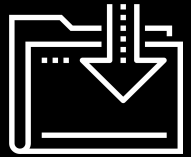


Course: Java

S1



# Learning Outcomes

---

By the end of this lesson, you will be able to:

01

Identify security concerns to consider when developing web applications.

02

Create a permissions diagram for a web application.

03

Discuss the basics of web application security:

- Penetration testing
- Ethical hacking
- OWASP Top 10

04

Discuss common security problems in web applications and their potential solutions.

05

Describe the role the Authentication Manager plays in Spring Security.

06

Describe the role the User Details Service plays in Spring Security.

# Learning Outcomes (continued)

---

07

Describe the role the Password Encoder plays in Spring Security.

08

Describe the role the Principal plays in Spring Security.

09

Describe the role the Users play in Spring Security.

10

Describe the role the Authorities play in Spring Security.

11

Describe role-based access control (RBAC).

12

Compare and contrast authentication and authorization.

13

Describe Cross-Site Request Forgery (CSRF).

14

Explain endpoint authorization in the context of Spring Security.

# Security Concerns



**How is access controlled to  
a multi-tenant office building?**



**How could someone break into the different parts of the building?**

# Security Basics

# Authentication

---



Are you *authentically* you? Are you who you claim to be?



Why do we use it?



How do we get into different parts of a building?

- Keys
- Keypad entry
- Thumbprint authentication
- Retina scan



Credential-based authentication





# Authentication (continued)

---



Authentication is positively identifying a user—in other words, making sure a user is who they say they are.



Computer/web-based authentication is just an extension of something we do in the analog world.

- We have government-issued picture IDs.
- We can be positively identified by our fingerprints or retina scans.
- We are asked to present our ID when cashing checks, checking into a hotel, flying on commercial airlines.



There are several different ways we can determine the identity of a user:

- Username/password
- Fingerprint
- Certificates/keys

# Authorization

---



What are you *authorized* to do? (What are you allowed to do?)



What does having authorization allow the users to do?



What are the risks when giving authorization?



Role-based access control (RBAC)



Secured resources



# Authorization (continued)

---



Authorization is all about who can do what, sometimes known as **permissions**.



Authorization assumes that authentication has worked and we know who the user is.



Permissions are generally not applied directly to users. Permissions are granted to groups or roles. User are assigned to groups or roles and assume the permissions of the groups or roles they are in.



This is referred to as **role-based access control (RBAC)**.



Time to Code

Create Spring Security Project

# Create a user with credentials (username, password)

---

Now, we will begin adding security to the app. We will do this in several steps.

01

Create a user with credentials (username, password).

02

Create authorization rules for that user.

03

Set up a security configuration.

04

Enable security on the application.

# Password Encoder

---

- Storing plain text passwords in a database is a security risk.
- A Password Encoder encodes incoming passwords before comparing them to the values in the database.
- Spring Security 5 requires us to choose and configure a Password Encoder.
- The best choice for this if there are no other constraints is the BCrypt encoder.
- The following must be true for encoded passwords to work:
  - Passwords must be encoded before being stored in the database.
  - Incoming passwords must be encoded before being compared to values in the database.
  - Incoming passwords and passwords in the database must be encoded using the same encoder.

6FE4861EGB2



Create a password encoder utility



Time to Code

Spring Security Schema and Enable Security





# Time to Code

## Add Endpoints

A close-up, high-angle shot of a computer keyboard. The central focus is a large, white, rectangular key with rounded corners. On this key, there is a dark blue icon of a coffee cup with three wavy lines above it representing steam. Below the icon, the word "Break" is printed in a dark blue, serif font. The key is set against a light-colored, textured keyboard surface. Other keys are visible in the background, including one with a double quote symbol and another with a dash/slash symbol, but they are out of focus.

Break

# Spring Security Concepts

# Users

---



- A **user** is someone who uses the system in question.
- In Spring, a user refers to the account associated with the person who uses the system.
- User accounts are generally identified by unique usernames.
- The most common way to authenticate a user account is via a password.

# Authorities

---



Spring uses the term **authorities** to describe the roles or groups to which a user can belong.



In Spring parlance, various authorities are granted to users.

