**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Many attacks come from outside sources that can enter the system through input. Validating data range and length can protect against attacks such as SQL Injection and Buffer Overflow. This can be done in login forms or file uploads, anywhere external data can enter the system. |
| 1. Heed Compiler Warnings | During builds and debugging, warning messages may appear in the console even though the code compiles normally. These warnings could point to issues in the code like uninitialized variables or memory leaks, which can become vulnerabilities. Reading the warnings and changing the code to address these bugs early can help prevent costly vulnerabilities or breaches later. |
| 1. Architect and Design for Security Policies | Include and enforce security policies in the design phase of the system from the beginning, not added later as an afterthought. This can be done by including security requirements in documentation and using security architecture like authentication and encryption. Systems handling personal data should follow standards like GDPR. Designing with security policies from the beginning helps ensure development decisions lead to a secure system. |
| 1. Keep It Simple | If the code is too complex, it can hide bugs and vulnerabilities and may be more difficult to fix once discovered. Simple systems are easier to maintain, and vulnerabilities are easier to find and fix. This also promotes better usability, making the system more accessible for users. For example, requiring users to use complicated passwords may seem like a good way to implement security, but if the requirements are too complicated, users may write down their passwords on notes or note apps, or reuse the same passwords on multiple sites, effectively reducing security. Two-factor authentication can be a workaround for this. |
| 1. Default Deny | Require systems to deny access by default, only granting it after specific conditions have been met, such as verifying credentials or confirming roles. This makes protection the starting point, like a castle wall that only lowers the drawbridge once identity and purpose are verified. It is harder to keep out malicious actors if the system is open access by default. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should be given only the minimum access necessary to perform their tasks. It is a similar concept to Default Deny. Instead of giving all users and systems full privilege, the default should be no privilege and only grant what is required. This limits the risk of accidental changes and reduces the damage a malicious actor could do if they gain access. It can be implemented through techniques like role-based access control (RBAC) and Just-In-Time access (JIT) to limit the time a user may have elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | In addition to sanitizing input data, when data is transferred within the system to other subsystems, such as from an application to a database, third-party API, or a logging tool, it should also be sanitized to prevent issues like injection attacks or data leaks. Untrusted characters that could introduce vulnerabilities should be escaped or encoded. |
| 1. Practice Defense in Depth | Instead of relying on a single strong layer of security, systems should employ multiple layers of defense. For example, a system could be protected by a hardware gateway, a software firewall, user authentication, and RBAC. If an attack gets past one layer, they may still be stopped by the next. This layered approach helps catch threats at different stages, giving developers time to detect and stop the attack before it causes further damage. |
| 1. Use Effective Quality Assurance Techniques | Apply QA techniques like code reviews, static analysis, penetration testing, and fuzz testing to ensure the code is not only functional but also secure. Performing tests early and often in the development cycle can catch bugs and vulnerabilities before they make it into production, where they are more expensive to fix and can damage a company’s reputation. |
| 1. Adopt a Secure Coding Standard | To ensure security is considered during all of the development lifecycle, use a secure coding standard like OWASP or CERT. These standards provide rules and best practices that help developers write secure, maintainable code. Tools like static code analyzers can enforce these standards by flagging code that does not follow the guidelines or introduces vulnerabilities, allowing developers to fix issues early in the development cycle. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **DCL52-CPP. Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Reference types cannot be changed, CPP will not allow it, so reference types do not need to be qualified with const or volatile. This can lead to undefined behavior, or confusion about the mutability of the referenced value. |

