

gnisreveR ortnI

Reverse Engineering

Crate-CTF

:)

What is reverse engineering

From wikipedia:

[...] is a process or method through which one attempts to understand through deductive reasoning how a [...] piece of software accomplishes a task [...]

Also known as:

- reversing
-

In a CTF context

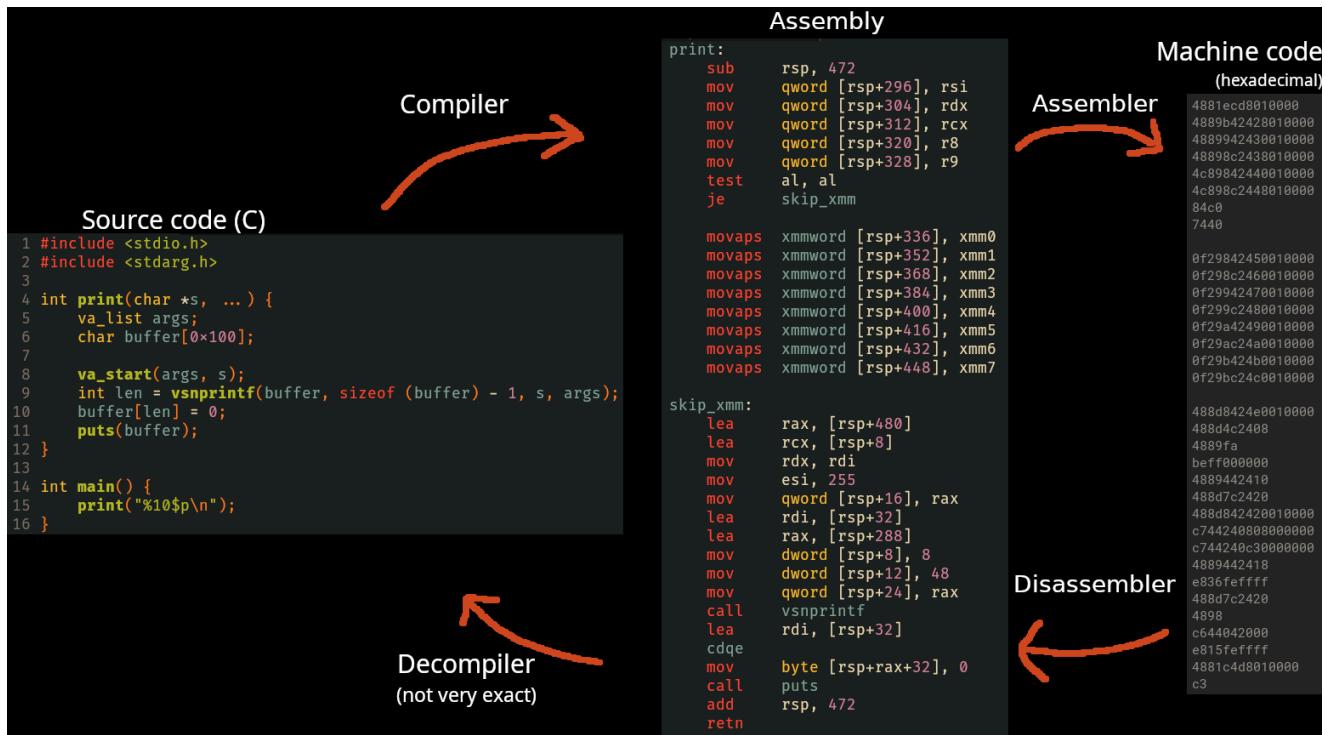
You get a program that:

- checks whether text you input to it is the flag, or
- outputs the flag if you supply the correct password, or
- does some arbitrary hocus pocus which somehow involve a flag

