



# Web Application Security

## Lecture:05

### Attacking Authentication

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

# Authentication Technologies

- HTML forms-based authentication
  - Multifactor mechanisms, such as those combining passwords and physical tokens
  - Client SSL certificates and/or smartcards
  - HTTP basic and digest authentication
  - Windows-integrated authentication using NTLM or Kerberos
  - Authentication services
- 
- **Over 90% of apps use name & password**

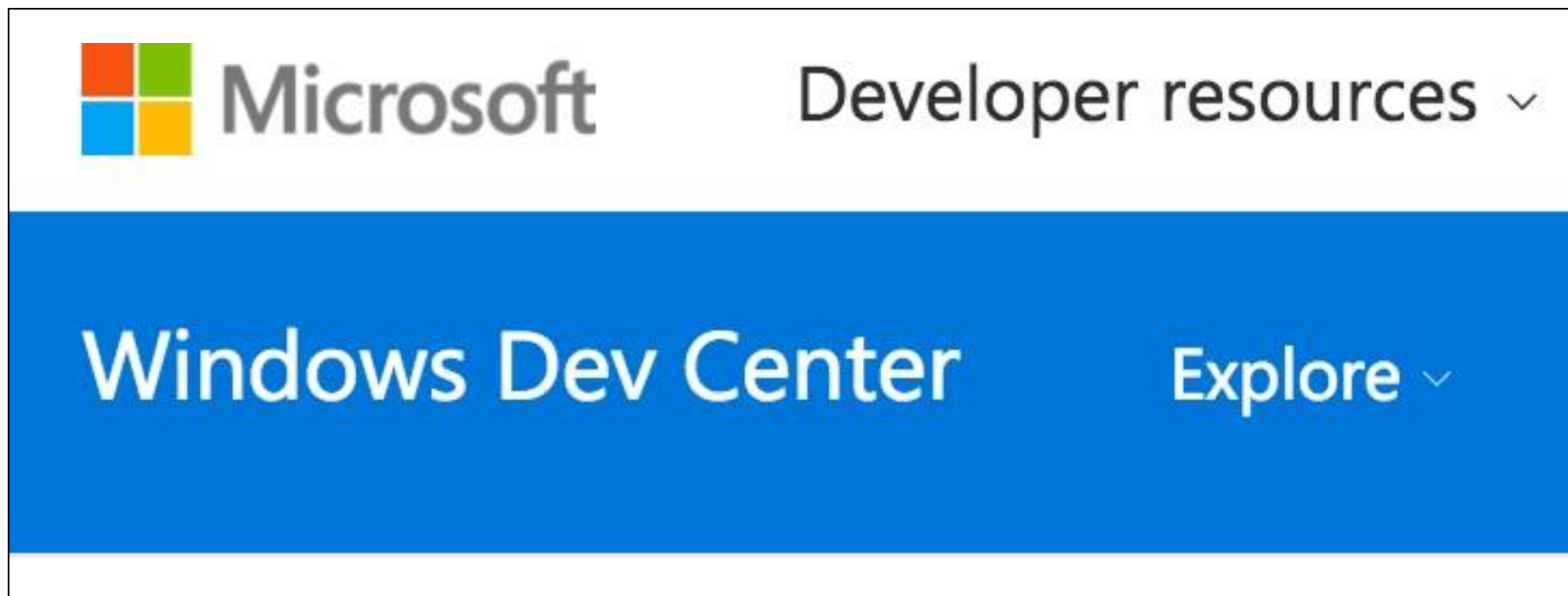
# Cryptographic Methods

- **Client-side SSL certificate**
- **Smartcards**
- **More expensive, higher overhead**

# HTTP Authentication


- **Basic, Digest, and Windows-integrated**
- **Rarely used on the Internet**
- **More common in intranets, especially Windows domains**
  - **So authenticated employees can easily access secured resources**


# Third-Party Authentication




Microsoft Passport and Windows Hello

# Third-Party Authentication

 SlideShare

 | Login with LinkedIn

 | Login with Facebook

or

Login with your SlideShare account

☒ Keep me logged in

Login

[Forgot password?](#)