| **Noncompliant Code** |
| --- |
| &const tries to apply const to the reference itself, which creates a const qualified reference to the char, leading to undefined behavior |
| #include <iostream>    **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |
| Here, &p correctly creates a reference to a const qualified char, but p is then modified in the following line. Attempting to modify it leads to errors. The const is applied to p, not the reference to p. |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Eliminate the unnecessary const qualifier. p can be modified safely, and the reference is already immutable, as enforced by C++ |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 2. Heed Compiler Warnings: A compiler warning would have been shown for this error. Address the warning to improve code readability.  4. Keep it Simple: Remove unnecessary qualifiers to reduce code complexity. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL52** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [**CERT C++: DCL52-CPP**](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **INT33-C/C++. Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | If the second operand is not checked to ensure it is not 0 in a division or remainder operation, a divide-by-zero error could occur and result in undefined behavior at runtime leading to abnormal program termination and leaving the program vulnerable to denial-of-service attacks. |

| **Noncompliant Code** |
| --- |
| While this code block prevents signed integer overflow, it does not check if the divisor s\_b is 0 before dividing to ensure that divide-by-zero division will not happen. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_a == LONG\_MIN) && (s\_b == -1)) {      /\* Handle error \*/    } **else** {      result = s\_a / s\_b;    }    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This code block checks first if s\_b is zero and handles the error if it is. If it is not zero, the code proceeds to check that an overflow will not happen. Checking before dividing the operands prevents divide-by-zero error, in addition to preventing signed integer overflow. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {      /\* Handle error \*/    } **else** {      result = s\_a / s\_b;    }    /\* ... \*/  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 1. Validate Input Data: Check that any divisor, especially if it comes from user input, is not zero before dividing.  4. Keep It Simple: Use straightforward checks to prevent undefined behavior |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **int-division-by-zero**  **int-modulo-by-zero** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT33** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | **LANG.ARITH.DIVZERO LANG.ARITH.FDIVZERO** | Division by zero Float Division By Zero |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule (In particular, it ensures that all operations involving division or modulo are preceded by a check ensuring that the second operand is nonzero.) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **DIVIDE\_BY\_ZERO** | Fully implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.15 | **zerodiv zerodivcond** |  |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.11.0 | **zerodiv zerodivcond**  **premium-cert-int33-c** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2025.2 | **C2830**  **C++2830**  **DF2831, DF2832, DF2833** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2025.2 | **DBZ.CONST** **DBZ.CONST.CALL** **DBZ.GENERAL** **DBZ.ITERATOR** **DBZ.ITERATOR.CALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **43 D, 127 D, 248 S, 629 S, 80 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-INT33-a** | Avoid division by zero |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rule INT33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint33c.html) | Checks for:   * Integer division by zero * Tainted division operand * Tainted modulo operand   Rule fully covered. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [**S3518**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-3518) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.37 | [**V609**](https://pvs-studio.com/en/docs/warnings/v609/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **division\_by\_zero** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/c37797b7)). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Buffer overflows can happen if a string is copied to a buffer that does not have enough space for it. This provides an opportunity for malicious attackers to exploit the overflow and access private data. |

| **Noncompliant Code** |
| --- |
| buf can hold up to 12 characters, but cin can accept input of any length which could lead to a buffer overflow |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |
| Using cin.width() may help, but bufOne may receive a truncated string, losing data or forming a vulnerability, and bufTwo may still overflow, unless .width() is called again. |
| #include <iostream>    **void** f() {  **char** bufOne[12];  **char** bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |
| Here, instead of user input, the input comes from a file. a char variable is set up with space for 32 characters and the file is read using that size but using std::basic\_istream<T>::read()does not guarantee that there will be a null character to terminate the string, leading to undefined behavior when using std::string str(buffer);. |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }      std::string str(buffer);    // ...  } |

| **Compliant Code** |
| --- |
| Replace char with string to avoid truncation and buffer overflow |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |
| Use std::string str(buffer, in.gcount()); to safely handle null truncation, assuming the input is at most 32 characters. |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }    std::string str(buffer, in.gcount());    // ...  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 1. Validate Input Data: Validate length and range of input data  4. Keep It Simple: Use methods from the C++ library like std::string to reduce the risk of overflow |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **MISC.MEM.NTERM LANG.MEM.BO LANG.MEM.TO** | No space for null terminator Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** | Partially checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **RTOS\_33 RTOS\_34 shadowVariable UNSAFE\_03 UNSAFE\_04** | Fully implemented |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **FIO30-C/CPP. Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | SQL Injection can happen through format string injection. User input embedded directly in format strings without validation or sanitization can be exploited, allowing attackers to inject malicious SQL commands to retrieve personal data or carry out other attacks. |

