Questions: XSS

Q1. What is the most basic indication that a cross-site scripting attack might be possible on a website?

The most basic indication that a cross-site scripting (XSS) attack might be possible on a website is when user inputs are not properly sanitized or validated before they are displayed on the website. This means that if a website takes user-generated content and directly inserts it into web pages without proper filtering or encoding, it can create an opening for an XSS attack.

Q2. What is one example of language that can be leveraged in an XSS attack?

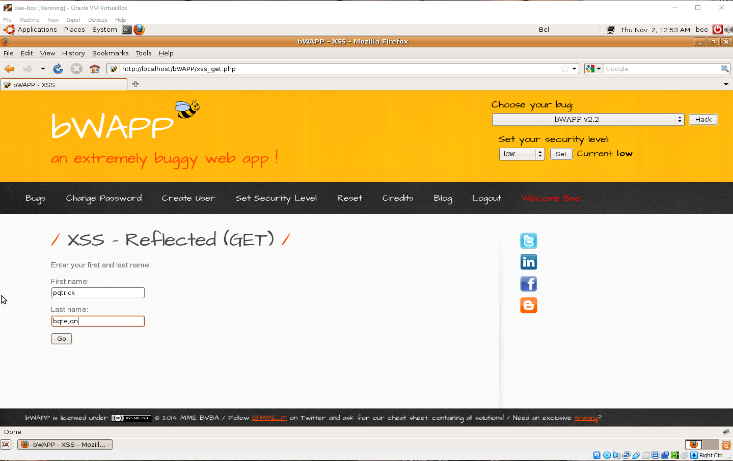
One example of a language that can be leveraged in an XSS attack is JavaScript. In an XSS attack, malicious JavaScript code is injected into a web page and then executed in the context of victim’s browser. This can allow an attacker to steal user data, manipulate the appearance of the website, or perform other malicious actions on behalf of the user.

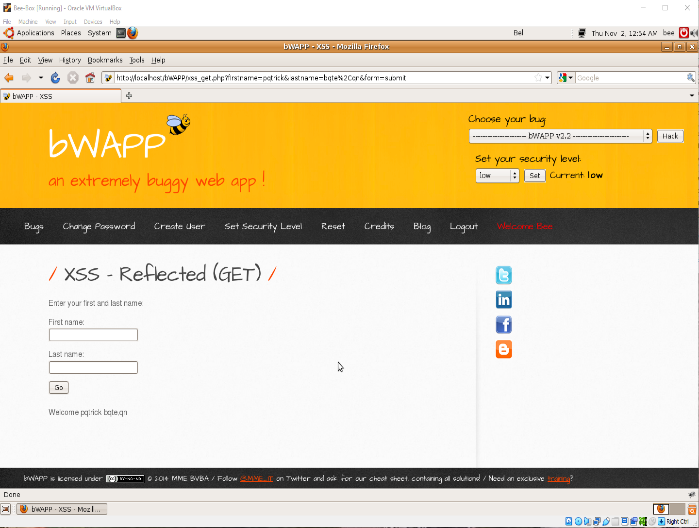
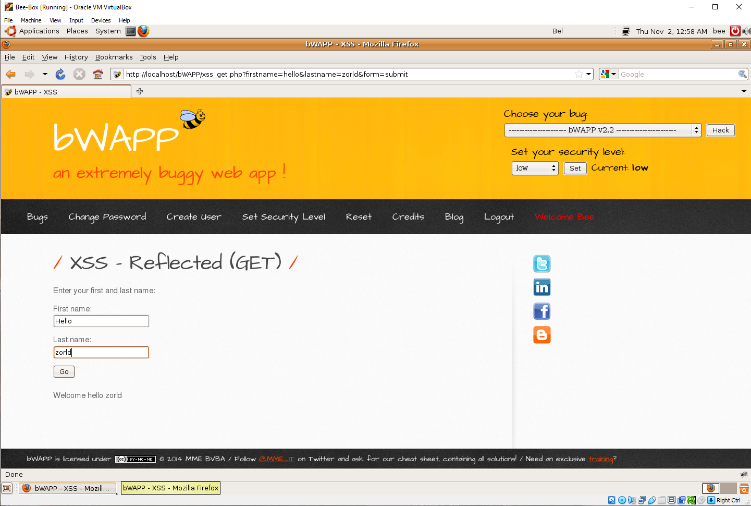
Q3. If the languages can be leveraged in an XSS attack, what shall we do then? Never use such languages?

