什么是水平越权和垂直越权?

水平越权和垂直越权是计算机安全领域中关于权限控制的两个术语，尤其在讨论访问控制和权限漏洞时经常提到。这两个概念通常用于描述不当的访问尝试或权限提升，这可能导致未授权用户访问或执行他们本不应有权限的操作。

1. **水平越权（Horizontal Privilege Escalation）**：这种情况发生时，攻击者或用户能够访问与他们同一权限级别的其他用户的数据或功能，但这些数据或功能并不属于他们。简而言之，水平越权指的是用户在不改变自己权限级别的情况下，访问或控制同一权限级别内其他用户的数据或功能。例如，一个网站用户改变浏览器中的URL或提交的表单数据，从而能够访问或修改另一个同等权限用户的账户信息。
2. **垂直越权（Vertical Privilege Escalation）**：在这种情况下，攻击者提升自己的权限级别，获得更高级别用户的权限，例如从普通用户变为管理员。垂直越权涉及攻击者从较低的权限级别提升到较高的权限级别，获取不应有的访问权限或执行高级功能。例如，一个普通用户通过某种漏洞获得管理员权限，从而能够执行本应只有管理员能执行的操作。

简单原理介绍

水平越权的原理：

* **同级访问控制缺失**：当应用程序未能充分检查用户对特定数据的访问权限时，就可能发生水平越权。这种情况下，尽管用户处于相同的权限级别，但他们可以访问或修改不属于自己的数据。
* **用户识别和数据隔离失败**：在没有有效区分用户身份和数据所有权的情况下，用户可能会访问到其他用户的信息。这通常是因为应用程序未能正确实施用户之间的数据隔离。
* **操作的非个性化处理**：当应用程序处理用户请求时，如果没有根据用户的具体身份来个性化处理，就可能允许用户执行不应该执行的操作。

垂直越权的原理：

* **权限检查缺陷**：垂直越权发生时，通常是因为应用程序在执行敏感操作或访问敏感数据之前，未能适当地验证请求者的权限等级。
* **提权漏洞**：某些情况下，攻击者可能会利用应用程序中的漏洞提升自己的权限级别。例如，通过注入攻击、会话劫持或错误的权限配置，攻击者可能

会绕过正常的权限检查流程。
* **身份验证和授权机制不严格**：如果身份验证和授权机制不够严格或易于绕过，攻击者可能会找到方法提升自己的权限级别。

代码示例

要在Spring Boot中使用Spring Authorization Server来防止水平和垂直越权，我们需要构建一个基本的授权服务器，并设置一些资源来模拟用户操作。以下是如何一步步构建这个示例的详细过程。

1. 设置Spring Authorization Server

首先，我们需要添加Spring Authorization Server的依赖项。在你的`pom.xml`文件中添加：

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-authorization-server</artifactId>
    <version>{version}</version> <!-- 使用当前的版本号 -->
</dependency>
```

2. 配置授权服务器

创建一个配置类来设置授权服务器：

```
@Configuration
public class AuthorizationServerConfig extends
AuthorizationServerConfigurerAdapter {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(AuthorizationServerSecurityConfigurer security)
```

```
throws Exception {
        security.tokenKeyAccess("permitAll()")
                .checkTokenAccess("isAuthenticated()");
    }

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws
Exception {
        clients.inMemory()
                .withClient("client-id")
                .secret("client-secret")
                .authorizedGrantTypes("password", "authorization_code",
"refresh_token")
                .scopes("read", "write");
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer
endpoints) throws Exception {
        endpoints.authenticationManager(authenticationManager);
    }
}
```

3. 设置用户和角色

假设我们有两种用户角色：`USER`和`ADMIN`。我们需要在用户详细信息服务中配置这些角色：

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        // 这里应从数据库加载用户信息，为了示例简化，我们直接创建
        if (username.equals("admin")) {
            return new User("admin", "password",
AuthorityUtils.createAuthorityList("ROLE_ADMIN"));
        } else if (username.equals("user")) {
            return new User("user", "password",
AuthorityUtils.createAuthorityList("ROLE_USER"));
        }
        throw new UsernameNotFoundException("User not found");
```

```
    }
}
```

4. 创建资源服务

现在，我们需要创建一些受保护的资源来模拟垂直和水平越权：

```
@RestController
@RequestMapping("/api")
public class TestController {

    // 垂直越权示例：只有ADMIN角色可以访问
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/admin")
    public String admin() {
        return "Admin data";
    }

    // 水平越权示例：用户只能访问自己的数据
    @GetMapping("/user/{userId}")
    public String userData(@PathVariable String userId, Authentication
authentication) {
        if (!authentication.getName().equals(userId)) {
            throw new AccessDeniedException("Not allowed to access
another user's data");
        }
        return "User data for " + userId;
    }
}
```

在这个例子中：

* 对于垂直越权，我们通过`@PreAuthorize("hasRole('ADMIN')")`确保只有管理员可以访问某些数据。
* 对于水平越权，我们比较了请求的用户ID和认证的用户名，以确保用户只能访问自己的数据。

5. 测试和验证

启动应用程序后，你可以尝试使用不同角色的用户访问这些端点，并验证安全控制是否按预期工作。例如，尝试用普通用户访问`/api/admin`应该被拒绝，而尝试用用户A的凭证访问用户B的数据也应该被拒绝。