| **Noncompliant Code** |
| --- |
| User input for username and password is received and immediately embedded into an SQL query using a format string. If the user had input something like username = ’; DROP TABLE users; -- and password = anything, the constructed format string would result in a query that deletes the entire users table from the database |
| #include <stdio.h>  #include <string.h>  void authenticateUser(const char \*username, const char \*password) {  char username[20];  char password[20];  char query[100];  scanf(“%19s”, username);  scanf(“%19s”, password);  // Vulnerable: directly embedding user input in SQL query  snprintf(query, sizeof(query),  "SELECT \* FROM users WHERE username = '%s' AND password =  '%s';",  username, password);  printf("Executing query: %s\n", query);  } |

| **Compliant Code** |
| --- |
| To avoid string injection, use parameterized queries, where user input is passed separately from the SQL query string. To do this, create a query with placeholders and pass the user input as separate parameters. Databases like sqlite3, MySQL, or libpq are required to implement parameter binding, this is a simplified example of safe string handling avoiding unsafe formatting. |
| #include <stdio.h>  #include <string.h>  void authenticateUser() {  char username[20];  char password[20];  const char \*query\_template = "SELECT \* FROM users WHERE username = ? AND password = ?;";  scanf("%19s", username);  scanf("%19s", password);  // Simulate parameterized query  printf("Executing query: %s [username=%s, password=%s]\n", query\_template, username, password);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 1. Validate Input Data: Input data may contain malicious characters, never embed them directly into format strings  7. Sanitize Data Sent to Other Systems: Safely escape or format user input before passing it to functions or databases |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported via stubbing/taint analysis |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FIO30** | Partially implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | **IO.INJ.FMT MISC.FMT** | Format string injection Format string |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_STRING** | Implemented |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.11.0 | **premium-cert-fio30-c** |  |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wformat-security flag is used |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2025.2 | **DF4916, DF4917, DF4918** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2025.2 | **SV.FMTSTR.GENERIC SV.TAINTED.FMTSTR** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **86 D** | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-FIO30-a** **CERT\_C-FIO30-b** **CERT\_C-FIO30-c** | Avoid calling functions printf/wprintf with only one argument other than string constant Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable Never use unfiltered data from an untrusted user as the format parameter |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **592** | Partially supported: reports non-literal format strings |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rule FIO30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio30c.html) | Checks for tainted string format (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.37 | [**V618**](https://pvs-studio.com/en/docs/warnings/v618/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **MEM52-CPP. Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | There are two kinds of new operators to allocate memory and it is important to use the right way to detect and handle the memory allocation errors. ::operator new(std::size\_t)throws an exception, std::bad\_alloc if allocation fails, while ::operator new(std::size\_t, const std::nothrow\_t &)does not throw an exception but returns nullptr. If the errors are not detected and handled properly, this may lead to abnormal program termination or leave the program vulnerable to denial-of-service attacks. |

| **Noncompliant Code** |
| --- |
| ::operator new(std::size\_t) is used to create an array of int in a noexcept function, but there is no check performed on the results of the allocation. If new throws an exception because the allocation failed, the program could terminate abnormally. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |
| Do not perform two memory allocations within the same expression as it makes error handling impossible and there is a risk of memory leak. If new B fails, the first allocation is leaked. |
| **struct** A { /\* ... \*/ };  **struct** B { /\* ... \*/ };    **void** g(A \*, B \*);  **void** f() {    g(**new** A, **new** B);  } |