# Design Flaws

- **Bad Passwords**
  - Very short or blank
  - Common dictionary words or names
  - The same as the username
  - Still set to a default value

# Brute-Force Attacks

- **Attackers use lists of common passwords**
- **Defense: account lockout rules after too many failed login attempts**

- password
- *website name*
- 12345678
- qwerty
- abc123
- 111111
- monkey
- 12345
- letmein



# Poor Attempt Counters

- **Cookie containing failedlogins=1**
- **Failed login counter held within the current session**
  - **Attacker can just withhold the session cookie to defeat this**
- **Sometimes a page continues to provide information about a password's correctness even after account logout.**

# Verbose Failure Messages



The image displays two side-by-side login forms, each enclosed in a brown border. Both forms have a 'Username:' label, a text input field, a 'Password:' label, another text input field, and a 'Login' button.

The left form shows a successful login attempt with the username 'daf' entered. Below the input fields, the message 'Password is incorrect.' is displayed in red text.

The right form shows a failed login attempt with the username 'zzz' entered. Below the input fields, the message 'User is not recognised.' is displayed in red text.

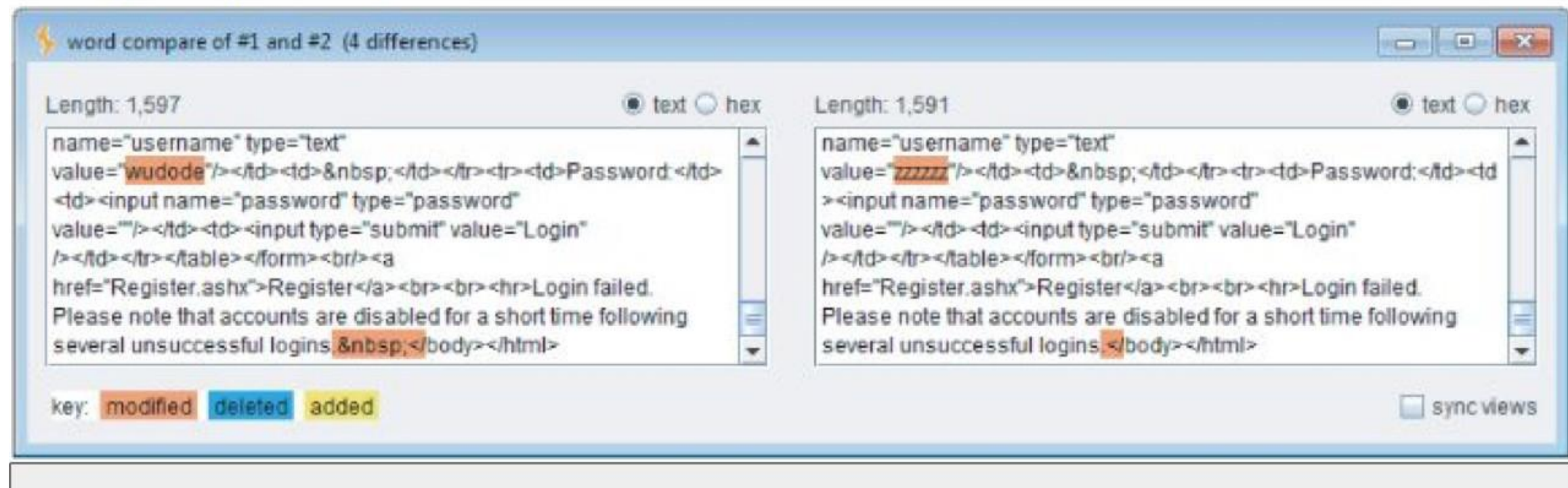
- **Friendly for legitimate users**
- **But helpful for attackers**

# Username Importance

- **Attacks that reveal valid usernames are called "username enumeration"**
- **Not as bad as finding a password, but still a privacy intrusion**
- **But could be used for social engineering attacks, such as spearphishing emails**

# Similar but Non-Identical Error Messages

**Figure 6.4** Identifying subtle differences in application responses using Burp Comparer



- Any difference can be exploited; even response time

# Example

Even when error messages look similar, slight variations can reveal information. For example, if a web application returns the same message for both valid and invalid login attempts, but the response time differs, an attacker can infer which usernames are valid based on the timing.

