**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 | Implement methods to ensure data being entered into a system from external sources is acceptable in form, type, and free of possibility of inducing unexpected system behaviors. Includes methods such as input size, type, and character limitations, among others. |
| 1. Heed Compiler Warnings | Compiler warnings ensure potential error-producing source code is properly addressed. As an early detection mechanism, compiler warnings serve to flag at-risk code for subsequent refactoring to avoid potential compile-time errors. Best practices dictate all delivered project source code is warning-free. Warnings may not cause compilation to fail, thus it is important to address such warnings with modifications to the code base for minimizing probability of errors. |
| 1. Architect and Design for Security Policies | Security principles and policies are to be an integrated component of system architecture and design from the system’s inception. Ensures proper security measures are incorporated during development to produce a safe and secure system the first time. Includes best practices such as system segmentation, two-factor authentication, and defense in depth. |
| 1. Keep It Simple | Avoid use of overly-complicated code implementations. Simplifying the code base while achieving requirements minimizes the opportunity for unforeseen system errors or behaviors while easing debugging and vulnerability detection. Simple code bases also allow for more adequate security measures. |
| 1. Default Deny | Adopt a security policy of denying all systems access by default. Accesses are granted only when certain system components are deemed allowable while the rest, if not all, are made authentication-required. |
| 1. Adhere to the Principle of Least Privilege | System is developed to only grant access to the least amount of required data and/or resources for a user to complete a particular task. |
| 1. Sanitize Data Sent to Other Systems | Ensure data being passed to an external or subsystem is formulated according to the sub-system’s policies and requirements. Avoids passing data that may cause unintended behaviors within the sub-system. |
| 1. Practice Defense in Depth | Employ multiple layers of security to protect system components. Layers of security components provide coverage for possible vulnerabilities existing within preceding or subsequent layers for more holistic system protection. |
| 1. Use Effective Quality Assurance Techniques | Utilize quality-assurance techniques such as unit testing, first and third-party security auditing, penetration, and fuzz testing to detect and subsequently address system vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Implement industry best practices, techniques, and standards as aligned by software industry, organizations, and secure coding community to ensure the safest and most secure system possible is developed. |

### C/C++ Ten Coding Standards

Source: SEI Cert Coding Standards - <https://wiki.sei.cmu.edu/confluence/display/seccode>

#### Coding Standard 1

| **Coding Standard** | **Label** | **Never qualify reference types with const or volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Declared reference types are inherently treated as const-qualified within the compiler and cannot be changed or modified. Avoid applying a const or volatile qualification to reference types that are immutable as assigning a change in value can cause errors. |

| **Noncompliant Code** |
| --- |
| Reference type variable &p is declared as a const-qualified variable. An attempt to initialize the value of the variable is later met with an error (compiler-dependent) due to value change. If compiler does not throw an error, the non-compliant code may cause unexpected behavior such as allowing the updated value to variable &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** |
| --- |
| Removing the const-qualifier for char reference variable &p allows for successful declaration and initialization of the variable without error. |
| #include <iostream>    void f(char c) {    char &p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Principles(s):**   * Heed Compiler Warnings: implementing non-compliant code results in a compiler warning C4427: anachronism used: qualifiers on reference are ignored. Use compiler warnings as an early-detection system to ensure code is within compliance and further minimize the possibility of undefined behaviors. * Keep it simple: Avoid implementing overly-complicated code to avoid undefined behaviors in the application. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-DCL52-a | - |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: DCL52-CPP | 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 | - | Automatic checks for rule violation results in error production. Flags or options not necessary. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | S3708 | - |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | All functions other than the “void” variety must return a value. There must be a path for the function returning a value to successfully exit thus the function should be coded with consideration to the possible outcomes of the value. |

| **Noncompliant Code** |
| --- |
| The function only accounts for one exit path where the value of variable “val” is less than zero, thus negative. However, it does not account for the possibility of variable “val” being a positive value. |
| int absolute\_value(int val) {    if (val < 0) {      return -val;    }  } |

| **Compliant Code** |
| --- |
| The function returns a value for both possible exit paths – one in which the variable “val” is a negative value, and the second possibility of variable “val” being a positive value. Thus, the function has accounted for all possible exit paths. |
| int absolute\_value(int val) {    if (val < 0) {      return -val;    }    return val;  } |

