa3bcbab0,0x51de5d58,0x28ef2eac,0x14
779756,0xa3bcbab,0xb11d39d8,0x588e9cec,0x2c474e76,0x1623a73b,0xbf110f90,0x5f8887c8,0x2fc443e4,0x17e221f2,0xbfb110f9,0
xb1f85471,0xecfcf635,0xc27ea717,0xd53f8f86,0x6a9fc7c3,0x814f3fec,0x40a79ff6,0x2053cfff,0xa4293bf0,0x52149df8,0x290a4
efc,0x1485277e,0xa4293bf,0xb12195d2,0x5890cae9,0x9848b979,0xf82480b1,0xc8129c55,0xd0099227,0xdc04151e
```

Y un archivo que cual esta cifrado sin texto claro.



Pasemos analizar el proyecto que se nos da que es el código fuente del ransomware, comenzamos por la función main():

```
int main(int argc, char* argv[]) {  
  
    const wchar_t* envPaths[] = {  
        L"%USERPROFILE%\\Downloads",  
        L"%USERPROFILE%\\Documents",  
        L"%USERPROFILE%\\Desktop"  
    };  
  
    std::vector<std::string> paths;  
  
    for (const auto& envPath : envPaths) {  
        try {  
            std::string expandedPath = ExpandEnvironmentVariable(envPath);  
            paths.push_back(expandedPath);  
        }  
        catch (const std::exception& ex) {  
        }  
    }  
  
    for (const auto& path : paths) {  
        char* p = new char[path.size() + 1];  
        std::strcpy(p, path.c_str());  
        find_files_recursive(p);  
    }  
}
```

Resumen:

- Primero establece un arreglo de paths
- Después crea un vector que obtiene los paths con las variables de entorno expandida
- Por ultimo comienza a iterar sobre los paths y pasarle el valor al método find_file_recursive() perteneciente a la clase worker.cpp

Analizamos método find_file_recursive() de la clase worker.cpp:

- Utiliza el método FindFirstFile y FindNextFile para empezar a iterar sobre los archivos del path dado
- Comprueba que el archivo no tenga la extensión “.extEsxi” pues pertenece al archivo ya cifrado
- Después vemos una serie de extensiones comentadas las cuales representan las extensiones originales del ransomware y por temas se pusieron así.
- Por último comprueba si el archivo se llama flag.txt

- Vemos que crea una copia de la información del archivo y además crea un hilo con los parámetros NULL, 0, encrypt_file, (void*)reinitialized_arg, 0, NULL
 - Donde encrypt_file es la función que se iniciara y reinitialized_arg son los argumentos que recibirá que en este caso contienen la info del archivo a cifrar

```
void find_files_recursive(char* begin) {
    WIN32_FIND_DATA findFileData;
    HANDLE hFind = INVALID_HANDLE_VALUE;
    char dirSpec[MAX_PATH];
    strcpy(dirSpec, begin);
    strcat(dirSpec, "\\*");

    hFind = FindFirstFile(CharToLPCWSTR(dirSpec), &findFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        return;
    }
    else {
        do {
            if (strcmp(WCHARToConstChar(findFileData.cFileName), ".") != 0 && strcmp(WCHARToConstChar(findFileData.cFileName), "..") != 0) {
                if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
                    char subDir[MAX_PATH];
                    strcpy(subDir, begin);
                    strcat(subDir, "\\");
                    strcat(subDir, WCHARToConstChar(findFileData.cFileName));
                    find_files_recursive(subDir);
                }
                else if (!(findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
                    if (!strcmp(WCHARToConstChar(findFileData.cFileName), ".extExi")) {
                        if (
                            //strstr(WCHARToConstChar(findFileData.cFileName), ".log") ||
                            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmdk") ||
                            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmem") ||
                            //strstr(WCHARToConstChar(findFileData.cFileName), ".vswap") ||
                            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmsn") ||
                            strstr(WCHARToConstChar(findFileData.cFileName), ".flag.txt")
                        ) {
                            files_all++;
                            char filePath[MAX_PATH];
                            strcpy(filePath, begin);
                            strcat(filePath, "\\");
                            strcat(filePath, WCHARToConstChar(findFileData.cFileName));

