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



# 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 | Validating input data is the process of making sure that input data is checked to ensure that it is expected and correct. This also involves sanitization of input data to eliminate any present SQL Injection or other threats. |
| 1. Heed Compiler Warnings | Warnings generated from the compiler address issues that will cause code to run in an unexpected or insecure way. These warnings typically do not indicate the code will fail to run, but they must be addressed in order to ensure proper and safe function. |
| 1. Architect and Design for Security Policies | Establishing a security policy is one of the first steps of development, and it has a direct relationship with the complexity of a project and its needs. A well established Security Policy will guarantee a plan is in place to address security concerns as they arise during development. |
| 1. Keep It Simple | Code design must be concise and follow the simplest possible design that is both functional and secure. Preventing overly complex code streamlines the code review process, enables easier security enhancement, and makes opportunities for exploits less likely. |
| 1. Default Deny | This represents denial of access privileges as a default state. Access privileges must be granted explicitly in order for a user to access program code or other vital information. This can limit the ability of an exploit to cause further damage by accessing additional parts of code. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege dictates that individuals should be granted only the minimum level of access necessary to perform their tasks. This approach effectively limits the scope of any potential issues to a more restricted area. |
| 1. Sanitize Data Sent to Other Systems | Prior to transmitting data to an external system, ensure that the data is both valid and formatted in accordance with the expected requirements of the receiving system. This practice reduces the likelihood of errors arising from inadequate input validation on the part of the receiving system. |
| 1. Practice Defense in Depth | Defense in depth refers to the strategy of implementing multiple layers of security measures within a system. This approach introduces redundancy, ensuring that if one layer fails, another layer remains in place to maintain the security of the system. |
| 1. Use Effective Quality Assurance Techniques | Employing comprehensive testing in conjunction with robust quality assurance methodologies ensures that potential issues are identified and resolved prior to exploitation. This approach significantly enhances the security of the program. |
| 1. Adopt a Secure Coding Standard | The implementation of a secure coding standard ensures the consistent delivery of well-secured code while adhering to established best practices. However, even the most robust security policies are ineffective if they are not adhered to; therefore, it is essential to establish enforceable requirements to guarantee compliance with these policies. |

### C/C++ Ten Coding Standards

Coding standards are sourced from [SEI CERT C++ Coding Standard](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)

#### Coding Standard 1

| **Coding Standard** | **Label** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Avoid cv-qualifying a reference type, as it leads to undefined behavior. While a conforming compiler must provide a diagnostic message, non-fatal diagnostics may result in unexpected outcomes, such as mutating the character referenced by p. |

| **Noncompliant Code** |
| --- |
| This noncompliant example properly declares as a reference to a const-qualified , but modifying renders the program ill-formed. |
| #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** |
| --- |
| This resolves the problem by removing the const qualifier. |
| #include <iostream>    void f(char c) {    char &p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Principles(s):** The principle that applies is #2 which states that Complier warnings that are shown in the IDE being used must always be heeded. |
| --- |

**Threat Level**

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

**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) | 2024.4 | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **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** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | A value-returning function must return a value along all code paths; failing to do so results in undefined behavior, including less-common paths like those in a function-try-block. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the programmer failed to return the input value for positive input, leaving some code paths without a return value. |
| int absolute\_value(int a) {    if (a < 0) {      return -a;    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, every code path ensures a return value. |
| int absolute\_value(int a) {    if (a < 0) {      return -a;    }    return a;  } |

| **Principles(s):** The principle that applies is #2Heed Complier warnings and #4Keep it Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | **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 |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.MRS LANG.STRUCT.NVNR** | Missing return statement Non-void noreturn, |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S935**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V591**](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-implicit** | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Range Check Element Access** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | The std::string index operators, const\_reference operator[](size\_type) const and reference operator[](size\_type), return the character at the specified position pos. If pos >= size(), they return a reference to a charT object with the value charT(). These operators are unchecked, meaning no exceptions are thrown for range errors, and modifying an out-of-range object leads to undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the value from get\_index() might exceed the string's element count, leading to undefined behavior. |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This compliant solution utilizes std::basic\_string::at(), functioning similarly to the index operator [], but it throws a std::out\_of\_range exception when pos >= size(). |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

