

ENSF 381

Full Stack Web Development

Lecture 33:

Security Practices for Web Applications

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

Outline

- Overview of web application security.
- Client-Side security.
- Server-Side security.
- Secure deployment practices.

What is Web Application Security?

- The process of securing websites and online services against various threats and vulnerabilities.
- It involves protecting data, users, and systems from unauthorized access, attacks, and misuse.
- Ensures the confidentiality, integrity, and availability of web applications and their data.

Securing your full-stack application is paramount

- **Protecting Sensitive Data:** full-stack applications often handle sensitive user data, such as personal information, financial details, or proprietary business data.
- **Preventing Unauthorized Access:** without proper security measures, attackers can gain unauthorized access to your application, its databases, or backend systems.
- **Maintaining User Trust:** users expect their data to be handled securely by the applications they use.
- **Mitigating Downtime and Damage:** security incidents can lead to downtime, data loss, and damage to your application and infrastructure.

Statistics

- Human error is responsible for **74%** of cybersecurity breaches.
- The average expense incurred by a data breach is **\$4.45 million** in 2023.
- Approximately 93% of all LinkedIn members, totaling 700 million users, had their personal information exposed in a data breach in 2021.
- In 2023, T-Mobile disclosed its second data breach, affecting 836 customers, following an earlier breach that impacted approximately 37 million customers.

Securing Full Stack Application

- Frontend

Frontend Security Practices

- **Input Validation:**

- Utilize client-side validation with JavaScript to perform immediate validation of user input in forms, ensuring that data meets specified criteria (e.g., format, length).
- Implement server-side validation to re-validate user input on the server, preventing malicious data from being processed even if client-side validation fails.