                            char* reinitialized_arg = (char*)malloc(strlen(filePath) + 1);
                            strcpy(reinitialized_arg, filePath);

                            printf("Encrypting: %s\n", reinitialized_arg);
                            CreateThread(NULL, 0, encrypt_file, (void*)reinitialized_arg, 0, NULL);
                            continue;
                        }
                    }
                    files_all++;
                    files_skipped++;
                }
            }
        } while (FindNextFile(hFind, &findFileData) != 0);
    }
}
```

Analizamos el método encrypt_file().

- Inicializa arreglos y objetos que serán usados en el proceso de cifrado
- Después vemos crea un numero aleatorio llamado seed (líneas 88 -90)
- Un valor hexadecimal llamado tap
- Después inicializa el objeto lfsr(seed,tap) el cual es una función generadora de números aleatorios
- Después vemos que hay 2 ciclos for donde el primero genera 200 números y el segundo 10000. Es posible que estos 200 números sean los que tenemos en nuestro archivo numbers.txt. (líneas 94-99)
- Después lo que hace es crear 32 números usando esta función aleatoria donde a cada numero generado le altera su orden es decir si era 0xAABBCCDD ahora sera 0xDDCCBBAA creando un numero de 32 bytes.
- Copia los valores de este numero a la variable llamada u_priv
- Ahora vemos que esta comentada la función csprng() la cual llenaba también la variable u_priv aquí nos damos cuenta que los atacantes originales no creaban esta variable a partir de la función LFSR por lo que esta función podría ser vulnerable.

- Después vemos una serie de operaciones sobre el valor `u_priv`. (línea 110-112)
- Después podemos notar que el valor `u_priv` es usada para ser cifrado por medio del algoritmo `curve25519` generando dos nuevas variables con el mismo valor llamadas `u_publ` y `u_secr`
- De la línea 116 a la 122 vemos como se comienza a inicializar los valores y parámetros para el algoritmo llamado `sosemanuk` donde podemos ver que el valor `u_secr` es el que se convierte en la llave del algoritmo
- De la línea 124 a la 135 se empieza a leer el contenido del archivo a cifrar en bloques de `0x20000000` bytes y son cifrados con el algoritmo `sosemanuk`. Después de cifrados los bytes son escritos en el archivo original
- De la línea 135 a la 149 se libera la memoria de los objetos
- Por último el archivo original es movido y renombrado agregándole la extensión `.extEsxi`

```

DWORD WINAPI encrypt_file(LPVOID lpParameter) {

    uint32_t wholeReaded = 0;
    size_t readed = 0;

    uint8_t u_publ[32];
    uint8_t u_priv[32];
    uint8_t u_secr[32];
    uint8_t se_key[32];
    LARGE_INTEGER fsize;

    sha256_context sc;
    sosemanuk_key_context kc;
    sosemanuk_run_context rc;

    char* path = static_cast<char*>(lpParameter);
    if (!path) {
        return 0;
    }

    HANDLE hFile = CreateFile(CharToLPCTSTR(path), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile != INVALID_HANDLE_VALUE) {
        if (GetFileSize(hFile, &fsize)) {
            if (uint8_t* f_data = (uint8_t*)malloc(CONST_BLOCK)) {
                if (std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
                    std::uniform_int_distribution<uint32_t> dist(0, (static_cast<int64_t>(1) << 63) - 1);
                    uint32_t seed = dist(rng);
                    uint32_t tap = 0xB400DC0D;
                    LFSR lfsr(seed, tap);
                    std::vector<uint32_t> sequence;
                    for (int i = 0; i < 200; ++i) {
                        sequence.push_back(lfsr.next());
                    }
                    for (int i = 0; i < 10000; ++i) {
                        lfsr.next();
                    }

