Decryptorium (Hard) - Rev

Sunday, June 11, 2023 3:11 PM

Description

I have created an incredibly powerful encryptor to secure my secret messages. However, I've encountered a setback as I have lost the decryptor. Presently, I am determined to decode my highly confidential message, armed only with the encryptor binary and an encrypted flag.

Solution

Here we are provided with a stripped binary that encrypts out text and provides the encrypted and hex version of the text.

```
┌──(kali㉿kali)-[~/…/rev/unused/decryptorium_r3.5/challenge]
└─$ file decryptorium
decryptorium: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a22f5ed754e949e080013
f6f78fb2c9ce8fb7081, for GNU/Linux 3.2.0, stripped
```

```
┌──(kali㉿kali)-[~/…/rev/unused/decryptorium_r3.5/challenge]
└─$ ./decryptorium
Welcome to the Super Cool C Encryptor!
Enter the text to encrypt: test
Encrypted Flag: ◆◆◆◆
Hex Format: 0×e3 0×e3 0×ee 0×e0
```

We are also provided with the encrypted flag.

```
┌──(kali㉿kali)-[~/…/rev/unused/decryptorium_r3.5/challenge]
└─$ cat enc
0×06 0×f6 0×3c 0×2a 0×e9 0×0d 0×2b 0×0f 0×d4 0×1a 0×0e 0×e4 0×2d 0×e1 0×0a 0×15 0×13 0×d0 0×19 0×d9 0×17 0×fd 0×eb 0×db 0×e5 0×e4 0×ec 0×e0 0×1d 0×23 0×e6 0×f3 0×0c 0x
f6
```

Let's reverse the binary now.

Looking around, we find the following to be the main function. I got a function that seems to be the main and renamed it for better understanding.

```

void main(void)

{
  size_t sVar1;
  byte input [112];
  char user_input [104];
  int local_10;
  int length_of_input;

  puts("Welcome to the Super Cool C Encryptor!");
  printf("Enter the text to encrypt: ");
  fgets(user_input,100,stdin);
  sVar1 = strlen(user_input);
  length_of_input = (int)sVar1;
  if (user_input[length_of_input + -1] == '\n') {
    user_input[length_of_input + -1] = '\0';
    length_of_input = length_of_input + -1;
  }
  strcpy((char *)input,user_input);
  xor_tilde_func(input,length_of_input);
  xor_func(input,length_of_input);
  reverse_func(input,length_of_input);
```

```
  printf("Encrypted Flag: %s\n",input);
  printf("Hex Format: ");
  for (local_10 = 0; local_10 < length_of_input; local_10 = local_10 + 1) {
    printf("%#04x ",(ulong)input[local_10]);
  }
  putchar(10);
  return;
}
```

```
I also renamed several other functions to better understand the code.

To our user_input, it first copies it in the input varaible. Then it passes the input variable to
xor_tilde_fun(), xor_func() and finally reverse_func().

```

```
void xor_tilde_func(long param_1,int param_2)

{
  int i;

  for (i = 0; i < param_2; i = i + 1) {
    *(byte *)(param_1 + i) = *(byte *)(param_1 + i) ^ (char)i + 1U;
    *(byte *)(param_1 + i) = ~*(byte *)(param_1 + i);
  }
  return;
}
```

```
This function basically takes our input, runs a loop upto the length of the input, and in the loop it first
performs xor operation as follows.

input[i] ^= (i+1)

Then it performs the tilde operation as follows.

input[i] = ~input[i]

To reverse this part of the code we have the following python code.

```
```
for i in range(len(l)):
    l[i] = ~l[i] & 0xFF
    l[i] ^= (i+1)
```

```
Here `l` is the user input.

Next up, we have the xor_func().

```

```
void xor_func(long param_1,int param_2)

{
  int j;
```

```
  int i;

  for (i = 0; i < param_2; i = i + 1) {
    *(byte *)(param_1 + i) = *(byte *)(param_1 + i) ^ 0xaa;
    for (j = 0; j < 8; j = j + 1) {
      if (((int)*(char *)(param_1 + i) >> ((byte)j & 0x1f) & 1U) != 0) {
        *(byte *)(param_1 + i) = *(byte *)(param_1 + i) ^ (byte)(1 << ((byte)j + 1 & 0x1f));
      }
    }
  }
  return;
}
```

The equivalent reverse operation in python is as follows.

```
for i in range(len(l)):
    for j in range(7,-1,-1):
            if l[i] >> j & 1:
                    l[i] ^= (1 << (j+1))
    l[i] ^= 0xAA
```

Finally we have reverse_func()

```
void reverse_func(long param_1,int param_2)

{
  undefined uVar1;
  int local_10;
  int i;

  local_10 = param_2 + -1;
  for (i = 0; i < local_10; i = i + 1) {
    uVar1 = *(undefined *)(param_1 + i);
    *(undefined *)(param_1 + i) = *(undefined *)(param_1 + local_10);
    *(undefined *)(local_10 + param_1) = uVar1;
    local_10 = local_10 + -1;
  }
  return;
}


```

This function just reverses the entire input.

So equivalent python code for this is as follows.

```
l = l[::-1]

```

Now we reversed each function separately, combining everything we get the following script.

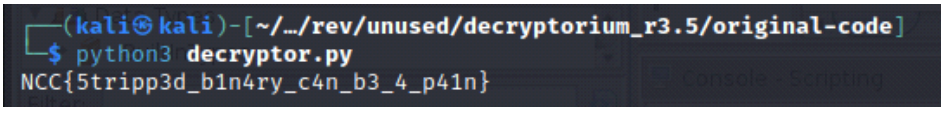
```
l = [6, 246, 60, 42, 233, 13, 43, 15, 212, 26, 14, 228, 45, 225, 10, 21, 19, 208, 25, 217, 23, 253, 235, 219,
229, 228, 236, 224, 29, 35, 230, 243, 12, 246]
l = l[::-1]
for i in range(len(l)):
    for j in range(7,-1,-1):
        if l[i] >> j & 1:
            l[i] ^= (1 << (j+1))
    l[i] ^= 0xAA

for i in range(len(l)):
    l[i] = ~l[i] & 0xFF
    l[i] ^= (i+1)

print("".join([chr(i) for i in l]))
```

Here, I converted the hex valued encrypted flag into integer list and named it as `l`.

Running the script, we'll get the flag.



Flag: NCC{5tripp3d_b1n4ry_c4n_b3_4_p41n}