**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 11](#_Toc52464062)

[Coding Standard 5 13](#_Toc52464063)

[Coding Standard 6 16](#_Toc52464064)

[Coding Standard 7 18](#_Toc52464065)

[Coding Standard 8 20](#_Toc52464066)

[Coding Standard 9 22](#_Toc52464067)

[Coding Standard 10 24](#_Toc52464068)

[Defense-in-Depth Illustration 26](#_Toc52464069)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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.

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | All input sources must be identified, specified, and validated. The specifications for untrusted data must describe the valid limits, minimum and maximum values and lengths, valid content, initialization and reinitialization requirements, and encryption requirements for storage and transmission. Input should also be validated as soon as possible. Each type of input shall be encapsulated and defined as a class to facilitate all validation. |
| 1. Heed Compiler Warnings | All code will be compiled with the maximum number of compiler warnings turned on as is possible with that particular compiler. All code must be changed so that it compiles cleanly with the maximum number of warnings turned on. |
| 1. Architect and Design for Security Policies | All projects and code will be designed with these security policies in mind in order to reduce security breaches, speed up response times, improve operational efficiency, and comply with industry regulations. This applies to all code down to the unit/function level. |
| 1. Keep It Simple | The design of projects, in general, and security mechanisms, in particular, should be kept relatively small and simple in order to facilitate easy implementation and verification. Complex designs increase the probability that errors will be made. |
| 1. Default Deny | Access should be denied, by default, and only authorized and explicitly permitted activities should be allowed. No user, device, or network is to be inherently trusted. Whitelisting shall be implemented, wherever possible. |
| 1. Adhere to the Principle of Least Privilege | Every program and every user of the system, program, or process should operate using the least set of privileges necessary to complete the task at hand. If the processes, systems, and/or programs are not more privileged than an attacker, the probability that a vulnerability can be exploited increases dramatically. Defaulting to the least privilege possible for users increases the probability that vulnerabilities cannot be exploited. |
| 1. Sanitize Data Sent to Other Systems | All data, including logs, exceptions, and API calls, shall be sanitized to ensure that data conforms to security-related requirements before being transmitted across a trust boundary. Output sanitization is to be performed in addition to, not in place of, input validation. Special attention should be given to complex subsystems such as command shells, relational databases, and commercial off-the-shelf applications as their complexity can hide vulnerabilities that would otherwise be apparent in simpler systems. |
| 1. Practice Defense in Depth | We understand that one tool cannot solve any problem, and that individual security tools have holes in their protection. As such, we will use a philosophy of Defense in Depth in order to create overlapping protection so that the individual holes of protection from individual tools do not overlap, thereby providing superior protection. |
| 1. Use Effective Quality Assurance Techniques | We will use effective quality assurance techniques such as penetration testing, source code audits, static analysis/dependency checking, and application scanning in order to identify and eliminate vulnerabilities. These techniques will be utilized before major changes or revisions are sent to production. |
| 1. Adopt a Secure Coding Standard | We will develop and/or apply a secure coding standard, such as this one, for each target programming language and platform. |

### 

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Never qualify a reference type with `const` or `volatile`. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| #include <iostream>    void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the `const` qualifier. |
| #include <iostream>    void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

**Principles(s)**

| 3. Architect and Design for Security Policies  4. Keep It Simple  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL52** |  |
| Helix QAC | 2024.2 | **C++0014** |  |
| Klocwork | 2024.2 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2024a | CERT C++: DCL52-CPP | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| 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 | 4.10 | **S3708** |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not access the bits of an object representation that are not part of the object's value representation. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the complete object representation is accessed when comparing two objects of type S. Per the C++ Standard, [class], paragraph 13 [ISO/IEC 14882-2014], classes may be padded with data to ensure that they are properly aligned in memory. The contents of the padding and the amount of padding added is implementation-defined. This can lead to incorrect results when comparing the object representation of classes instead of the value representation, as the padding may assume different unspecified values for each object instance. |
| #include <cstring>    struct S {  unsigned char buffType;  int size;  };    void f(const S &s1, const S &s2) {  if (!std::memcmp(&s1, &s2, sizeof(S))) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, S overloads operator==() to perform a comparison of the value representation of the object. |
| struct S {  unsigned char buffType;  int size;    friend bool operator==(const S &lhs, const S &rhs) {  return lhs.buffType == rhs.buffType &&  lhs.size == rhs.size;  }  };    void f(const S &s1, const S &s2) {  if (s1 == s2) {  // ...  }  } |

