Security: Introduction to reverse engineering

Reverse engineering is the process of analysing and dissecting a software, device or system 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 analyse the performance of a system or process to reconstruct, understand and possibly improve it. Examples of reverse engineering applications include analysing software for bugs, trying to bypass security, modifying applications or exploring communication protocols in computer networks.

Source:Wikipedia: Reverse engineering

Contents of file main2.c

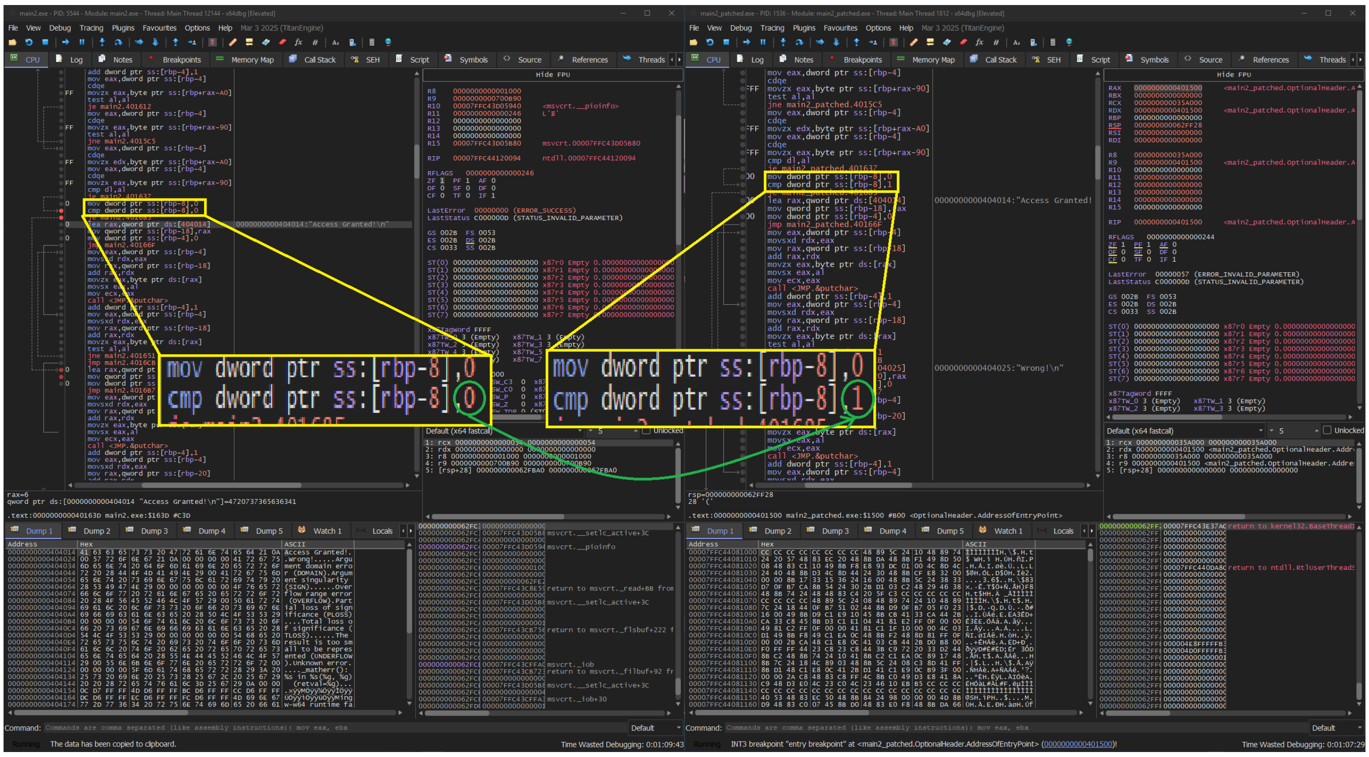
This example presents the source code of a simple program that verifies the correctness of the entered licence key. The program reads the entered data character by character and compares it with the reference key. Finally, depending on the result of the comparison, it displays the message „Access Granted!” 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

This is exactly how changing one bit causes the program to accept any key and still display the access granted message. I am aware that this program can be broken with the simple strings tool, but that is not what this presentation is about. Below is a picture of the two debuggers, on the left the original program and on the right its modified version. I have marked in the illustration the place where one bit has to be changed to break 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 a change of one bit changes the hash value completely.

NO	Filename	Filepath	Filehash MD5	Filesize	Description
5	main2.exe	D:^{3333}.	A4723B8F536E7E1CB1950D7D7D14DC4C	128.78 KiB	boot file unchanged after clean compilation
6	main2_patched.exe	D:^{33}.	9D5217F3D421A2E2D69D2B67E52B8747	128.78 KiB	file with one bit changed, size remains the same hash is completely different

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

files:

main2.c

- source code

main2.exe

- compiled code for the Windows platform

main2_patched.exe
- the version with the change that accepts any key