

A large, stylized graphic with the word 'Welcome' in a dark blue, cursive font. The text is set against a background of colorful, abstract splashes in shades of orange, red, and blue, all on a light yellow background.

# Welcome

# Web Application Security



## Lecture:01

### Web Application Security Introduction

**Miss Maryam Malik**  
**Lecturer**  
**Department of cyber security**  
**Air university Islamabad**

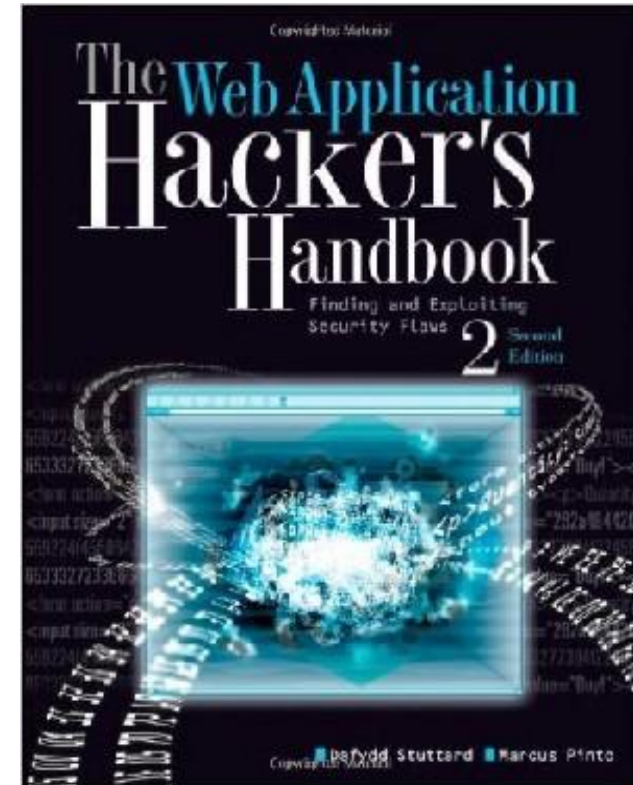
# Text Books



- ❑ The Web Application Hacker's Handbook: : Finding and Exploiting Security Flaws by Dafydd Stuttard and Marcus Pinto
  - 7<sup>th</sup> edition

- ❑ Read the textbook!

- Key for learning and obtaining a good grade



# Class

# Information

Assessments

Weightage

Assignments	10 %
Mid Term	25 %
Quiz	10 %
Project	10 %
Final Exam	45 %

**GCR Code: 4vupndu**



Note: plagiarized assignments and quizzes will be awarded straight zero.

# Agenda

- Web Application Evolution, Web 1.0, Web 2.0.
- Web App Functions
- OWASP Top 10 Vulnerabilities
- Class Activity
- Authentication
- Session Management and problems,
- Access Control

# Securing your Web Application



- Creating a Web application is easy, but creating a secure Web application is hard and tedious.
- Because of the multi-tiered architecture, security flaws may appear at many levels.
- You need to secure your database, your server, your application, and your network.
- Result: To create a secure Web application, you need to examine every layer.

# Static Website: Web 1.0

- Information flows one-way
- Users don't log in, shop, or submit comments
- An attacker who exploits flaws in the Web server software can
- Steal data on the Web (e.g., public data anyway)
- Deface the site





# Modern Web App

- Two-way information flow
- Users log in, submit content
- Content dynamically generated and tailored for each user
- Much data is sensitive and private (e.g. passwords)
- Most apps developed in-house





# Common Web App Functions

- Shopping (Amazon)
- Social networking (Facebook)
- Banking (Citibank)
- Web search (Google)
- Auctions (eBay)
- Gambling (Betfair)
- Web logs (Blogger)
- Web mail (Gmail)
- Interactive information (Wikipedia)

# Internal Web Apps ( "Cloud" Services)



- HR -- payroll information, performance reviews
- Admin interfaces to servers, VMs, workstations
- Collaboration software (SharePoint)
- Enterprise Resource Planning (ERP)
- Email web interfaces (Outlook Web Access)
- Office apps (Google Apps, MS Office Live)