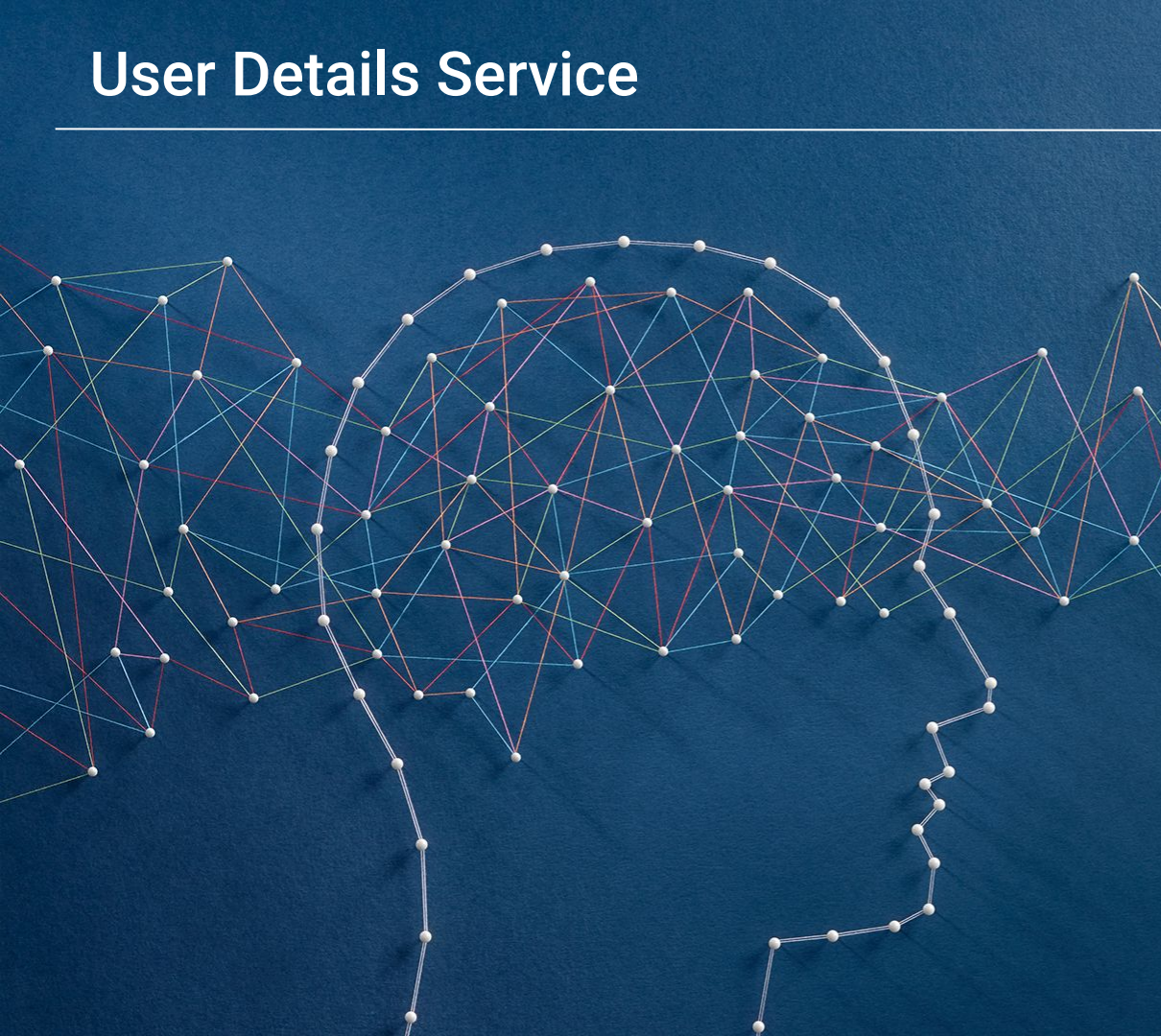


As we've seen, Authorities are arbitrary strings that have no inherent meaning. We get to define the authorities for our applications and we get to assign meaning to these authorities as we wish.



# User Details Service

---

- 
- The **User Details Service** is the Spring Security component responsible for retrieving information about logged in users.
  - We can create custom User Details Service components to fit whatever user repository our application uses.
  - Spring Security does have a default implementation if you use a standard user store and access it in a standard way.

# Authentication Manager

---

- This component is responsible for authenticating users in the system.
- It integrates with whatever user repository our application uses and has the ability to find users by username and find their associated authorities.
- Spring has some default implementations for common user repositories.



# Security Schema

---



Spring Security supplies a standard SQL schema that can be used for user and authority data. This is the schema we've defined already.



This works well for projects that are being built from scratch or that don't already have a user repository.



Spring Security makes it easy to incorporate this schema into the Authentication Manager for your application.



This is a great choice if you don't already have a user/authority store.



# Data Source

This provides database connection configuration if you choose to use the Spring Security default SQL schema.