| **Compliant Code** |
| --- |
| Here we are using std::nothrow in a noexcept function, so instead of an exception, a nullptr will be returned if there is an error. This block shows appropriate testing of the allocation of memory to ensure it is not nullptr before referencing the pointer. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |
| Another way to handle it without std::nothrow is using a try/catch block to catch the exception and handle it. Alternatively, the function may also be marked noexcept(false)or remove the specifier altogether. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy;  **try** {      copy = **new** **int**[size];    } **catch**(std::bad\_alloc) {      // Handle error  **return**;    }    // At this point, copy has been initialized to allocated memory    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |
| Use std::unique\_ptr to manage memory allocation automatically. |
| #include <memory>    **struct** A { /\* ... \*/ };  **struct** B { /\* ... \*/ };    **void** g(std::unique\_ptr<A> a, std::unique\_ptr<B> b);  **void** f() {    g(std::make\_unique<A>(), std::make\_unique<B>());  } |
| Do not rely on memory allocation and pass the objects by reference. |
| **struct** A { /\* ... \*/ };  **struct** B { /\* ... \*/ };    **void** g(A &a, B &b);  **void** f() {    A a;    B b;    g(a, b);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 3. Architect and Design for Security Policies: Handle memory failure properly from the beginning  8. Practice Defense in Depth: Combine exception handling, safe memory allocation and other safety techniques to ensure security |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2025.2 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.37 | [**V522**](https://pvs-studio.com/en/docs/warnings/v522/)**,**[**V668**](https://pvs-studio.com/en/docs/warnings/v668/) |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **DCL03-C. Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | The C++ version was marked VOID, but this principle still applies through static\_assert. While assert checks conditions at runtime and is useful during development, static\_assert checks them at compile time and catches bugs even if the code path is never triggered. |

| **Noncompliant Code** |
| --- |
| This code only uses assert during execution, and only if that code path runs, which makes it easy to miss if func() is never called during testing. |
| #include <cassert>  **struct** Timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func() {  **assert**(**sizeof**(Timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  return 0;  } |

| **Compliant Code** |
| --- |
| Use #error preprocessor directives to generate clear, user-defined messages during compilation. This stops compilation and displays the message |
| **struct** Timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(Timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |
| Use static\_assert to diagnose assumptions at compile time to avoid runtime errors and receive clear diagnostic messages. |
| **struct** Timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(Timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                "Structure must not have any padding"); |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 4. Keep it Simple: Resolve issues before the program runs using simple, clean assertions  9. Use Effective Quality Assurance Techniques: Catch errors at compile time instead of runtime |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/c/Security+Reviewer+-+Static+Reviewer) | 6.02 | **C13 C14 C15 C52 C129 C130 C132 C133 C135 C154 C155** |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **ERR51-CPP. Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When an exception is thrown, the program searches for a matching exception handler. If none is found, the program terminates abnormally, which can lead to denial-of-service vulnerabilities. To avoid this, ensure that all exceptions are caught by matching exception handlers. |

| **Noncompliant Code** |
| --- |
| No matching handler is found to catch the exception thrown by throwing\_func(), so the program terminates abnormally. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| main() has a try/catch block wrapped around the call for the f() function, which catches and handles the exception. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 3. Architect and Design for Security Policies: Use consistent error handling to avoid unhandled exceptions  9. Use Effective Quality Assurance Techniques: Using proper error handling prevents unexpected program termination |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.STRUCT.UCTCH PARSE.MBDH** | Masked by handler Masked by default handler |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **C35** | Fully implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **DCL50-CPP. Do not define a C-style variadic function** |
| --- | --- | --- |
| **Declarations** | STD-008-CPP | Variadic functions use a C-style ellipses at the end of a function’s parameters so that the function can accept any number of arguments. While flexible, these are dangerous, as the arguments cannot be checked for type safety and could lead to undefined behavior or used to run arbitrary code. |