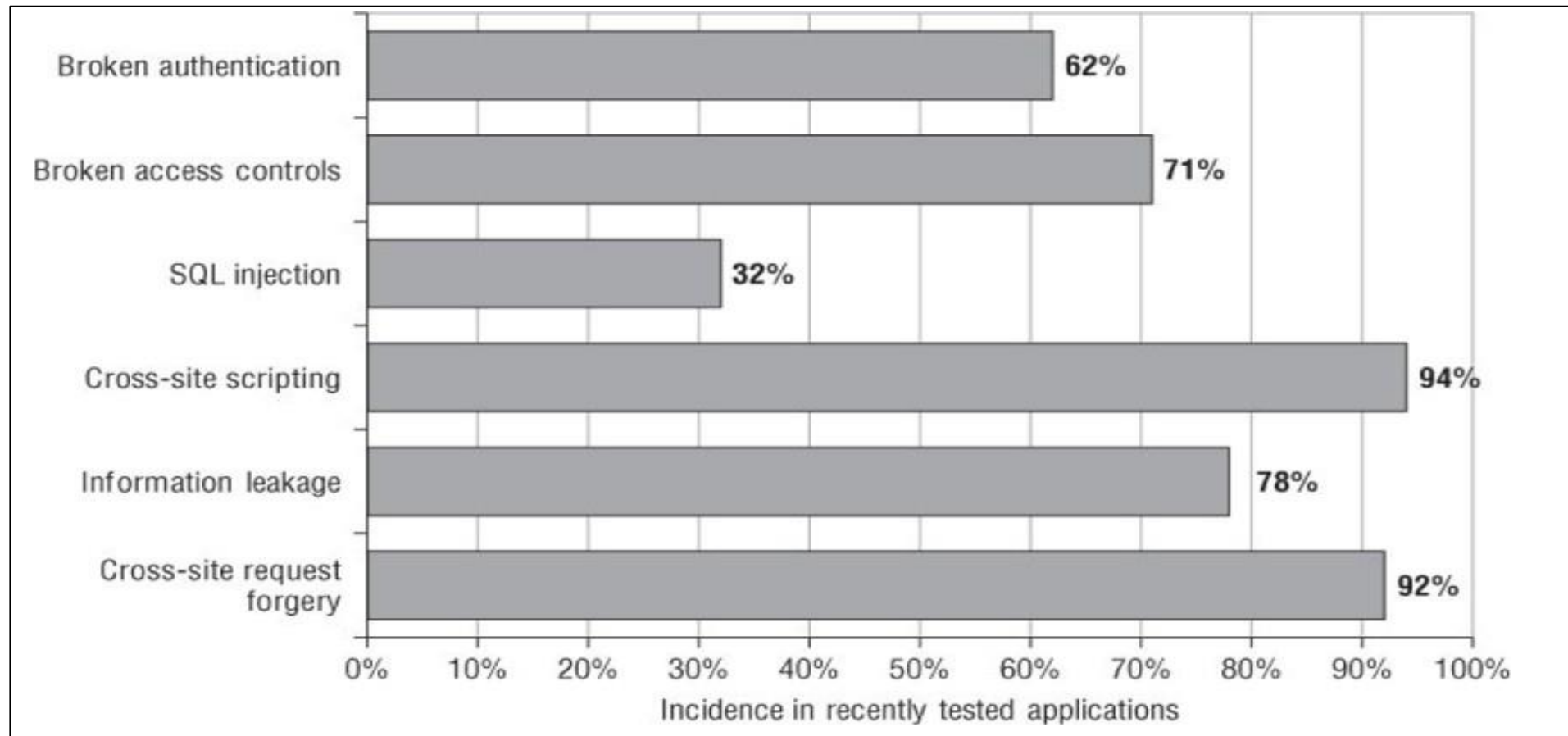
# Benefits of Web Apps

- HTTP is lightweight and connectionless
- Resilient in event of communications errors
- Can be proxied and tunnelled over other protocols
- Web browsers run on many devices, highly functional, easy to use
- Many platforms and development tools available

# Web App Security

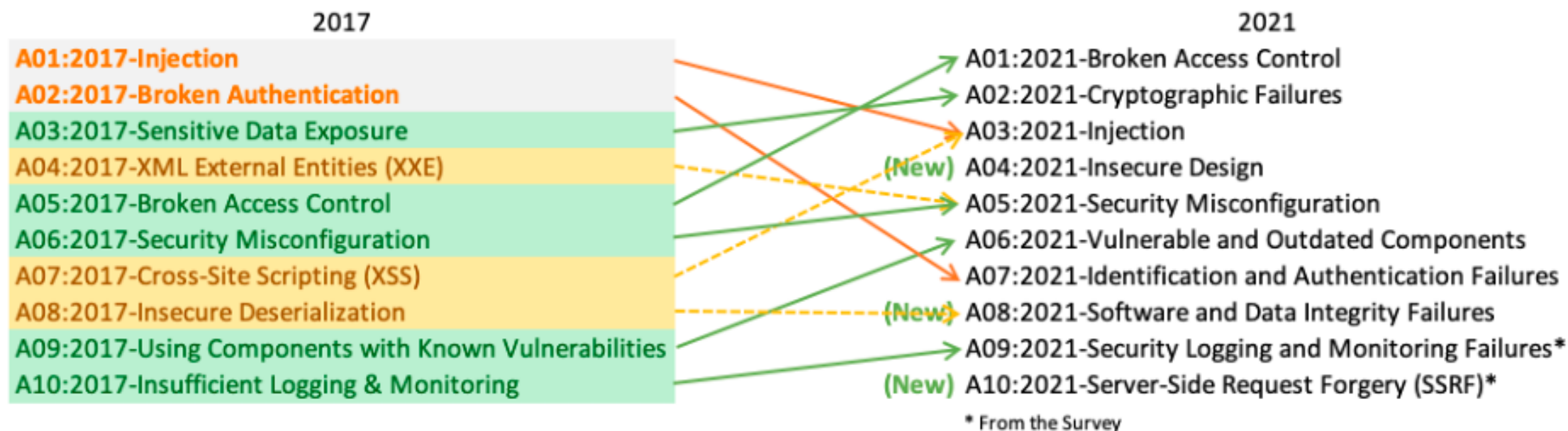
- Breaches are common
- Attackers gets sensitive data, possibly complete control of back-end systems
- Denial of Service at Application Level