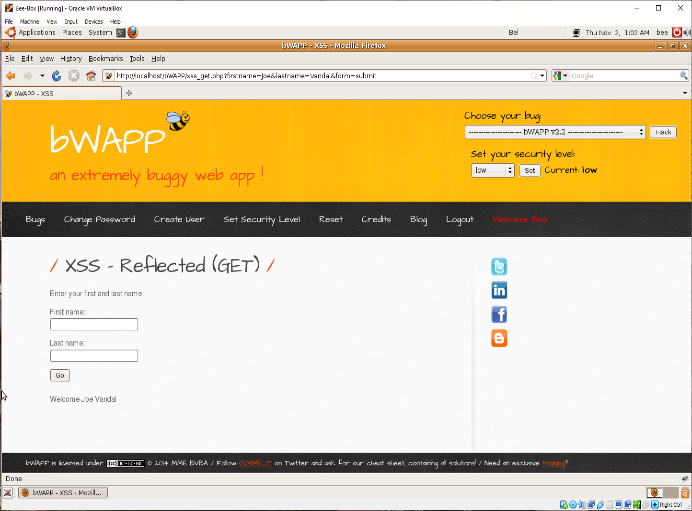
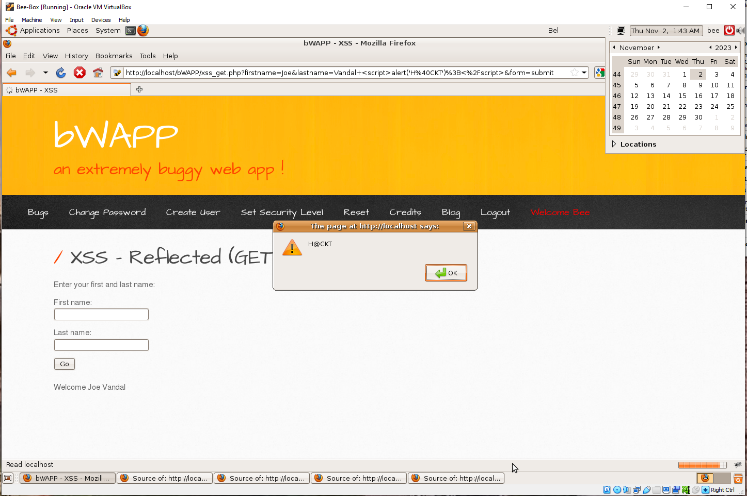
Its not necessary to completely avoid using languages like JavaScript or other technologies that can be leveraged in XSS attacks. Instead, you should follow best practices to protect you web applications from XSS vulnerabilities. Here are some key measures to take:

1. Input Validation: Always validate and sanitize user inputs to ensure that they do not contain malicious code.
2. Output Encoding: Encode or escape user-generated content before it is displayed on web pages. This ensures that even if an attacker injects malicious code, it will be treated as data rather than exe code.
3. Content Security Policy (CSP): Implement a content security Policy to control which scripts can be executed on your website. CSP headers can help prevent the execution of unauthorized scripts.
4. Regular Security Audits: Periodically perform security audits and vulnerability assessments on your website to identify and address any potential XSS vulnerabilities.

Activity 1: XSS Attack

1. Go to <http://www.bwapp.com/login.php>
2. Login: bee password: bug
3. From “choose your bug.” Drop-down list, select “Cross-Site Scripting – Reflected (GET)”
4. Get security level to low and click Hack
5. Provide different combinations of first name and last name and look for the potential vulnerability.