| **Noncompliant Code** |
| --- |
| Using a C-style variadic function allows any number of integers to be passed to the function to add them up, but they cannot be checked to ensure they are integers, or that a 0 has been passed after the first two arguments to indicate it can stop reading arguments. |
| #include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| Use template parameter pack and type traits to ensure only integers are accepted and the function is type-safe. The recursion adds all he values without needing a sentinel. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |
| Use a braced initializer list that also uses a template to ensure type safety. This avoids recursion and does not need a sentinel. |
| #include <type\_traits>    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg i, Arg j, Ts... all) {  **int** values[] = { j, all... };  **int** r = i;  **for** (auto v : values) {      r += v;    }  **return** r;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 1. Validate Input Data: Use safe, typed parameters to validate the arguments passed into the function. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **function-ellipsis** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++2012, C++2625** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2025.2 | **MISRA.FUNC.VARARG** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **41 S** | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL50-a** | Functions shall not be defined with a variable number of arguments |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl50cpp.html) | Checks for function definition with ellipsis notation (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **function-ellipsis** | Fully checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **UNSAFE\_09** | Fully implemented |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**FunctionEllipsis**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-923) |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **MSC50-CPP. Do not use std::rand() for generating pseudorandom numbers** |
| --- | --- | --- |
| **Security** | STD-009-CPP | std::rand() does not guarantee truly random or unpredictable numbers. Avoid using it to generate random numbers for security related applications such as cryptography. |

| **Noncompliant Code** |
| --- |
| The numbers generated by the rand() function are not truly random nor unpredictable. There is a possibility of generating the same ID multiple times. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters  // "ID" followed by a random integer  // in the range [0-10000].    id += std::to\_string(std::**rand**() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| Use std::random\_device, the Mersenne Twister algorithm, and std::uniform\_int\_distribution to generate safer, more random numbers, avoiding the modulo bias from rand()%10000. |
| #include <random>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters  // "ID" followed by a random integer  // in the range [0-10000].    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 3. Architect and Design for Security Policies: Use proper security structures during design  9. Use Effective Quality Assurance Techniques: Test to verify randomness meets the requirements for the application |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | **BADFUNC.RANDOM.RAND** | Use of rand |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC30** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++5028** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **CERT.MSC.STD\_RAND\_CALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MSC50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc50cpp.html) | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **RTOS\_07** | Fully implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **ERR50-CPP. Do not abruptly terminate the program** |
| --- | --- | --- |
| **Error Handling** | STD-010-CPP | Avoid using functions like std::abort(), std::quick\_exit(), std::\_Exit(), or throwing uncaught exceptions, as these can cause the program to abruptly terminate without proper memory cleanup, buffer flushing or stack unwinding. |

| **Noncompliant Code** |
| --- |
| When the program exits, it calls f() as an exit handler. However f() calls throwing\_func(); which might throw an exception, leading to the program calling std::terminate(), abruptly terminating it. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| Catch exceptions in a try/catch block inside atexit()to avoid abrupt termination. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** 3. Architect and Design for Security Policies: Follow error handling best practice and encase code in simple try/catch blocks to handle exceptions gracefully  4. Keep It Simple: Implement straightforward error handling logic |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m CERT\_CPP-ERR50-n** | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.37 | [**V667**](https://pvs-studio.com/en/docs/warnings/v667/)**,**[**V2014**](https://pvs-studio.com/en/docs/warnings/v2014/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stdlib-use** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S990**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-990) |  |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation makes it easy to quickly and consistently test software using methods like scans, dependency checks, and patching to build secure software systems. It should be used throughout the entire development life cycle to ensure that security policies are consistently enforced at every stage. This approach falls under the Shift Left Security principle, which encourages applying security earlier in the development life cycle rather than leaving it to the end.

Automation reduces human error and can catch vulnerabilities that developers may miss. However, it does not detect every error, such as “STD-003-CPP: STR50-CPP: Guarantee that storage for strings has sufficient space for character data and the null terminator.” Automation must be used to complement human testing and code reviews rather than replacing them, following the Defense in Depth principle.

