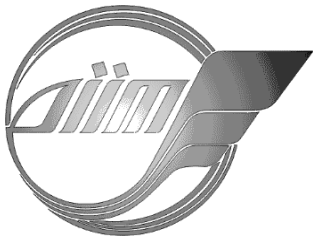
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



УДУНТ ННІ ДІІТ

Кафедра «Комп'ютерні інформаційні технології»

Лабораторна робота №2

з дисципліни «Безпека програм та даних »

на тему: «Фундаментальні основи хакерських атак»

Виконав:
студент гр.ПЗ1911
Сафонов Д.Є.
Прийняв:
Разносілін В.В.

Дніпро, 2022

Тема. Фундаментальні основи хакерських атак.
Мета. Ознайомитись з фундаментальними основами хакерських атак.

Лабораторна робота виконана на ОС linux тому використане ПЗ відрізняється від наведеного в методичних вказівках, але хід роботи збережено. Бінарні файли відредаговано з допомогою онлайн утиліти hexed.it

rodata

Створимо програму, яка буде перевіряти пароль — main1.cpp.

```
[dazzlemon@dazzlemonarch 2]$ bin/main1
Enter password: wrong
Wrong password
Enter password: not correct
Wrong password
Enter password: invalid
Wrong password
[dazzlemon@dazzlemonarch 2]$ bin/main1
Enter password: myGOODpassword
Password OK
```

Спробуємо знайти наш пароль у сегменті rodata: виконаємо extract_rodata.sh, на виході отримуємо main1.rodata.txt та main2.rodata.txt. rodata означає read only data.

#pragma data_seg

Якщо перемістити пароль до іншого сегменту (у випадку g++ це завжди .data) отримаємо main2.cpp
Пароля тепер можна дістатися трохи іншим чином — extract_data.sh — main2.data.txt

Дізасемблер

Щоб дізасемблювати main2.cpp виконаємо disassemble.sh, на виході отримуємо main2.disassembled.txt та main2.cracked.disassembled.txt

Ось наш пароль:

```
0000000000004040 <pwd>:
    4040:   6d                      insl   (%dx),%es:(%rdi)
    4041:   79 47                   jns    408a <_end+0x2a>
    4043:   4f                      rex.WRXB
    4044:   4f                      rex.WRXB
    4045:   44 70 61                rex.R jo 40a9 <_end+0x49>
    4048:   73 73                   jae    40bd <_end+0x5d>
    404a:   77 6f                   ja     40bb <_end+0x5b>
    404c:   72 64                   jb     40b2 <_end+0x52>
    404e:   0a 00                   or     (%rax),%al
```

Ось де він перевіряється:

```
    11cd:   48 8d 15 6c 2e 00 00    lea    0x2e6c(%rip),%rdx        # 4040 <pwd>
    11d4:   48 89 d6                mov    %rdx,%rsi
    11d7:   48 89 c7                mov    %rax,%rdi
    11da:   e8 71 fe ff ff          call   1050 <strcmp@plt>
    11df:   85 c0                   test   %eax,%eax
    11e1:   74 11                   je     11f4 <main+0x7b>
    11e3:   48 8d 05 2b 0e 00 00    lea    0x0e2b(%rip),%rax        # 2015 <_IO_stdin_used+0x15>
    11ea:   48 89 c7                mov    %rax,%rdi
```

Знайдемо це місце у виконуваному файлі:

```
    11cd:   48 8d 15 6c 2e 00 00    lea    0x2e6c(%rip),%rdx        # 4040 <pwd>
    11d4:   48 89 d6                mov    %rdx,%rsi
    11d7:   48 89 c7                mov    %rax,%rdi
    11da:   e8 71 fe ff ff          call   1050 <strcmp@plt>
    11df:   85 c0                   test   %eax,%eax
    11e1:   74 11                   je     11f4 <main+0x7b>
    11e3:   48 8d 05 2b 0e 00 00    lea    0x0e2b(%rip),%rax        # 2015 <_IO_stdin_used+0x15>
```

```
000011B0  FF 48 8B 15 98 2E 00 00 48 8D 45 90 BE 64 00 00
000011C0  00 48 89 C7 E8 A7 FE FF FF 48 8D 45 90 48 8D 15
000011D0  6C 2E 00 00 48 89 D6 48 89 C7 E8 71 FE FF FF 85
000011E0  C0 74 11 48 8D 05 2B 0E 00 00 48 89 C7 E8 6E FE
```

Замінимо test на xor - тепер наша програма сприймає будь який пароль, як правильний.

```
[dazzlemon@dazzlemonarch 2]$ bin/main2.cracked
Enter password: wrong password
Password OK
```

Порівняємо виконувані файли за допомогою cmp_bin.sh

```
[dazzlemon@dazzlemonarch 2]$ sh scripts/cmp_bin.sh
 4576 205  63
```

Порівняємо дізасембльовані версії виконуваних файлів за допомогою cmp_disassembled.sh

```
[dazzlemon@dazzlemonarch 2]$ sh scripts/cmp_disassembled.sh
2c2
< bin/main2:     file format elf64-x86-64
---
> bin/main2.cracked:     file format elf64-x86-64
610c610
<     11df:     85 c0                   test   %eax,%eax
---
>     11df:     33 c0                   xor    %eax,%eax
```

Symbol demangling

Створимо просту програму main3.cpp, main3.disassembled.txt, main3.disassembled.demangled.txt

```
0000000000001040 <_ZSt1lsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>:
    1040:   ff 25 c2 2f 00 00       jmp    *0x2fc2(%rip)        # 4008 <_ZSt1lsISt11char_traitsIcEERSt13basic_ostreamIcT
    1046:   68 01 00 00 00          push   $0x1
    104b:   e9 d0 ff ff ff          jmp    1020 <_init+0x20>

0000000000001050 <_ZNSt8ios_base4InitC1Ev@plt>:
    1050:   ff 25 ba 2f 00 00       jmp    *0x2fba(%rip)        # 4010 <_ZNSt8ios_base4InitC1Ev@GLIBCXX_3.4>
    1056:   68 02 00 00 00          push   $0x2
    105b:   e9 c0 ff ff ff          jmp    1020 <_init+0x20>
```

Щоб розшифрувати ці ідентифікатори скористуємося аргументом -C objdump

```
0000000000001040 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basi
    1040:   ff 25 c2 2f 00 00       jmp    *0x2fc2(%rip)        # 4008 <std::basic_ostream<char, std::char_traits<char>
    1046:   68 01 00 00 00          push   $0x1
    104b:   e9 d0 ff ff ff          jmp    1020 <_init+0x20>

0000000000001050 <std::ios_base::Init::Init()@plt>:
    1050:   ff 25 ba 2f 00 00       jmp    *0x2fba(%rip)        # 4010 <std::ios_base::Init::Init()@GLIBCXX_3.4>
    1056:   68 02 00 00 00          push   $0x2
    105b:   e9 c0 ff ff ff          jmp    1020 <_init+0x20>
```