                    std::vector<uint8_t> bytes;
                    for (int i = 0; i < 32; ++i) {
                        uint32_t lfsr_value = lfsr.next();
                        for (int j = 0; j < 4; ++j) {
                            bytes.push_back((lfsr_value >> (j * 8)) & 0xFF);
                        }
                    }
                    std::copy(bytes.begin(), bytes.end(), u_priv);
                    //csprng(u_priv, 32);
                    u_priv[0] &= 248;
                    u_priv[31] &= 127;
                    u_priv[31] |= 64;
                }
            }
        }
    }
}

```

A partir del análisis y pensando en que los 200 números obtenidos son los generados por el primer ciclo del LFSR podemos tratar de hacer una recuperación de la semilla con la que fue cifrado el archivo y así poder recuperar la llave del algoritmo `sosemanuk`.

Para nuestra fortuna contamos con el código del PRNG en la clase LFSR, el cual solo consiste en manipular la variable `state` que corresponde a un inicio a la `seed` y pasarla por 3 operaciones un AND, un Shift a la derecha y un xor con el `tap`.

```

#include "LFSR.h"

LFSR::LFSR(uint32_t seed, uint32_t tap) : state(seed), tap(tap) {}

uint32_t LFSR::next() {
    uint32_t lsb = state & 1;
    state >>= 1;
    if (lsb) {
        state ^= tap;
    }
    return state;
}

uint32_t LFSR::getState() const {
    return state;
}

```

Copiamos este código a una nueva solución y ahora hay que crear un método que nos permita iterar sobre los posibles valores que pueda tener la seed la cual puede ir desde 0x10000000 al 0xffffffff ya que siempre es usado un valor hexadecimal de 8 caracteres. Los 200 numeros que tenemos nos permitirán validar que la seed sea la correcta.

```

uint32_t LFSR::recoverSeed(const std::vector<uint32_t>& sequence, uint32_t tap) {
    assert(sequence.size() >= 2);
    //std::cout << "Semilla recuperada: " << std::hex << (1 << 31) << std::endl;

    for (uint32_t candidate = 0x10000000; candidate < 0xffffffff; ++candidate) {
        //std::cout << "Semilla recuperada: " << std::hex << candidate << std::endl;

        LFSR lfsr(candidate, tap);
        bool match = true;
        for (size_t i = 0; i < sequence.size(); ++i) {
            if (lfsr.next() != sequence[i]) {
                match = false;
                break;
            }
        }
        if (match) {
            return candidate;
        }
    }
    return 0;
}

```