| **Principles(s):** The principle that applies is #9 Use Effective Quality Assurance Techniques and #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++3162, C++3163, C++3164, C++3165** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | [String data sent to complex subsystems may include special characters that could trigger commands or actions, creating software vulnerabilities. Therefore, all string data must be sanitized to ensure it is harmless within its intended context. |

| **Noncompliant Code** |
| --- |
| Effective data sanitization demands familiarity with both the data being processed and the subsystem's functionalities. John Viega and Matt Messier illustrate this with an example where an application stores an email address in a buffer and uses the resulting string as a parameter in a system() call. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| To ensure security, all valid data should be accepted while potentially harmful data is rejected or sanitized. This can be challenging when valid characters or sequences also carry special meanings in the subsystem, potentially requiring data validation against a grammar. When no overlap exists, whitelisting can be applied to remove harmful characters.  The whitelisting method involves defining a set of acceptable characters and discarding any that do not meet the criteria. Typically, the valid input set is predictable, well-defined, and manageable in size. This compliant solution, inspired by Wietse Venema's tcp\_wrappers package, demonstrates the whitelisting approach. |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {    \*cp = '\_';  } |

| **Principles(s):** The principle that applies is #1 Validate Input Data and #4 Keep it Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.4 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | [STD-nnn-LLL] | The C programming language offers multiple functions for memory allocation, including std::malloc(), std::calloc(), and std::realloc(), which can also be used in C++ programs. However, it provides only one function to free allocated memory: std::free(). Refer to MEM31-C (free dynamically allocated memory when no longer needed) and MEM34-C (only free memory allocated dynamically) for specific rules on allocation and deallocation in C. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the local variable space is provided as the expression for the placement new operator. The resulting pointer is subsequently passed to ::operator delete(), causing undefined behavior because ::operator delete() tries to free memory that was not allocated by ::operator new(). |
| #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** |
| --- |
| This compliant solution eliminates the use of ::operator delete() and explicitly invokes s1's destructor, one of the rare cases where explicitly calling a destructor is justified. |
| #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):** The principle that applies is #8 Practice DID and #9 Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | **P18** | **L1** |

**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** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **ALLOC.DF ALLOC.TM ALLOC.LEAK** | Double free Type mismatch Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **CL.FFM.ASSIGN** **CL.FFM.COPY** **CL.FMM** **CL.SHALLOW.ASSIGN** **CL.SHALLOW.COPY** **FMM.MIGHT** **FMM.MUST** **FNH.MIGHT** **FNH.MUST** **FUM.GEN.MIGHT** **FUM.GEN.MUST** **UNINIT.CTOR.MIGHT** **UNINIT.CTOR.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MEM51-a** **CERT\_CPP-MEM51-b** **CERT\_CPP-MEM51-c** **CERT\_CPP-MEM51-d** | Use the same form in corresponding calls to new/malloc and delete/free Always provide empty brackets ([]) for delete when deallocating arrays Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor Properly deallocate dynamically allocated resources |
| [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++: MEM51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem51cpp.html) | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V515**](https://pvs-studio.com/en/docs/warnings/v515/), [**V554**](https://pvs-studio.com/en/docs/warnings/v554/), [**V611**](https://pvs-studio.com/en/docs/warnings/v611/), [**V701**](https://pvs-studio.com/en/docs/warnings/v701/), [**V748**](https://pvs-studio.com/en/docs/warnings/v748/), [**V773**](https://pvs-studio.com/en/docs/warnings/v773/), [**V1066**](https://pvs-studio.com/en/docs/warnings/v1066/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1232**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1232) |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Since assert() invokes abort(), cleanup functions registered via atexit() are not executed. To ensure proper cleanup following a failed assertion, runtime assertions should be replaced with static assertions when feasible (refer to DCL03-C for using static assertions on constant expressions). For assertions relying on runtime data, replace assert() with a runtime check that follows the adopted error-handling strategy (see ERR00-C for implementing a consistent and comprehensive error-handling policy). |

