**Define: What is a security vulnerability?**

Security vulnerability refers to a weakness, flaw, or error that can be exploited by threats to compromise assets. With the increasing reliance on multiple technologies to develop a single application or system aiming to enhance user experience and system performance, there are growing concerns about security. Each programming language, framework, library, or package utilized can introduce risks such as breaches, denial of service, unauthorized access, or exploitation. These vulnerabilities often arise due to issues in hardware, software, networks, or procedural.

**Identify: What kinds of vulnerabilities would be identifiable in C++ code?**

Numerous vulnerabilities in C++, C, and Objective-C have been recognized and mitigated by security experts and the computing community due to their significant impact on users, the economy, and information security. Some of the most notable vulnerabilities include:

**Buffer overflow:** This occurs when data written to a buffer surpasses its allocated size, causing adjacent memory to be overwritten. This can result in system crashes or the execution of arbitrary code, causing serious security risks.

**Integer overflow:** This happens when the result of an arithmetic operation exceeds the maximum value that can be stored in the allocated memory space, leading to unexpected behavior and incorrect results.

**Dangling pointers:** These arise when a pointer references memory that has already been deallocated or freed, causing undefined behavior such as crashes, data corruption, or security vulnerabilities.

**Memory leaks:** These occur when allocated memory is not properly deallocated, gradually exhausting available resources and potentially causing application crashes or denial of service.

**Sql Injection:** A common vulnerability in software systems where malicious SQL scripts are injected through user inputs, enabling unauthorized access to databases and compromising sensitive information.

**Race Conditions:** These occur when multiple threads attempt to access the same resource simultaneously without proper concurrency controls, potentially leading to security breaches or inconsistent system states.

**Purpose: Why would you be looking for vulnerabilities during legacy to C++ conversion rather than during testing?**

There are multiple purposes looking for vulnerabilities during legacy to C++ conversion rather than during testing. Some of the most important purposes are:

**Early detection and mitigations:** allows programmers to address security issues in the code, reducing the risk of ship vulnerabilities to new system.

**Compliance and Standards:** Many industries like banking and healthcare have rigorous compliance requirements, working on these requirements during code conversion early will reduce the risk of re-writing the program.

**Code Modernization:** During the conversion process we can upgrade, add or remove 3rd party dependencies that can cause security vulnerabilities in the system or address a replacement for them to keep us current.

**Code Quality:** During the conversion process we can refactor the original code to increase readability, modularity, apply separation of concern, remove replicated code and enforce input validation to prevent multiple vulnerabilities.

**Solutions: How do you determine the appropriate fix to a security vulnerability?**

Fortunately, every revealed vulnerability has received a fix or mitigation for almost all programming languages, libraries, packages or dependencies out there used in the industry.

My approach to determining the fix could be as follows.

1. Determine if the vulnerability is related to the language itself or coding issue, for example it’s recommended to use smart pointers over regular pointers to avoid memory leaks. That has been addressed by the creator of C++ and the community; therefore, I need to follow and check lasted updates about the language. If the issue raises from the code itself like sanity check; I would update the code to prevent vulnerability.
2. Static vulnerability security testing could come handy to discover which 3rd party, package or library might introduce a vulnerability. That can be resolved by updating the package, replacing it with a secure one or writing it myself.
3. Apply multiple sanity checks to prevent overflow issues.
4. Interacting with Database must be scrutinized and apply SQL injection prevention according to OWASP.

Basically, for each vulnerability I will check the appropriate way to fix it according to best practices either by official organization or tech giant leading companies’ practices.

**References:**

<https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html>