**Access Control: Database Development Mitigation SOP**

Access Control vulnerabilities occur when access to a resource is granted to an attacker. This can include access to the database, restricted packages or executing arbitrary code. Vulnerabilities can be exploited through a database when the execution of a SQL statement containing a user-controlled primary key allows an attacker to view unauthorized records. The system is vulnerable to this breach in security when data enters the system from an untrusted source or data is used to specify the value of a primary key in a SQL query.

**Defense Against Access Control: Database**

Per IA Guidance, these defects can be considered invalid if authorization is present on the presentation later (end point), therefore providing access control to the database. To determine if the defect is valid, locate the enclosing function listed in the defect and find all usages. Follow the usage chain until you find an @Authorize, ProjecPep.isAuthorized(...) or ProjecPep.enforce(...). If you reach a top-level controller method (is public and has @RequestMapping) and none of the authorization checks are present on the controller method this is a valid defect. Resolve the defect by adding the appropriate authorization to the controller method.

In some instances, the enclosing function never bubbles up to a controller. Some of these instances are simply that the method is never used. In other instances, the calling hierarchy endpoint is not part of the presentation layer (e.g. batch services). Currently specific guidance has not been provided to handle these cases. The general assumption is that these endpoints are used for machine-to-machine communications and therefore not vulnerable.

In order for the defect to be considered invalid by IA, provide a call trace from the enclosing function to each endpoint and justify why each individual endpoint is secure.

Additionally, all access should be denied by fault and require explicit grants to specific roles for access to each function. Entitlements and permissions should not be hardcoded.

**Example**

@ResponseBody

@RequestMapping(value = “/saveRoCapabilities”, method = RequestMethod.POST)

public RegionalOfficeCapacity saveRoCapabilities (…){

…

}

@Override

public BucketEntity findById(final long id){

Query namedQuery = em.createNamedQuery(FIND\_BY\_BUCKET\_ID);

namedQuery.setParameter(“id”, id);

}

**Explanation**

When findById is called, the long, id, comes in from the saveRoCapabilities method above. The saveRoCapabilities method lacks the @Authorize tag above it. To modify the saveRoCapabilities method for proper access control, it is recommended to add the @Authorize tag.

**Recommendation**

@Authorize(policy = ProjPolicy.EDIT\_CAPACITY)

@ResponseBody

@RequestMapping(value = “/saveRoCapabilities”, method = RequestMethod.POST)

public RegionalOfficeCapacity saveRoCapabilities (…){

…

}

**@Core Component Custom Access Controls**

The above information is relative to all components. However, Core has “developer-specific” or “custom-coded” access control annotations. @Authorize and the method projpep.enforce() are core specific authorizations. The other annotations that can provide access control are spring annotations, @PreAuthorize, @PreFilter, @PostAuthorize and @PostFilter. Please remember that Core has “custom” or “developer-specific” annotations.

**References**

1. [HP Enterprise Security – Access Control: Database](https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.access_control_database#Java%2fJSP)
2. IA Defect SOP on Dev Wiki
3. [https://docs.spring.io/spring-security/site/docs/3.2.0.CI-SNAPSHOT/reference/html/el-access.html](https://slack-redir.net/link?url=https%3A%2F%2Fdocs.spring.io%2Fspring-security%2Fsite%2Fdocs%2F3.2.0.CI-SNAPSHOT%2Freference%2Fhtml%2Fel-access.html)