OWASP Top 10 Security Guide for Java Spring Applications

Important Note: While the user requested OWASP Top 10 2024, after researching official OWASP sources, the current authoritative version is **OWASP Top 10 2021**, with the next update expected to be OWASP Top 10 2025 (releasing late summer/early fall 2025). OWASP

Executive Summary

This comprehensive reference guide provides production-ready security implementations for Java Spring applications, covering all OWASP Top 10 2021 vulnerabilities. Each section includes vulnerable anti-patterns, secure implementations, Spring-specific features, configuration examples, testing approaches, and dependency management strategies using Spring Boot 3.x, Spring Security 6.x, and the broader Spring ecosystem.

1. A01:2021 - Broken Access Control

Vulnerability Overview

Ranking: #1 (moved up from #5 in 2017) OWASP OWASP Foundation
Impact: Most common vulnerability affecting 94% of tested applications OWASP Foundation +3

Access control failures lead to unauthorized information disclosure, modification, or destruction of data. GeeksforGeeks owasp

🔴 Vulnerable Anti-Pattern

java

```
```

```java
// DON'T DO THIS - Missing access control
@RestController
public class VulnerableController {
  @GetMapping("/users/{id}")
  public ResponseEntity<User> getUser(@PathVariable Long id) {
    // No access control - any authenticated user can access any user's data
    User user = userService.findById(id);
    return ResponseEntity.ok(user);
  }
}
```

✅ Secure Implementation

java

```java
// SECURE - Proper access control implementation
@RestController
@RequestMapping("/api/v1")
public class SecureUserController {

  @GetMapping("/users/{id}")
  @PreAuthorize("hasRole('ADMIN') or #id == authentication.principal.id")
  public ResponseEntity<User> getUser(@PathVariable Long id) {
    User user = userService.findById(id);
    return ResponseEntity.ok(user);
  }

  @DeleteMapping("/admin/users/{id}")
  @PreAuthorize("hasRole('ADMIN')")
  @PostAuthorize("@userService.canDelete(returnObject, authentication)")
  public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
    userService.deleteUser(id);
    return ResponseEntity.noContent().build();
  }
}
```

Spring Security Configuration

```
java
```

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class AccessControlSecurityConfig {

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
      .authorizeHttpRequests(auth -> auth
        .requestMatchers("/public/**").permitAll()
        .requestMatchers("/api/v1/admin/**").hasRole("ADMIN")
        .requestMatchers("/api/v1/user/**").hasRole("USER")
        .anyRequest().authenticated()
      )
      .sessionManagement(session -> session
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
        .maximumSessions(1)
        .maxSessionsPreventsLogin(false)
      );
    return http.build();
  }
}
```

Testing Access Control

```
java
```

```
@SpringBootTest
@AutoConfigureMockMvc
class AccessControlTest {
  @Autowired
  private MockMvc mockMvc;

  @Test
  @WithMockUser(username = "user1", roles = "USER")
  void testUserCannotAccessOthersData() throws Exception {
    mockMvc.perform(get("/api/v1/users/2"))
        .andExpect(status().isForbidden());
  }
}
```

2. A02:2021 - Cryptographic Failures

Vulnerability Overview

Ranking: #2 - Root causes of cryptographic failures leading to sensitive data exposure OWASP owasp

🔴 Vulnerable Patterns

```java
```

```java
// DON'T DO THIS - Weak encryption
@Service
public class VulnerableCryptoService {
  public String hashPassword(String password) {
    return DigestUtils.md5Hex(password); // VULNERABLE - MD5 is broken
  }

  private static final String SECRET_KEY = "MySecretKey123"; // VULNERABLE - hardcoded
}
```

✅ Secure Implementation

```
java
```

```java
// SECURE - Strong cryptographic practices
@Service
public class SecureCryptoService {

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12); // Strong cost factor
  }