| **Noncompliant Code** |
| --- |
| This noncompliant example includes a function designed to execute cleanup tasks before the program terminates: |
| void cleanup(void) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {    if (atexit(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/      assert(/\* Something bad didn't happen \*/);      /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, assert() is replaced with an if statement that invokes exit() to guarantee the execution of proper termination routines. |
| void cleanup(void) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {    if (atexit(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/      if (/\* Something bad happened \*/) {      exit(EXIT\_FAILURE);    }      /\* ... \*/  } |

| **Principles(s):** The principle that applies is #3 Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | **P4** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **bad-function bad-macro-use** | Supported |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.36 | [**V2021**](https://pvs-studio.com/en/docs/warnings/v2021/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **bad-function bad-macro-use** | Supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When an exception is thrown, control passes to the nearest handler with a matching type. If no matching handler is found within the try block where the exception occurred, the search dynamically extends to surrounding try blocks in the same thread. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, neither f() nor main() handle exceptions thrown by throwing\_func(). As a result, with no matching handler available, std::terminate() is invoked. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, ensuring the stack is unwound to the main() function and enabling graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

| **Principles(s):** The principle that applies is #3 Architect and Design for Security Policies and #9 Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | **P4** | **L3** |

**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) | 9.0p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **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 |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not modify the standard namespaces** |
| --- | --- | --- |
| Namespace Control | [STD-008-CPP] | Namespaces create new declarative regions for declarations, minimizing identifier conflicts with other regions. A key feature of namespaces is their ability to be extended, even across separate translation units. For example, the following declarations are valid. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, x is declared within the std namespace, leading to undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| This compliant solution presumes that the programmer intended to place the declaration of x in a namespace to avoid conflicts with global identifiers. Rather than using the std namespace, the declaration is placed within a namespace that does not have a reserved name. |
| namespace nonstd {  int x;  } |

| **Principles(s):** The principle that applies is #3 Architect and Design for Security Policies, #4 Keep it Siple, and #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V1061**](https://pvs-studio.com/en/docs/warnings/v1061/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3470**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Guarantee that library functions do not overflow** |
| --- | --- | --- |
| Library Overflow | [STD-009-CPP] | Copying data into a container that lacks sufficient capacity leads to a buffer overflow. To avoid this issue, the amount of data copied should be limited based on the destination container's size, or ideally, the destination container should be ensured to be large enough to accommodate the data. |

| **Noncompliant Code** |
| --- |
| STL containers are prone to similar vulnerabilities as array data types. The std::copy() algorithm lacks built-in bounds checking, potentially causing buffer overflows. In this noncompliant example, a vector of integers is copied from src to dest using std::copy(). Since std::copy() does not automatically resize the dest vector, a buffer overflow occurs as soon as the first element is copied. |
| #include <algorithm>  #include <vector>    void f(const std::vector<int> &src) {    std::vector<int> dest;    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Compliant Code** |
| --- |
| To use std::copy() correctly, the destination container must be large enough to accommodate all elements being copied. This compliant solution ensures this by increasing the vector's capacity before performing the copy operation. |
| #include <algorithm>  #include <vector>  void f(const std::vector<int> &src) {    // Initialize dest with src.size() default-inserted elements    std::vector<int> dest(src.size());    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Principles(s):** The principle that applies is #1 Validate Input Data, #2 Heed Compiler Warnings, and #7 Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **invalid\_pointer\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **BADFUNC.BO.\* LANG.MEM.BO LANG.MEM.TBA** | A collection of warning classes that report uses of library functions prone to internal buffer overflows. Buffer Overrun Tainted Buffer Access |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.4 | **ITER.END.OUTPARAM.MIGHT ITER.END.OUTPARAM.MUST** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C++: CTR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr52cpp.html) | Checks for library functions overflowing sequence container (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Write constructor member initializers in the canonical order** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | The member initializer list in a class constructor enables members to be initialized with specified values and allows base class constructors to be invoked with particular arguments. However, the initialization order is predetermined and is unaffected by the order in which members appear in the initializer list. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the derived class D attempts to initialize the base class B1 using a value derived from the base class B2. However, since B1 is initialized prior to B2 based on the declaration order in the base class specifier list, the behavior becomes undefined. |
| class B1 {    int val;    public:    B1(int val) : val(val) {}  };    class B2 {    int otherVal;    public:    B2(int otherVal) : otherVal(otherVal) {}    int get\_other\_val() const { return otherVal; }  };    class D : B1, B2 {  public:    D(int a) : B2(a), B1(get\_other\_val()) {}  }; |

