



# Programming Presentation

By: Bandr AlSwyan



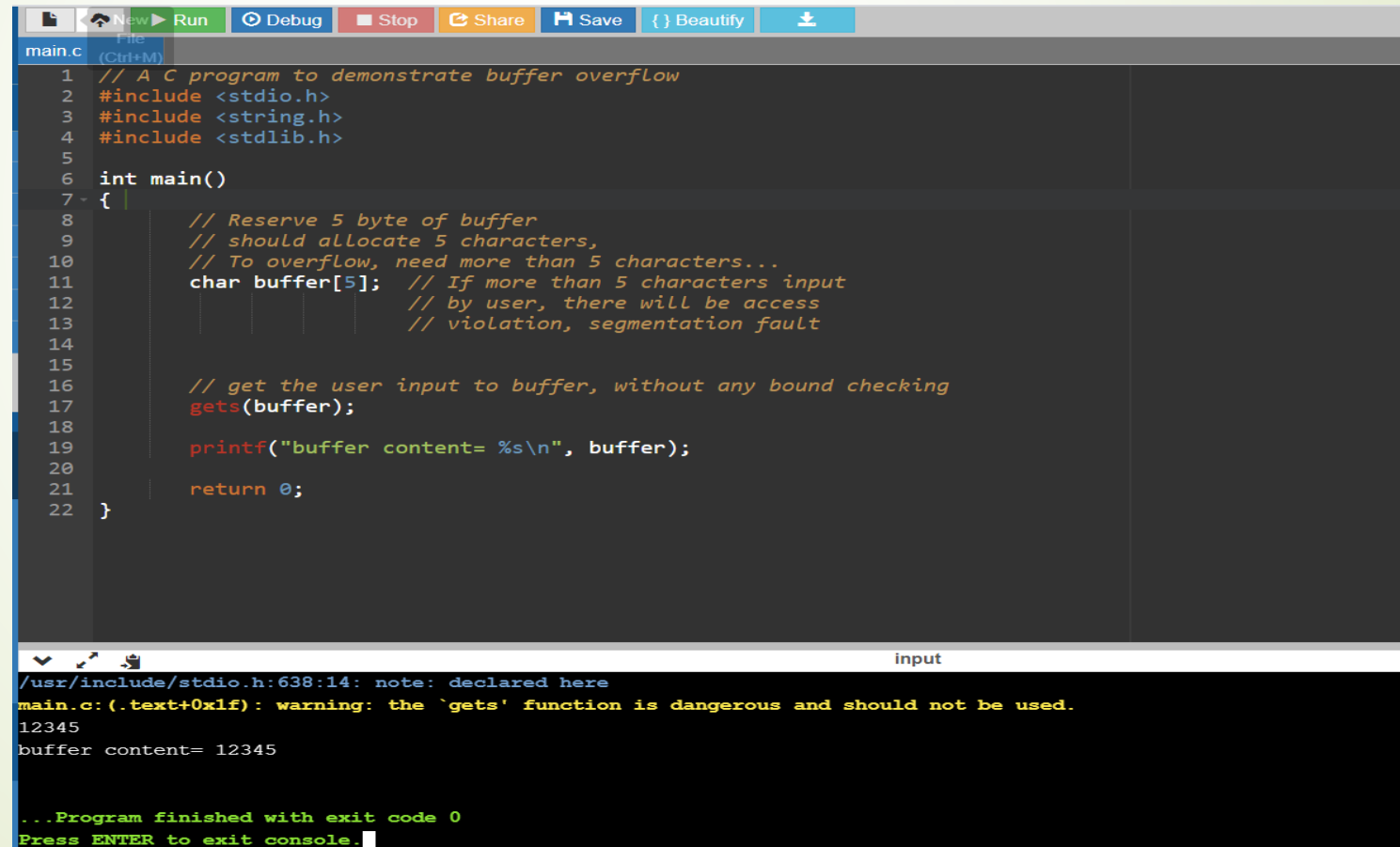
# Buffer Overflows Attack

From my essay:

- **Introduction:** A buffer overflow occurs when a programmer writes more data in memory that exceeds its space that is allocated in a given program.
- A program's memory is divided into different segments, including stack, heap, and buffer.
- The buffer is the memory storage region that holds a program's data as it is transferred to another location.
- The buffer overruns occur when a given program attempts to fill a buffer with more data that was not designed to hold.