| **Principles(s):**   * Adopt a Secure Coding Standard: keeping secure coding practices at the forefront during development emphasizes a focus on industry standards and best practices. One such best practice includes ensuring all exit paths to a function are accounted for – further lending to a secure implementation of code. * Keep it Simple: applying this principle means being mindful of implementing only the necessary code needed to accomplish the task without over-complication. Doing so can mitigate potential issues with a function(s) exit paths and avoid undesired behaviors such as infinite loops due to issues with conditional statements. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Return-implicit | Fully checked |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 2 D, 36 S | Fully implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: MSC52-CPP | Checks for missing return statements (partial coverage) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | Return-implicit | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Starting with C++11, all string data types are terminated with a null terminator. Ensure proper handling of string data types by assuring that all methods involving a buffer for string storage/manipulation uses a buffer large enough for the target data. |

| **Noncompliant Code** |
| --- |
| This loop traverses a string and copies the character data to a buffer that is insufficient in size to also store the null-termination character. This will produce an off-by-one error as the buffer will copy all string data sans the null-termination character at the end. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {     size\_t i;       for (i = 0; src[i] && (i < n); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| The for-loop exit condition has been modified to account for the null termination character to allow for successful copying of the source string into the targeted string buffer. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {     size\_t i;       for (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Principles(s):**   * Validate Input Data: When working with data input from external sources, ensure proper memory allocation for variables storing and manipulating character array and string data-types to avoid undefined behaviors and buffer overflows. Apply industry-standard validation such as size limits and character limitations. * Sanitize Data Sent to Other Systems: Mitigate potential security vulnerabilities and undefined behaviors via practicing data cleanliness. Ensure data being processed and/or passed to other sub-systems meet expected sub-system specific criteria. * Practice Defense in Depth: Utilize multiple tools and methods to ensure buffer overflows do not occur when working with char array and string data-types. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 24.04 | - | All buffer overflows resulting from insufficiently sized buffers are reported when copying data too large for the buffer. |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity?_gl=1*j0f7ad*_gcl_au*MjM1NTYzMDkyLjE3NTQzNTI4OTk.*_ga*MTM4NDg1MDgwNy4xNzU0NDQ4OTQz*_ga_87WECW6HCS*czE3NTQ0NDg5NDMkbzEkZzEkdDE3NTQ0NDk1MDQkajU4JGwwJGgw) | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 489 S, 109 D, 66 X, 70 X, 71 X | Partially implemented |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | When passing data to an external subsystem, the manner in which the subsystem processes the data has to be considered. This is because passed data may contain special characters or other data formatted in a way that triggers a vulnerability in the target system. |

| **Noncompliant Code** |
| --- |
| Sample code originating from ticket VU#881872 for Sun Solaris TELNET daemon – execl function allows for user login but passes unsanitized data via USER environment variable. This code allows malicious actors to pass further command line options to the function. |
| (void) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    (AuthenticatingUser != NULL ? AuthenticatingUser :    getenv("USER")),    0); |

| **Compliant Code** |
| --- |
| String “- -“ is added to separate command-line arguments from the passed data from the USER environment variable thus preventing unexpected behaviors from the login process. |
| (void) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    "--",    (AuthenticatingUser != NULL ? AuthenticatingUser :    getenv("USER")), 0); |

| **Principles(s):**   * Architect and Design for Security Policies: Implementing security at all steps from system conception to completion ensures the architecture and sub-systems utilized are considered at project inception. Allows for proper planning of implementation to mitigate security risks with system-to-system interactions. * Validate Input Data: Ensure proper processing of input data when passing into sub-systems and components. Ensures data passed to sub-systems is formatted in a manner accepted by the sub-system’s criteria – ultimately avoiding undefined behavior, errors, and security vulnerabilities. * Practice Defense in Depth: Apply data validation, exception handling (try/catch), input limits, and other techniques and tools to ensure clean data can be passed to sub-systems. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity?_gl=1*j0f7ad*_gcl_au*MjM1NTYzMDkyLjE3NTQzNTI4OTk.*_ga*MTM4NDg1MDgwNy4xNzU0NDQ4OTQz*_ga_87WECW6HCS*czE3NTQ0NDg5NDMkbzEkZzEkdDE3NTQ0NDk1MDQkajU4JGwwJGgw) | 6.5 | TAINTED\_STRING | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 108 D, 109 D | Partially implemented |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protects against command injection  Protects against file name injection  Protects against SQL injection |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 24.04 | - | Supported by stubbing/taint analysis |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Use appropriate deallocator functions to free up memory resources used during dynamic use of allocator objects. Functions such as delete(), deallocate(), and free() can be used with their associated allocator functions to ensure memory resources are not tied up by objects that are no longer needed. |

