Reverse Engineering Report

Lakshit Verma

August 14, 2024

1 LIT CTF — revsite2

We were given a website which incremented a variable by clicking a button. When the variable reached $10 \wedge 18$ we would get the flag. Inspecting the script running the site we found it ran WASM, which was the code we had to reverse to get the flag.

Things I learned in this challenge:

1. **WASM decompiling:** Using tools such as `wat2wasm` and `wasm2js` to decompile and convert WASM to more readable languages.
2. **WASM reversing:** Understanding WASM syntax and logic.
3. **Using the Ghidra WASM plugin:** Found this out after the CTF ended.

After looking at other writeups and on the Discord it turned out it performing the summation

$$y = \sum_{x=0}^{10^{18}-1} (8x^3 + 3x^2 + 3x + 8)$$

After getting the value of y, the program performs several bit-shifts, XORs and other operations on it. We reverse those to get the flag.

```
def make_flag(value):
    expr1 = (((value >> 0x29) & 0x1ff ^ 0x110))
    expr2 = (((value >> 0x2b) & 0x1ff ^ 0x144))
    expr3 = (((value >> 0x24) & 0x1ff ^ 0x131))
    expr4 = (((value >> 0x1c) & 0x1ff ^ 0x1e))
    expr5 = (((value >> 5) & 0x1ff ^ 0xd2))
    expr6 = (((value >> 0x17) & 0x1ff ^ 0xb))
    expr7 = (((value >> 0x1c) & 0x1ff ^ 0x2d))
    expr8 = (((value >> 0x23) & 0x1ff ^ 0x151))
    expr9 = (((value >> 1) & 0x1ff ^ 0x68))
    expr10 = (((value >> 0x34) & 0x1ff ^ 0x1f0))
    expr11 = (((value >> 0x2b) & 0x1ff ^ 0x1ff))
    expr12 = (((value >> 0x29) & 0x1ff ^ 0xbb))
    expr13 = (((value >> 0x13) & 0x1ff ^ 0x16a))
    expr14 = (((value >> 0x34) & 0x1ff ^ 0x199))
    expr15 = (((value >> 0x18) & 0x1ff ^ 0xa8))
    values = [expr1, expr2, expr3, expr4, expr5, expr6, expr7, expr8, expr9, expr10, expr11, expr12, expr13,
    flag_chars = []
    flag_chars.append(chr(((((value >> 1) & 0xFF)  ^ 0x75)) & 0xFF))
    flag_chars.append(chr(((((value >> 0x2e) & 0xFFFFFFFF) ^ 199))& 0xFF))
    flag_chars.append(chr(((((value >> 9) & 0xFF) ^ 0x69))& 0xFF))
    flag_chars.append(chr(((((value >> 0x2f) & 0xFFFFFFFF) ^ 0xa7))& 0xFF))
    flag_chars.append(chr(((((value >> 0x12) & 0xFF )^ 0x82))& 0xFF))
    flag_chars.append(chr(((((value >> 0x17) & 0xFF ) ^ 6))& 0xFF))
    flag_chars.append(chr(((((value >> 0x2e) & 0xFFFFFFFF) ^ 0xc5))& 0xFF))
    flag_chars.append(chr(((((value >> 0x36) & 0xFFFFFFFF) ^ 0x2d))& 0xFF))
```

```
    flag_chars.append(chr(((((value >> 0x33) & 0xFFFFFFFF) ^ 0x6c))& 0xFF))
    flag_chars.append(chr(((((value >> 0x27) & 0xFFFFFFFF) ^ 0xf))& 0xFF))
    flag_chars.append(chr(((((value >> 0x1e) & 0xFF ) ^ 0x16))& 0xFF))
    flag_chars.append(chr(((((value >> 0x2e) & 0xFFFFFFFF) ^ 0xc4))& 0xFF))
    flag_chars.append(chr(((((value >> 0x17) & 0xFF ) ^ 0x42))& 0xFF))
    flag_chars.append(chr(((((value >> 0xb) & 0xFF ) ^ 0xec))& 0xFF))
    flag_chars.append(chr(((((value >> 0x2e) & 0xFFFFFFFF) ^ 0x8d))& 0xFF))
    flag_map = dict(zip(values, flag_chars))
    sorted_flag = sorted(flag_map.items())
    print(sorted_flag)
    flag = 'LITCTF{' + ''.join(char for _, char in sorted_flag)
    print(flag)

n=1000000000000000000

bn = 8 * (((n-1)*n)//2)**2
bn += 3 * ((n-1)*n*(2*n-1))//6
bn += 3 * ((n-1) * n) //2
bn += 8 * n
bn += 3

make_flag(bn)

LITCTF{s0_l457minute!}
```

CTFZone — Try to Easy Crackme

Given is a memory dump. We extract a file `Crackme.exe` and find that its a WinRAR SFX (Self-Extracting Executable). Things I learned in this challenge:

1. **Memory Forensics:** How to extract files from memory dumps using Moneta, I had only used Volatility before.
2. **SFX:** Self-Extracting Executables and how to crack their passwords with the same methods we use for regular RAR files.

The solution turned out to be a rockyou.txt brute-force on the `Crackme.exe` file, then a few rounds of UPX executable unpacking and reversing BAT2EXE, a program to convert Windows Batch files to executables.