# Buffer Overflows Attack Simulation

We used the below C code for showing the buffer overflow attack. We used “**gets**” method to get the input from the user by console. As you see in the first run of the program in the first image, we entered “12345” as the input and because its size is less than or equal to buffer size, this code runs successfully.



```
main.c
1 // A C program to demonstrate buffer overflow
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 int main()
7 {
8     // Reserve 5 byte of buffer
9     // should allocate 5 characters,
10    // To overflow, need more than 5 characters...
11    char buffer[5]; // If more than 5 characters input
12                  // by user, there will be access
13                  // violation, segmentation fault
14
15
16    // get the user input to buffer, without any bound checking
17    gets(buffer);
18
19    printf("buffer content= %s\n", buffer);
20
21    return 0;
22 }
```

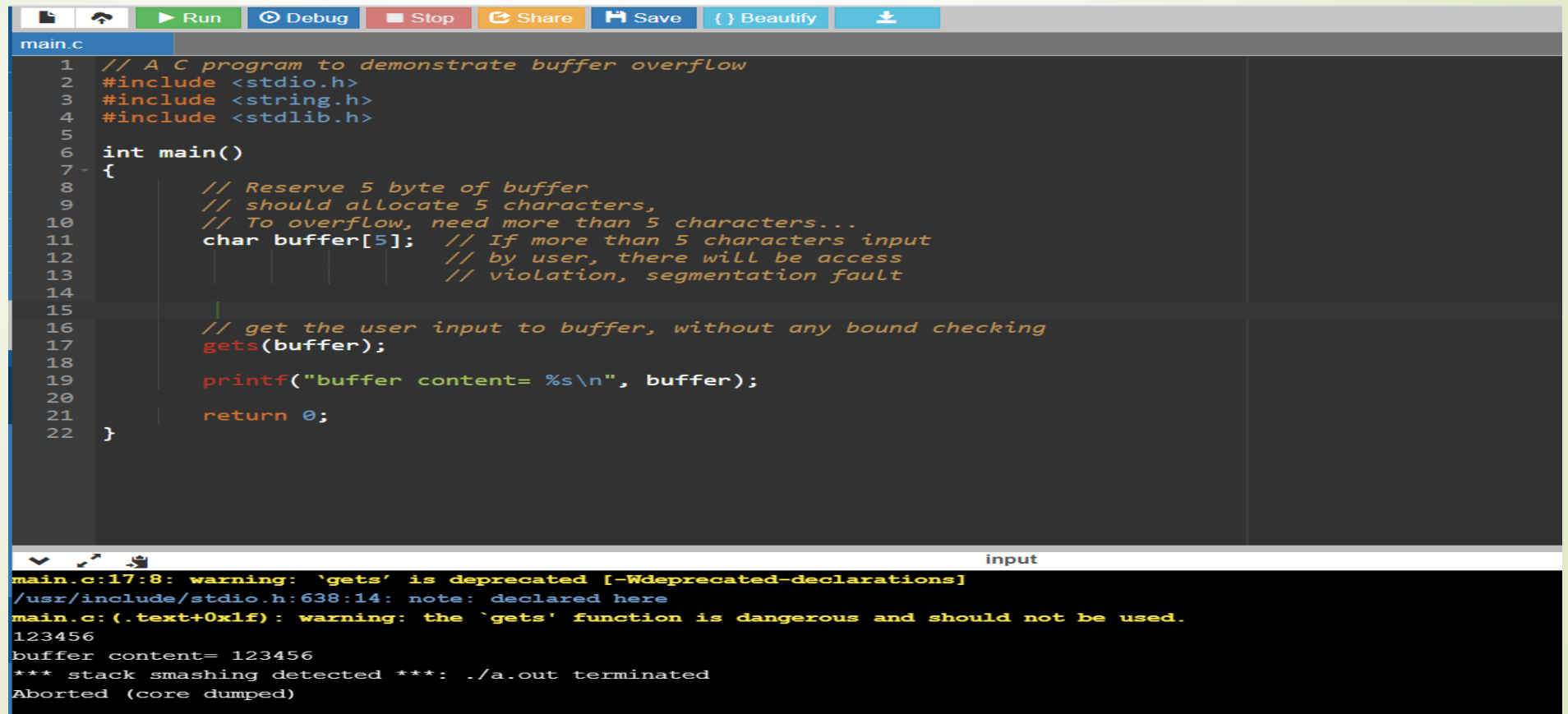
input

```
/usr/include/stdio.h:638:14: note: declared here
main.c:(.text+0x1f): warning: the `gets' function is dangerous and should not be used.
12345
buffer content= 12345

...Program finished with exit code 0
Press ENTER to exit console.
```

# Buffer Overflows Attack Simulation

In the second run, we entered a larger input. It is larger than the size of the buffer and as we see, it goes to the Stack Smashing Error. It means that we allocate characters more than buffer size and compiler prevent doing this by this error. As you see the compiler aborted the program and doesn't let the other lines to run and the next lines (like printf) never runs in this scenario.