# Common rate of Attacks



# What is OWASP?

- Open Web Application Security Project
  - ▶ Promotes secure software development
  - ▶ Support application security risk decision making
  - ▶ Focused on the security of web applications as software products of the SDLC
- Provides free resources to developer teams
- Encourages active participation and information sharing





# Broken Access Control

- Principle of Least Privilege Violation: Users are granted excessive permissions instead of only those necessary for their roles.
- Bypassing Access Control: Attackers can manipulate URLs, API requests, or HTML forms to bypass access checks.
- Insecure Direct Object References: Users can access or edit another user's account by providing their unique identifier.
- Missing Access Controls: APIs lacking proper checks for methods like POST, PUT, and DELETE can expose sensitive operations.
- Elevation of Privilege: Users can perform actions as an admin or another user without proper authentication.
- Metadata Manipulation: Tampering with tokens (e.g., JWTs) or cookies can lead to unauthorized access or privilege escalation.
- CORS Misconfiguration: Poorly configured Cross-Origin Resource Sharing (CORS) policies may allow access from untrusted origins.
- Force Browsing: Unauthorized users can access protected pages through direct URLs.

# Example

- An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page.
- <https://example.com/app/getappInfo>
- [https://example.com/app/admin\\_getappInfo](https://example.com/app/admin_getappInfo)
- If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

# Cryptographic Failures

A cryptographic failure occurs when a cryptographic system does not provide the intended level of security, which can lead to unauthorized access, data breaches, or integrity violations

- Weak Algorithms
- Poor Key Management
- Improper Implementation
- Insecure Configuration
- Failure to Validate
- Exploitable Error Messages

## Example

An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

# Injection

Injection vulnerabilities occur when untrusted data is sent to an interpreter as part of a command or query. Attackers can exploit these flaws to execute arbitrary commands or retrieve sensitive data.

## **Examples:**

SQL injection, Command injection

## **Impact:**

Data loss, data corruption, denial of service, and unauthorized access.

# Injection

## Example

- the attacker modifies the 'id' parameter value in their browser to send: ' UNION SLEEP(10);--.  
For example:
- `http://example.com/app/accountView?id=' UNION  
SELECT SLEEP(10);--`

# Insecure Design

Insecure design encompasses weaknesses resulting from "missing or ineffective control design." It highlights the need for proper security measures at the design stage.

## Insecure Design vs. Insecure Implementation:

- **Insecure Design:** Refers to flaws in the design phase where necessary security controls were never established.
- **Insecure Implementation:** Involves defects in coding or configuration that occur despite a secure design.

## Impact of Insecure Design:

An insecure design cannot be remedied by perfect implementation, as it lacks the foundational controls needed to defend against specific threats.

Vulnerabilities arising from insecure design stem from inadequate risk profiling and the failure to identify required security levels.

# Insecure Design

**Example 1:** An online form that accepts user input without checking for malicious content. For instance, a search box that allows SQL queries without sanitization can lead to SQL injection attacks.

**Example 2:** Storing passwords in plain text in a database instead of using hashing. This design flaw can lead to massive data breaches if the database is compromised.



# Security Misconfiguration

Security misconfiguration occurs when security settings in software, applications, networks, or systems are set up incorrectly or not maintained properly, leaving them vulnerable to attacks.

This often results from:

- Default or incomplete security settings
- Unnecessary features enabled (such as open ports or unused pages)
- Outdated or unpatched software

# Security Misconfiguration

**Scenario:** Many web applications, devices (such as routers), and databases come with default usernames and passwords like "admin" and "password." If administrators do not change these credentials after installation, attackers can easily guess them and gain unauthorized access.

**Case:** In 2015, a security researcher exploited misconfigurations in many MongoDB databases because the databases were left open to the internet with default settings. This allowed anyone to access sensitive data without a password.

**Misconfiguration:** The database was set up with default settings that didn't require authentication for access, exposing the data to the internet.