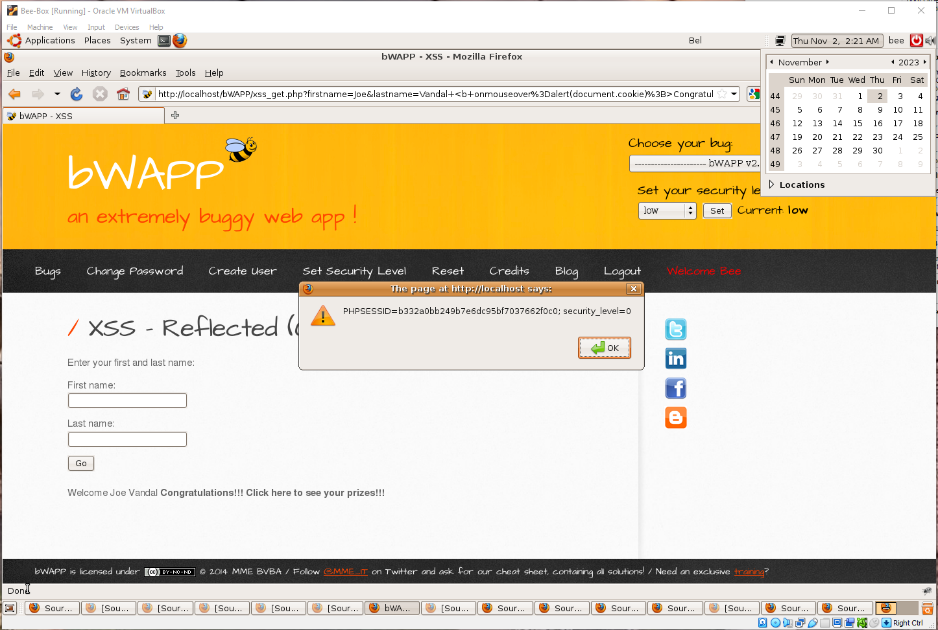
Challenge I: Modified XSS Attack

As a Continuation to previous activity, try to inject some input parameters such that

1. Inputs values are returned back to the user itself with the welcome message as follows:

Welcome message as follows:

Welcome Joe Vandal Congratulations !!! Click here to see your prizes!!!