**Invalid username:** "Login failed. Please check your credentials." (Response time: 1 second)

**Valid username but incorrect password:** "Login failed. Please check your credentials." (Response time: 3 seconds)

An attacker might notice that a particular username always results in a longer response time, indicating that it's valid even though the password is incorrect.

# Vulnerable Transmission of Credentials

- **Eavesdroppers may reside:**

- On the user's local network
- Within the user's IT department
- Within the user's ISP
- On the Internet backbone
- Within the ISP hosting the application
- Within the IT department managing the application

# HTTPS Risks

- **If credentials are sent unencrypted, the eavesdropper's task is trivial, but even HTTPS can't prevent these risks:**
  - **Credentials sent in the query string are likely to appear in server logs, browser history, and logs of reverse proxies**
  - **Many sites take credentials from a POST and then redirect it (302) to another page with credentials in the query string**

# HTTPS Risks

- **Cookie risks:**
  - **Web apps may store credentials in cookies**
  - **Even if cookies cannot be decrypted, they can be re-used**
- **Many pages open a login page via HTTP and use an HTTPS request in the form**



# Password Change Functionality

- **Periodic password change mitigates the threat of password compromise (a dubious claim)**
- **Users need to change their passwords when they believe them to be compromised**

# Vulnerable Password Change Systems

- **Reveal whether the requested username is valid**
- **Allow unrestricted guesses of the "existing password" field**
- **Validate the "existing password" field before comparing the "new password" and "confirm new password" fields**
  - **So an attacker can test passwords without making any actual change**

# Password Change Decision Tree

- **Identify the user**
- **Validate "existing password"**
- **Integrate with any account lockout features**
- **Compare the new passwords with each other and against password quality rules**
- **Feed back any error conditions to the user**
- **Often there are subtle logic flaws**

# Forgotten Password Functionality

- **Often the weakest link**
  - **Uses a secondary challenge, like "mother's maiden name"**
  - **A small set of possible answers, can be brute-forced**
  - **Often can be found from public information**

# Forgotten Password Functionality

- **Often users can write their own questions**
  - **Attackers can try a long list of usernames**
  - **Seeking a really weak question, like "Do I own a boat?"**
- **Password "Hints" are often obvious**

# Forgotten Password Functionality

- **Often the mechanism used to reset the password after a correct challenge answer is vulnerable**
- **Some apps disclose the forgotten password to the user, so an attacker can use the account without the owner knowing**
- **Some applications immediately let the user in after the challenge--no password needed**

# Forgotten Password Functionality

- **Some apps allow the user to specify an email for the password reset link at the time the challenge is completed**
  - **Or use email from a hidden field or cookie**
- **Some apps allow a password reset and don't notify the user with an email**

# "Remember Me"

- **Sometimes a simple persistent cookie, like**
  - `RememberUser=jsmith`
  - `Session=728`
- **No need to actually log in, if username or session ID can be guessed or found**



# "Remember Me"

- **Even if the userid or session token is properly encrypted, it can be stolen via XSS or by getting access to a user's phone or PC**

# User Impersonation

User impersonation functionality allows certain users, typically administrators or support staff, to "act as" another user within an application. This lets them access that user's data or take actions on their behalf.

This means they temporarily view and access the app as if they were the customer. When they do this, the support person can:

1. See the customer's transactions
2. Help them navigate the application
3. Make limited account changes, if necessary

# User Impersonation

Risk associated

## **1. Hidden or Poorly Protected Function**

The bank's application has a URL, /admin/ImpersonateUser.jsp, that allows any user who knows the URL to impersonate another user.

## **2. Trusting User-Controllable Data**

When a support staff member accesses a customer's account, the application relies on a cookie that stores the customer's account ID. If an attacker can alter this cookie value, they may be able to switch accounts without proper authentication, accessing sensitive data from other user accounts.