**Impact:** Sensitive information was publicly available.

# Vulnerable and Outdated Components



- **Unknown Versions:** If you don't keep track of the versions of all the software components (both on the client side and server side) you are using, including nested dependencies, you might be using outdated or vulnerable components without realizing it.
- **Outdated or Unsupported Software:** If any part of your system—operating system (OS), web servers, application servers, databases, APIs, or libraries—is vulnerable, no longer supported, or out of date, you are at risk. Attackers often target systems using outdated software with known security holes.
- **Not Regularly Scanning for Vulnerabilities:** If you don't regularly check for security vulnerabilities in the components you use or keep up-to-date with security bulletins, you could be unaware of known issues that need fixing.
- **Delayed Patching and Updates:** If you don't fix or update software, libraries, or frameworks in a timely manner, especially when known vulnerabilities exist, you leave your system exposed for longer than necessary. In some organizations, patches might only be applied on a monthly or quarterly basis, leaving systems vulnerable for weeks or months.

# Identification and Authentication Failures



Identification and authentication failures occur when an application does not adequately verify a user's identity or fails to manage user sessions securely. These weaknesses can lead to a variety of attacks, including unauthorized access, credential stuffing, brute force attacks, and session hijacking. Proper authentication controls are crucial to ensuring that only legitimate users can access sensitive resources.

**Example:** Credential Stuffing Attack a common real-world scenario for identification and authentication failures is credential stuffing. Many users reuse the same password across multiple sites. Attackers compile lists of stolen usernames and passwords from previous breaches, then attempt to log in to other sites using these credentials. If the site does not have protection

# Software and Data Integrity Failures



**Description:** Software and data integrity failures occur when code and infrastructure fail to protect against integrity violations. These vulnerabilities arise when applications or systems rely on untrusted sources for code, plugins, libraries, or data, allowing attackers to compromise the integrity of the system or manipulate the data.

## Common Causes of Software and Data Integrity Failures:

### Use of Untrusted Components:

**Example:** An application uses libraries or plugins from untrusted sources, repositories, or **Content Delivery Networks (CDNs)** without verifying their integrity.

Attackers can inject malicious code into these components, compromising the application.

# Security Logging and Monitoring Failures



Security logging and monitoring failures occur when applications, systems, or networks lack proper logging of important security events or fail to monitor and analyze logs in real-time. This prevents organizations from detecting, investigating, or responding to suspicious activities and security incidents in a timely manner.

## Example

Marriott Starwood Data Breach (2018): Attackers had been accessing the Starwood Hotels reservation database for four years without detection. Marriott's failure to detect suspicious activity sooner allowed attackers to steal data on **500** million guests.

# Server-Side Request Forgery (SSRF)





# Class Activity



Identify the Owasp vulnerabilities in following incidents

## Snapchat

On New Year's Day, the website Snapchat DB.info released the usernames and redacted phone numbers of 4.6 million U.S. Snapchat users. Months earlier, an Australian security outfit called Gibson Security published a thorough account of the security vulnerabilities plaguing the company had revealed 4.6 million usernames and phone numbers.

Why was this significant?

The attack seems to be motivated at least partly by Snapchat's assertion that the attack was theoretical, and they had not taken any action. This resulted in a data leakage of phone numbers and users details that could be valuable for various uses.

## The Panama Papers incident (Apr 2016)

The Panama Papers are a collection of 11.5 million records from Mossack Fonseca, originally leaked to German journalist Bastian Obermyer in 2015. Due to the sheer size of the data, the International Consortium of Investigative Journalists were approached.

Why was this significant?

Many public figures, present and past, had their financial dealings exposed, linking them to terrorists, drug cartels and tax havens. Some public figures

# The Core Security Problem

- Users can submit arbitrary input
- Alter parameters, cookies, HTTP headers
- Client-side controls can't be trusted
- Developers must assume all input is malicious
- Attackers have attack tools like Burp; they are not restricted to using browsers

# Key Problem Factors

- Underdeveloped Security Awareness
- Custom Development
- Deceptive Simplicity
- Easy to make a website, but hard to secure it
- Rapidly Evolving Threat Profile
- Resource and Time Constraints
- Overextended Technologies
- Increasing Demands on Functionality

# Core Defense Elements

- Limiting user access to app's data and functionality
- Limiting user input to prevent exploits that use malformed input
- Frustrating attackers with appropriate behaviour when targeted
- Administrative monitoring and configuring the application