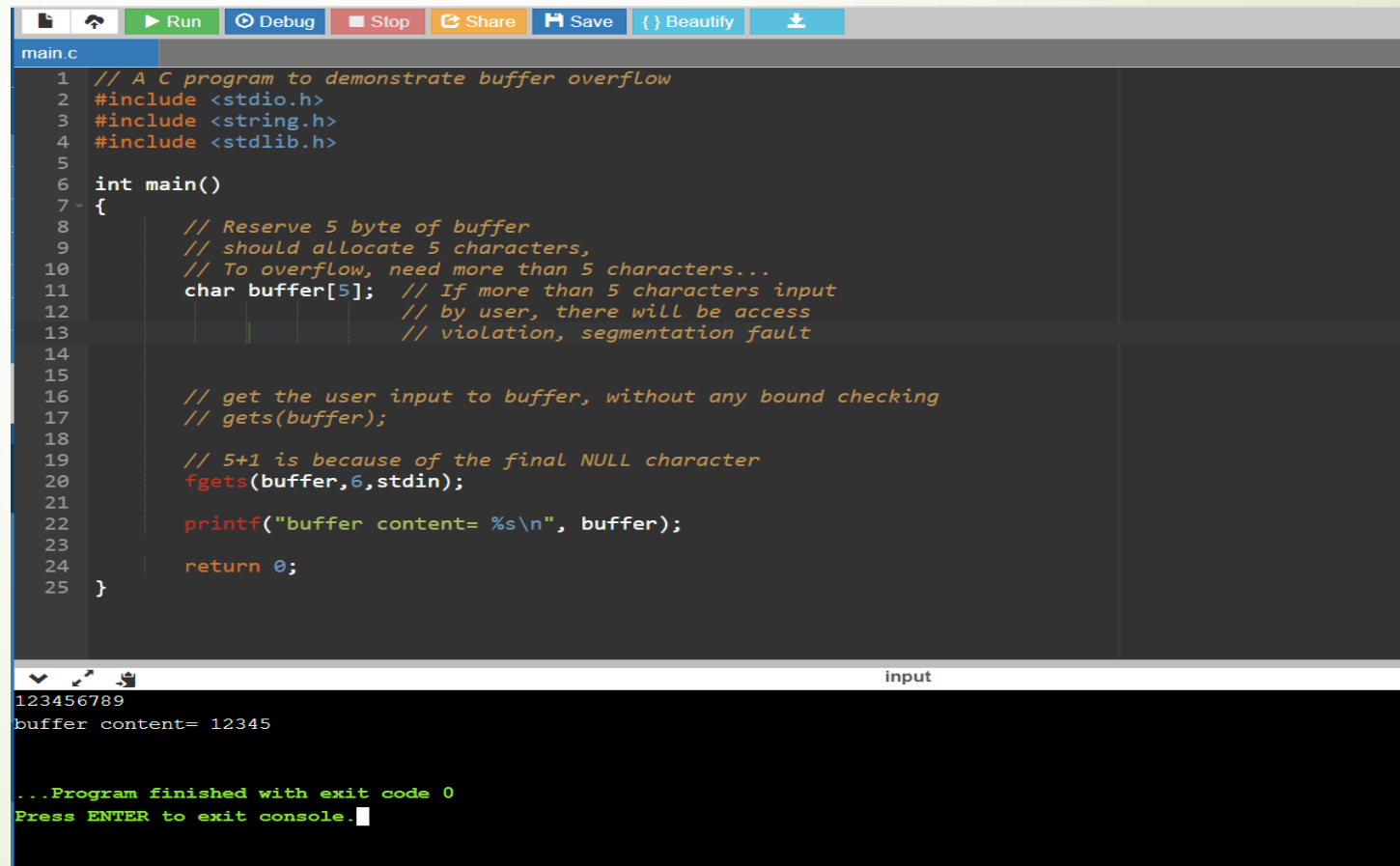
```
main.c
1 // A C program to demonstrate buffer overflow
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 int main()
7 {
8     // Reserve 5 byte of buffer
9     // should allocate 5 characters,
10    // To overflow, need more than 5 characters...
11    char buffer[5]; // If more than 5 characters input
12                    // by user, there will be access
13                    // violation, segmentation fault
14
15
16    // get the user input to buffer, without any bound checking
17    gets(buffer);
18
19    printf("buffer content= %s\n", buffer);
20
21    return 0;
22 }
```

input

```
main.c:17:8: warning: 'gets' is deprecated [-Wdeprecated-declarations]
/usr/include/stdio.h:638:14: note: declared here
main.c:(.text+0x1f): warning: the 'gets' function is dangerous and should not be used.
123456
buffer content= 123456
*** stack smashing detected ***: ./a.out terminated
Aborted (core dumped)
```