## **3. Backdoor Password**

Some systems may have a "master password" that support staff can use with any username to log in.

# Incomplete Validation

## Truncated Passwords

Some applications truncate passwords without telling users

- **Stripping Special Characters**

If the application removes

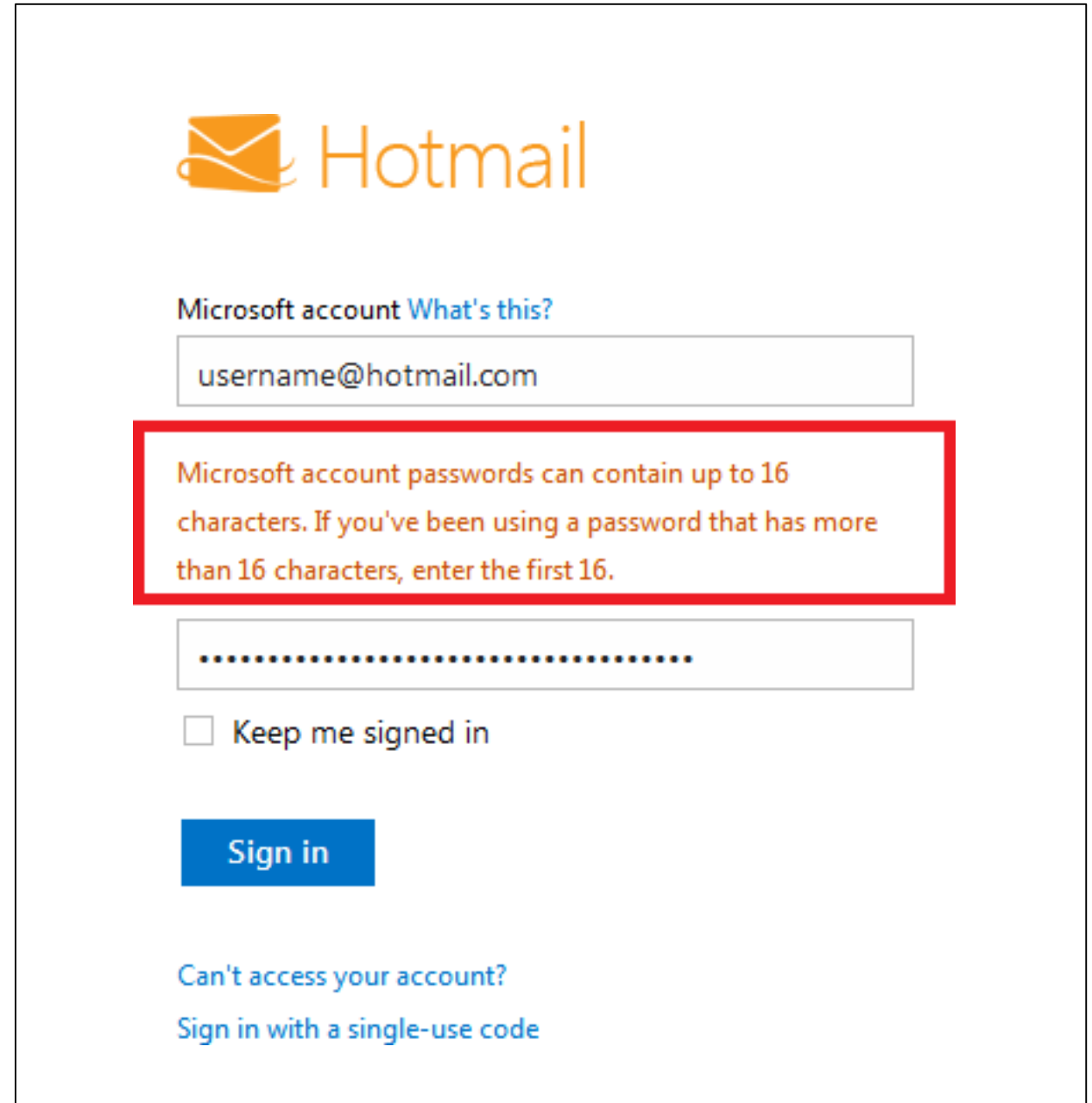
- special characters (like !@#\$%)

from passwords, then a

password like "P@ssw0rd!"

might be treated as "Pssw0rd".