```
int main() {
    uint32_t tap = 0x8000C80; // Tap de 32 bits (polinomio de retroalimentación)
    std::vector<uint32_t> numbers = { 0xa4d9ec9, 0xb1261369, 0xec93d5b9, 0xc24936d1, 0xd5244765, 0xd92ffbf9, 0xd4b49a3d2, 0x6da4d1e9, 0x82d2b4f9, 0xf5698671, 0xc6b41f35, 0xd35ad397, 0xddadb5c6, 0x6ed6dae3, 0x836bb17c, 0x40000000 };
    uint32_t recovered_seed = LFSR::recoverSeed(numbers, tap);
    std::cout << "Semilla recuperada: " << std::hex << recovered_seed << std::endl;
    LFSR lfsr2(recovered_seed, tap);
    std::vector<uint32_t> sequence2;
    for (int i = 0; i < 200; ++i) {
        sequence2.push_back(lfsr2.next());
    }
    for (int i = 0; i < 10000; ++i) {
        lfsr2.next();
    }
    std::cout << "Números generados con semilla: " << std::endl;
    for (const auto& num : sequence2) {
        std::cout << std::hex << num << " ";
    }
    std::cout << std::endl;
    std::vector<uint8_t> bytes2;
    for (int i = 0; i < 32; ++i) {
        uint32_t lfsr_value = lfsr2.next();
        //std::cout << "Way 2: " << std::endl;
        std::cout << std::hex << lfsr_value << " ";
        // Extraer bytes individuales del valor de 32 bits
        for (int j = 0; j < 4; ++j) {
            bytes2.push_back((lfsr_value >> (j * 8)) & 0xFF);
        }
    }
    std::cout << "Bytes generados 2: " << std::endl;
    for (const auto& byte : bytes2) {
        std::cout << std::hex << static_cast<int>(byte) << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

```

Semilla recuperada: 149b3d92
N°meros generados con semilla:
a4d9ec9 b1261369 ec93d5b9 c24936d1 d5244765 de92ffbf db49a3d2 6da4d1e9 82d2b4f9 f5698671 ceb41f35 d35ad397 ddadb5c6 6ed6dae3 83
6bb17c 41b5d8b6 20daec5f a46daa22 5236d511 9d1bb685 fa8d074f c9a565fa 64a32fd5 8e6514be7 f72879fe 7b943c9f 89cac272 44e56139 967
26c91 f3f9a585 cb9c292f d1cec89a 68e7614d 80736e2b f4396b18 7a1cb58c 3d0e5ac6 1a872d63 bb343abc 5d0a1a5e 2ed0d2af a368b55a 51b
5aad 9cdaf15b fad6da4a 7d36d250 3e9b6928 1f4db491 fad6da4a 7d36d25 b7e96a9f eff1u69u2 77fa3a71 d8ffcd65d f3fe2f32 cdfffc39c 66ffe1c
e 337f0e7f adbf247e 56df923f 9f6f1512 4fb78a89 93db1949 fded50a9 caf67459 d17be621 dcd27f1d da5e4b83 d92ff923 c6977fec 646fe7c
af252334 5792919a 2bc948cd ale4786b e4f2e038 7279701c 393cb80e 1c9e50cf ba4ff20e 5db7f907 9a93208e 4dd499047 92a4142e 49520a17
90a9d906 485b1c9c 902aaac4 84155526 240aa93f a605994a 5302ca4d 29816251 ab0c6d25 e460ea9f c630a942 61318541 858c6f5d 66fca6723 c
f638f99e 67b7e7ce 33d8e3e7 adecafe6 56f656ff 9f7fb772 f4bdfbfb 93d2e1d1 fdffcc5e caf73af7 d17b1432 6b8da099 8059c411 f42fda2d c
17311b d30b4u40 6985a2u0 34c2d120 1a616890 d30b4u8 6985a2u 34c2d12 1a61689 b0d39479 ee693749 c334u7d9 d59affel decda3fd db660df
3 d9b3da4 6cd9ed7a 366cf6bd af36a753 e39b8fa4 71cddc7d 38e6e3e9 a873ad9f a0390af1 c41c5975 d0ef0f07 d07a456 6f83d22b 83c13518
41ea9ba8 20f40d46 107826a3 bc3ccff5 5e1e67ae 2f0833d7 a38745e6 51c3a2f3 9ce10d74 d670886a 2738343d a79cfa4a e7cea2de 73e7516e
39f3ab87 a8f90856 547c842b 93e9e3e1 41f14f0c 278fa786 13c7d3c3 6b2335ec 5ef19af6 2f78cd7b abc3bab0 51de5d58 28ef2eac 14797756 a
3bcbab b11d39d8 588e9ce c247e476 1623a7fb b11f099f 5f88878c 2f4cu4454 17e22f12 bf1109f b185u741 eecf63f5 c27ea71 52f3f886 6a9f
c7c3 814f3fec 04a79ff6 2053cffb a4293bf0 52149df8 290a4efc 1485277e a4293bf b12195d2 5890cae9 9848b979 f82480b1 c8129c55 d00992
27 dc0a151e
a326a921 e593889d c6c91843 d764502c 6bb22816 35d9140b aeec5608 57762b04 2bbb1582 15dd8ac1 beee196d eb77d0bb c1bb3450 60dd9a28 3
06cd14 1837668a c1bb345 b20d05af ed065eda 76832f6d 8f414bbb f3a079d0 79d03ce8 3ce81e74 1e740f3a f3a079d b39ddf3c edce33ec 76e7
19f6 3b73c8f8 a9b91a70 54dc8d38 Bytes generados 2:
21 a9 26 a3 9d 88 93 e5 43 18 a9 c6 2c 50 64 d7 16 28 b2 6b b 14 d9 35 8 56 ec a4 2b 76 57 82 15 bb 2b c1 8a dd 15 6d 19 ee b
e bb d0 77 5b 34 bb c1 28 a9 c6 d0 64 1c 6e 30 8a 66 37 18 45 d3 1b c af 5 d b2 da 5e 6 ed 62 2f 83 76 bb 4b 41 8f d0 79 a0
f3 e8 3c d0 79 74 1e e8 3c 3a f 74 1e 9d 7 3a f c3 d8 fd 9d b3 ec 33 ce ed f6 19 e7 76 fb 8c 73 3b 70 1a b9 a9 38 8d dc 54

