CS-437 / Lab-3

In this homework, you are asked to create vulnerable source code samples with their exploitations and patched versions. Please read this document carefully and ensure you submit your work as asked.

Task-1: Stack Buffer Overrun

In this task, you need to create 2 samples to show Stack Buffer Overrun vulnerability. You need to find and use 2 different functions that lead to the vulnerability (1 function for each sample).

1) First, write your vulnerable program, the program needs to be a simple program which takes a command line input. Although your program is simple, it should be a program where you will not only take the input and print the output but also show it through simple scenarios to show the vulnerability in different contexts, an example of what is wanted from you can be found at the end of this document (you should not write a code like in '*Copy of Cybersecurity Practices and Applications CS437 / SEC537- Chapter 9* '-- slide 9 or any other slide&lab like in lecture, otherwise you cannot gain any points for that.).(screenshot&explanation needed)

2) After writing your program compile it with *gcc* compiler. You can use your Kali machine or your Ubuntu server to use *gcc*. Some of the versions of *gcc* have protection for memory-related vulnerabilities, you can use certain flags that can disable specific security features please search for them (you need to submit your binary). (screenshot&explanation needed)

3)Exploit the code with an appropriate input (here we do not want you to create a shell code or any other harmful payload, you need to do just a PoC). Then make sure that the exploit worked well using the *gdb* debugger, see the registers. (screenshot&explanation needed)

4)Patch the vulnerability in your code in another file. (screenshot&explanation needed)(you need to submit your patched binary)

5)Show that the fixation works by trying the same exploitation method.(screenshot&explanation needed)

WHAT YOU NEED TO SUBMIT FOR TASK-1:
1- A well-structured **PDF** report contains your screenshot step by step with their explanation.
- Which line contains the vulnerability
- Which function leads to vulnerability
- How did u patch it
- Explanation of your programs' scenario

2- 4 files for every single vulnerability an example can be found below:
- task_1_function_0_vulnerable.c
- task_1_function_0_patched.c
- Task_1_function_0_payload.txt
- Vulnerable_program_function_0 (binary)

Task-2: In this task we want you to use ChatGPT or another LLM of your choice to generate 2 vulnerable code pieces which could have the following vulnerabilities (choose 2 of them):
- Heap overflow
- Buffer overflow
- Integer overflow
- SQLi

- Cmd Injection
- Null point dereference
- String Format Injection Vulnerability

Note: You are not going to mention that you want a vulnerable **C** code in your prompt, instead you will force him to give you that could contain any of the vulnerabilities above by confusing it or finding a prompt that works. We know that LLMs can give vulnerable codes while normal coding requests made. We want you to trigger this behaviour by finding a good prompt which does not specifically ask the desired vulnerability.
What you need to submit is:

- Link to your conversation (Most of the LLMs include chat sharing if you cannot share it you need to include all of the screenshots of your conversation)
- Your prompt in written format.
- Explanation about why the given code is vulnerable (need to mention about variable, line in other words be specific)
- Code itself.