| **Noncompliant Code** |
| --- |
| In this example, function f() makes use of a pointer s1 which is subsequently passed into the delete function. However, because the new operator used when declaring the value for pointer s1 would not have returned a memory address, we can expect undefined behavior when the delete() function is executed. |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      delete s1;  } |

| **Compliant Code** |
| --- |
| In this example, the s1 pointer is deallocated using its deconstructor to successfully free up memory resources. |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      s1->~S();  } |

| **Principles(s):**   * Heed Compiler Warnings: Ensure warnings and errors as they pertain to using the proper operator for freeing up dynamically allocated resources. * Architect and Design for Security Policies: Consider dynamically allocated resources during system components identification and architecture building to ensure resource allocation is properly executed. Avoids creating possible security vulnerabilities resulting from memory leaks and possible undefined behaviors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Invalid\_dynamic\_memory\_allocation  Dangling\_pointer\_use | - |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 7.2.0 | CertC++=MEM51 | - |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | - | - | Runtime Detection |
| [Security Reviewer – Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | wcsdupCalled | Fully implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate diagnostic tests using assertions** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Validate code for proper behavior by utilizing the assert() macro with a passed expression. Provides a boolean result and relevant information for failed assertions such as source file, argument text, line number, and name of calling function. |

| **Noncompliant Code** |
| --- |
| Code example attempts to utilize an assertion to verify memory allocation has taken place. However, memory allocation and deallocation occurs at many points within a given system and using an assertion to confirm it has taken place could lead to undefined behaviors up to and including termination of the program. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char \*)malloc(len + 1);    assert(NULL != dup);      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| Instead of utilizing an assertion, a conditional statement can be used in this scenario to confirm proper memory allocation has taken place. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char\*)malloc(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Principles(s):**   * Use Effective Quality Assurance Techniques: Validate functions and other code using assertions, unit testing, and other quality assurance techniques to ensure expected behavior is observed when testing code. Can help to identify potential vulnerabilities early, as well as confirming proper functionality of the code. * Adopt a Secure Coding Standard: Ensure proper quality testing is integrated into coding standards to incorporate proper validation as a part of the development cycle. Sets a standard of secure coding implementations via vulnerability detection and mitigation. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity?_gl=1*j0f7ad*_gcl_au*MjM1NTYzMDkyLjE3NTQzNTI4OTk.*_ga*MTM4NDg1MDgwNy4xNzU0NDQ4OTQz*_ga_87WECW6HCS*czE3NTQ0NDg5NDMkbzEkZzEkdDE3NTQ0NDk1MDQkajU4JGwwJGgw) | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | In the event of an exception, a handler of the appropriate type is found via search to handle the exception occuring within the try/catch block. When an appropriate handler cannot be found, the program can ultimately be terminated by the default terminate handler which abruptly ends program execution, also leaving the system open to denial-of-service attacks. |

| **Noncompliant Code** |
| --- |
| In the example, neither function handles the thrown exception of a matching data type which in turn causes the terminate handler to abruptly terminate program execution. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| Main function handles all exceptions using a try/catch block where the f() function is also called. Doing so allows for the program to ultimately terminate “gracefully” or with better handling of resources as opposed to a sudden termination. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

| **Principles(s):**   * Use Effective Quality Assurance Techniques: Ensure use of try/catch blocks to handle possible exceptions appropriately. Focus on differentiating exceptions hindering proper execution of the application and exceptions that can be caught and allow the program to continue. Use assertions when necessary to check for expected program behaviors to further bolster mitigation of issues arising from thrown exceptions. * Heed Compiler Warnings: consider any potential warnings and/or errors appearing for unhandled exceptions. Ensures coverage of exceptions to lessen the likelihood of execution-breaking errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P6 | L2 |

**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 |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 527 S | Partially implemented |
| [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** | **Do not attempt to modify string literals** |
| --- | --- | --- |
| **String Correctness** | [STD-008-CPP] | String literals are generally stored in read-only memory. When a program is compiled, string literals are referred to by a pointer to the memory address of the array of characters that make up the string literal. If a string literal is modified, what occurs is considered undefined behavior and may result in an access violation error. |

| **Noncompliant Code** |
| --- |
| In this example, a pointer is assigned the memory address of a string containing “string literal.” It then attempts to change the character at position zero within the character array which causes undefined behavior and possible access violation to read-only memory. |
| char \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| The character array is first stored into an array variable where any of data within the array’s index can then be changed. In this case, the string literal itself is not being modified. |
| char str[] = "string literal";  str[0] = 'S'; |

| **Principles(s):**   * Keep It Simple: Ensure adherence to rule dictating that string literal pointers are immutable. If modification of a string literal is needed, create a copy of the string literal using a non-pointer variable. * Heed Compiler Warnings: Modifying string literals will likely cause compiler warnings/errors for access violations. Ensure focus is on warnings to ensure any potential issues with accessing string literals is avoided. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | String-literal-modification-write-to-string-literal | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-STR30 | Fully implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 24.04 | String-literal-modification | Partially checked |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 489, 1776 | Partially supported |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Integer Correctness** | [STD-009-CPP] | Unsigned integer wrapping can occur when an operation between two operands of unsigned integer type is conducted and the resulting value cannot be represented by the container data type, thus resulting in unexpected values. |

| **Noncompliant Code** |
| --- |
| In the example, the sum of the values for arguments ui\_a and ui\_b can eventually require more memory than is available for storing or executing further operations – leading to undefined behavior and exposing the system to external exploits. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Importing the limits header allows for proper error handling when the sum operation results in a wrapped value, thus allowing detection of the issue and for the result of the function to be handled accordingly. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum;    if (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } else {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |

| **Principles(s):**   * Validate Input Data: Ensure adequate input validation is put in place to avoid possible buffer overflow/underflow conditions causing undetermined behaviors. * Use Effective Quality Assurance Techniques: Utilize assertions to validate for proper behavior of operations on input data. Unit testing adds an additional layer for confirming the program behaves as intended. * Heed Compiler Warnings: Where applicable, focus on warnings from the compiler noting possible buffer over/under flows that may affect the correct representation of values after an operation. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 24.04 | Integer-overflow | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-INT30 | Implemented |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity?_gl=1*j0f7ad*_gcl_au*MjM1NTYzMDkyLjE3NTQzNTI4OTk.*_ga*MTM4NDg1MDgwNy4xNzU0NDQ4OTQz*_ga_87WECW6HCS*czE3NTQ0NDg5NDMkbzEkZzEkdDE3NTQ0NDk1MDQkajU4JGwwJGgw) | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 493 S, 494 S | Partially Implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not declare variables inside a switch statement before the first case label** |
| --- | --- | --- |
| **Declarations** | [STD-010-CPP] | Conditional statements enclosed in a switch statement block execute via jumps to lines of code within the switch body and are designated to execute statements based on the control expression. Thus declaring and/or initializing variables prior to the first case clause in the switch statement results in indeterminate values. |

| **Noncompliant Code** |
| --- |
| In this example integer variable ‘i’ is declared and initialized after the opening switch statement and prior to the first case, thus the variable is not initialized and results in an unexpected value when output. |
| #include <stdio.h>    extern void f(int i);    void func(int expr) {    switch (expr) {      int i = 4;      f(i);    case 0:      i = 17;      /\* Falls through into default code \*/    default:      printf("%d\n", i);    }  } |

| **Compliant Code** |
| --- |
| Declaration and initialization of the variable and function moved outside of the switch statement thus successfully initializing the values prior to the switch statement executing and modifying the value. |
| #include <stdio.h>    extern void f(int i);    int func(int expr) {    /\*     \* Move the code outside the switch block; now the statements     \* will get executed.     \*/    int i = 4;    f(i);      switch (expr) {      case 0:        i = 17;        /\* Falls through into default code \*/      default:        printf("%d\n", i);    }    return 0;  } |

| **Principles(s):**   * Use Effective Quality Assurance Techniques: Utilize Assertions and 3rd party testing libraries to test for expected results from switch/case conditional statements. Ensures code implementation reflects the intended behavior from the application. * Keep it Simple: Avoid overly complicated switch/case statement implementations. Minimizes risk of introducing a security vulnerability via missed code-blocks due to incorrect placement within statement. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 24.04 | Switch-skipped-code | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL41 | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2025.2 | C2008, C2882, C3234 | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2025.2 | CERT.DCL.SWITCH.VAR\_BEFORE\_CASE | Fully implemented |

### Defense-in-Depth Illustration

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



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.

Per this policy, Green Pace is shifting the paradigm from a core DevOps infrastructure to a DevSecOps infrastructure. This means that the policies set forth in this document are to be integrated in all facets of project management within the firm. We strive to achieve this implementation of security into all phases of the development cycle to ensure security is at the forefront of the systems and components we build.

Within the “Assess and Plan” phase of development, we account for potential threat landscapes and focus on well-defined threat assessments for potentials both new and old. During the “Design” and “Build” phases, security is further integrated via system concepts made with adequate security testing using industry-standard best practices in mind. These include, but are not limited to, testing practices such as Google’s test suite, OWASP implementations, and unit testing. During the build, a focus on trusted libraries, repositories and sources further ensures that the system and its components are build with security at their core. Furthermore, during he “Verify and test” phase, we ensure compliance to our rigorous security standards and the policies set forth in this document, while applying the aforementioned test suites to ensure our products live up to our Quality Assurance standards.

Security does not stop at a project’s conception and implementation, but continues well into its lifecycle. With this in mind, once a system has reached the production phase, it is important to conduct regular security-oriented audits. Strategies such as penetration testing, log collection, intrusion detection, and fail-safe measures verifications such as attack blocking and stabilization activities further build upon the overall security of the system. Such activities are part of the “Transition and health check”, “Monitor and detect”, “Respond”, and “Maintain and stabilize” phases of the development cycle.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Medium | Probable | High | P8 | L2 |
| STD-003-CPP | High | Likely | High | P9 | L2 |
| STD-004-CPP | High | Likely | High | P9 | L2 |
| STD-005-CPP | High | Likely | High | P9 | L2 |
| STD-006-C | Low | Unlikely | Low | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P6 | L2 |
| STD-008-CPP | Low | Likely | Low | P6 | L2 |
| STD-009-CPP | High | Likely | High | P9 | L2 |
| STD-010-CPP | Medium | Unlikely | Low | P6 | L2 |

### Policies for Encryption and Triple A

Source: <https://jatheon.com/blog/data-at-rest-data-in-motion-data-in-use/>

| 1. **Encryption** | **What it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Ensures encryption of data “at rest.” Applies to data in storage such as on mobile devices, desktop and/or laptop computers, and any other medium holding data that is not currently in motion or in use. Encryption algorithms should be in place when storing such data, as well as recalling the data for use. This ensures data manipulation is unlikely and that the data at rest is secured. |
| Encryption in flight | Applies to encrypting data that is in motion. Examples include file transfers to and from cloud storage, network data transfers, emails, messaging, client-to-server communications, and more. Proper encryption schemes and security features such as firewalls, TLS/SSL, IPsec, and asymmetric encryption, among others, ensures that data in motion is as secure as possible. |
| Encryption in use | Refers to data actively being modified, processed, or otherwise categorized as “in use.” Strategies such as applying authentication and authorization as a prerequisite for using such data minimizes security risks and assures only the parties allowed to use said data is doing so. Applying encryption prior-to and post-modification of such data also bolsters their security. |

Source:<https://www.strongdm.com/blog/aaa-security>

| 1. **Triple-A Framework\*** | **What it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Refers to verifying a user’s identity prior to allowing access to system components. Techniques such as 2-factor authentication, biometric authentication, strong password enforcement, one-time or single-use passwords, and even dedicated physical authentication tokens are used to properly authenticate users and secure systems. |
| Authorization | Allows for the specification of what a given user of a system is allowed access to. Can apply to system components, data, and functionality. Limits access based on principle of least privilege to secure system assets against unwarranted accesses to potentially sensitive data. Techniques such as role definitions, tiered accesses, and permission-management solutions ensure parties are only given access to data that is needed to complete a particular task – further securing the system. |
| Accounting | Tracks system activity for potential threats or lapses in security. Helps to gain insight as to when, where, and potentially why a vulnerability exists. Accounting also helps to identify if a vulnerability has been utilized to the system’s possible detriment. Strategies such as logging system activities, monitoring of data transfers, scanning data for potential risk, and applying timestamps where possible allow for close auditing of system security as well as traceability if a security event takes place. |

**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 |  |
| 1.1 | 07/20/2025 | Coding Standards | Walter Briones |  |
| 1.2 | 08/05/2025 | Risk Assessments,  Automated Detection,  Automation,  Summary of Risk Assessments  Mapping to Principles  Encryption Policies  Triple A\* Policies | Walter Briones |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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