[XSS] - [Cross-Site Scripting]

Risk Analysis

Total Risk |High Severity |High Probability |High Fix effort |Low

Vulnerability Description

Cross-Site Scripting (XSS) is a security vulnerability that occurs when an attacker injects malicious scripts into web applications. These scripts can then be executed by users who view the affected pages, leading to potential unauthorized access, data theft, or manipulation. XSS attacks can be classified into three main types:

Stored XSS- is an attack where a malicious script is injected into a website and permanently stored on the server. When other users access the page, the stored script is executed, posing risks like unauthorized access or data theft. Prevention involves input validation, output encoding, and secure handling of user-generated content.

Reflected XSS involves injecting a malicious script that is immediately reflected in the web page's response. The script executes when users access the manipulated URL, potentially leading to security vulnerabilities.

DOM XSS - is a type of XSS attack where the manipulation of the Document Object Model (DOM) in a web page leads to the execution of malicious scripts. It occurs when the client-side script modifies the DOM, posing security risks.

Each pose varying degrees of risk to web applications. Preventative measures include input validation, output encoding, and implementing secure coding practices.

Vulnerability Details

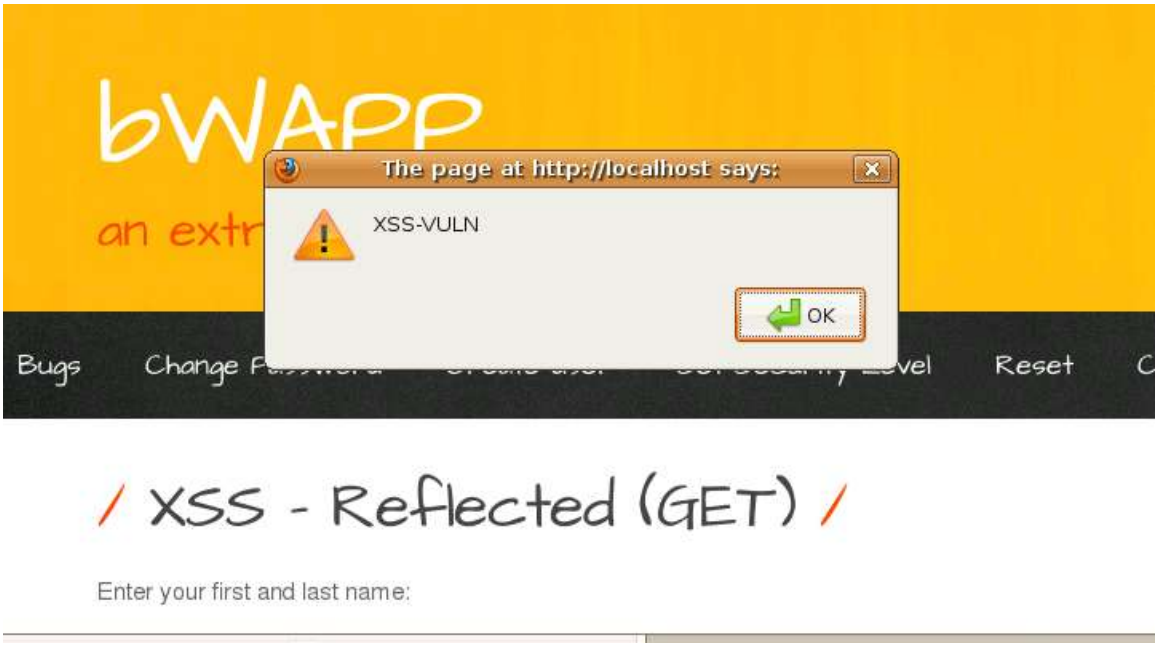
In our controlled environment, we found a vulnerability within our websites HTML output, where a potential attacker could inject malicious scripts which would then get executed by other users' browsers when accessing the compromised page.

Execution Demonstration

Within our controlled environment, we visit **http://[ip-address]/bWAPP/xss_get.php,** here we can insert data into the login form.

The data inserted gets reflected back to the user, we can tell by the bottom of our webpage there is "Welcome user password" this shows us the reflection.

An attacker has the ability to input a inject on the client-side and we demonstrate this by inputting our very own malicious code, which for demonstration purposes will actually just create a pop-up , or as you see within the code an "alert", this is to a visualization of the sucess of the injected script. We use <script>alert("XSS-VULN")</script, which simulates our sucessful injection with a pop-up message instead.



RECOMMENDED RECTIFICATION

* Validate and sanitize user inputs both within the client and server side, using input validation mechanisms to ensure the data entered by the user adheres to the expected formart.

* Setting the "HTTPonly" flag on cookies to prevent client-side scripts from accessing sensitive cookie data, reducing the impact of XSS attacks.

* Neutralizing potential script elements and characters (e.g., '<' to '<', '>' to '>')

* Implement and enforce a Content Security Policy (CSP) that controls what resources are allowed to be loaded, and pulled from trusted sources.