# Buffer Overflows Attack Simulation

So, what is the solution? For solving this problem different programming languages provide some secure methods which strongly recommended. For example, we can use **fgets** instead of gets. By calling fgets we pass the appropriate size of the input and if the entered input was larger than that, it will cut it and use only the first matched part. You can see this process in the following code:

The image shows a code editor window with a dark theme. The title bar of the editor shows icons for Run, Debug, Stop, Share, Save, and Beautify. The code is in a file named 'main.c'. It includes standard headers and defines a main function. Inside main, a buffer of size 5 is declared. Comments explain that to overflow, more than 5 characters are needed. The code uses fgets to read input from stdin, specifying a size of 6 (5 for the buffer plus 1 for the null terminator). It then prints the buffer content. The terminal output at the bottom shows the input '123456789' and the output 'buffer content= 12345', demonstrating that the input was truncated to fit the buffer. The program then finishes with exit code 0.

```
1 // A C program to demonstrate buffer overflow
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 int main()
7 {
8     // Reserve 5 byte of buffer
9     // should allocate 5 characters,
10    // To overflow, need more than 5 characters...
11    char buffer[5]; // If more than 5 characters input
12                   // by user, there will be access
13                   // violation, segmentation fault
14
15
16    // get the user input to buffer, without any bound checking
17    // gets(buffer);
18
19    // 5+1 is because of the final NULL character
20    fgets(buffer,6,stdin);
21
22    printf("buffer content= %s\n", buffer);
23
24    return 0;
25 }
```

input

123456789  
buffer content= 12345

...Program finished with exit code 0  
Press ENTER to exit console.

# Buffer Overflows Attack Simulation

Below is the list of such functions and, if they exist, their safe equivalents:

gets() -> fgets() - read characters

strcpy() -> strncpy() - copy content of the buffer

strcat() -> strncat() - buffer concatenation

sprintf() -> snprintf() - fill buffer with data of different types

(f)scanf() - read from STDIN

getwd() - return working directory

realpath() - return absolute (full) path





# Cross-Site Scripting Attack

From my essay:

- **Introduction:** Cross-Site Scripting (XSS) is a website attack where malicious codes get injected into existing websites.
- This type of cyber-attack is also known as a client-side code injection attack.
- In the event of an XSS occurrence, the attacker sends malicious codes and scripts to the user's website. The method aims to run and execute the codes on the end user's device
- The process utilizes unvalidated inputs to change the outputs from the unsuspecting user. The common avenues where XSS attacks occur include message boards, forums, and web pages that permit comments

# Cross-Site Scripting Attack Simulation

In this task we ask user to enter a query in our python code. The query will pass to this website:

<https://xss-game.appspot.com/level1/frame>



The screenshot shows a web browser window with the address bar displaying `xss-game.appspot.com/level1/frame`. The page content features the logo **FourOrFour** in a stylized purple and green font. Below the logo is a search interface consisting of a text input field with the placeholder text "Enter query here..." and a "Search" button.



# Cross-Site Scripting Attack Simulation

Every input in our python code will pass to this website and placed inside the input box. The problem is that user can add a script in the input box and instead of running a query, running a code in our website and attack to the private information like cookies.

As you see in the below image, we run the python code and pass a simple script into it:

```
<script>alert(123)</script>
```