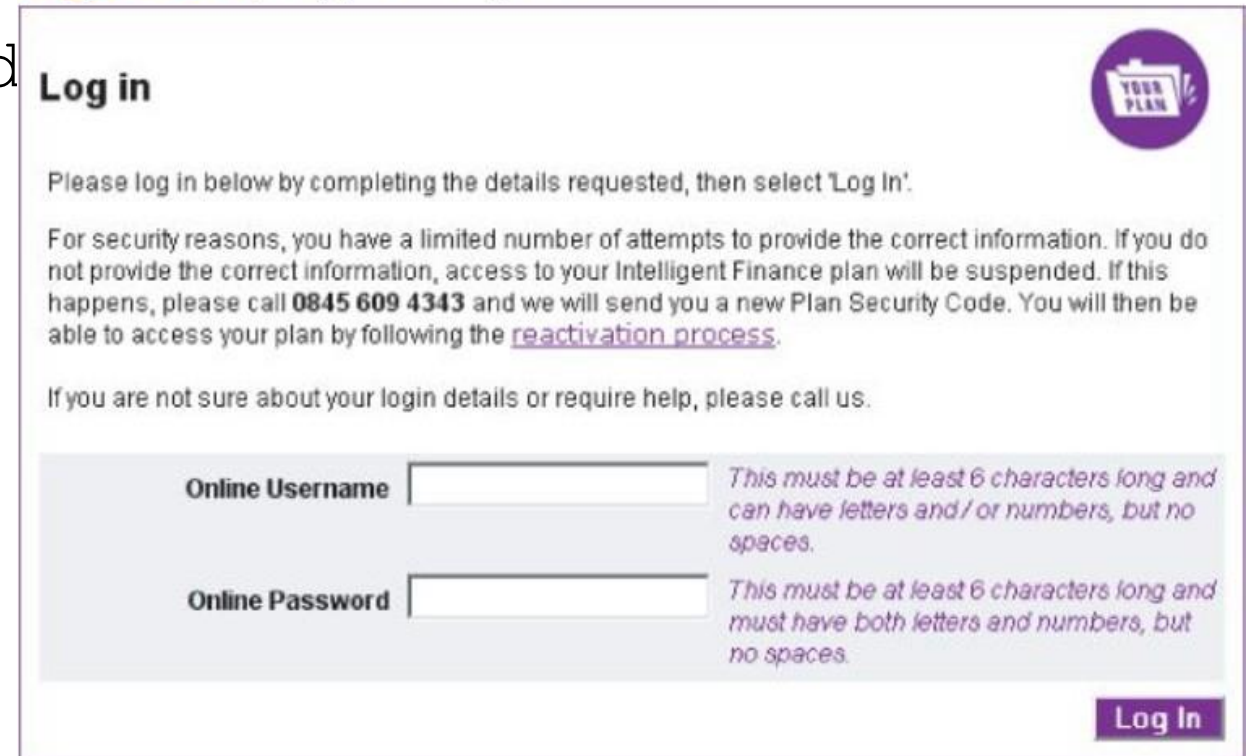
# Handling User Access

- Authentication
- Session management
- Access Control

# Authentication

- Username and password is most common method
- Better: additional credentials and multistage login
- Best: client certificates, smart cards, challenge-response tokens
- Also: self-registration, account recovery, password change

**Figure 2.1** A typical login function

The image shows a web-based login interface. At the top right is a circular purple icon with a white document and the text 'YOUR PLAN'. Below this, the heading 'Log in' is displayed. A paragraph of text instructs the user to log in by completing details and selecting 'Log In'. A security warning follows, stating that incorrect information will lead to account suspension and provides a phone number (0845 609 4343) and a link to the reactivation process. Below the text are two input fields: 'Online Username' and 'Online Password'. To the right of each field is a purple text note specifying requirements: at least 6 characters, letters and numbers, no spaces. At the bottom right is a purple 'Log In' button.

**Log in**

Please log in below by completing the details requested, then select 'Log In'.

For security reasons, you have a limited number of attempts to provide the correct information. If you do not provide the correct information, access to your Intelligent Finance plan will be suspended. If this happens, please call **0845 609 4343** and we will send you a new Plan Security Code. You will then be able to access your plan by following the [reactivation process](#).

If you are not sure about your login details or require help, please call us.

**Online Username**  *This must be at least 6 characters long and can have letters and / or numbers, but no spaces.*

**Online Password**  *This must be at least 6 characters long and must have both letters and numbers, but no spaces.*

**Log In**

# Common Login Problems

- Predictable usernames
- Password that can be guessed
- Defects in logic



# Session Management

- **Session Creation:** After successful authentication, the server creates a session for the user and generates a unique session token.
- **Token Issuance:** The session token is sent to the user's browser, commonly through:
  - HTTP Cookies: Most common method for managing sessions.
  - Hidden Form Fields: Less common but still used.
  - URL Query Strings: Can expose tokens in logs and history.
- **Token Submission:** The browser automatically sends the session token with each subsequent request, allowing the server to associate requests with the authenticated user.
- **Session Expiration:** Sessions should ideally expire after a certain period of inactivity to reduce the risk of unauthorized access.

