Security: Introduction to reverse engineering

Reverse engineering is the process of analyzing and breaking down software, hardware, or a system into its component parts in order to understand its operation, structure, or function, often in the context of recovering system information, patching, modifying, or securing software.

> Reverse engineering is the process of extracting knowledge or redesigning systems based on available, existing components. The goal of reverse engineering is to analyze the operation of a system or process, which allows for its reconstruction, understanding, and possible improvement. Examples of reverse engineering applications include analyzing software to detect errors, attempting to circumvent security measures, modifying applications, and examining communication protocols in computer networks.

Source: Wikipedia: Reverse engineering

Contents of file main2.c

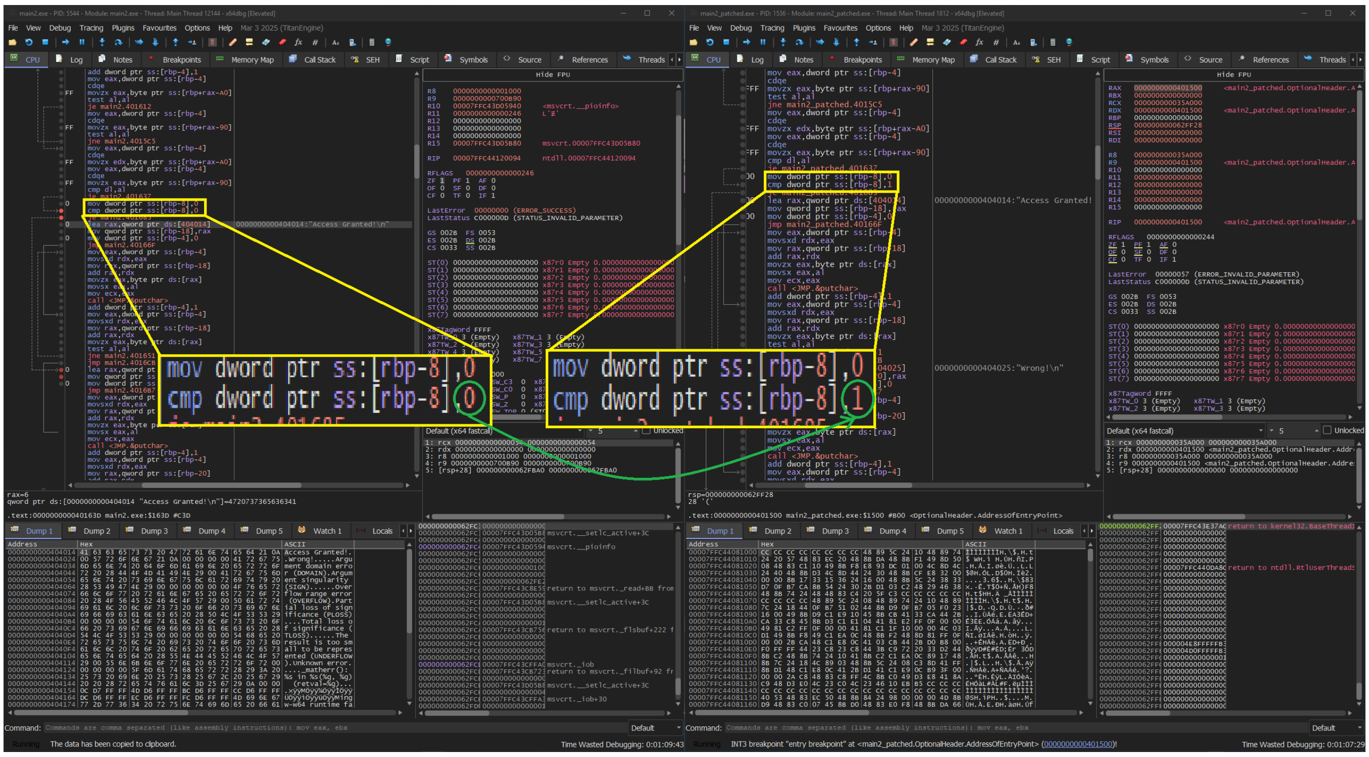
This example shows the source code for a simple program that verifies the validity of an entered license key. The program reads the entered data character by character and compares it with the reference key. Finally, depending on the result of the comparison, it displays the message “Access Granted!” or “Wrong!”.

main.c

```c
#include <stdio.h>

int main() {
    char key[100];
    char valid[] = "AAAA-Z10N-42-OK";
    int i = 0;
    int match = 1;
    int ch;

    printf("Enter license key: ");

    // Read input manually, character by character
    while (i < 100) {
        ch = getchar();
        if (ch == '\n' || ch == EOF) {
            key[i] = 0;
            break;
        }
        key[i] = ch;
        i++;
    }

    // Compare each character manually
    i = 0;
    while (valid[i] != 0 && key[i] != 0) {
        if (valid[i] != key[i]) {
            match = 0;
            break;
        }
        i++;
    }

    // Also check if lengths match
    if (valid[i] != key[i]) {
        match = 0;
    }

    // Output result
    if (match) {
        const char *msg = "Access Granted!\n";
        for (i = 0; msg[i] != 0; i++) {
            putchar(msg[i]);
        }
    } else {
        const char *msg = "Wrong!\n";
        for (i = 0; msg[i] != 0; i++) {
            putchar(msg[i]);
        }
    }
    getchar();
    return 0;
}
```

Changing one bit

Exactly, changing one bit causes the program to accept any key and still display the message “access granted.” I am aware that this program can be cracked with a simple strings tool, but that is not the point of this presentation. Below is a photo of two debuggers: on the left is the original program, and on the right is the modified version. I have marked the place in the illustration where you need to change one bit to crack the program.



```
mov dword ptr ss:[rbp-8],0
cmp dword ptr ss:[rbp-8],0
```

```
mov dword ptr ss:[rbp-8],0
cmp dword ptr ss:[rbp-8],1
```

x64dbg - An open-source x64/x32 debugger for windows.

Verification of file hashes

Using the QuickHash GUI tool, we can verify that changing a single bit completely changes the hash value.

NO	Filename	Filepath	Filehash MD5	Filesize	Description
5	main2.exe	D:\bin\	A4723B8F536E7E1CB1950D7D7D14DC4C	128,78 KiB	file without any changes
6	main2_patched.exe	D:\bin\	9D5217F3D421A2E2D69D2B67E52B8747	128,78 KiB	

https://www.quickhash-gui.org/

files:

main2.c

- source code

main2.exe

- cource code compiled for windows

main2_patched.exe

- version that will work with any license code