  @Component
  public class DataEncryption {
    private static final String ALGORITHM = "AES/GCM/NoPadding";
    private final SecretKeySpec secretKey;

    public DataEncryption(@Value("${app.encryption.key}") String key) {
      this.secretKey = new SecretKeySpec(
        Base64.getDecoder().decode(key), "AES"
      );
    }

    public String encrypt(String data) throws Exception {
      Cipher cipher = Cipher.getInstance(ALGORITHM);
      cipher.init(Cipher.ENCRYPT_MODE, secretKey);

      byte[] iv = cipher.getIV();
      byte[] encrypted = cipher.doFinal(data.getBytes(StandardCharsets.UTF_8));

      byte[] result = new byte[iv.length + encrypted.length];
      System.arraycopy(iv, 0, result, 0, iv.length);
      System.arraycopy(encrypted, 0, result, iv.length, encrypted.length);

      return Base64.getEncoder().encodeToString(result);
    }
  }
}
```

Application Configuration

```yaml
# application.yml - Secure configuration
server:
 ssl:
  enabled: true
  key-store: classpath:keystore.p12
  key-store-password: ${SSL_KEYSTORE_PASSWORD}
  key-store-type: PKCS12
 servlet:
  session:
   cookie:
    secure: true
    http-only: true
    same-site: strict

app:
 encryption:
  key: ${ENCRYPTION_KEY} # 32-byte Base64 encoded key from environment
```

3. A03:2021 - Injection

Vulnerability Overview

Ranking: #3 - SQL, NoSQL, XSS, OS command, and LDAP injection attacks OWASP owasp

🔴 Vulnerable Patterns

```java
// DON'T DO THIS - SQL Injection vulnerabilities
@Repository
public class VulnerableUserDao {
  public List<User> findByName(String name) {
    String sql = "SELECT * FROM users WHERE name = '" + name + "'"; // VULNERABLE
    return jdbcTemplate.query(sql, new UserRowMapper());
  }
}
```

✅ Secure Implementation

```java
// SECURE - Injection prevention through parameterized queries
@Repository
public interface SecureUserRepository extends JpaRepository<User, Long> {

  // Method queries - automatically parameterized
  List<User> findByNameContaining(String name);

  // JPQL with named parameters - SECURE
  @Query("SELECT u FROM User u WHERE u.email = :email")
  Optional<User> findByEmail(@Param("email") String email);

  // Native query with parameters - SECURE
  @Query(value = "SELECT * FROM users WHERE name = :name", nativeQuery = true)
  List<User> findByNameNative(@Param("name") String name);
}
```

Input Validation

```java
@RestController
@RequestMapping("/api/v1/users")
@Validated
public class SecureUserController {

  @PostMapping
  public ResponseEntity<User> createUser(@Valid @RequestBody UserRequest request) {
    User user = userService.createUser(request);
    return ResponseEntity.status(HttpStatus.CREATED).body(user);
  }
}

// Input validation DTO
public class UserRequest {
  @NotBlank(message = "Username is required")
  @Size(min = 3, max = 20)
  @Pattern(regexp = "^[a-zA-Z0-9_]+$")
  private String username;

  @NotBlank
  @Email
  private String email;

  @Pattern(regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]+$")
  private String password;
}
```

4. A04:2021 - Insecure Design

Vulnerability Overview

New Category: Design-level security flaws that cannot be fixed by implementation alone OWASP owasp

Secure Design Implementation

```java
```

```java
// SECURE DESIGN - Defense in depth architecture
@Service
@Transactional
public class SecureBankingService {

  // Rate limiting for sensitive operations
  @RateLimited(value = 5, window = 60, unit = TimeUnit.SECONDS)
  @PreAuthorize("hasRole('ACCOUNT_HOLDER')")
  public TransferResult transferMoney(MoneyTransferRequest request) {

    // Multi-factor authentication verification
    if (request.getAmount().compareTo(new BigDecimal("10000")) > 0) {
      if (!mfaService.verifyToken(request.getMfaToken())) {
        throw new MfaRequiredException("Large transfer requires MFA");
      }
    }

    // Business logic validation with security implications
    Account fromAccount = accountService.findById(request.getFromAccountId());

    // Authorization check
    if (!accountOwnershipService.canAccess(getCurrentUser(), fromAccount)) {
      throw new UnauthorizedAccountAccessException();
    }