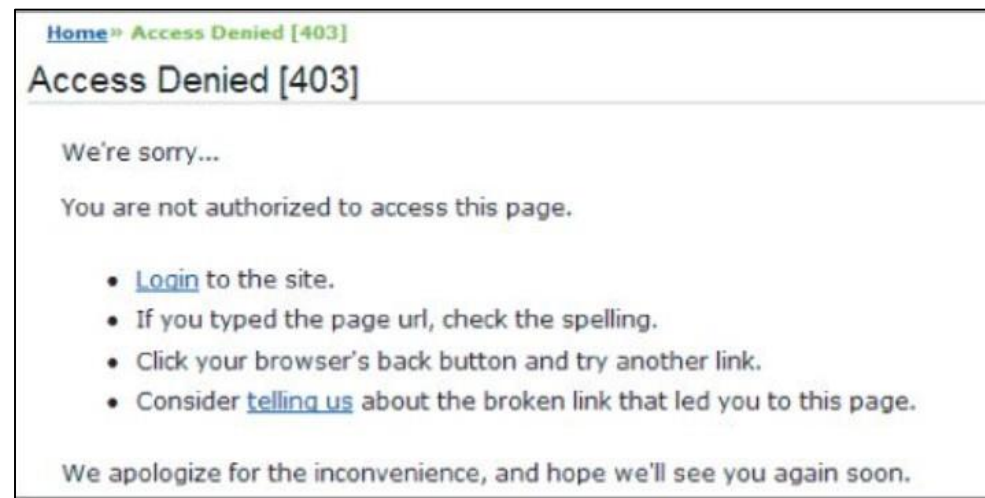
# Common Session Problems

- Tokens are predictable (not random)
- Tokens poorly handled, so an attacker can capture another user's token



# Access Control

- Each request must be permitted or denied
- Multiple roles within an application
- Frequent logic errors and flawed assumptions



# Handling User Input

- Input Validation" is the most common solution

First Name	<input type="text" value="a"/>	Must contain at least 4 characters
Last Name	<input type="text" value="a"/>	Must contain at least 4 characters
Email	<input type="text" value="a"/>	Please provide a valid email address
Phone number	<input type="text" value="a"/>	Must contain only numbers

# Reject Known Bad

- Also called "blacklisting"
- Least effective method
- Difficult to identify all bad inputs

- If `SELECT` is blocked, try `seLeCt`
- If `or 1=1--` is blocked, try `or 2=2--`
- If `alert('xss')` is blocked, try `prompt('xss')`

```
%00<script>alert(1)</script>
```

# Accept Known Good

- Also called "whitelisting"
- Most effective technique, where feasible
- However, sometimes you can't do it
- Human names really contain apostrophes, so you can't filter them out

# Sanitization

- Render dangerous input harmless
- HTML-Encoding: One of the most common uses of data sanitization is to prevent Cross-Site Scripting (XSS) attacks, where attackers inject malicious scripts into web pages. To protect against XSS, an application sanitizes user input by HTML-encoding dangerous characters, such as `<`, `>`, `'`, and `"`.
- Difficult if several kinds of data may be present within an item of input
- Boundary validation is better

# Safe Data Handling

- Write code that can't be fooled by malicious data
  - SQL parameterized queries
  - Don't pass user input to an OS command line
- Effective when it can be applied

Incorrect

```
SELECT * FROM users WHERE username =  
'input_from_user';
```

Correct

```
SELECT * FROM users WHERE username = ?;
```



# Semantic Checks

- Some malicious input is identical to valid input
- Such as changing an account number to another customer's number
- Data must be validated in context
- Does this account number belong to the currently logged-in user?

# Difficulties with Simple Input Validation



- Data coming from user is "bad" or "untrusted"
- The server-side app is "good" and trusted
- Many different types of input with different filtering requirements
- Apps may chain several processing steps together
- Data may be harmless at one stage, but be transformed into harmful data at another stage