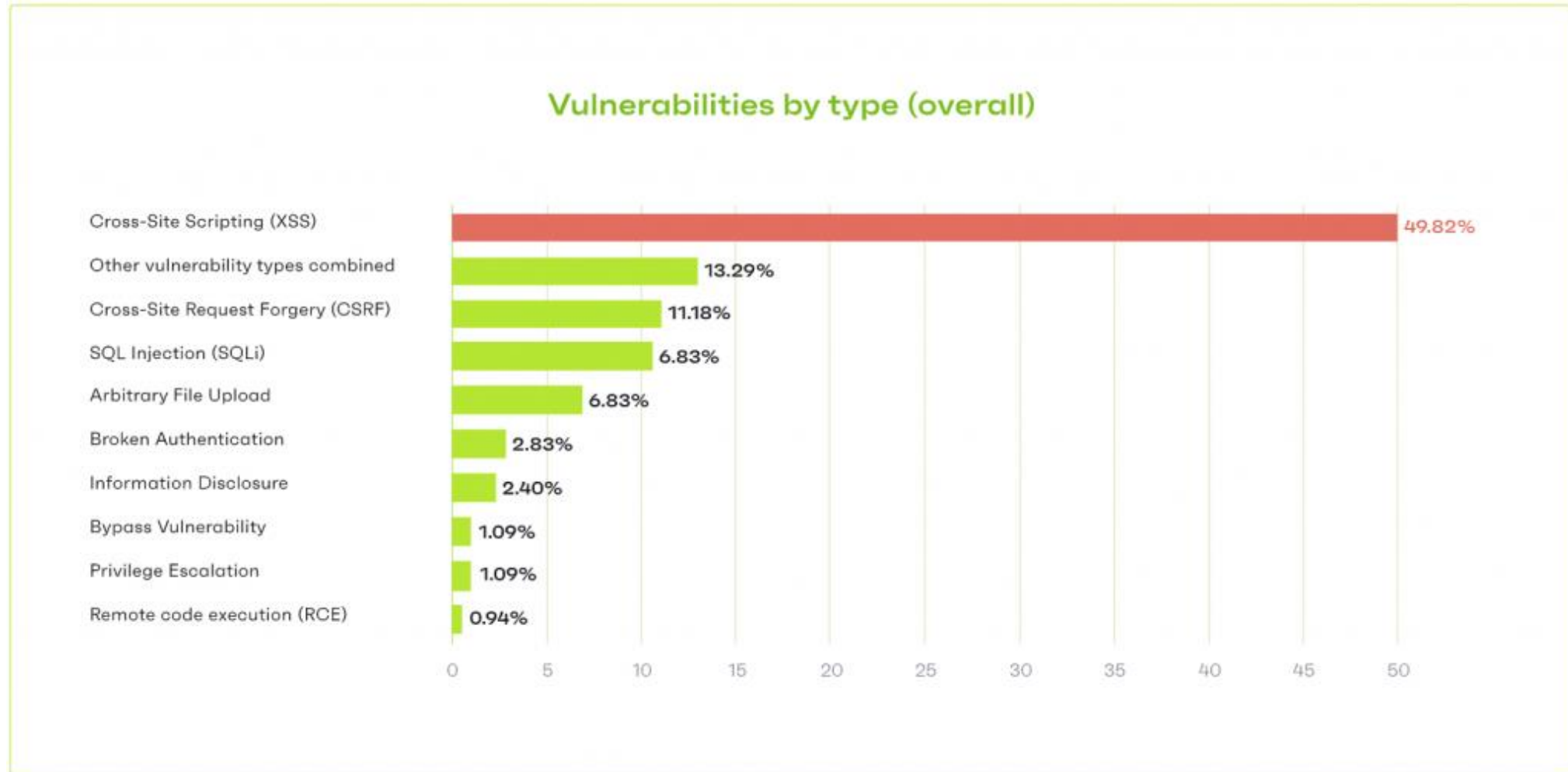
- **File Type Validation:**

- Only allow file types that are necessary for the application's functionality.
- Rename uploaded files using a randomized or hashed naming scheme to prevent attackers from predicting or manipulating file paths.

- **Content Security Policy (CSP):**

- Prevent Cross-Site Scripting (XSS) attacks by defining a Content Security Policy that restricts the sources from which content, such as scripts, stylesheets, and images, can be loaded.
- XSS occurs when attackers inject malicious scripts into web pages viewed by other users.

Frontend Security Practices – cont.

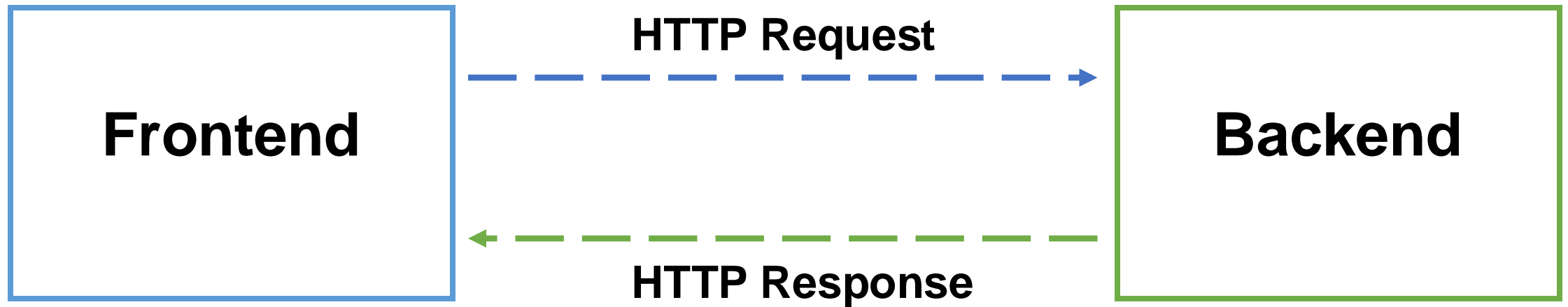


Frontend Security Practices

- Client-side code, including React components, runs in the **user's browser** and can be accessed and manipulated by users.
- Avoid storing sensitive data (e.g., authentication tokens, API keys) in client-side code.
- Use third-party libraries carefully and keep them updated to avoid vulnerabilities.
- Avoid sending sensitive information, such as passwords or authentication tokens, in plain text over insecure connections.

Recap: HTTP - Example

Hypertext Transfer Protocol (HTTP): the protocol used for transmitting data on the web, including requests and responses between frontend and backend.



Hypertext Transfer Protocol Secure (HTTPS)

- HTTPS is an extension of HTTP that adds a layer of security through encryption.
- HTTPS encrypts data using SSL/TLS, preventing unauthorized access and ensuring privacy.
- Ensure that data remains unchanged during transmission.

