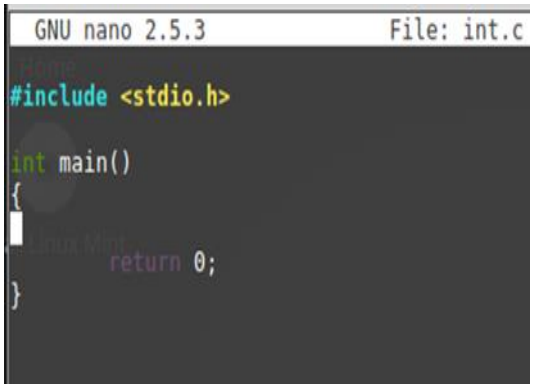
VULNERABILTIIES

BUFFER OVERFLOW

SCREWING WITH NUMBERS

You will make a program to calculate your pay! This is an example of integer overflow and why we need to take this exploit into consideration. We will program this lab in C.



STEP 1

Open a linux terminal. I am using Linux Mint 18.1 64 bit. In the command line, type:

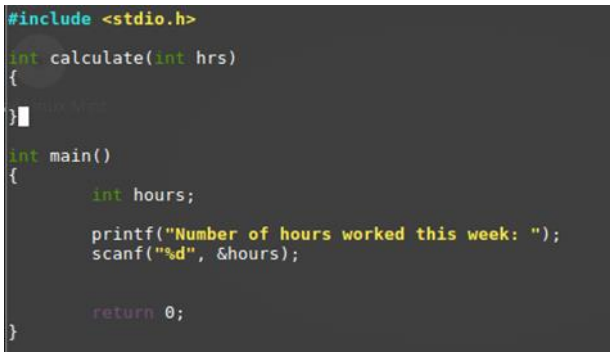
- `nano int.c`

STEP 2

Write out the basics of main and include the header <stdio.h>

STEP 3

Declare an integer named `hours`. Use a printf statement to ask the user for number of hours worked and scan that into `hours` using scanf (not secure, but for this little exercise it is fine).



```
#include <stdio.h>

int calculate(int hrs)
{

}

int main()
{
        int hours;

        printf("Number of hours worked this week: ");
        scanf("%d", &hours);


        return 0;
}
```

STEP 4

Go above main to start a new function. Let's name it "calculate" with an integer as the parameter. This function will return an integer.

```c
#include <stdio.h>

int calculate(int hrs)
{
        int pay;
        int hrlypay;

        printf("What is the pay per hour? ");
        scanf("%d", &hrlypay);
        pay = hrs * hrlypay;

        return pay;
}
```

STEP 5

Define two new variables in the calculate function: `pay` and `hrlypay`. Use a printf statement to ask the user what the pay per hour is and scan that into `hrlypay` (again, fgets is much better than scanf but this will suffice). Write an equation that calculates pay from `hrs` multiplied with `hrlypay`. Return `pay` to main.

STEP 6

Back in the main function, define a new variable named `result` that will store the returned value from the calculate function. We will send `hours` to the calculate function and make this call equal to `result`. Finally, use a printf statement to print out the `pay` variable.

```c
int main()
{
        int hours;
        int result;

        printf("Number of hours worked this week: ");
        scanf("%d", &hours);

        result = calculate(hours);

        printf("Your pay is: %d\n", result);


        return 0;
}
```

STEP 7

Hit ctrl-x and hit y. This will save it. Now type "gcc int.c" and hit enter to compile the file. If it gives an error, then go back and fix your file. It will most likely be spelling mistakes.

STEP 8

To run the file, type "./a.out" then press enter.

OUTPUT

This is how we expect the program to go.

```
mint@mint ~ $ gcc int.c
mint@mint ~ $ ./a.out
Number of hours worked this week: 10
What is the pay per hour? 12
Your pay is: 120
mint@mint ~ $
```

But what if we intentionally overflow it…?

```
mint@mint ~ $ gcc int.c
mint@mint ~ $ ./a.out
Number of hours worked this week: 3294832843248394823984023
What is the pay per hour? 9382934832904890328409328490832
Your pay is: 1
mint@mint ~ $ ./a.out
Number of hours worked this week: 291999991999999999999999999999999999
What is the pay per hour? 29
Your pay is: -29
mint@mint ~ $ ./a.out
Number of hours worked this week: 200
What is the pay per hour? 22222222222983892890398098982973816265462137289174892
Your pay is: -200
mint@mint ~ $
```

ADD BOUNDS CHECKING

STEP 1

Open your file back up

a. `nano int.c`

STEP 2

Go into main and make `hours = -1`. Wrap a while loop around the printf and scanf statements with the condition:

`hours < 0 || hours > 150`

```
int main()
{
        int hours = -1;
        int result;

        while(hours < 0 || hours > 150)
        {
                printf("Number of hours worked this week: ");
                scanf("%d", &hours);
        }

        result = calculate(hours);
```

STEP 3

```
int calculate(int hrs)
{
        int pay;
        int hrlypay = 5;

        while(hrlypay < 6 || hrlypay > 100)
        {
                printf("What is the pay per hour? ");
                scanf("%d", &hrlypay);
        }
        pay = hrs * hrlypay;
```

Go into the calculate function and make `hrlypay = 5`. Wrap a while loop around the printf and scanf statements with the condition:

`hrlypay < 6 || hrlypay > 100`

STEP 4

These conditions will make the program repeat the question if the wrong numbers are entered. Note, this is not sufficient bounds checking in case of illegal characters. This only prevents the user from entering too many NUMBERS. This will NOT catch letters. It is only an example.

STEP 5

Hit ctrl-x then y to save it. Type "gcc int.c" then hit enter to compile. Test the program with different numbers to make sure the while loops are correct. We get more reasonable numbers when we can control what the user can enter.

```
mint@mint ~ $ gcc int.c
mint@mint ~ $ ./a.out
Number of hours worked this week: 200
Number of hours worked this week: -3
Number of hours worked this week: 49
What is the pay per hour? -2
What is the pay per hour? 200
What is the pay per hour? 44
Your pay is: 2156
mint@mint ~ $
```

ANSWER THE FOLLOWING QUESTIONS

QUESTION 1

What would make a better bounds checking than the one used here?

QUESTION 2

What services would need to take integer overflow into consideration when using software?

QUESTION 3

What makes fgets() better than scanf()? What parameters go in the parenthesis of fgets() to make it work?

WHAT TO SUBMIT

In a word document, submit screenshots of the output from your code and the answers to the questions. Also submit your code in a notepad text file.