Security: Wstęp do inżynierii wstecznej

Inżynieria wsteczna to proces analizowania i rozkładania na części składowe oprogramowania, urządzenia lub systemu w celu zrozumienia jego działania, struktury lub funkcji, często w kontekście odzyskiwania informacji o systemie, łatania, modyfikacji lub zabezpieczania oprogramowania.

> Inżynieria wsteczna (ang. reverse engineering) to proces wydobywania wiedzy lub przeprojektowywania systemów na podstawie dostępnych, już istniejących elementów. Celem inżynierii wstecznej jest analiza działania systemu lub procesu, który pozwala na jego rekonstrukcję, zrozumienie oraz ewentualne ulepszanie. Przykłady zastosowań inżynierii wstecznej obejmują analizę oprogramowania w celu wykrycia błędów, próby obejścia zabezpieczeń, modyfikację aplikacji czy badanie protokołów komunikacyjnych w sieciach komputerowych.

Źródło: Wikipedia: Inżynieria wsteczna

Zwartość pliku main2.c

W niniejszym przykładzie zaprezentowano kod źródłowy prostego programu, który weryfikuje poprawność wprowadzonego klucza licencyjnego. Program wczytuje wprowadzone dane znak po znaku i porównuje je z kluczem referencyjnym. Na końcu, w zależności od wyniku porównania, wyświetla komunikat „Access Granted!” lub „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
```

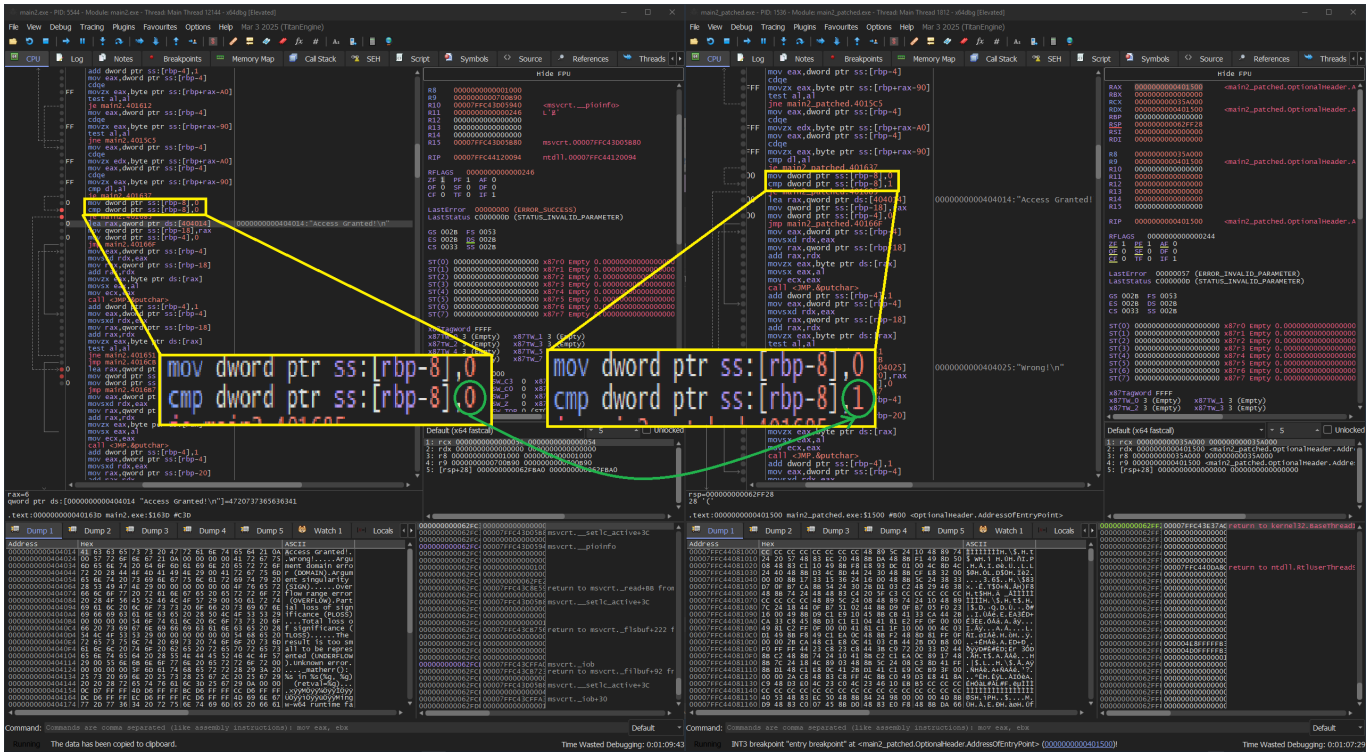
```
    i = 0;
    while (valid[i] != 0 && key[i] != 0) {
        if (valid[i] != key[i]) {
            match = 0;
            break;
        }
        i++;
    }

    // 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;
}
```

Zmiana jednego bitu

Dokładnie tak zmiana jednego bitu powoduje że program przyjmie jakikolwiek klucz a mimo to wyświetli komunikat access granted. Jestem świadomy że ten program można złamać prostym narzędziem strings, ale nie o to w tej prezentacji chodzi. Poniżej zdjęcie dwóch debuggerów po lewej program oryginalny po prawej jego wersja z zmianą. Zaznaczyłem na ilustracji miejsce w którym trzeba zmienić jedn bit aby złamać program.



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

Weryfikacja hashy plików

Za pomocą narzędzia QuickHash GUI możemy sprawdzić że zmiana jednego bitu zmienia całkowicie wartość hasha.

NO	Filename	Filepath	Filehash MD5	Filesize	Opis
5	main2.exe	D:\bin\	A4723B8F536E7E1CB1950D7D7D14DC4C	128,78 KiB	plik startowy bez zmian po czystej kompilacji
6	main2_patched.exe	D:\bin\	9D5217F3D421A2E2D69D2B67E52B8747	128,78 KiB	plik z zmienionym jednym bitem, rozmiar pozostaje ten sam hash jest kompletnie inny

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

pliki:

main2.c

- kod źródłowy

main2.exe

- skompilowany kod dla platformy Windows

main2_patched.exe

- wersja z zmianą która powoduje przyjęcie jakiegokolwiek klucza