| **Compliant Code** |
| --- |
| This compliant solution avoids dependency on the initialization order of base classes by initializing both base classes with the same value from the constructor's parameter list. |
| class B1 {    int val;    public:    B1(int val) : val(val) {}  };    class B2 {    int otherVal;    public:    B2(int otherVal) : otherVal(otherVal) {}  };    class D : B1, B2 {  public:    D(int a) : B1(a), B2(a) {}  }; |

| **Principles(s):** The principle that applies is #2 Heed Compiler Warnings, and #3 Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | **P4** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **initializer-list-order** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP53** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreorder |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.INIT.OOMI** | Out of Order Member Initializers |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4053** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.OOP.CTOR.INIT\_ORDER** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **206 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-OOP53-a** | List members in an initialization list in the order in which they are declared |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: OOP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop53cpp.html) | Checks for members not initialized in canonical order (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **initializer-list-order** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3229**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3229) |  |

### Defense-in-Depth Illustration

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



### Automation

This illustration provides a visual representation of the practices used in automation of security protocols.



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.

The CI/CD pipeline will be enhanced with the implementation of automated tools to generate reports based on successful or failed tests. Static code analysis will also be completed in order to detect common errors or known exploitations. This will also detect vulnerabilities with dependencies.

After builds and tests have been completed, the CI/CD pipeline will be used to create an automated configuration and deployment of the Production ready version of the app. This represents the Transition and health check phase. Tools will also be implemented to perform penetration testing and optimization of security settings. Coding standards will be referenced when running these tests.

Tools will be used to automate the generation of console logs that provide the information necessary to monitor and detect any issues. Any detected issues will trigger the appropriate response and subsequent maintenance.

Maintenance will consist of the use of tools to perform automated checks based on specified issues detected in the monitor and detect phase. It will also consist of manual review by coding experts trained in the coding principles of this security policy. Any changes will be posted to the pipeline to notify all included developers and generate historical documentation.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | **P3** | **P3** |
| STD-002-CPP | Medium | Probable | Medium | **P8** | **L2** |
| STD-003-CPP | High | Unlikely | Medium | **P6** | **L2** |
| STD-004-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-005-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-006-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | High | Unlikely | Medium | **P6** | **L2** |
| STD-009-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-010-CPP | Medium | Unlikely | Medium | **P4** | **L3** |

### Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This policy represents encryption of data when it is in static storage and awaiting a call. This ensures that exploits that grant access to storage have an additional layer of encryption preventing use of illegally obtained data. |
| Encryption in flight | This policy represents data that is travelling between mediums. An example is data actively traveling from server-side applications to client-side applications. This data must also be encrypted so that exploits focused on monitoring and accessing data traveling between nodes is not useable. |
| Encryption in use | This form of encryption is focused on data actively being used by the client or server-side applications. This ensures that data is encrypted when in active use so that exploits focused on accessing the live application are unable to use the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy is focused on the implementation of validating the identity of users attempting to access the application. The most common form of this is login credentials. |
| Authorization | This policy ensures that the queries and calls made by a user are appropriate given the security clearance of the user. This is typically implemented as access levels that dictate the difference between standard users and users with elevated privileges like admins. |
| Accounting | This represents the implementation of logging protocols to ensure that a record of all queries and application actions are recorded. This creates a history which can be associated with every user that accesses the application. This provides developers with a way to research any unexpected events. |

## 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 | 03/20/2025 | Completed Core Security Principles as well as DID and partial Coding Standards | Benjamin Sturgeon | Benjamin Sturgeon |
| 1.2 | 04/13/2025 | Completed Remainder of Policy | Benjamin Sturgeon | Benjamin Sturgeon |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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