### **BIT: Check to see if ((...) & 0) == 0 (BIT\_AND\_ZZ)**

### This method compares an expression of the form (e & 0) to 0, which will always compare equal. This may indicate a logic error or typo.

1. **& <-- verifies both operands  
   && <-- stops evaluating if the first operand evaluates to false since the result will be false**

(x != 0) & (1/x > 1) <-- this means evaluate (x != 0) then evaluate (1/x > 1) then do the &. the problem is that for x=0 this will throw an exception.

(x != 0) && (1/x > 1) <-- this means evaluate (x != 0) and only if this is true then evaluate (1/x > 1) so if you have x=0 then this is perfectly safe and won't throw any exception if (x != 0) evaluates to false the whole thing directly evaluates to false without evaluating the (1/x > 1).

1. **BSHIFT: Possible bad parsing of shift operation (BSHIFT\_WRONG\_ADD\_PRIORITY)**
   1. The code performs an operation like (x << 8 + y). Although this might be correct, probably it was meant to perform (x << 8) + y, but shift operation has a lower precedence, so it's actually parsed as x << (8 + y).

### **SA: Double assignment of local variable (SA\_LOCAL\_DOUBLE\_ASSIGNMENT)**

This method contains a double assignment of a local variable; e.g.

public void foo() {

int x,y;

x = x = 17;

} Assigning the same value to a variable twice is useless, and may indicate a logic error or typo.

### **SA: Self assignment of local variable (SA\_LOCAL\_SELF\_ASSIGNMENT)**

This method contains a self assignment of a local variable; e.g.

public void foo() {

int x = 3;

x = x;

} Such assignments are useless, and may indicate a logic error or typo.