Spring Security

What is Spring Security?

What is Spring Security?

- A powerful and highly customizable authentication and access-control framework
- Build on top of Spring Framework
- De-facto standard for securing Spring-based applications

Spring Security Setup

Add Dependencies

- Adding Spring Security Starter to your project and getting dependencies for Spring Security.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Setup

- Starter configures infrastructre Spring beans automatically
  - SpringSecurityFilterChain
  - Global method security
  - BasicAuth
    - For initial password see console

```
2017-11-16 00:23:28.130  INFO 16039 --- [  restartedMain] b.a.s.Authenticat

Using default security password: a5e53135-6899-4953-9b78-7ef597fa37cb

2017-11-16 00:23:28.136  INFO 16039 --- [  restartedMain] o.s.s.web.Default
```

    - Or set user and password explicit

```
security.basic.enabled=true
security.user.name=user
security.user.password=secret
```

DEPRECATED

Fundamentals

Fundamentals (1)

- Principal
  - User or system that performs the action
- Authentication
  - Confirming truth of credentials
- Authorization
  - Define access policy for principal
- Authority
  - Role or permission to be checked
- Secured Item
  - Resource with URL or a method

Fundamentals (2)

- Authentication
  - Principal in a Spring Security-specific manner
- GrantedAuthority
  - Application-wide permissions granted to a principal
- SecurityContext
  - Holds the Authentication and other security information
- SecurityContextHolder
  - Provides access to SecurityContext

Web Security

Custom Security Configuration

- Override default security configuration by Spring Boot

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .mvcMatchers("/","/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    } …
```

More on authorizeRequests()

- Chain multiple restrictions
- Evaluated in listed order
- First match is used, specific matches should be first

```
protected void configure(HttpSecurity http)throws Exception{
   http
   .authorizeRequests()
       .mvcMatchers("/signup","/about").permitAll()
       .mvcMatchers("/employees/edit*").hasRole("ADMIN")
       .mvcMatchers("/employees/**").hasAnyRole("USER",”ADMIN”)
       .anyRequest().authenticated();
}
```

Skipping Security

- Static resources do not need to be secured
- Using permitAll() still processed through Spring Security
- Skipping security checks completely

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter{

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().mvcMatchers("/css/**","/images/**",
            "/javascript/**");
    }
…
}
```

Login page

- POST username and password to predefined URL
- ViewController serves view /template/login.html

```
<!DOCTYPEhtml>
<htmlxmlns=http://www.w3.org/1999/xhtml
    xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Login</title>
</head>
<body>
<div th:if="${param.error}">Invalid username and password.</div>
<div th:if="${param.logout}">You have been logged out.</div>
<form th:action="@{/login}" method="post">
    <label>User Name :<input type="text" name="username"/></label>
    <label>Password:<input type="password" name="password"/></label>
    <input type="submit" value="Sign In"/>
</form>
</body>
</html>
```

Loading users with roles

- InMemoryUserDetailsManager used for development
- Use LdapUserDetailsManager or JdbcUserDetailsManager
- For production apps use PasswordEncoder

```
…

    @Bean
    public InMemoryUserDetailsManager InMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withDefaultPasswordEncoder()
            .username("user")
            .password("password")
            .roles("USER").build());
    }
}
```

Method Security

Enable Method Security

- Creating security proxies for @Secured and @PreAuthorized on a @Configuration class

```
@EnableGlobalMethodSecurity(securedEnabled=true,
    prePostEnabled=true)
```

- Typically securing services, can be declared on any Spring bean

Securing a @Service

```
@Service
public class EmployeeService{

    @Secured("ROLE_USER")
    public String secure() {
        return "Hello Security";
    }

    @PreAuthorize("hasRole('ADMIN')")
    public String authorized() {
        return"Hello World";
    }

    @PreAuthorize("false")
    public String denied() {
        return"Goodbye World";
    }
}
```

Securing Spring Data REST

Securing Spring Data REST

- Using global method security on an interface ...

```
@PreAuthorize("hasRole('USER')")
public interface EmployeeRepository extends
    CrudRepository<Employee, Long>{


    @PreAuthorize("hasRole('ADMIN')")
    @Override
    Employee save(Employee s);


    @PreAuthorize("hasRole('ADMIN')")
    @Override
    void delete(Long id);
}
```

Important notes

Important notes

- Portable across containers
- Can be deployed as-is
- Runs in standalone environment
- Business logic is decoupled from security concern
- Authentication and authorization are decoupled
- Supports different authentication mechanisms e.g. basic, form, cookies, SSO
- Supports different user data storage e.g. DB, LDAP, In-Memory, etc.