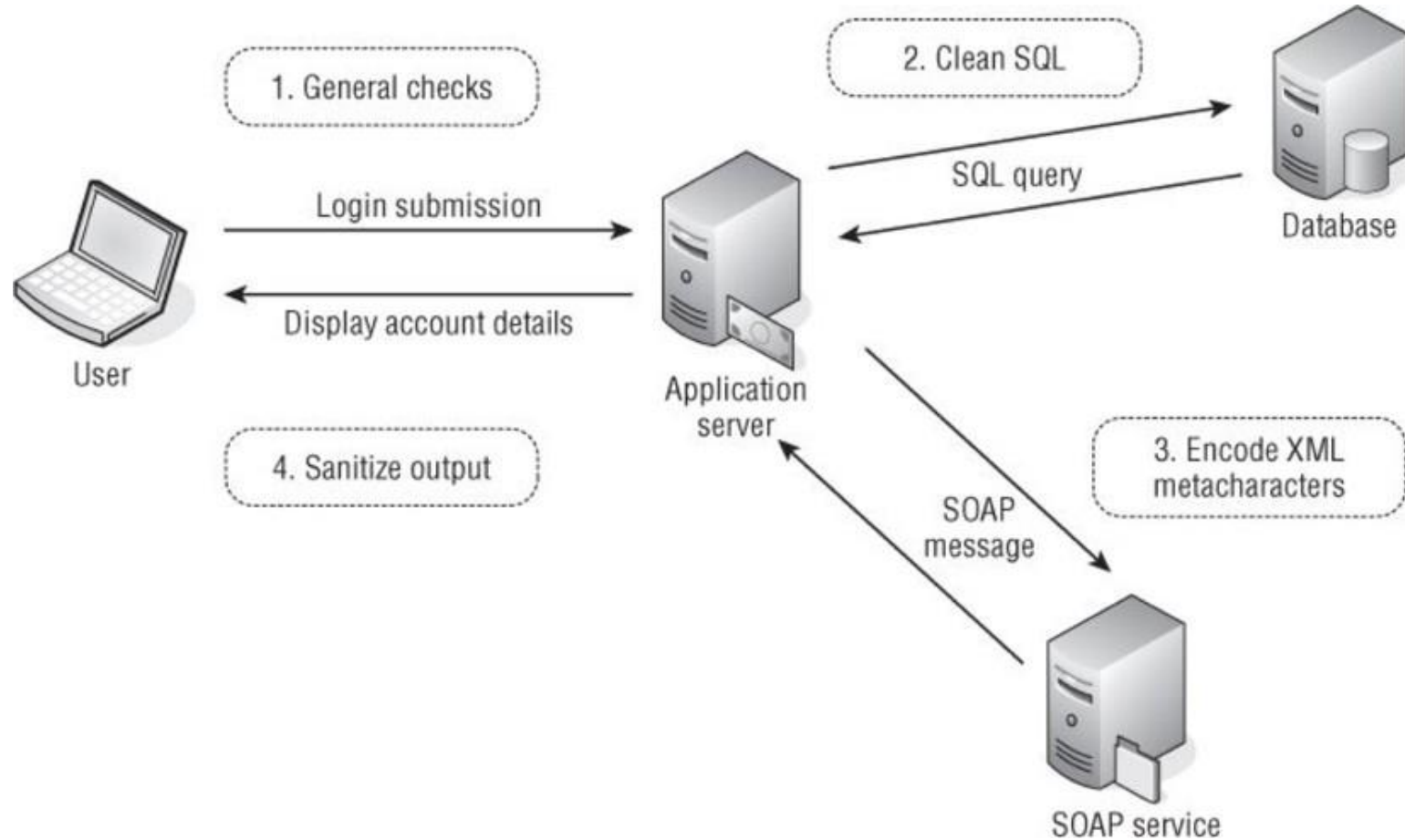
# Boundary Validation

- Trust boundary
- Divides a trusted zone from an untrusted zone  
Clean data that passes a boundary
- Such as from the user into an application

# Boundary Validation

- Each component treats its input as potentially malicious
- Data validation performed at each trust boundary
- Not just between client and server

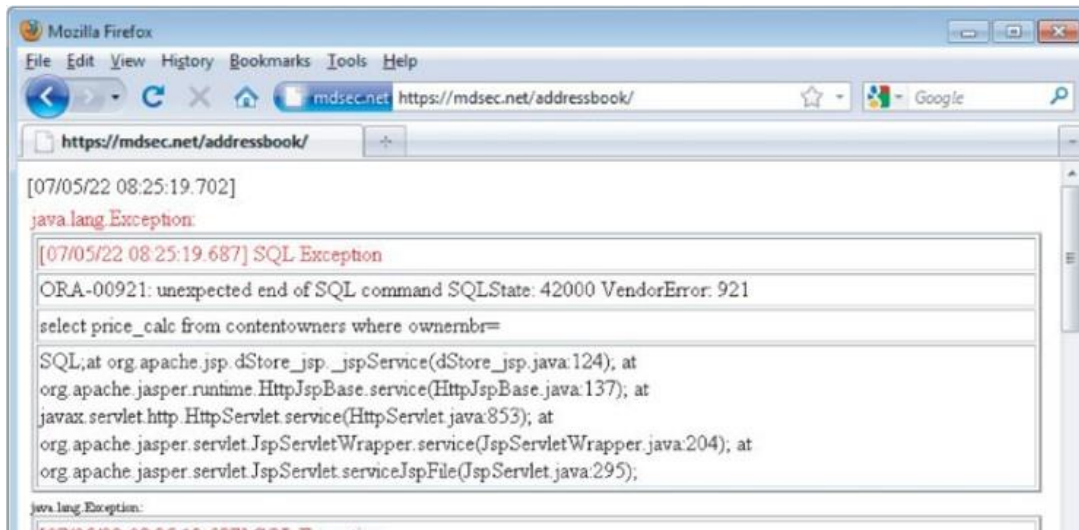
# Example



# Handling Attackers

- Handling errors
- Maintaining audit logs
- Alerting administrators
- Reacting to attacks