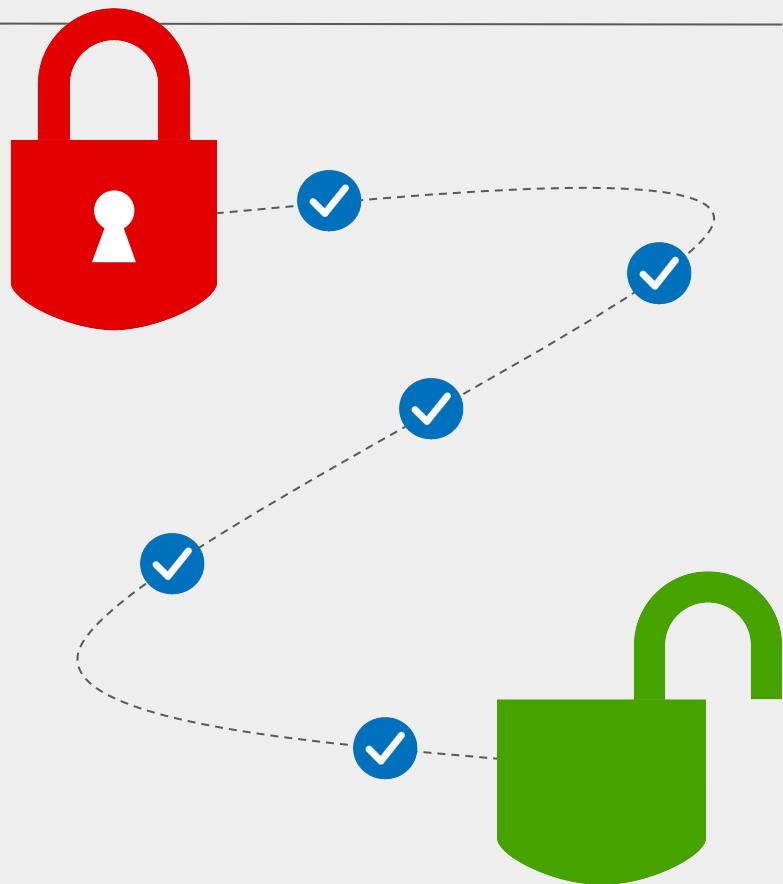
This is the same data source used for general JDBC database connectivity.



# Endpoint Authorization

---

- Spring Security applies authorization rules to individual or groups of endpoints.
- This is done by defining which authorities (typically referred to as roles in this context) a user must have in order to access a given endpoint.
- We have the freedom to:
  - Leave some endpoints open to everyone.
  - Leave some endpoints open to all authenticated users.
  - Limit access to some endpoints to only those users who have been granted particular authorities.



# Principal

---

- The **Principal** is an object that represents a logged in user of the system.
- The Principal includes information such as the user's name and granted authorities.
- This is the object we use to get information about the logged in user in our code.



# Logout

---



Spring Security supports the ability to log users out of the system.



We can define the following:

- The URL to be accessed when requesting to be logged out.
- The URL to which the user should be redirected upon successful logout.
- Cookies we want to delete.
- Whether or not the HTTP session should be invalidated on logout.

# Cross-Site Request Forgery (CSRF)

---

According to OWASP, the definition of Cross-Site Request Forgery attacks is:

**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. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application."

- Spring Security has CSRF protection turned on by default.
- The CSRF protection is configurable so that it is compatible with a wide variety of UI/client technologies.



# Time to Code

Configure CSRF Protection,  
Logout, and Principal



# Time to Code

Use principal, and implement logout





# Administrative Features

# Administrative Features

---

Now, we have the end-user functionality running and secure in our application. We need to add administrative features, too.

We need a new role for event publishers. To start, add a new endpoint, only available to event publishers, that allows them to view a list of guests registered for an event.





## Add an Endpoint that Requires a New Authority

Suggested Time:

---



Time's Up! Let's Review.

# Access is explicitly granted, and not inherently hierarchical

---

Now, what happens when a publisher to register for a private event?

01

Have your new publisher user (billtries) try to use the `/privateEvent` endpoint. Go to <http://localhost:8080/privateEvent>

02

Observe that the user can't.

- i. The HTTP response is 403 - Forbidden.
- ii. This is because authorities are not inherently hierarchical.
- iii. In other words, the user with the most access doesn't automatically get basic access.
- iv. We can make it appear this way (hierarchical) by assigning all lower level access to "upper level" users.



