

C my mind

Razvan

Overview

This challenge implies founding the key and iv for the aes cbc encryption used for encrypting the so file that has the flag in it.

1 Dumping the encrypted so file

To dump the encrypted so file you need pwndbg to set the breakpoint at main function and from there to run the program until you see that something is loaded in memory with dl.Pwndbg will show you the pid and the filedescriptor of the encrypted so file, with which you can dump it with cp /proc/15989/fd/3 /Desktop/cmymind.bin



The screenshot shows the pwndbg debugger interface. The assembly window displays the main function's code, including calls to `do_global_dtors_aux` and `dlopen`. Registers RDI, RSI, RDX, R8, R9, R10, R11, R12, R13, R14, R15, RBP, and RSP are shown with their corresponding memory addresses. The stack window shows the current stack state, and the backtrace window shows the call stack starting from the main function. The command line at the bottom shows the command `!dumpRegs`.

```
rdi 0x7fffffd900 -> 0x7fffffd840 <- '/proc/self/fd/3'
rsi 1
r8 0x64
r9 0
r10 0
r11 0
r12 1
r13 0
r14 0x555555558d58 (<_do_global_dtors_aux_fini_array_entry>) -> 0x555555555260 (<_do_global_dtors_aux>) <- endbr4
r15 0x7fffffd800 (<_rtld_global>) -> 0x7fffffe0ed -> 0x555555554000 -> 0x10102464c457f
rbp 0x7fffffd900 -> 0xfffffffdb30 -> 0xfffffffdb90 <- 0
rsp 0x7fffffd900 -> 0x300000014
*rdi 0x5555555551e (main+230) -> mov rdi, rax
[ DISASM / x86-64 / set emulate on ]
0x555555555f04 <main+200>    mov    rdi, rax      RDI => 0x7fffffd840 <- 0
0x555555555f07 <main+211>    mov    eax, 0      RAX => 0
0x555555555f0e <main+216>    call   _snprintf@plt      EAX => 0
[snprintf@plt]
0x555555555f11 <main+221>    lea    rax, [rbp - 0x50]  RAX => 0x7fffffd840 <- '/proc/self/fd/3'
0x555555555f15 <main+225>    mov    est, 1      ES1 => 1
0x555555555f18 <main+230>    mov    rdi, rax      RDI => 0x7fffffd840 <- '/proc/self/fd/3'
0x555555555f1d <main+233>    call   _dlopen@plt      <dlopen@plt>
[dlopen@plt]
0x555555555f22 <main+238>    mov    qword ptr [rbp - 0xd8], rax
0x555555555f29 <main+255>    cmp    qword ptr [rbp - 0xd8], 0
0x555555555f31 <main+253>    jne    main+300      <main+300>
0x555555555f33 <main+255>    lea    rax, [rip + 0x1148]  RAX => 0x555555557082 <- 'Oh no i lost it :((('
[ STACK ]
00:0000| rbp 0x7fffffdab0 <- 0x300000014
01:0008-0bd 0x7fffffdab0 <- 0x40 /* '0' */
02:0010-0dd 0x7fffffdac0 <- 0x800000
03:0018-0cb 0x7fffffdac8 <- 8
04:0020-0cb 0x7fffffdad0 <- 0xfffffffffffffff
05:0028-0bd 0x7fffffdad8 <- 0x40 /* '0' */
06:0030-0bd 0x7fffffdad8 <- 0xc /* '\x0c' */
07:0038-0bd 0x7fffffdad8 <- 0xa0000
[ BACKTRACE ]
▶ 0 0x555555555f1a main+230
1 0x7ffff762a1ca __libc_start_call_main+122
2 0x7ffff762a28b __libc_start_main+139
3 0x5555555551e5 __start+37
pwndbg> !dumpRegs
```

2 Finding and decrypting the the iv and key for the aes

Looking through the decompiled compiled code you will see cbc aes so for finding the key and iv you need to look through the encrypt function there you will see 4 functions and find 8 hex values in every function the first 2 functions contains the encrypted key value and in the next 2 functions the hex values represent the iv. In the same functions where you find the hex values you will find the encryption algorithms used to encrypt them. In the original code they were called in the same order so your mission is to analyze them and reproduce them, that part will be pretty easy so here i will show the part that divides the key and calls the function that calls the transpose of the matrix.

```
155 }
156
157 int main() {
158     srand(seed: time(timer: NULL));
159
160     char input[BLOCK_SIZE + 1] = "5457538754346744";
161     char encrypted[BLOCK_SIZE];
162
163     char decrypted[BLOCK_SIZE];
164     char matrix[SIZE][SIZE];
165
166     printf(format: "Original: %s\n", input);
167
168     for (int i = 0; i < BLOCK_SIZE; i++) {
169         matrix[i / SIZE][i % SIZE] = input[i];
170     }
171     encrypt(matrix);
172
173     for (int i = 0; i < BLOCK_SIZE; i++) {
174         encrypted[i] = matrix[i / SIZE][i % SIZE];
175     }
176     printf(format: "Encrypted (hex): ");
177     printHex(data: encrypted, len: BLOCK_SIZE);
178
179     for (int i = 0; i < BLOCK_SIZE; i++) {
180         matrix[i / SIZE][i % SIZE] = encrypted[i];
181     }
182     decrypt(matrix);
183
184     for (int i = 0; i < BLOCK_SIZE; i++) {
185         decrypted[i] = matrix[i / SIZE][i % SIZE];
186     }
187     decrypted[BLOCK_SIZE] = '\0';
188
189     printf(format: "Decrypted: %s\n", decrypted);
190
191     return 0;
192 }
```

complete code of the algorithm in the private folder algo.c]

[You can find the

3 Decrypting encrypted so file with aes ”

After you find the key and iv you need to make a function to decrypt the previous encrypted file that you obtained. To do this I use but you can use any programming language.

```
void aes_encrypt_cbc(const unsigned char *plaintext, int pt_len,
                     const unsigned char *key, const unsigned char *iv,
                     unsigned char *ciphertext) {
    AES_KEY aes_key;
    AES_set_encrypt_key(userKey: key, bits: 128, key: &aes_key);
    AES_cbc_encrypt(in: plaintext, out: ciphertext, length: pt_len, key: &aes_key, iv: (unsigned char *)iv, enc: AES_ENCRYPT);
}

void aes_decrypt_cbc(const unsigned char *ciphertext, int ct_len,
                     const unsigned char *key, const unsigned char *iv,
                     unsigned char *plaintext) {
    AES_KEY aes_key;
    AES_set_decrypt_key(userKey: key, bits: 128, key: &aes_key);
    AES_cbc_encrypt(in: ciphertext, out: plaintext, length: ct_len, key: &aes_key, iv: (unsigned char *)iv, enc: AES_DECRYPT);
}
```

■ Full code can be seen

After you decrypted the so file you will find half of the flag and an array with hex values to find the last part of the flag you need to find the encryption algorithm with was used to encrypt theme and reverse

The screenshot shows the Immunity Debugger interface with the assembly window open. The assembly code is as follows:

```
Function name: _int16_fastcall_sub_140C(unsigned int8 al)
1:     {
2:         return (((unsigned int16)(116 * (al ^ 0x83)) ^ 0x2A) >> 8) | (((unsigned int16)(116 * (al ^ 0x83)) ^ 0x2A) << 8))
3:         + 1440;
4:     }
5:
6: }
```

seen in extra.c]

[Full code can be

5 UVT{1n_m3mory_w3_trust_wh3n_dySk_1s_ly1ng}