SCL201 - DEFENDING SCALA

Course Learning Objectives

Defending Scala is a course designed to provide you with an overview of security concepts that affect the Scala programming language. In module one, we explore the different authentication and authorization vulnerabilities. In module two, we look at best practices for defending against common vulnerabilities and framework-related malpractices, including Cross-site Request Forgery and SQL injection attacks. In module three, we examine different vulnerabilities related to data validation and logging and different ways of validating data. In module four, we discuss how improper file handling could compromise application security. And finally, in module five, we look at different cryptographic methods that are available in Scala.

Description

While Scala and Java share many similarities in terms of security, many Scala-focused security solutions are not yet ready for production. Defending Scala covers best security practices along the SDLC when working with Scala and is targeted to Developers and Testers. This course is recommended for intermediate learners who have completed AppSec Fundamentals and the OWASP Top 10, or equivalent.

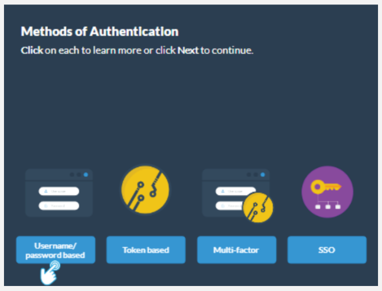
Audience

Scala Developers

Testers

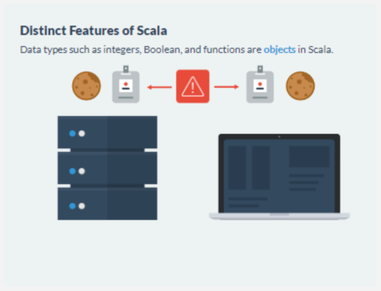
Time Required

Tailored learning - 55 minutes total (approx.)



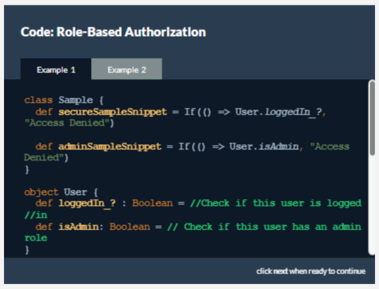
Methods of Authentication

Click on each to learn more or click Next to continue.



Distinct Features of Scala

Data types such as integers, Boolean, and functions are objects in Scala.



Code: Role-Based Authorization

```
class Sample {
  def secureSampleSnippet = If(() => User.loggedIn_?,
"Access Denied")

  def adminSampleSnippet = If(() => User.isAdmin, "Access
Denied")
}

object User {
  def loggedIn_? : Boolean = //Check if this user is logged
//in
  def isAdmin: Boolean = // Check if this user has an admin
role
}
```

SCL201 - DEFENDING SCALA

Course Outline

1. Authentication and Authorization

- Distinct features
- Common attacks
- Web application session in Play
- Defenses: session hijacking
- Code: injection attack
- Example serializer class
- Session fixation
- Cross-Site Scripting
- Cross Site Request Forgery
- Authentication and authorization types
- Methods of authentication
- Authentification implementation
- User authentification
- User and UserDAO
- UUID generation
- Authorization for public and private pages
- Play session management
- Play framework security class
- Apache Shiro library
- Scala Lift framework
- Broken Access Control
- Authorization in Scala Lift framework
- OWASP
- Scala authentication and authorization best practices

2. CSRF and Injection Attacks

- Handling session and server-side data
- Code: payment application
- Code: improved application
- Session timeout
- Code: set session timeout
- Storing unencrypted sessions
- Exposure of sensitive data
- Code: connect to database with hardcoded password
- Code: write user credentials in config file
- Code: sample build.sbt dependency injection
- Scala encryption libraries
- Code: encrypt sensitive data
- Code: known ECB vulnerability
- SQL injection
- Code: user input
- Code: sanitize input
- Session hijacking risks
- Code: directly access session data
- Session hijacking: best practices
- Web filters in Scala
- Scala Play framework defaults
- Code: play.filters
- Code: business logic filters

3. Data Validation and Logging

- User input validation
- Input validation strategies
- Regular expressions
- Code example: Using Scala's built-in Regex class
- Code: Syntactic email validation
- Semantic email validation
- Allow and deny lists
- Code: declare filters
- Code: implement a denylist
- Allow list filter example
- Code: validate user-entered data
- Denial of service attack
- Code: mitigate against server resource consumption
- XML parsing attacks
- Code: XML External Entity attack prevention
- XML parsing attacks
- SQL injection
- Code: SQL query
- Mitigate SQL injection
- Logging
- Code: Syntactic email validation
- Code: Logging framework improperly configured
- Log injection
- Code: Log injection
- Use updated libraries
- Cats or scalaz validated
- Use cases and pattern matching

Security Compass

SCL201 - DEFENDING SCALA

Course Outline

4. File Handling

- Securely implementing File IO
- Code: file Inclusion
- File inclusion vulnerability
- File inclusion attacks: best practices
- API functionalities
- File attacks in Scala
- XML file attacks
- File size handling
- Path traversal attack
- Path traversal attack: best practices
- File handling: best practices
- Using vulnerable components
- Vulnerable components: best practices

5. Cryptography and Key Management

- Numerical computations
- Secure random function
- Code: Random and SecureRandom classes
- Cryptographic functions
- Java Cryptography Architecture
- Code: symmetric encryption
- Asymmetric encryption
- Code: RSA encryption
- Secrets and certificate management
- Code: TypeSafe config library
- Code: development and production environments
- Secrets management: best practices
- Certificate management: best practices
- Code: cipher streams

SecurityCompass