snyk Spring Boot Security Cheat Sheet

www.snyk.io

1. Use HTTPS in Production

To use HTTPS in your Spring Boot app, extend `WebSecurityConfigurerAdapter` and require a secure connection (Note: this forces HTTPS in development also):

```
@Configuration
public class WebSecurityConfig extends
                      WebSecurityConfigurerAdapter {
 @Override
 protected void configure(HttpSecurity http)
                                 throws Exception {
    http.requiresChannel().requiresSecure();
  }
}
```

2. Test Your Dependencies

Ensure your application does not use dependencies with known vulnerabilities. Use a tool like Snyk to:

- Test your app dependencies for known vulnerabilities.
- Automatically Fix issues that exist.
- Continuously Monitor for new vulns.

3. Enable CSRF Protection

Spring Security enables CSRF support by default. If you use a JavaScript framework, configure the `CookieCsrfTokenRepository` so cookies are not HTTP-only.

```
@EnableWebSecurity
public class WebSecurityConfig extends
                    WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http)
                                 throws Exception {
  http.csrf()
      .csrfTokenRepository(CookieCsrfTokenRepository
                          .withHttpOnlyFalse());
  }
}
```

4. Use a Content Security Policy

Enable to avoid XSS attacks.

Spring Security provides a number of security headers by default, but not CSP. Enable in your Spring Boot app as follows:

```
@EnableWebSecurity
public class WebSecurityConfig extends
                        WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http)
                                      throws Exception {
  http.headers().contentSecurityPolicy("script-src
'self' https://trustedscripts.example.com; object-src
https://trustedplugins.example.com; report-uri /csp-
report-endpoint/");
  }
}
```

5. Use OpenID Connect

OpenID Connect (OIDC) provides user information via an ID token in addition to an access token.
Query the `/userinfo endpoint` for additional user information.

6. Use Password Hashing

Don't store passwords in plain text. Spring Security doesn't allow plain text passwords by default.

`PasswordEncoder` is the main interface for password hashing in Spring Security:

```
public interface PasswordEncoder {
  String encode(String rawPasswd);
  boolean matches(String rawPasswd, String encPasswd);
}
```

7. Use the Latest Releases

The start.spring.io site automatically uses the latest versions of libraries for new apps.

For existing apps, when upgrades aren't possible, consider patches from a security vendor, like Snyk.

8. Store Secrets Securely

Store secrets in Vault by HashiCorp or Spring Vault

Extract secrets from the Spring Vault using annotations.

```
@Value("${password}")
String password;
```

9. Pen Test Your App

The OWASP ZAP security tool is a proxy that performs penetration testing. It runs in Spider and Active Scan modes to identify and map all hyperlinks in your app, and automatically test your selected targets against a list of potential vulnerabilities.

10. Have Your Security Team do a Code Review

Code reviews are essential. Ensure all your code changes undergo:

- a security team code review.
- Automatic testing on every pull request using Snyk

Authors:


@sjmaple
Java Champion and Developer Advocate at Snyk


@mraible
Java Champion and Developer Advocate at Okta