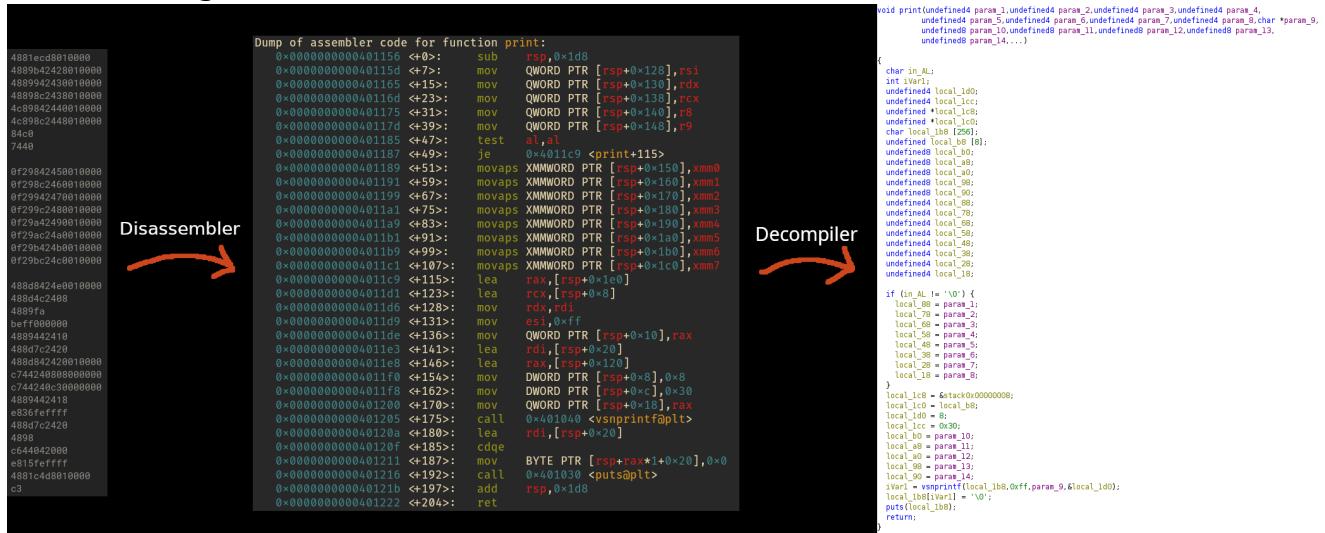
Usually, you get a compiled program

Is to some (often lesser) extent required for binary exploitation (pwn) challenges

What is compilation?



But when we go backwards...



(Basically) everything is compiled, even JavaScript and Python!

	5	0 RESUME	0	
	6	2 BUILD_LIST	0	
		4 STORE_FAST	2 (decoded)	
	7	6 LOAD_GLOBAL	1 (NULL + range)	
		18 LOAD_FAST	1 (diameter)	
		20 PRECALL	1	
		24 CALL	1	
		34 GET_ITER		
	>>	36 FOR_ITER	37 (to 112)	
		38 STORE_FAST	3 (i)	
	8	40 LOAD_FAST	0 (message)	
		42 LOAD_FAST	3 (i)	
		44 LOAD_CONST	0 (None)	
		46 LOAD_FAST	1 (diameter)	
		48 BUILD_SLICE	3	
		50 BINARY_SUBSCR		
		60 GET_ITER		
	>>	62 FOR_ITER	23 (to 110)	
		64 STORE_FAST	4 (c)	
	9	66 LOAD_FAST	2 (decoded)	
		68 LOAD_METHOD	1 (append)	
		70 LOAD_FAST	4 (c)	
		72 PRECALL	1	
		76 CALL	1	
		106 POP_TOP		
		108 JUMP_BACKWARD	24 (to 62)	
	8	>> 110 JUMP_BACKWARD	38 (to 36)	
	10	>> 112 LOAD_CONST	1 ('')	
		114 LOAD_METHOD	2 (join)	
		136 LOAD_FAST	2 (decoded)	
		138 PRECALL	1	
		142 CALL	1	
		152 RETURN_VALUE		

9700 6700 7d02 7401 0000 0000 0000 0000
0000 7c01 a601 0000 ab01 0000 0000 0000
0000 4400 5d25 7d03 7c00 7c03 6400 7c01
8503 1900 0000 0000 0000 0000 4400 5d17
7d04 7c02 a001 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
ab01 0000 0000 0000 0000 0100 8c18 8c26
6401 a002 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 7c02 a601 0000 ab01
0000 0000 0000 0000 5300