Frontend Security Practices

- Client-side code, including React components, runs in the user's browser and can be accessed and manipulated by users.
- Avoid storing sensitive data (e.g., authentication tokens, API keys) in client-side code.
- Use third-party libraries carefully and keep them updated to avoid vulnerabilities.
- Avoid sending sensitive information, such as passwords or authentication tokens, in plain text over insecure connections.
- Verify that the user interacting with your website or application is a human, rather than a bot or automated script.

reCAPTCHA - Example

```
import React, { useState } from 'react';  
import ReCAPTCHA from 'react-google-recaptcha';
```

→ Install the 'react-google-recaptcha' package.

```
function MyComponent() {  
  const [verified, setVerified] = useState(false);
```

```
  function handleRecaptchaChange(value) {  
    console.log('Recaptcha value:', value);  
    // Set the verified state based on the recaptcha value  
    setVerified(value !== null);  
  };
```

```
  function handleSubmit() {  
    // Perform form submission logic here  
    console.log('Form submitted!');  
  };
```

```
  return (  
    <div>  
      <h1>My Component</h1>  
      <ReCAPTCHA  
        sitekey="SECRET KEY"  
        onChange={handleRecaptchaChange}  
      />  
      <button onClick={handleSubmit} disabled={!verified}>Submit</button>  
      {!verified && <p>Please complete the ReCAPTCHA</p>}  
    </div>  
  );  
};
```


```
export default MyComponent;
```

reCAPTCHA - Example

localhost:3000

My Component

☐ I'm not a robot


reCAPTCHA
[Privacy](#) - [Terms](#)

Submit

Please complete the ReCAPTCHA

Securing Full Stack Application

- Frontend
- Backend

Backend Security Practices

- **Preventing SQL Injection:** utilize parameterized queries or prepared statements to prevent malicious input from being interpreted as SQL commands. Consider using an Object-Relational Mapping (ORM) library to abstract away direct database interactions and mitigate SQL injection risks.
- **Secure File Uploads:** validate file types and sizes to prevent malicious file uploads.
- **Secure Error Handling:** avoid revealing sensitive information in error messages to attackers. Implement proper logging mechanisms to track and monitor errors, without exposing sensitive data.
- **Rate Limiting and IP Banning:** implement rate limiting to mitigate brute-force attacks on authentication mechanisms.

Backend Security Practices

- Implement secure API endpoints with proper authentication and authorization mechanisms.
- Store uploaded files in a secure location outside the web root directory to prevent direct access and execution.
- Conduct regular security audits and penetration testing to identify and mitigate vulnerabilities in your Flask application.
- Store user passwords securely using strong hashing algorithms (e.g., bcrypt) with a unique salt for each user.

Hashing

- Hashing is the process of converting input data (such as passwords or other sensitive information) into a fixed-size string of characters, typically through a mathematical algorithm called a hash function.
- Secure Hash Algorithm 256 (SHA-256): SHA-256 is part of the SHA-2 family and is widely used for secure hashing. It produces a 256-bit hash value.
 - Example: e59e31e90a0d68e0d15c866edf167bc4e6f8b6255188122284fda2fc640dee7c
- bcrypt: bcrypt is a password-hashing function designed specifically for secure password storage.
 - Example: \$2b\$12\$luLwZ25eKhbVHeG5n9H7VOu8b76Ddw4N9dU9bue9ZpH4W5Npf9wHm

Securing Full Stack Application

- Frontend
- Backend
- Deployment

Secure deployment practices

- **Secure Server Configurations:** properly configure web servers (e.g., Apache) and application servers to minimize security risks. Implement firewall configurations to restrict unauthorized access to server resources.
- **Regular Software Updates and Patch Management:** keep all software components, including operating systems, web servers, application frameworks, and libraries, up to date with the latest security patches.
- **Use of Security Monitoring Tools:** deploy Intrusion Detection Systems to detect and respond to potential security threats in real-time.
- **Continuous Security Testing:** conduct automated security testing using tools to identify and remediate security vulnerabilities in your application. Also, perform manual code reviews and security audits to uncover potential security flaws that automated tools may miss.

Questions