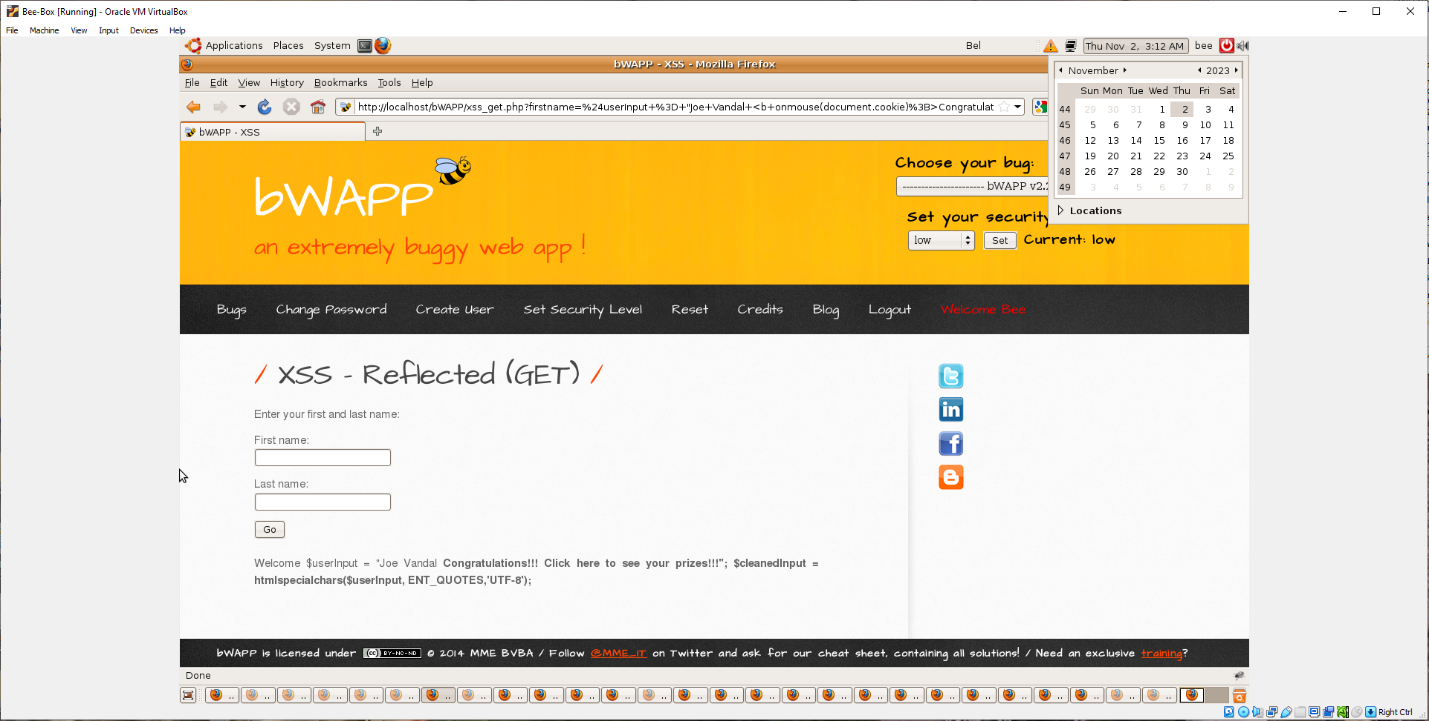
1. When user performs mouse-over action on Congratulations!!! Click here to see your prizes!!! , it displays cookie information for the user as an alert.

Challenge II: Mitigate Challenge I Attack

Try sanitizing inputs such that the scripts you executed in the challenge 1 acts just as a normal script.

A screenshot of a computer

Description automatically generatedFor this challenge, the equivalent php file is located at: /var/www/html/bW APP/public.html/xss\_get.php



Activity II: CSRF Attack Part I

1. From “Choose your bug.” Drop-down list, select “Cross-Site Request Forgery (Change Password)”
2. Set security level to low and click Hack
3. Change the password from “bug” to something else. You should be able to do it
4. Right Click on the browser and view the source of the webpage
5. Copy Web content as shown below