Static and dynamic analysis

Static analysis

Just look at the program

```
s = input()
a = "".join(chr(ord(c) + 1) for c in s[::-2]) + s[:-1][::-2]
assert a == "oiz!!fny!fj!uotpilpeettlte"
```

Dynamic analysis

Run parts of the program and look at the behavior and/or intermediary results.

```
s = "".join(
    chr(((ord(c) * (109 + i)) % 127))
    for i, c in enumerate("H&X[\x14ll^A\x112D\x17jd[")
)
assert input() == s
```

Things you get to reverse in CTFs

- Source code

- C, C++, Rust, Zig compiled to machine code (most often for linux)
- Python, Java, C# compiled to bytecode
- Obfuscated JavaScript
- Web Assembly

Tools

- 🧠
 - Ghidra/Binary Ninja, strings, gdb, strace, ltrace
 - decompyle3/python-decompile3, jd-gui, ILSpy
 - <https://obf-io.deobfuscate.io/> (I learned about this way too recently)
 - wasm2wat & wasm-decompile from wabt, chrome developer tools
-

Reversing demonstration?

(The source code was not shown but it seems easier to include text than a compiled binary in a pdf)

```
gcc -O1 -o static static.c && strip static

#include <stdio.h>
#include <string.h>

void encrypt(char *f, int len) {
    for (int i = 0; i < len; i++) {
        f[i] = f[i] + i * 3;
    }
}

int main() {
    // "flag{Please do play CrateCTF. That would make me happy}";
    char enc_flag[] = {
        0x66, 0x6f, 0x67, 0x70, 0x87, 0x5f, 0x7e, 0x7a, 0x79, 0x8e, 0x83, 0x41,
        0x88, 0x96, 0x4a, 0x9d, 0x9c, 0x94, 0xaf, 0x59, 0x7f, 0xb1, 0xa3, 0xb9, 0xad,
        0x8e, 0xa2, 0x97, 0x82, 0x77, 0xae, 0xc5, 0xc1, 0xd7, 0x86, 0xe0, 0xdb, 0xe4,
        0xde, 0xd9, 0x98, 0xe8, 0xdf, 0xec, 0xe9, 0xa7, 0xf7, 0xf2, 0xb0, 0xfb, 0xf7,
        0x9, 0xc, 0x18, 0x1f, 0
    };

    char input[256];
    fgets(input, 256, stdin);
    *strchr(input, '\n') = 0;
    int input_len = strlen(input);
    encrypt(input, input_len);
```

```

    if (input_len == sizeof (enc_flag) - 1 && strcmp(input, enc_flag) == 0) {
        printf("Correct!\n");
    } else {
        printf("Wrong!\n");
    }
}

```

```
gcc -O1 -o dynamic dynamic.c && strip dynamic
```

```

#include <stdio.h>
#include <string.h>

void decrypt(char *f, int len) {
    for (int i = 0; i < len; i++) {
        f[i] = f[i] - i * 2;
    }
}

int main() {
    // "flag{omg I need to come up with another flag}";
    char flag[] = {
        0x66, 0x6e, 0x65, 0xd, 0x83, 0x79, 0x79, 0x75, 0x30, 0x5b, 0x34, 0x84,
        0x7d, 0x7f, 0x80, 0x3e, 0x94, 0x91, 0x44, 0x89, 0x97, 0x97, 0x91, 0x4e, 0xa5,
        0xa2, 0x54, 0xad, 0xa1, 0xae, 0xa4, 0x5e, 0xa1, 0xb0, 0xb3, 0xba, 0xb0, 0xaf,
        0xbe, 0x6e, 0xb6, 0xbe, 0xb5, 0xbd, 0xd5, 0
    };

    char input[256];
    fgets(input, 256, stdin);
    *strchr(input, '\n') = 0;
    int input_len = strlen(input);
    decrypt(flag, sizeof (flag) - 1);

    if (input_len == sizeof (flag) - 1 && strcmp(input, flag) == 0) {
        printf("Correct!\n");
    } else {
        printf("Wrong!\n");
    }
}

```