This reduces the complexity of the password, making it easier to guess.



The screenshot shows the Hotmail login interface. At the top is the Hotmail logo. Below it is a link for "Microsoft account What's this?". A text input field contains the placeholder "username@hotmail.com". Below the input field is a red-bordered box containing the message: "Microsoft account passwords can contain up to 16 characters. If you've been using a password that has more than 16 characters, enter the first 16." Below this box is a password input field with dots. Underneath the password field is a checkbox labeled "Keep me signed in". At the bottom is a blue "Sign in" button. Below the button are two links: "Can't access your account?" and "Sign in with a single-use code".

# Nonunique Usernames

Suppose two people register with the username "john\_doe" and both set the password "Password123!". If the application allows this, it either:

**Rejects** one user's password (revealing that another user has the same credentials).

**Allows** both users to log in with identical credentials, potentially giving each other access to the other's account.

An attacker registers "admin" with several different passwords. If they receive a response indicating that "admin" and a specific password combination already exist, they've learned the correct password without directly attacking the login.

# Predictable Usernames

When usernames follow a clear, sequential pattern, attackers can deduce other users' usernames without interacting much with the system. They can use this knowledge as a basis for further attacks, such as brute-force password guessing or phishing attacks.

## Example

Imagine a banking app that assigns new customers usernames like:

**User1001**

**User1002**

**User1003**

An attacker who notices this pattern can predict other usernames by increasing the numbers, generating a list of potential users for the app.

# Predictable Initial Passwords

Predictable initial passwords occur when an application assigns temporary passwords that are easily guessed or follow a predictable pattern. This is common in environments where many accounts are created at once, like a corporate intranet, and each user is given a generated password.

**Username-Based Passwords:** Sometimes, applications generate passwords based on the username, like combining the username with a number or a predictable suffix (e.g., for user jdoe, the password might be jdoe2023). If attackers recognize this pattern, they can predict other users' passwords based on their usernames.

**Sequential or Simple Patterns:** Generated passwords might follow a pattern like "Password001," "Password002," and so on. If attackers discover a few sample passwords, they could guess other users' passwords in the batch.

# Insecure Distribution of Credentials

**Emailing Both Username and Password:** A new user receives an email that includes both their username and password with no prompt to change the password upon first login. If this email is accidentally forwarded or accessed by someone else, they could log in as the user.

Risk: Many users don't change these initial credentials, meaning the email remains an open security risk if not deleted.

## **Predictable Account Activation URLs:**

A new user gets an email with a unique activation link to set their password. However, if the activation URLs follow a predictable sequence, an attacker could register multiple dummy accounts, observe the pattern, and predict other users' activation URLs.

Risk: The attacker could hijack accounts by accessing these activation URLs before the intended user sets their password.

these links should be unique and difficult to guess.

ExampleImagine an application generates account activation links in a sequence like:

<https://app.com/activate?user=1001>

<https://app.com/activate?user=1002>

<https://app.com/activate?user=1003>



# Implementation Flaws

# Fail-Open Login

```
public Response checkLogin(Session session) {  
    try {  
        String uname = session.getParameter("username");  
        String passwd = session.getParameter("password");  
        User user = db.getUser(uname, passwd);  
        if (user == null) {  
            // invalid credentials  
            session.setMessage("Login failed. ");  
            return doLogin(session);  
        }  
    }  
    catch (Exception e) {}  
  
    // valid user  
    session.setMessage("Login successful. ");  
    return doMainMenu(session);  
}
```

# Fail-Open Login

If there's an error (an "exception") in the code, such as if `db.getUser()` fails or there's a missing parameter, the `catch (Exception e) {}` block catches the error but does nothing with it. Instead of denying access, the program proceeds as if the login was successful by default, setting the "Login successful" message and letting the user into the main menu (`doMainMenu(session)`).

