# CWE Detail – CWE-111

## Description

When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java.

## Extended Description

Many safety features that programmers may take for granted do not apply for native code, so you must carefully review all such code for potential problems. The languages used to implement native code may be more susceptible to buffer overflows and other attacks. Native code is unprotected by the security features enforced by the runtime environment, such as strong typing and array bounds checking.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Bypass Protection Mechanism — Notes:

## Potential Mitigations

**•** Implementation: Implement error handling around the JNI call. (Effectiveness: N/A)

**•** Implementation: Do not use JNI calls if you don't trust the native library. (Effectiveness: N/A)

**•** Implementation: Be reluctant to use JNI calls. A Java API equivalent may exist. (Effectiveness: N/A)

## Applicable Platforms

**•** Java (Class: None, Prevalence: Undetermined)

## Demonstrative Examples

**•** Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of gets(), which does not check the length of its input.