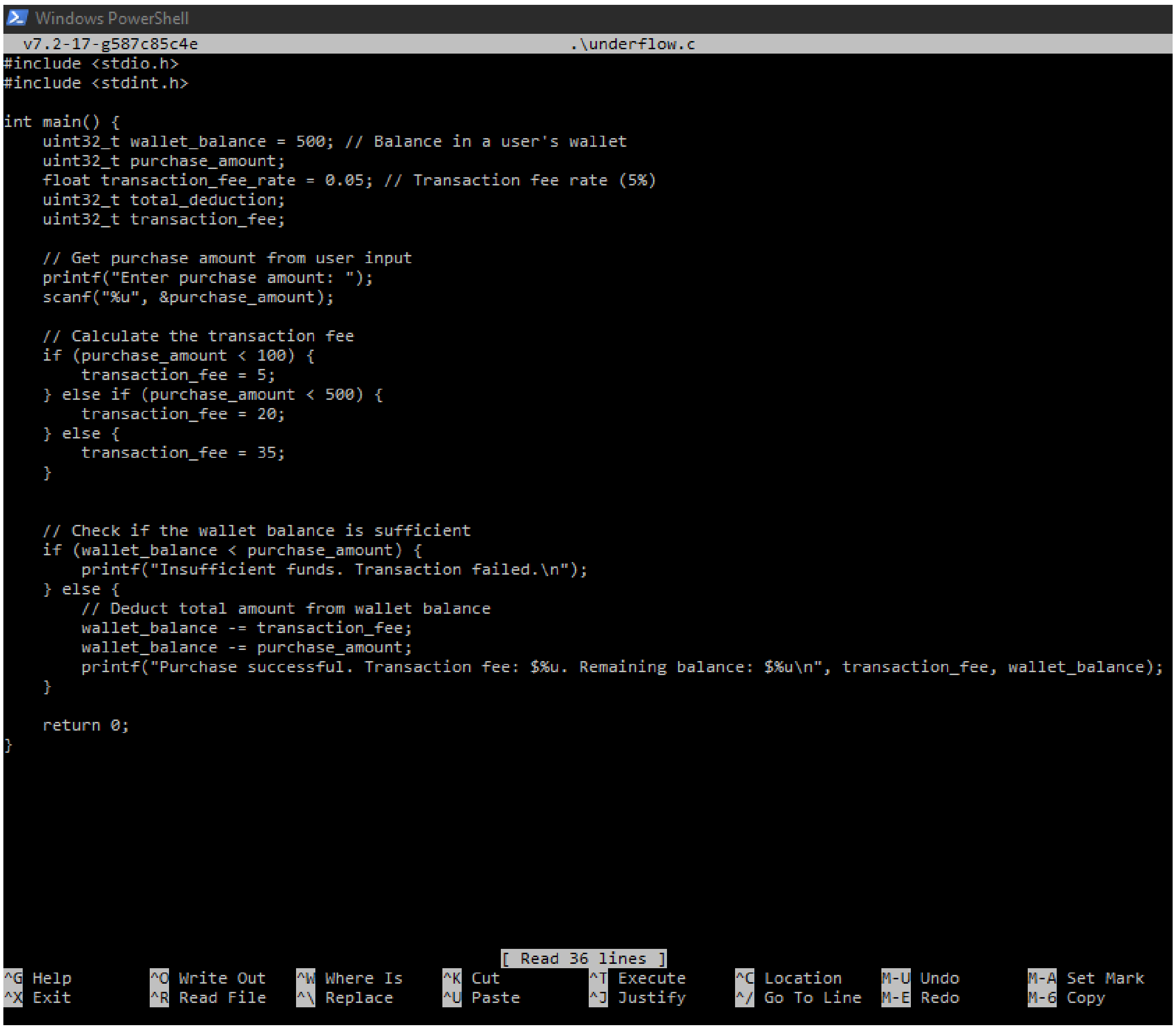
ZIP your report and files before submitting!!! 2 FOLDER AS TASK1, TASK2

So lastly your submission should contain the followings:

```
|
|---TASK1
|     |
|     |---- 4 Files for every single vulnerability with appropriate name.
|
|---TASK2
|    |
|    |----The code from LLM.
|
|--- Report.pdf
```

There will be point deduction if you do not respect the above structure.