Time to Code

Make Access Hierarchical



# Time to Code

## Add an admin endpoint

# Control Access Based on HTTP Method





# Time to Code

## Control Multiple HTTP Methods

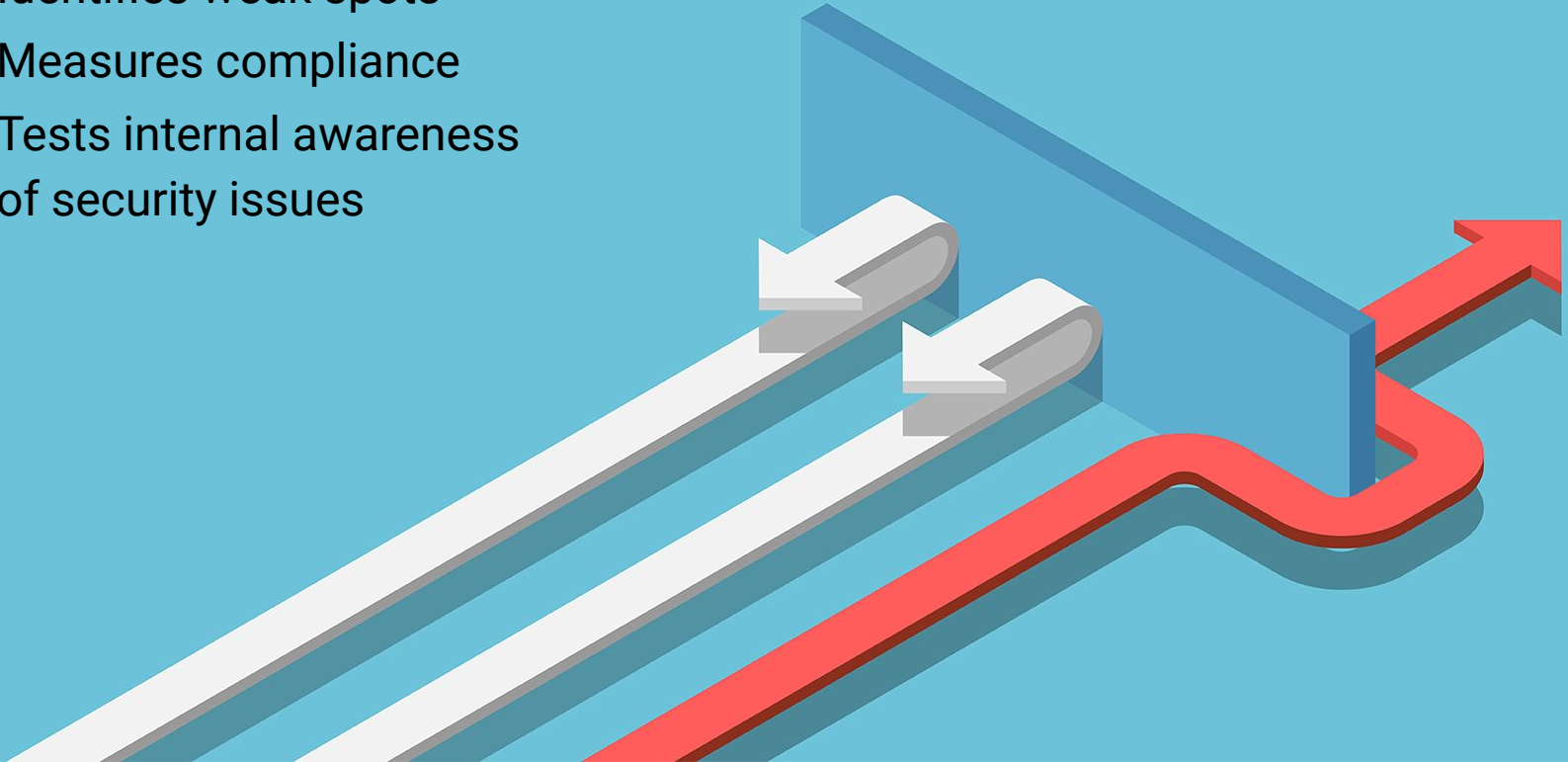


Break

# Penetration Testing aka Pen Testing

---

- Simulated cyber attack
- Identifies weak spots
- Measures compliance
- Tests internal awareness of security issues



# Ethical Hacking

---

“What is  
Ethical Hacking?”

<https://www.eccouncil.org/ethical-hacking/>



**“The Open Web Application Security Project is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.” The OWASP Top 10 is a “broad consensus about the most critical security risks to web applications.”**

[https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)



# OWASP Top 10

---

01

**\*\*Injection \*\***

02

Broken Authentication

03

Sensitive Data Exposure

04

XML External Entities

05

Broken Access Control

06

Security Misconfiguration

07

Cross-site Scripting

08

Insecure Deserialization

09

Using Components with  
Known Vulnerabilities

10

Insufficient Monitoring  
& Logging

# Database Security



“**Anderson's Rule** means you cannot construct a database with scale, functionality and security because if you design a large system for ease of access it becomes insecure, while if you make it watertight it becomes impossible to use.”

Ross J. Anderson

Source: <https://www.theguardian.com/commentisfree/henryporter/2009/aug/10/id-card-database-breach>





**“SQL injection** is a code injection technique that might destroy your database. SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.”



Source: [https://www.w3schools.com/sql/sql\\_injection.asp](https://www.w3schools.com/sql/sql_injection.asp)

“A **prepared statement** is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.”

Source: [https://www.w3schools.com/php/php\\_mysql\\_prepared\\_statements.asp](https://www.w3schools.com/php/php_mysql_prepared_statements.asp)





# Questions?