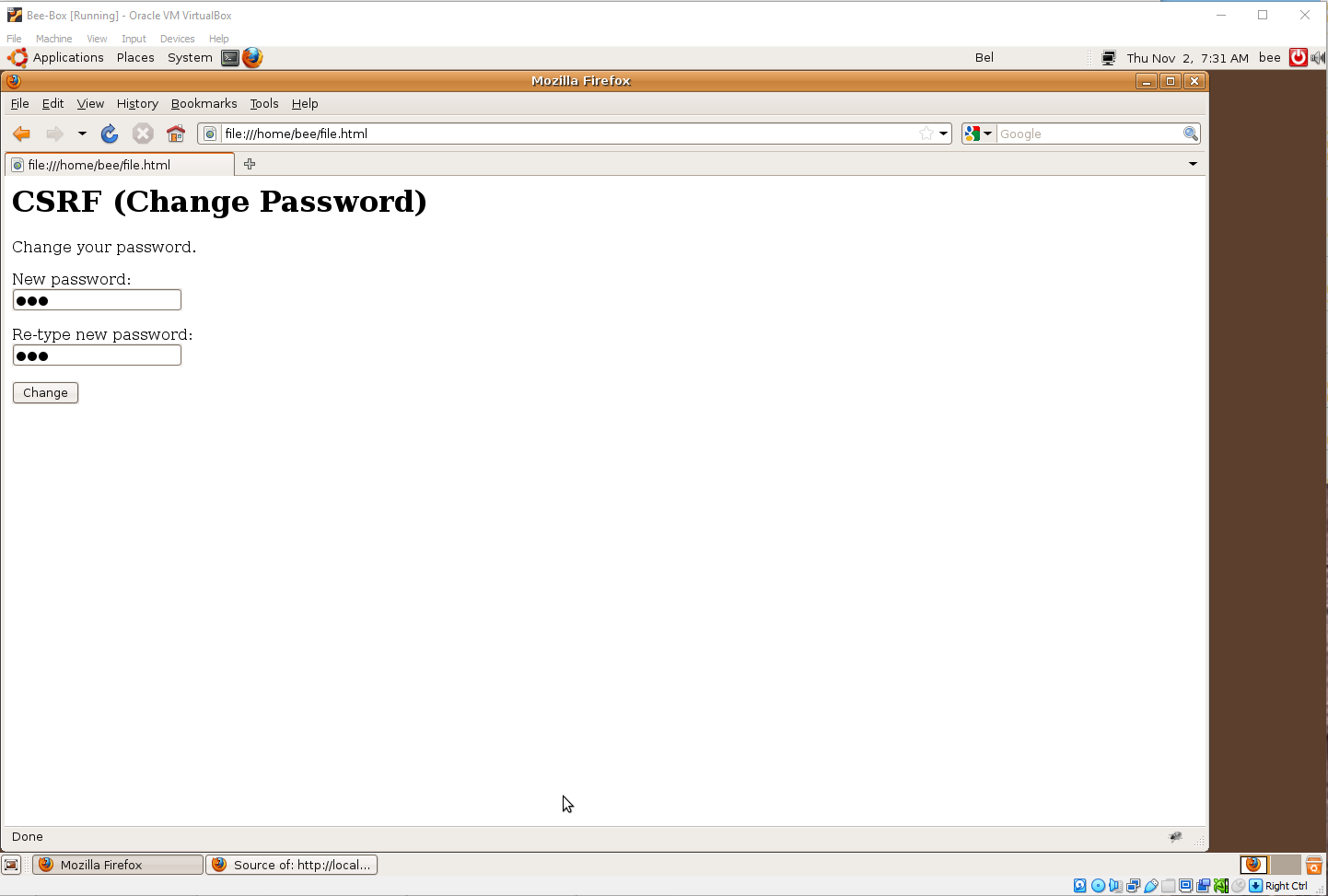
A screenshot of a computer

Description automatically generated

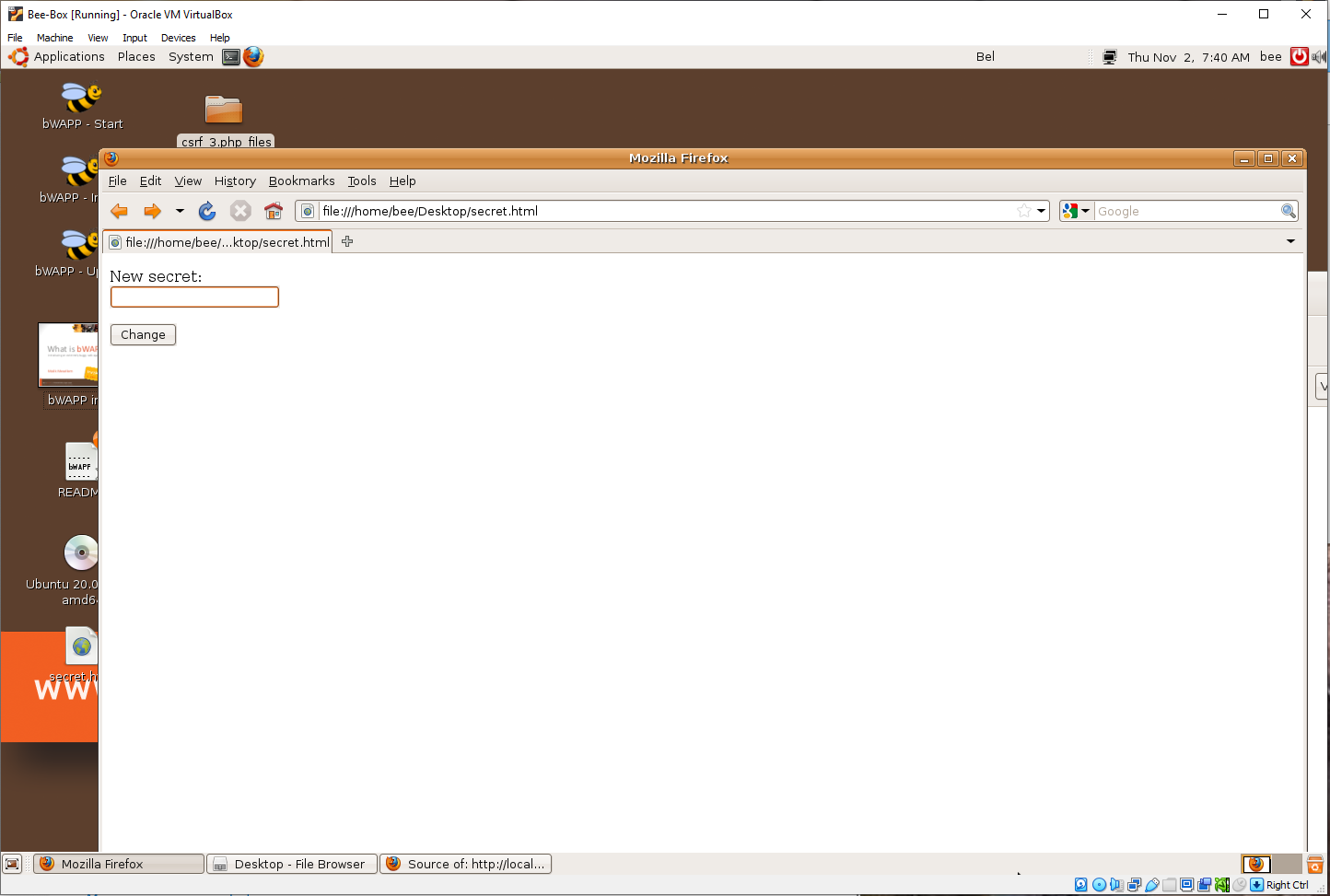
Activity II: CSRF Attack Part II

1. Modify the copied content as shown in the notes section below. Note the following changes:
   1. Action field in the form tag
   2. Addition of value attributes in the input tags
2. Save it as html file in desktop
3. Open the html file and press Change

Attackers tend to execute such process in the background silently where your password is being compromised secretly.

Mitigating Attack from Activity II

Challenge III: Modified CSRF Attack

1. From Choose your bug, select Cross-Site Request Forgery(Change Secret)
2. Set security level to low and then press Hack
3. As we did in previous tutorial, create a HTML page that can be injected using CSRF, and execute html page such that it change the secret statement to something else.

Questions: CSRF

What was the problem with the following pages?

1. Page where the password was changed
   1. It allowed for the user to force unwanted HTML file onto the webpage in which we had current authentication. The password was forcefully updated and changed
2. Page where the secret was changed
   1. This created the same situation as the password webpage but instead it prompted for a change in the secret instead of the password element

Can you think about the ways to mitigate the problems in the above pages?

Token Synchronization

* Attackers would not be able to make requests to the backend without valid tokens

Double-Submitting Cookies

* It is a stateless, easily implemented technique that sends random values twice

Same-Site Cookies

* Help defend against CSRF attacks by restricting the cookies sent alongside each request.

Challenge IV: Mitigate Challenge III Attack <>

Use one-time token implementation such that the system uses tokens that change as per the session thus, grabbing tokens from the web-application of the target victim doesn’t work

* The page associated with this challenge is csrf\_3.php located in /var/www/html/www.bwapp.com/public\_html/