## Fail-Open Logic:

In this example, if the code encounters an unexpected issue (like missing login information), it "fails open," meaning it grants access rather than denying it. The code treats the lack of login data as a successful login instead of blocking access.

# Multistage Login

- **Enter username and password**
- **Enter specific digits from a PIN (a CAPTCHA)**
- **Enter value displayed on a token**

# Multistage Login Defects

## **Unsafe Assumptions Between Stages:**

The application might assume that if a user has reached the third stage, they must have successfully completed the first two stages. An attacker could exploit this assumption by directly accessing the third stage after only completing the first stage. If they enter correct information for the third stage, they might be granted access without needing the first two credentials.

**Risk:** This allows attackers to bypass necessary checks and gain unauthorized access with incomplete authentication.

# Multistage Login Defects

Some systems use randomly selected security questions to further enhance security. However, there are issues with this approach that can undermine its effectiveness:

## 1. Storing Questions Insecurely:

**Example:** The application may display a randomly chosen question but store the details in a hidden form field or cookie. When the user submits an answer, they also submit the question itself.

**Risk:** An attacker can capture the user's input (the question and answer) and later choose which question to answer, thus gaining access.

## 2. Recycling Questions:

**Example:** If a user fails to answer a question and tries to log in again, the application may generate a different question. An attacker could cycle through multiple login attempts until they receive a question they know the answer to.

# Insecure Credential Storage

- **Plaintext passwords in database**
- **Hashed with MD5 or SHA-1, easily cracked**

# Securing Authentication

Implementing a secure authentication solution is a complex task that requires careful consideration of various security objectives, user experience, and cost factors.

- **Will users tolerate inconvenient controls?**
- **Cost of supporting a user-unfriendly system**
- **Cost of alternatives, compare to revenue generated or value of assets**



# Strong Credentials

- **Minimum password length, requiring alphabetical, numeric, and typographic characters**
- **Avoiding dictionary words, password same as username, re-use of old passwords**
- **Username should be unique**
- **Automatically generated usernames or passwords should be long and random, so they cannot be guessed or predicted**
- **Allow users to set strong passwords**

# Handle Credentials Secretively

- **Protect them when created, stored, and transmitted**
- **Use well-established cryptography like SSL, not custom methods**
- **Whole login page should be HTTPS, not just the login button**
- **Use POST rather than GET**
- **Don't put credentials in URL parameters or cookies**

# Handle Credentials Secretively

- **Don't transmit credentials back to the client, even in parameters to a redirect**
- **Securely hash credentials on the server**

# Hashing

- **Password hashes must be salted and stretched**
- **Salt: add random bytes to the password before hashing it**
- **Stretched: many rounds of hashing (Kali Linux 2 uses 5000 rounds of SHA-512)**

# Handle Credentials Secretively

- **Client-side "remember me" functionality should remember only nonsecret items such as usernames**
- **If you allow users to store passwords locally, they should be reversibly encrypted with a key known only to the server**
  - **And make sure there are no XSS vulnerabilities**

# Handle Credentials Secretively

- **Force users to change passwords periodically**
  - *No longer recommended*
- **Credentials for new users should be sent as securely as possible and time-limited; force password change on first login**
- **Capture some login information with drop-down lists instead of text fields, to defeat keyloggers**

# Validate Credentials Properly

- **Validate entire credential, with case sensitivity**
- **Terminate the session on any exception**
- **Review authentication logic and code**
- **Strictly control user impersonation**

# Multistage Login

- **All data about progress and the results of previous validation tasks should be held in the server-side session object and never available to the client**
- **No item of information should be submitted more than once by the user**
- **No means for the user to modify data after submission**



# Multistage Login

- **The first task at every stage should be to verify that all prior stages have been correctly completed**
- **Always proceed through all stages, even if the first stage fails--don't give attacker any information about which stage failed**

