Kiuwan Vulnerability Types

This guide explains what a vulnerability is and which types are being used by the Kiuwan Solutions.

Contents:

What is a Vulnerability

Every Kiuwan rule checks for specific source code defects (flaws, faults, bugs and/or improvements) according to the software characteristics addressed by Kiuwan (Efficiency, Maintainability, Portability, Reliability, and Security).

Some of these rules are directly related to security issues (rules that find defects that can be directly used by a hacker to gain access to a system or network).

But there are other rules not directly related to a security issue but that could lead to a security issue (if a hacker knew how to exploit it). A typical example is a buffer management error that would cause a segmentation fault but that could be exploited by a hacker to inject malicious code to gain access to the system.

> To group both kinds of defects , Kiuwan introduces the concept of **Vulnerability**, i.e. a source code defect that can lead, directly or indirectly, to a security issue.
>
> Following this approach, Kiuwan Code Security inspects your source code by applying all the rules that can (directly or not) highlight a security issue.
>
> That's the reason why you can find also non-security rules in Kiuwan Code Security.

Vulnerability types

Every vulnerability detected by Kiuwan is classified under a **category**, i.e. a **Vulnerability Type**.

You can find many different taxonomies to classify software vulnerabilities (CWE, OWASP, CERT, SANS, Pernicious Kingdoms, etc.).

When looking to adopt a taxonomy for Kiuwan vulnerabilities, we found that most of them were overly complex.

The Kiuwan Vulnerability Types are based on **Wheeler & Moorthy**'s paper (*State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation*) and related works (NSA Center for Assured Software, *NSA-2012*, and *NSA-2011*).

Kiuwan thus considers the following **Vulnerability Types**:

Vulnerability Type	Description
Buffer handling	Improper buffer handling can lead to attackers crashing or gaining complete control of a system. An example would be a buffer overflow that allows an adversary to execute his/her code. This type mostly applies to C/C++ languages. Rules belonging to this vulnerability type allow finding buffer access violations in the source code.
Control flow management	Control flow management deals with timing and synchronization issues that can cause unexpected results when the code is executed. An example would be a race condition. One possible consequence of a race condition is a deadlock which leads to a denial of service. Rules belonging to this vulnerability type allow finding issues in the order of execution within the source code.
Design error	This covers design errors that lead to unintentional vulnerabilities.

Encryption and randomness	Encryption is used to provide data confidentiality. However, if a weak or wrong encryption algorithm is used, an attacker may be able to convert the ciphertext into its original plain text. An example would be the use of a weak Pseudo-Random Number Generator (PRNG). Using a weak PRNG could allow an attacker to guess the next number that is generated. Rules belonging to this vulnerability type check for proper encryption and randomness in the source code.
Error handling and fault isolation	Error handling is used when a program behaves unexpectedly. However, if a program fails to handle errors properly it could lead to unexpected consequences. An example would be an unchecked return value. If a programmer attempts to allocate memory and fails to check if the allocation routine was successful, then a segmentation fault could occur if the memory failed to allocate properly. Rules belonging to this vulnerability type check for proper error handling within the source code.
File handling	File handling deals with reading from and writing to files. An example would be reading from a user-provided file on the hard disk. Unfortunately, adversaries can sometimes provide relative paths that contain periods and slashes. An attacker can use this method to read or write to a file in a different location on the hard disk than the developer expected. Rules belonging to this vulnerability type check for proper file handling within the source code.
Information leaks	Information leaks can cause unintended data to be made available to a user. For example, developers often use error messages to inform users that an error has occurred. Unfortunately, if sensitive information is provided in the error message, an adversary could use it to launch future attacks on the system. Rules belonging to this vulnerability type check for information leaks within the source code.
Initialization and shutdown	Initializing and shutting down resources occurs often in source code. For example, in C/C++ if memory is allocated on the heap it must be deallocated after use. If the memory is not deallocated, it could cause memory leaks and affect system performance. Rules belonging to this vulnerability type check for proper initialization and shutdown of resources in the source code.
Injection	Code injection can occur when user input is not validated properly. One of the most common types of injection flaws is cross-site scripting (XSS). An attacker can place query strings in an input field that could cause unintended data to be displayed. This can often be prevented using proper input validation and/or data encoding. Rules belonging to this vulnerability type check for injection weaknesses in the source code.
Misconfiguration	This covers common parameters not properly configured (for example, in web.xml file in J2EE apps or web.config in .NET, etc) that lead to unintentional vulnerabilities.
Number handling	Number handling issues include incorrect calculations as well as number storage and conversions. An example is an integer overflow. On a 32-bit system, a signed integer's maximum value is 2,147,483,647. If this value is increased by one, its new value will be a negative number rather than the expected 2,147,483,648 due to the limitation of the number of bits used to store the number. Rules belonging to this vulnerability type check for proper number handling in the source code.

Permission, privileges and access controls	Attackers can gain access to a system if the proper authentication and access control mechanisms are not in place. An example would be a hardcoded password or a violation of the least privilege principle. Rules belonging to this vulnerability type check whether or not the source code is preventing unauthorized access to the system.
Pointer and reference handling	Pointers are often used in source code to refer to a block of memory without having to reference the memory block directly. One of the most common pointer errors is a NULL pointer dereference. This occurs when the pointer is expected to point to a block of memory, but instead, it points to the value of NULL. This can cause an exception and lead to a system crash. Rules belonging to this vulnerability type check for proper pointer and reference handling.
System element isolation	This covers errors involving allowing system elements unfettered access to each other.
Other	Vulnerabilities in this class do not fit into any of the other types. An example would be an assignment instead of a comparison. Control flow statements allow developers to compare variables to certain values to determine the proper path to execute. Unfortunately, they also allow developers to make variables assignments. If an assignment is made where a comparison was intended then it could lead to the wrong path of execution.