An example program for integer underflow:

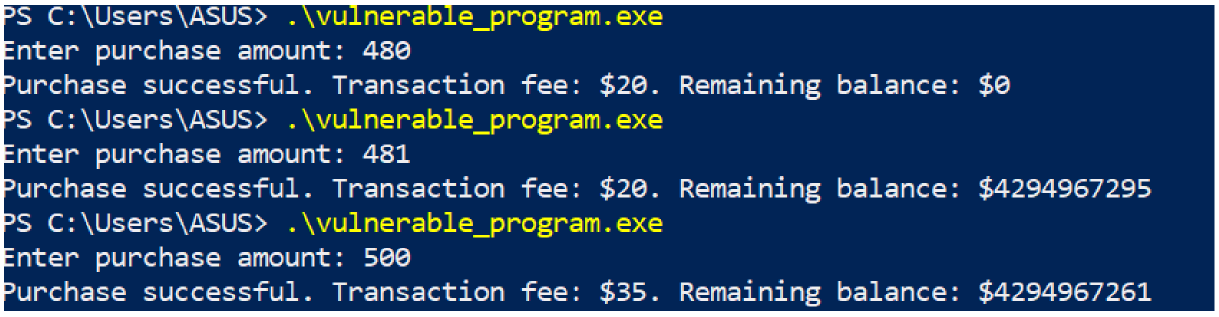


```
Windows PowerShell
 v7.2-17-g587c85c4e                                      .\underflow.c
#include <stdio.h>
#include <stdint.h>

int main() {
    uint32_t wallet_balance = 500; // Balance in a user's wallet
    uint32_t purchase_amount;
    float transaction_fee_rate = 0.05; // Transaction fee rate (5%)
    uint32_t total_deduction;
    uint32_t transaction_fee;

    // Get purchase amount from user input
    printf("Enter purchase amount: ");
    scanf("%u", &purchase_amount);

    // Calculate the transaction fee
    if (purchase_amount < 100) {
        transaction_fee = 5;
    } else if (purchase_amount < 500) {
        transaction_fee = 20;
    } else {
        transaction_fee = 35;
    }


    // Check if the wallet balance is sufficient
    if (wallet_balance < purchase_amount) {
        printf("Insufficient funds. Transaction failed.\n");
    } else {
        // Deduct total amount from wallet balance
        wallet_balance -= transaction_fee;
        wallet_balance -= purchase_amount;
        printf("Purchase successful. Transaction fee: $%u. Remaining balance: $%u\n", transaction_fee, wallet_balance);
    }

    return 0;
}

                                                    [ Read 36 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy
```

As you can see the program that you write will be simple with a simple scenario. In this example we just use a transaction scenario with some additional check such as defining transaction fees and only checking if the balance is enough to pay the transaction fee but did not check the total amount of purchase and its transaction fee.

The code is vulnerable to integer underflow, it deducts the transaction fee and purchase amount from the wallet_balance without first checking the sum of these deductions doesn't exceed the available balance.
In C, subtracting a larger number from a smaller one in an unsigned integer (uint32_t in this case) leads to an underflow, in other words, the variable rolls over to a very high value instead of going negative, as it can't represent negative numbers. The code only compares wallet_balance with purchase_amount and doesn't consider the additional transaction_fee in its sufficiency check.

To exploit the vulnerability as the purchase amount is checked we cannot give more than 500. But any value between 480 and 500 will result in underflow vulnerability.



```
PS C:\Users\ASUS> .\vulnerable_program.exe
Enter purchase amount: 480
Purchase successful. Transaction fee: $20. Remaining balance: $0
PS C:\Users\ASUS> .\vulnerable_program.exe
Enter purchase amount: 481
Purchase successful. Transaction fee: $20. Remaining balance: $4294967295
PS C:\Users\ASUS> .\vulnerable_program.exe
Enter purchase amount: 500
Purchase successful. Transaction fee: $35. Remaining balance: $4294967261
```

Please be careful to only show the asked vulnerabilities to create the samples. Your code should not contain any other vulnerabilities.

Links and materials that can be useful:

1- SuCourse - *CS437 SLIDES*
2- https://www.youtube.com/watch?v=1S0aBV-Waeo to make a recap and see how the buffer overflow vulnerability can be used to run malicious payloads.
3- https://stackoverflow.com/questions/2340259/how-to-turn-off-gcc-compiler-optimization-to-enable-buffer-overflow