# Handling Errors



## Best Practices for Error Handling

- **User-Friendly Messages:** When an error occurs, present users with simple, non-technical messages indicating that something went wrong, without revealing internal details.
- **Consistent Error Codes:** Use consistent error codes or identifiers that can help support teams diagnose issues without exposing sensitive information.
- **Logging Sensitively:** While logs are crucial for debugging, be cautious not to log sensitive information (e.g., passwords, credit card numbers) to avoid data leaks.
- **Regular Audits:** Periodically review error handling and logging practices to ensure they comply with security standards and address any vulnerabilities.
- **Testing Error Scenarios:** Include tests for error handling during development,

# Audit Logs

- Authentication events: login success and failure, change of password
- Key transactions, such as credit card payments
- Access attempts that are blocked by access control mechanisms
- Requests containing known attack strings
- For high security, log every client request in full



# Protecting Logs

- Logs should contain time, IP addresses, and username
- May contain cookies and other sensitive data
- Attackers will try to erase and/or read logs
- Log must be protected
- Place on an autonomous system that only accepts update messages
- Flush logs to write-once media

# Alerting Administrators

- Usage anomalies, like a large number of requests from the same IP address or user
- Business anomalies, such as a large number of funds transfers to/from the same account
- Requests containing known attack strings
- Requests where hidden data has been modified

# Firewalls

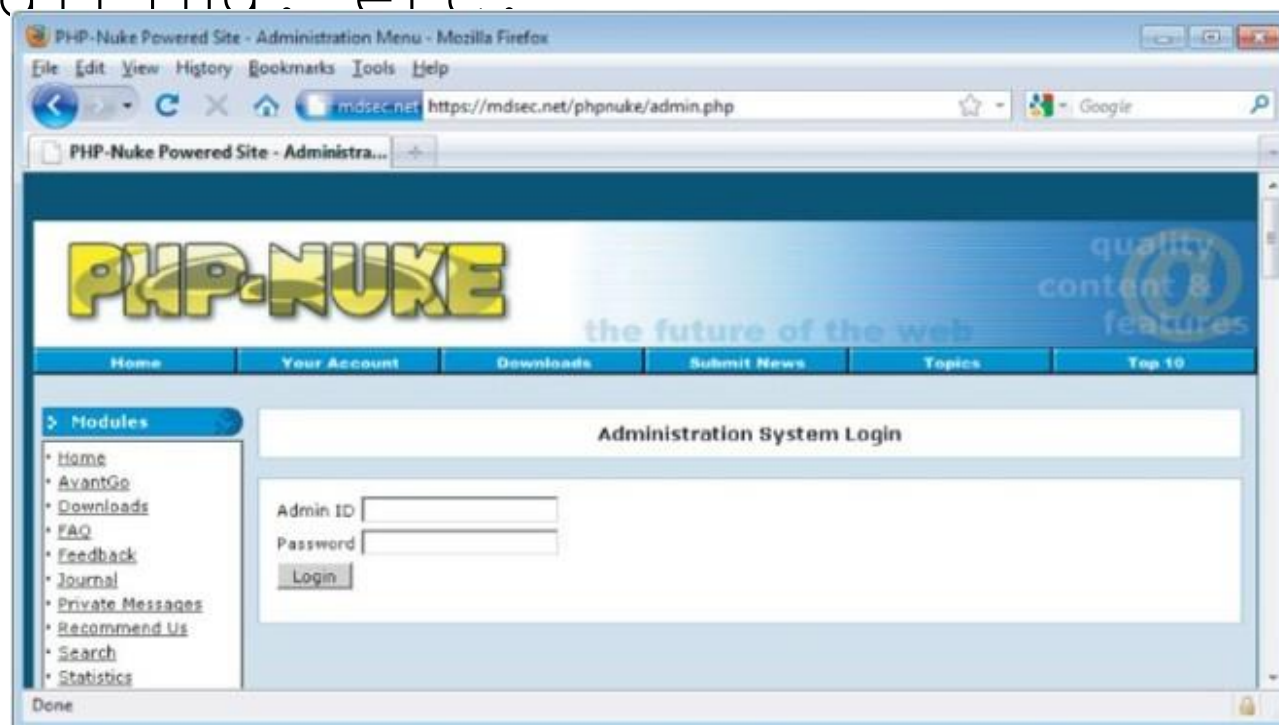
- Web App Firewalls can detect generic attacks
- But not subtle ones that are specific to your app
- Most effective security control is integrated with app's input validation mechanisms
- Check for valid values of your parameters

# Reacting to Attacks

- Attackers probe for vulnerabilities
- Sending many similar requests
- Automated defense
- Respond increasingly slowly to requests
- Terminate the attacker's session

# Managing the Application

- Management interface allows administrator to control user accounts, roles, access monitoring, auditing, etc.



# Attacking the Administration Panel



- Defeat weak authentication
- Some administrative functions might not require high privileges
- XSS flaws can allow cookie theft and session hijacking