Automation can be used in the Design and Build stages of DevSecOps by choosing automation tools according to the company’s coding standards to enforce policies and guide developers as they write code. Static Application Security Testing (SAST) tools like Cppcheck can help enforce standards and find vulnerabilities early. In the Verify and Test stage, vulnerability scanning tools can find weaknesses before deployment. Automation can also be helpful in the Monitor and Detect stage by helping detect threats quickly, and it can even automate responses to common threats.

Active Directory (AD) stores private user information, and the Lightweight Directory Access Protocol (LDAP) is a protocol for reading and modifying that user information. They can both be automated to make it easier to confirm user identity within the system, helping to implement Authentication, Authorization, and Accounting (AAA) security.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | Low | 1 |
| STD-002-CPP | Low | Likely | Medium | Medium | 3 |
| STD-003-CPP | High | Likely | High | Medium | 5 |
| STD-004-CPP | High | Likely | Medium | High | 5 |
| STD-005-CPP | High | Likely | Low | High | 5 |
| STD-006-CPP | Low | Unlikely | Low | Low | 1 |
| STD-007-CPP | Low | Probable | Low | Medium | 3 |
| STD-008-CPP | High | Probable | Medium | High | 4 |
| STD-009-CPP | Medium | Unlikely | Medium | Low | 2 |
| STD-010-CPP | Low | Probable | High | Low | 2 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest uses encryption algorithms such as Advanced Encryption Standard (AES) to protect data which is stored on a device or system such as a hard drive or database. It is useful for protecting stored sensitive personal data like passwords or credit card numbers by encrypting it with an encryption key. If the device it is stored in is lost, stolen, or hacked, malicious actors cannot read the stored data without the decryption key. This policy should be applied any time sensitive data is stored on a drive, server, or database. |
| Encryption in flight | Encryption in flight, also known as in transit, protects data while it moves between devices or systems. It is encrypted before transmission and decrypted by the receiver. This protects the data in case it is intercepted by an attacker, as it remains unreadable without the decryption key. This policy should be implemented whenever data is transmitted over networks, such as sending emails, text messages, or transmitting payment information. |
| Encryption in use | Encryption in use involves encrypting data while it is being used and processed by applications. Often data is at its most vulnerable when it is in use, as it needs to be decrypted to be able to be read and used. Through partially homomorphic encryption (PHE) and fully homomorphic encryption (FHE), computers can run operations on encrypted data without fully decrypting it, protecting it from threats like memory scraping. This policy should be enforced when processing sensitive data, or when data is being processed in an untrusted environment, like public clouds. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication uses credentials such as usernames and passwords to verify a user before granting them access to a system or network. Other ways to prove identity include biometrics like fingerprint scans. Active Directory is a database that stores user credentials and validates them during login. This policy applies at all login points to keep unverified users from accessing the system and sensitive data. Multifactor authentication (MFA) should be enforced, and passwords should be appropriately complex for safety. |
| Authorization | The next step after authentication, authorization grants users access to resources based on their roles using Role-Based Access Control (RBAC) to limit permission following the principle of Least Privilege. For example, a standard employee may read data, but only an Admin level user can make changes to the database. When new users are added, they must only be granted the minimum level of access needed to perform their roles in the system. Authorization should be reviewed periodically to remove unnecessary permissions or add necessary ones. |
| Accounting | Accounting gathers data from the actions of users that have been authenticated and authorized in the system to keep accountability. Logs must keep track of events like logins, file access and downloads, and database changes. This keeps accountability of any changes in the system or database and what files were accessed by which user. These accountability logs should be periodically reviewed to see patterns of usage and ensure users are using the system appropriately. Any suspicious activity must be flagged and investigated immediately. If a breach happens, these logs could help track the source. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 07/20/2025 | Security principles, coding standards | Angela Karina Vegega Ortiz |  |
| 3.0 | 08/10/2025 | Encryption, Triple-A framework, Automation, Risk Assessment, Principles | Angela Karina Vegega Ortiz |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |