**Redundant Null Check Development Mitigation SOP**

Redundant null check vulnerabilities occur when a check-after-deference error occurs because the program dereferences an object that could be null before doing a null check. If an attacker is able to intentionally trigger a null pointer deference, they may be able to use the resulting exception to bypass security logic or cause the application to reveal debugging information that will be valuable for other potential attacks.

**Defense Against Redundant Null Check**

Checks should be made before dereferencing objects that may be null. It is best practice to implement abstract null wrappers around code that manipulate resources to ensure that they are applied in all cases and to decrease the chances of mistakes.

**Example**

if( foo == null ) {

foo.setBar( val );

…

}

Explanation:

1. In the code above, the programmer confirms the variable foo is null because setting the value in the if statement, causing a null deference to occur
2. This code will cause a null pointer exception, making the check for null redundant

**References**

1. [HP Enterprise Security – Redundant Null Check](http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/null_dereference_check_after_dereference.html)