**Code Correctness: Constructor Invokes Overridable Function**

A constructor of the class calls a function that can be overridden.

**Explanation**

When a constructor calls an overridable function, it may allow an attacker to access the this reference prior to the object being fully initialized, which can in turn lead to a vulnerability.

**Example**

The following calls a method that can be overridden.

class User {

private String username;

private boolean valid;

public User(String username, String password){

this.username = username;

this.valid = validateUser(username, password);

}

public boolean validateUser(String username, String password){

//validate user is real and can authenticate

...

}

public final boolean isValid(){

return valid;

}

}

Since the function validateUser and the class are not final, it means that they can be overridden, and then initializing a variable to the subclass that overrides this function would allow bypassing of the validateUser functionality. For example:

class Attacker extends User{

public Attacker(String username, String password){

super(username, password);

}

public boolean validateUser(String username, String password){

return true;

}

}

...

class MainClass{

public static void main(String[] args){

User hacker = new Attacker("Evil", "Hacker");

if (hacker.isValid()){

System.out.println("Attack successful!");

}else{

System.out.println("Attack failed");

}

}

}

**Recommendations**

Constructors should not call functions that can be overridden, either by specifying them as final, or specifying the class as final. Alternatively if this code is only ever needed in the constructor, the private access specifier can be used, or the logic could be placed directly into the constructor of the superclass.

**Example**

The following makes the class final to prevent the function from being overridden elsewhere.

final class User {

private String username;

private boolean valid;

public User(String username, String password){

this.username = username;

this.valid = validateUser(username, password);

}

private boolean validateUser(String username, String password){

//validate user is real and can authenticate

...

}

public final boolean isValid(){

return valid;

}

}

This example specifies the class as final, so that it cannot be subclassed, and changes the validateUser() function to private, since it is not needed elsewhere in this application. This is programming defensively, since at a later date it may be decided that the User class needs to be subclassed, which would result in this vulnerability reappearing if the validateUser() function was not set to private.

**References**

* http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/code\_correctness\_constructor\_invokes\_overridable\_function.html