    // Fraud detection
    if (fraudDetectionService.isSuspicious(request)) {
      fraudDetectionService.flagForReview(request);
      throw new SuspiciousActivityException();
    }

    return executeTransfer(fromAccount, toAccount, request.getAmount());
    }
}
```

5. A05:2021 - Security Misconfiguration

Vulnerability Overview

Ranking: #5 - 90% of applications had some form of misconfiguration OWASP owasp

✅ Secure Configuration

```
java
```

```java
@Configuration
@EnableWebSecurity
public class SecureConfig {

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
      .authorizeHttpRequests(auth -> auth
        .requestMatchers("/actuator/health", "/actuator/info").permitAll()
        .requestMatchers("/actuator/**").hasRole("ADMIN")
        .anyRequest().authenticated()
      )
      .sessionManagement(session -> session
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
        .sessionFixation().migrateSession()
        .maximumSessions(1)
      )
      .headers(headers -> headers
        .frameOptions().deny()
        .contentTypeOptions().and()
        .httpStrictTransportSecurity(hsts -> hsts
          .maxAgeInSeconds(31536000)
          .includeSubdomains(true)
        )
        .contentSecurityPolicy("default-src 'self'")
      )
      .csrf(Customizer.withDefaults());
    return http.build();
  }
}
```

Production Configuration

yaml

```yaml
# application-prod.yml
server:
  port: 8443
  ssl:
    enabled: true
  servlet:
    session:
      cookie:
        secure: true
        http-only: true
        same-site: strict
      timeout: 30m
  error:
    include-stacktrace: never
    include-message: never

management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics
  endpoint:
    health:
      show-details: when-authorized
      roles: ADMIN

logging:
  level:
    org.springframework.security: WARN
  file:
    name: /var/log/spring-app/application.log
```

6. A06:2021 - Vulnerable and Outdated Components

Dependency Management Strategy

Maven Security Configuration

xml

```
<project>
  <properties>
    <spring-boot.version>3.2.1</spring-boot.version>
    <spring-security.version>6.2.1</spring-security.version>
  </properties>

  <build>
    <plugins>
      <!-- OWASP Dependency Check -->
      <plugin>
        <groupId>org.owasp</groupId>
        <artifactId>dependency-check-maven</artifactId>
        <version>9.0.7</version>
        <configuration>
          <failBuildOnCVSS>7</failBuildOnCVSS>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>check</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Automated Updates

```
yaml
```

```yaml
# .github/workflows/dependency-updates.yml
name: Dependency Security Updates
on:
 schedule:
  - cron: '0 2 * * 1' # Weekly
jobs:
 security-updates:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4
   - name: OWASP Dependency Check
    run: mvn org.owasp:dependency-check-maven:check
   - name: Update dependencies
    run: mvn versions:use-latest-versions -DallowMajorUpdates=false
```

7. A07:2021 - Identification and Authentication Failures

Comprehensive Authentication System

```
java
```

```
@Configuration
@EnableWebSecurity
public class AuthenticationSecurityConfig {

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12);
  }

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
      .formLogin(form -> form
        .loginPage("/login")
        .successHandler(customAuthenticationSuccessHandler())
        .failureHandler(customAuthenticationFailureHandler())
      )
      .sessionManagement(session -> session
        .sessionFixation().migrateSession()
        .maximumSessions(1)
        .sessionRegistry(sessionRegistry())
      );
    return http.build();
  }
}
```

Login Attempt Protection

```
java
```

```java
@Service
public class LoginAttemptService {
  private final int MAX_ATTEMPT = 5;
  private final int LOCKOUT_DURATION_MINUTES = 15;
  private final Cache<String, Integer> attemptsCache;

  public void loginFailed(String key) {
    Integer attempts = attemptsCache.get(key, k -> 0);
    attempts++;
    attemptsCache.put(key, attempts);

    if (attempts >= MAX_ATTEMPT) {
      auditService.logAccountLockout(key, attempts);
    }
  }

  public boolean isBlocked(String key) {
    return attemptsCache.get(key, k -> 0) >= MAX_ATTEMPT;
  }
}
```

JWT Token Security

```java
```

```java
@Component
public class JwtTokenProvider {

    public TokenResponse generateTokens(Authentication authentication) {
        UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();
        Date expiryDate = new Date(System.currentTimeMillis() + jwtExpirationInMs);

        String accessToken = Jwts.builder()
            .setSubject(Long.toString(userPrincipal.getId()))
            .setIssuedAt(new Date())
            .setExpiration(expiryDate)
            .claim("roles", userPrincipal.getAuthorities())
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();

        return TokenResponse.builder()
            .accessToken(accessToken)
            .tokenType("Bearer")
            .expiresIn(jwtExpirationInMs / 1000)
            .build();
    }

    public boolean validateToken(String authToken) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
            return true;
        } catch (SignatureException | MalformedJwtException | ExpiredJwtException ex) {
            return false;
        }
    }
}
```

8. A08:2021 - Software and Data Integrity Failures

Vulnerability Overview

New Category: Software updates, critical data, and CI/CD pipeline integrity (OWASP) (owasp)

Secure Serialization

```java
// ✅ SECURE - Safe data handling
@RestController
public class SecureDataController {

    @Autowired
    private ObjectMapper objectMapper;

    @PostMapping("/data")
    public ResponseEntity<DataResponse> processData(@Valid @RequestBody DataRequest request) {
      // Use JSON serialization instead of Java serialization
      try {
        String json = objectMapper.writeValueAsString(request);
        DataResponse response = processBusinessLogic(request);
        return ResponseEntity.ok(response);
      } catch (JsonProcessingException e) {
        throw new InvalidDataFormatException("Invalid data format", e);
      }
    }
}
```

CI/CD Pipeline Integrity

```yaml
# .github/workflows/secure-build.yml
name: Secure Build and Deploy
jobs:
 security-checks:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4
   - name: OWASP Dependency Check
    run: mvn org.owasp:dependency-check-maven:check
   - name: Static Analysis
    run: mvn spotbugs:check
   - name: Sign artifacts
    run: gpg --batch --yes --detach-sign target/*.jar
```

9. A09:2021 - Security Logging and Monitoring Failures

Comprehensive Security Logging

```java
```

```
@Component
public class SecurityEventListener {
  private static final Logger securityLogger = LoggerFactory.getLogger("SECURITY");

  @EventListener
  public void handleAuthenticationSuccess(AuthenticationSuccessEvent event) {
    String username = event.getAuthentication().getName();
    String clientIp = getClientIpAddress();

    securityLogger.info("Authentication success: user={}, ip={}, timestamp={}",
      username, clientIp, Instant.now());
  }

  @EventListener
  public void handleAuthenticationFailure(AbstractAuthenticationFailureEvent event) {
    String username = event.getAuthentication().getName();
    String reason = event.getException().getMessage();

    securityLogger.warn("Authentication failed: user={}, reason={}", username, reason);

    if (isSuspiciousActivity(username, getClientIpAddress())) {
      alertService.sendSecurityAlert("Suspicious login activity detected");
    }
  }
}
```

Structured Logging Configuration

```
xml
```

```
<!-- logback-spring.xml -->
<configuration>
  <appender name="SECURITY_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/var/log/app/security.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>/var/log/app/security.%d{yyyy-MM-dd}.gz</fileNamePattern>
      <maxHistory>90</maxHistory>
    </rollingPolicy>
    <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
      <providers>
        <timestamp/>
        <logLevel/>
        <message/>
        <mdc/>
      </providers>
    </encoder>
  </appender>

  <logger name="SECURITY" level="INFO">
    <appender-ref ref="SECURITY_FILE"/>
  </logger>
</configuration>
```

10. A10:2021 - Server-Side Request Forgery (SSRF)

Vulnerability Overview

New Addition: Enables attackers to make requests to internal systems (OWASP) (owasp)

🔴 Vulnerable Pattern

```
java
```

```java
// DON'T DO THIS - SSRF vulnerability
@RestController
public class VulnerableWebhookController {
  @PostMapping("/webhook")
  public ResponseEntity<String> processWebhook(@RequestParam String url) throws Exception {
    // DANGEROUS - No URL validation
    URL target = new URL(url);
    HttpURLConnection connection = (HttpURLConnection) target.openConnection();
    return ResponseEntity.ok(IOUtils.toString(connection.getInputStream()));
  }
}
```

✅ Secure Implementation

java

```java
// SECURE - SSRF prevention with URL validation
@RestController
public class SecureWebhookController {

  @Autowired
  private UrlValidator urlValidator;

  @PostMapping("/webhook")
  public ResponseEntity<String> processWebhook(@RequestParam String url) {
    // Validate URL before processing
    if (!urlValidator.isUrlSafe(url)) {
      throw new InvalidUrlException("URL not allowed: " + url);
    }

    String response = secureHttpClient.fetchContent(url);
    return ResponseEntity.ok(response);
  }
}
```

URL Validation Service

```
java
```

```java
@Service
public class UrlValidator {
  private static final Set<String> BLOCKED_HOSTS = Set.of(
    "localhost", "127.0.0.1", "0.0.0.0", "::1",
    "169.254.169.254", // AWS metadata service
    "metadata.google.internal" // Google Cloud metadata
  );

  public void validateUrl(String urlString) {
    try {
      URL url = new URL(urlString);

      // Check scheme
      if (!Set.of("http", "https").contains(url.getProtocol())) {
        throw new InvalidUrlException("Unsupported URL scheme");
      }

      // Check host
      String host = url.getHost().toLowerCase();
      if (BLOCKED_HOSTS.contains(host) || isPrivateIpAddress(host)) {
        throw new InvalidUrlException("Blocked host: " + host);
      }

    } catch (MalformedURLException e) {
      throw new InvalidUrlException("Malformed URL", e);
    }
  }

  private boolean isPrivateIpAddress(String host) {
    try {
      InetAddress addr = InetAddress.getByName(host);
      return addr.isSiteLocalAddress() || addr.isLoopbackAddress();
```

```
        } catch (UnknownHostException e) {
            return false;
        }
    }
}
```

Security Testing Framework

Integration Test Suite

```
java
```

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class OwaspTop10IntegrationTest {

  @Autowired
  private TestRestTemplate restTemplate;

  @Test
  void testA01_BrokenAccessControl() {
    String userToken = loginAndGetToken("user@example.com", "password");

    ResponseEntity<String> response = restTemplate.exchange(
      "/api/v1/admin/users",
      HttpMethod.GET,
      createEntityWithToken(userToken),
      String.class
    );

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.FORBIDDEN);
  }

  @Test
  void testA03_Injection() {
    String sqlInjectionPayload = "admin'; DROP TABLE users; --";

    ResponseEntity<String> response = restTemplate.getForEntity(
      "/api/v1/users?name=" + sqlInjectionPayload,
      String.class
    );

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
  }
  @Test
  void testA10_ServerSideRequestForgery() {
    String maliciousUrl = "http://127.0.0.1:8080/actuator/env";

    ResponseEntity<String> response = restTemplate.postForEntity(
      "/api/v1/webhook?url=" + maliciousUrl,
      null,
      String.class
    );

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
  }
}
```

Production Deployment Checklist

- [ ] SSL/TLS properly configured with strong ciphers
- [ ] All management endpoints secured or disabled
- [ ] Database connections encrypted and credentials managed securely
- [ ] Security headers properly configured
- [ ] Rate limiting implemented on sensitive endpoints
- [ ] Comprehensive security logging enabled
- [ ] OWASP dependency scan integrated into build process Medium
- [ ] Security tests passing in CI/CD pipeline
- [ ] Secrets management system in place
- [ ] Monitoring and alerting configured for security events

Key Implementation Guidelines

1. **Defense in Depth**: Implement multiple security layers

2. **Secure by Default**: Use Spring Security's secure defaults Form.io
3. **Input Validation**: Apply comprehensive validation at all boundaries Snyk Stackademic
4. **Least Privilege**: Grant minimum necessary permissions
5. **Fail Securely**: Ensure failures don't expose sensitive information
6. **Security Testing**: Integrate security tests into CI/CD pipelines spring
7. **Monitoring and Logging**: Implement comprehensive security event logging
8. **Regular Updates**: Maintain current versions of all dependencies
9. **Configuration Management**: Externalize and secure all configuration
10. **Incident Response**: Prepare for security incidents with proper logging

This comprehensive guide provides production-ready security implementations for Java Spring applications covering all OWASP Top 10 2021 vulnerabilities with extensive code examples, configuration templates, and testing approaches designed for enterprise applications. Medium