```

1. Modificamos el buscador de archivos haciendo que en lugar de buscar flag.txt busque flag.txt.extEsxi

1. Modificamos el buscador de archivos haciendo que en lugar de buscar flag.txt busque flag.txt.extEsxi

```

else if (!(findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
    if (!strstr(WCHARToConstChar(findFileData.cFileName), ".extEsxi2")) {
        if (
            //strstr(WCHARToConstChar(findFileData.cFileName), ".log") ||
            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmdk") ||
            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmem") ||
            //strstr(WCHARToConstChar(findFileData.cFileName), ".vswp") ||
            //strstr(WCHARToConstChar(findFileData.cFileName), ".vmsn")
            strstr(WCHARToConstChar(findFileData.cFileName), "flag.txt.extEsxi")
        ) {
            files_all++;
            char filePath[MAX_PATH];
            strcpy(filePath, begin);
            strcat(filePath, "\\");
            strcat(filePath, WCHARToConstChar(findFileData.cFileName));

            char* reinitialized_arg = (char*)malloc(strlen(filePath) + 1);
            strcpy(reinitialized_arg, filePath);

            printf("Encrypting: %s\n", reinitialized_arg);
            CreateThread(NULL, 0, encrypt_file, (void*)reinitialized_arg, 0, NULL);
            continue;
        }
    }
}

```

2. Modificamos nuestra seed para que sea la encontrada

```

84 HANDLE hFile = CreateFile(CharToLPCWSTR(path), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
85 if (hFile != INVALID_HANDLE_VALUE) {
86     if (GetFileSizeEx(hFile, &fsize)) {
87         if (uint8_t* f_data = (uint8_t*)malloc(CONST_BLOCK)) {
88             //std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
89             //std::uniform_int_distribution<uint32_t> dist(0, (static_cast<int64_t>(1) << 63) - 1);
90             uint32_t seed = 0x140b3d92;
91             uint32_t tap = 0xB40DC0D;
92             LFSR lfsr(seed, tap);

```





3. Modificamos el archivo de salida

```

149     CloseHandle(hFile);
150
151     char locked_name[MAX_PATH];
152     strcpy(locked_name, path);
153     strcat(locked_name, ".txt");
154     MoveFile(CharToLPCWSTR(path), CharToLPCWSTR(locked_name));
155 }

```

Corremos la solución y tendremos nuestro archivo llamado flag.txt.extEsxi.txt

Nombre	Fecha de modificación	Tipo	Tamaño
 codigo_esxi	06/08/2024 10:55 p. m.	Carpeta de archivos	
 codigo_esxi.zip	31/07/2024 01:23 a. m.	Carpeta comprimida (e...	34 KB
 flag.txt.extEsxi.txt	06/08/2024 10:55 p. m.	Archivo TXT	1 KB
 numbers.txt	31/07/2024 12:01 a. m.	Archivo TXT	3 KB

Y si vemos su contenido podremos ver el valor de nuestra flag.

```

numbers.txt x flag.txt.extEsxi.txt x
hackdef{N3v3r_Us3_W34k_PRNG<INSERTA_SEED>}<0x04>5ti~$<0x7f>d4""J<0x8d>»Mdi8<0x1b><0x07>G3<0x08>t|ô%bG
%ÇX<0x03>LA,ë}=êHCÿî"XhæÉ°ô¹5°Ü<0x1e><0x0c>$7iài<0x19>|

```