Python code pass this to the website and website thinks this is a simple search query and run it. But by running this code the script will run in the system and do what we want in the website.



```
1 import webbrowser
2
3 if __name__ == "__main__":
4
5     value = input("Please enter a query:\n")
6     # <script>alert(123)</script>
7     value = "https://xss-game.appspot.com/level1/frame?query="+value
8
9     webbrowser.open(value, new=2)
```

main

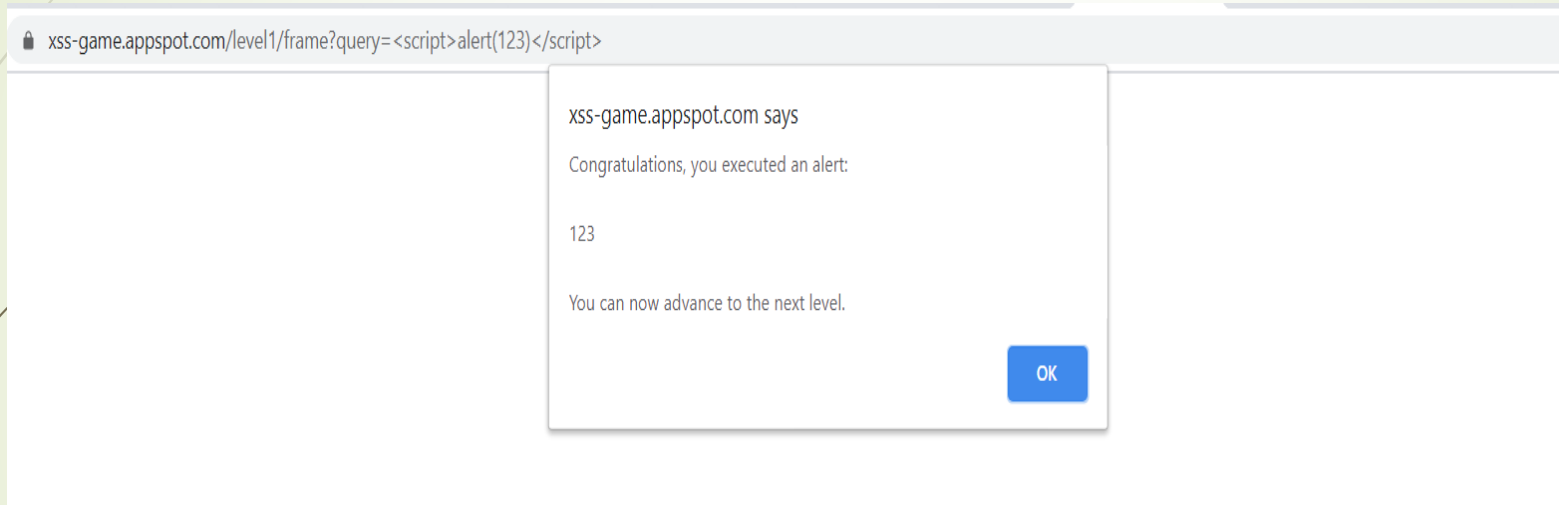
C:\Users\User\PycharmProjects\XSS\venv\Scripts\python.exe C:/Users/User/PycharmProjects/XSS/main.py

Please enter a query:

<script>alert(123)</script>

# Cross-Site Scripting Attack Simulation

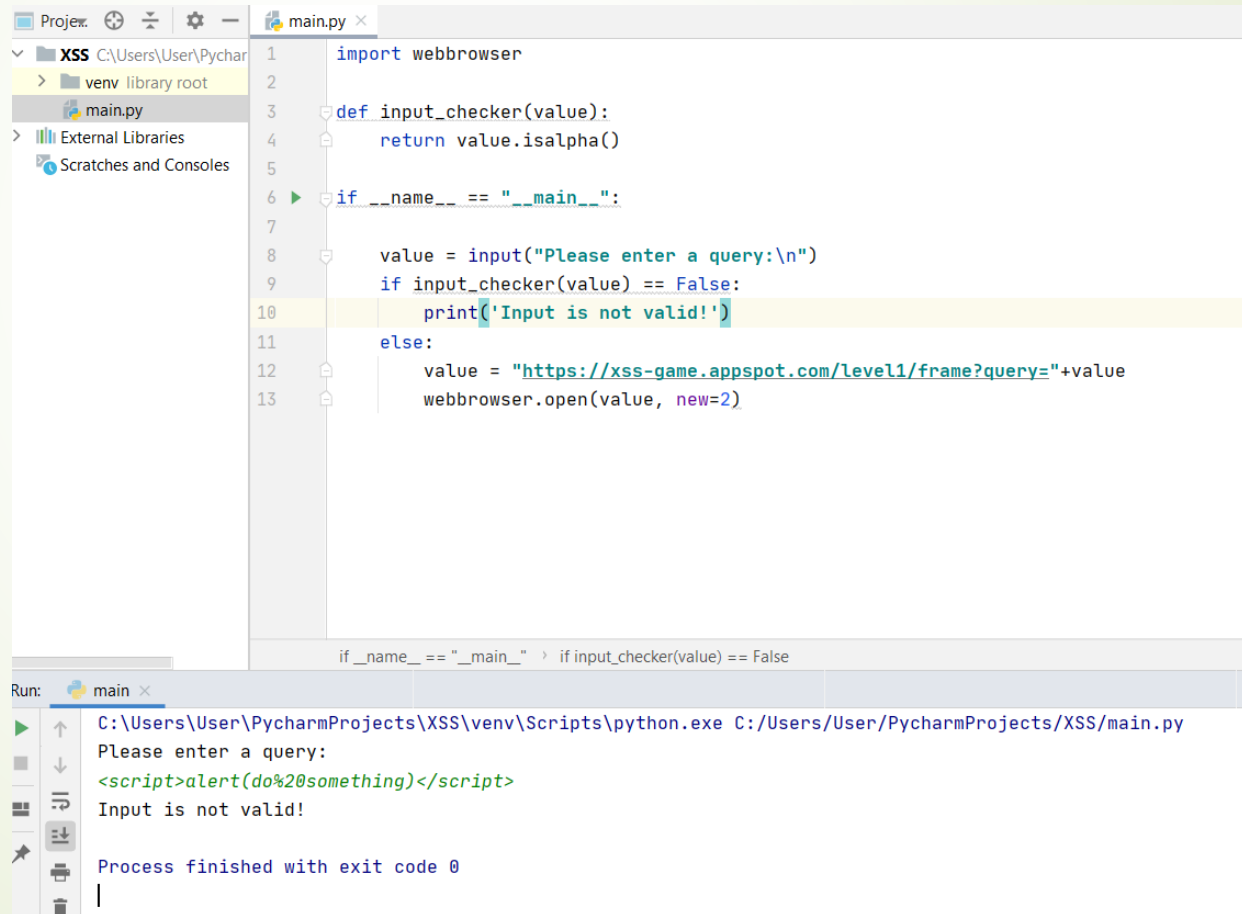
As you see in below image, the website will show the alert box with our message ("123").



# Cross-Site Scripting Attack Simulation

There are two things to correct this attack:

1- Input validation: We need to decide which character set is valid for the input box and restrict the inputs, for example we can restrict the input data to only the uppercase and lowercase letters.



```
1 import webbrowser
2
3 def input_checker(value):
4     return value.isalpha()
5
6 if __name__ == "__main__":
7
8     value = input("Please enter a query:\n")
9     if input_checker(value) == False:
10         print('Input is not valid!')
11     else:
12         value = "https://xss-game.appspot.com/level1/frame?query="+value
13         webbrowser.open(value, new=2)
```

Run: main ×

```
C:\Users\User\PycharmProjects\XSS\venv\Scripts\python.exe C:/Users/User/PycharmProjects/XSS/main.py
Please enter a query:
<script>alert(do%20something)</script>
Input is not valid!

Process finished with exit code 0
```

