**Code Correctness: Hidden Method Development Mitigation SOP**

Code correctness vulnerabilities occur when an Object API is not used properly or as intended.

Code correctness vulnerabilities can occur when static methods are called as an instance because they may appear as hidden. Static methods cannot be overridden because they belong to the class rather than an instance of the class.

**Defense Against Code Correctness: Hidden Method**

Avoid giving static methods a signature similar to those in superclasses to avoid confusion. It is best practice to refactor the code to make it clear by using separate final classes, which cannot be subclassed, or by making the constructor private to prevent the static methods being called against an instance of a class instead of the class itself.

**Examples**

**General Example**

class AccessLevel {

public static final int ROOT = 0;

public static final int NONE = 9;

}

class User {

private static int access;

public User() {

access = AccessLevel.ROOT;

}

public static int getAccessLevel() {

return access;

}

…

}

class RegularUser extends User {

private static int access;

public RegularUser() {

access = AccessLevel.NONE;

}

public static int getAccessLevel() {

return access;

}

public static void escalatePrivilege() {

access = AccessLevel.ROOT;

}

…

}

class SecureArea {

public static void doRestrictedOperation(User user) {

if(user instanceof RegularUser ) {

if(**user.getAccessLevel()** == AccessLevel.ROOT) {

System.out.println(“Doing a privileged

operation.”);

} else {

throw new RuntimeException();

}

}

}

}

**Explanation**

The code above tries to define an API for authenticating users. When calling the method getAccessLevel() against the instance user and not against the classes User or RegularUser, it will mean that this condition will always return true, and the restricted operation will be performed even though instanceof was used in order to get into this part of the if/else block.

**Recommendation**

The code below refactors the code to make permissions clearer and uses the principle of least privilege to make sure it fails securely:

class AccessLevel {

public static final int ROOT = 0;

public static final int NONE = 9;

}

class User {

protected int access;

public User() {

access = AccessLevel.NONE;

…

}

public int getAccessLevel() {

return access;

}

…

}

final class RegularUser extends User { … }

final class AdminUser extends User {

public AdminUser() {

access = AccessLevel.ROOT;

}

}

class SecureArea {

…

public void doRestrictedOperation(User user) {

if(user.getAccessLevel() == AccessLevel.ROOT) {

//do operation

…

} else {

throw new PermissionNotAllowedException();

}

}

}

**Example**

**Resources**

1. [HP Enterprise Security – Code Correctness: Hidden Method](https://vulncat.fortify.com/en/detail?id=desc.structural.java.code_correctness_hidden_method#Java%2fJSP)