# Prevent Information Leakage

- **Ensure that error messages or application behavior does not reveal which specific login credential (username, password, etc.) was incorrect.**
- **Use a single, consistent error message for all login failures**
- **Avoid disclosing account lockout details (e.g., “Your account has been locked for 15 minutes”) in a way that allows attackers to confirm usernames or optimize brute-force attempts.**

# Self-Registration

- **Allowing users to choose usernames permits username enumeration.**

**Better methods:**

- **Generate unique, unpredictable usernames**
- **Use Email-Based Registration with Verification**

# Prevent Brute-Force Attacks

## **Enforce Controls Across All Authentication Points**

If the login page has CAPTCHA but the password reset page doesn't, attackers may focus their brute-force attempts on the unprotected page to bypass login security.

## **Use Unpredictable Usernames and Block Enumeration**

Making usernames unpredictable or limiting the ability to enumerate usernames makes it harder for attackers to target specific accounts.

## **Temporary Account Suspension for Failed Logins**

After three failed login attempts, an account might be suspended for 30 minutes. This approach reduces successful brute-force attacks but minimizes user inconvenience.

## **Implement CAPTCHA**

CAPTCHA challenges on key authentication pages help prevent automated scripts from launching brute-force attacks.

Example: A login page with CAPTCHA prevents bots from automating password guesses. Even though CAPTCHA can sometimes be bypassed, it deters many attackers from continuing.

# Prevent Brute-Force Attacks

- **Password Spray attack**
  - **Use many different usernames with the same password, such as "password"**
  - **Avoids account lockouts**
- **Defenses: strong password rules, CAPTCHA**



# Password Change

**To prevent misuse of the password change function, it's essential to secure the mechanism against unauthorized use, accidental errors, and certain types of attacks.**

- **The password change function should only be accessible when the user is logged in.**
- **Require user to enter the old password**
- **Require new password twice (to prevent mistakes)**
- **Same error message for all failures**
- **Users should be notified out-of-band (such as via email) that the password has been changed**

# Account Recovery

## **Use Out-of-Band Recovery for High-Security Applications:**

For highly sensitive applications, like online banking, out-of-band recovery adds an extra layer of protection, such as requiring a phone call or mailing a reactivation code. This approach reduces the chance that an attacker could reset the password online without direct access to the account owner.

**Don't use "password hints"**

**Email a unique, time-limited, unguessable, single-use recovery URL**

# Account Recovery

- **Challenge questions**
  - **Don't let users write their own questions**
  - **Don't use questions with low-entropy answers, such as "your favorite color"**



# Log, Monitor, and Notify

## **Comprehensive Logging of Authentication Events**

### **What to Log:**

The application should log all authentication-related actions, including:

- Successful and failed logins
- Password changes
- Password resets
- Account suspensions
- Account recovery

**Details to Include:** Relevant information should be recorded, such as:

- Username
- IP address
- Timestamp of the event

# Log, Monitor, and Notify

## **Real-Time Anomaly Detection**

**Monitoring for Anomalies:** The application should have mechanisms to detect unusual patterns in authentication events that could indicate security threats, such as:

- Multiple failed login attempts from a single IP address (brute-force attack)
- Logins from unusual geographic locations

**Alerting Administrators:** When anomalies are detected, real-time alerts should be sent to application administrators. This enables them to take immediate action, such as temporarily suspending accounts or blocking suspicious IP addresses.

# Log, Monitor, and Notify

## **Out-of-Band User Notifications**

**Critical Security Events:** Users should receive notifications through a secure, out-of-band method (like email) for critical events, such as:

- Password changes
- Account recoveries

**Purpose:** These notifications help users to quickly identify unauthorized actions taken on their accounts, allowing them to react promptly (e.g., changing their password if they did not initiate the change).

# Log, Monitor, and Notify

## **In-Band Notifications for Users**

**Frequent Security Events:** Users should be notified within the application of ongoing security events, such as:

- The time and IP address of their last successful login
- The number of failed login attempts since their last login

**Rationale:** In-band notifications raise user awareness about potential attacks on their account. If users are informed that their account is being targeted, they may be motivated to change their passwords more frequently and choose stronger passwords.