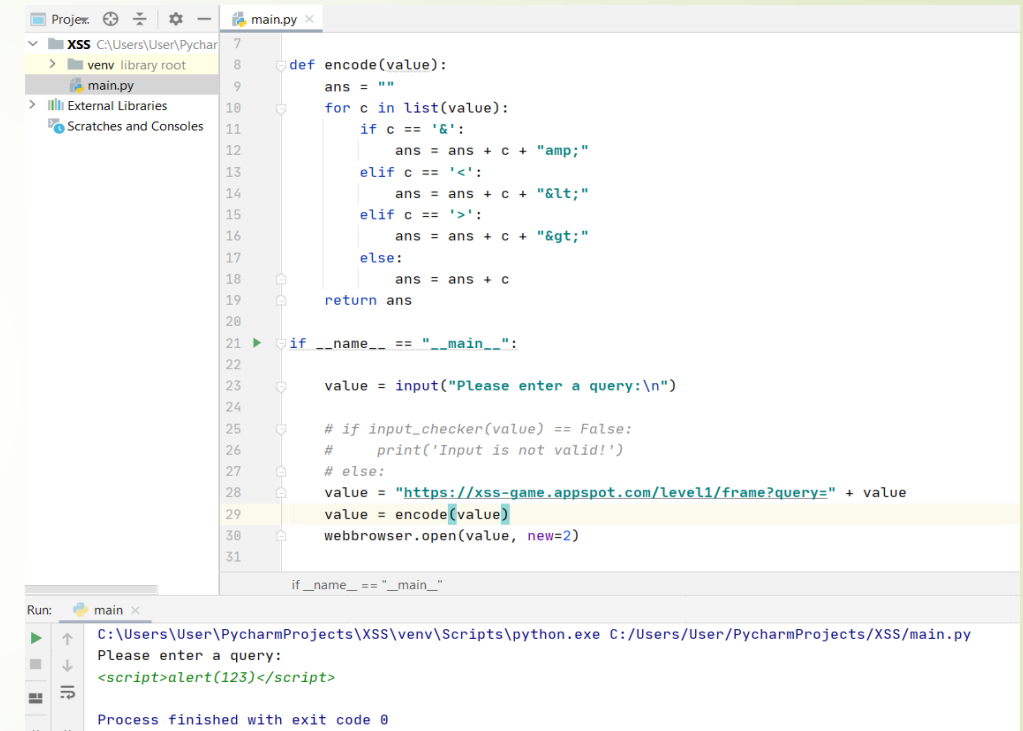
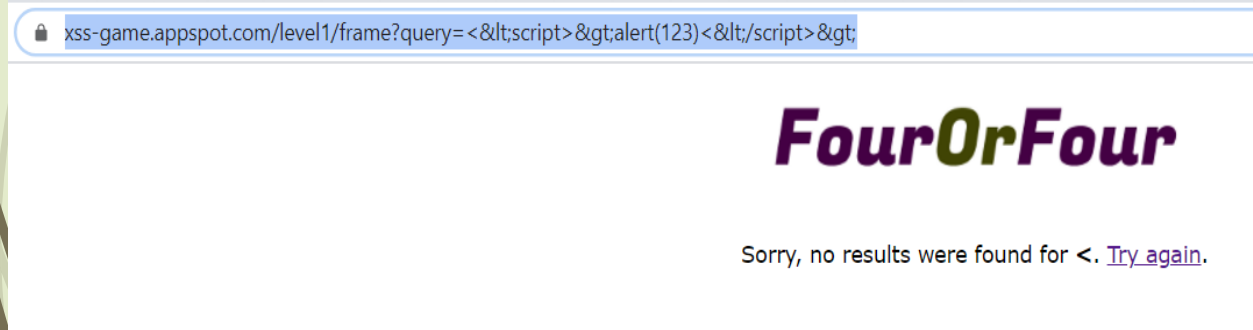
# Cross-Site Scripting Attack Simulation

Encode source before displaying in the context. For example, we can do these changes:

& → &amp;  
< → &lt;  
> → &gt;

As you see in the images next, we do the encoding and by running the program with previous input we will not see the alert and we will only have a simple search even by receiving a script as input.

The result are below:





# SQL Injection Attack

From my essay:

- **Introduction:** The SQL Injection attack depends on malicious SQL code for accessing data, which was not previously intended for display.
- The attack occur with the help of backend database manipulation, and the information displayed may include sensitive company information, including the lists of private customers and their personal data.
- The attackers insert arbitrary SQL into their chosen companies databases to either delete, modify, or copy database content.

# SQL Injection Attack Simulation

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution. As an example of this type of attack we can take a look at the following example.

- If we have the following code for handling email and password for login:

```
SELECT *  
FROM users  
WHERE email = 'user@email.com'  
AND pass = 'password' LIMIT 1
```



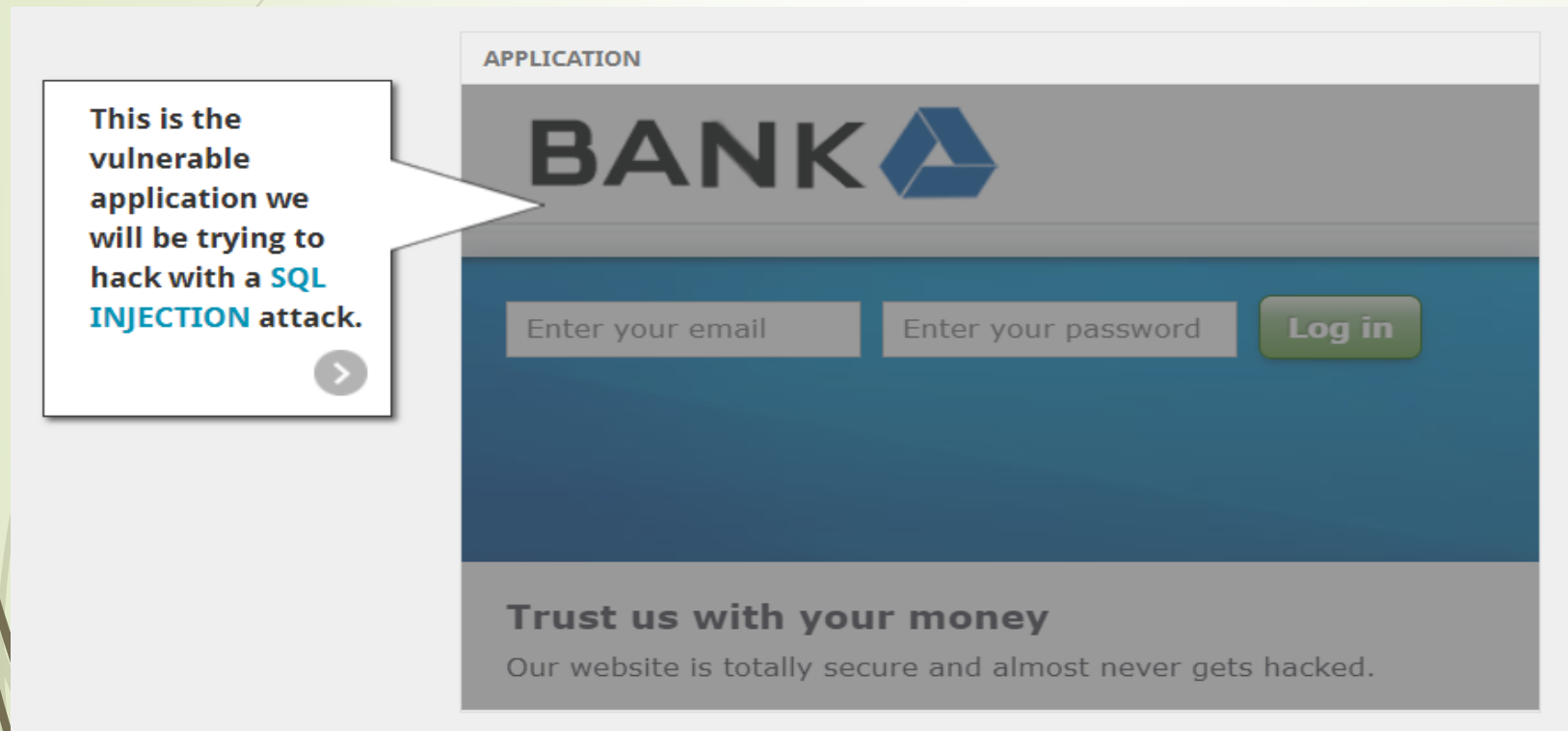
# SQL Injection Attack Simulation

- If system does not check the password, we can enter following password as the password of any email address:
  - ' or 1=1—
- If you replace password with the above, we will have:
  - ```
SELECT *  
FROM users  
WHERE email = 'user@email.com'  
AND pass = " or 1=1--" LIMIT 1
```

    - And by running this code by sql you can easily break the password without having to guess the password.

# SQL Injection Attack Simulation

- ▶ You can see the implementation here:  
<https://www.hacksplaining.com/exercises/sql-injection>



# SQL Injection Attack Simulation

Enter the following credentials and click "Log in":

**Email**

user@email.com

**Password**

' or 1=1--



## APPLICATION

**BANK** 

An unexpected error occurred.

user@email.com

.....

Log in

' or 1=1--

**Trust us with your money**

Our website is totally secure and almost never gets hacked.

## LOGS

```
rendering login page
Checking supplied authentication details for
user@email.com.
Finding user in database.
Authentication details confirmed,
establishing session for this user.
Logging out of session.
...done.
```

## CODE

```
SELECT *
FROM users
WHERE email = 'user@email.com'
AND pass = '' or 1=1--' LIMIT 1
```

# SQL Injection Attack Simulation

And we are in! We successfully gained access to the application without having to guess the password, using **SQL INJECTION**.



## APPLICATION

**BANK** 

Log out

### Bank Accounts

| Account  | Available Balance | Present Balance |
|----------|-------------------|-----------------|
| Checking | \$16,100.44       | \$16,100.44     |
| Savings  | \$50,895.96       | \$50,895.96     |

Transfer Funds!

## LOGS

```
Finding user in database.  
Authentication details confirmed,  
establishing session for this user.  
Logging out of session.  
...done.  
Finding user in database.  
Authentication details confirmed,  
establishing session for this user.
```

## CODE

```
SELECT *  
FROM users  
WHERE email = 'user@email.com'  
AND pass = '' or 1=1--' LIMIT 1
```



Thank you