**Principles(s)**

| 3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **invalid\_pointer\_dereference** **uninitialized\_variable\_use** |  |
| CodeSonar | 8.1p0 | **BADFUNC.MEMCMP**  **BADFUNC.MEMSET** | Use of memcmp  Use of memset |
| Helix QAC | 2024.2 | **DF4726, DF4727, DF4728, DF4729, DF4731, DF4732, DF4733, DF4734** |  |
| Klocwork | 2024.2 | **CERT.MEMCMP.PADDED\_DATA** **CWARN.MEM.NONPOD** |  |
| LDRA tool suite | 9.7.1 | **618 S** | Partially implemented |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-EXP62-a** | Do not compare objects of a class that may contain padding bits with C standard library functions |
| Polyspace Bug Finder | R2024a | CERT C++: EXP62-CPP | Checks for access attempts on padding and vtable bits (rule fully covered). |
| PVS-Studio | 7.33 | V598, **V780, V1084** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the unformatted input function std::basic\_istream<T>::read() is used to read an unformatted character array of 32 characters from the given file. However, the read() function does not guarantee that the string will be null terminated, so the subsequent call of the std::string constructor results in undefined behavior if the character array does not contain a null terminator. |
| #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** |
| --- |
| This compliant solution assumes that the input from the file is at most 32 characters. Instead of inserting a null terminator, it constructs the std::string object based on the number of characters read from the input stream. If the size of the input is uncertain, it is better to use std::basic\_istream<T>::readsome() or a formatted input function, depending on need. |
| #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());  // ...  } |

**Principles(s)**

| 1. ValidateInput Data  2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| CodeSonar | 8.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| Helix QAC | 2024.2 | **C++5216**  **DF2835, DF2836, DF2839** |  |
| Klocwork | 2024.2 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| LDRA tool suite | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| Parasoft C/C++test | 2023.1 | **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 | R2024a | CERT C++: STR50-CPP | 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 | 22.10 | **stream-input-char-array** | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | **S3519** |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevent SQL injection with parameterized queries. |

| **Noncompliant Code** |
| --- |
| If an attacker enters “‘ OR ‘1’=’1” as the username in the following query, the query transforms into: |
| SELECT \* FROM users WHERE username = '' OR '1' = '1' AND password = 'password'; |

| **Compliant Code** |
| --- |
| Separate Parameters: Pass the actual user input as separate parameters alongside the SQL query: |
| std::string username = "ajay";  std::string password = "Whdu83uK";  // Assuming a connection object 'conn' is established  sql::PreparedStatement\* stmt = conn->prepareStatement(query);  stmt->setString(1, username);  stmt->setString(2, password); |

**Principles(s)**

| 1. ValidateInput Data  3. Architect and Design for Security Policies  7. Sanitize Data Sent to Other Systems  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| CodeSonar | 8.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| Coverity | 7.5 | **SQLI** **FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| Findbugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| Fortify | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| Klocwork | 2024.2 | **SV.DATA.DB** **SV.SQL** **SV.SQL.DBSOURCE** | Implemented |
| Parasoft Jtest | 2024.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| SonarQube | 9.9 | **S2077**  **S3649** | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |
| SpotBugs | 4.6.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** **SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |
| CodeSonar | 8.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Properly deallocate dynamically allocated resources. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in undefined behavior due to ::operator delete() attempting to free memory that was not returned 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 removes the call to ::operator delete(), instead explicitly calling s1's destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #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)**

| 2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **invalid\_dynamic\_memory\_allocation** **dangling\_pointer\_use** |  |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MEM51** |  |
| 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 | 8.1p0 | **ALLOC.FNH** **ALLOC.DF** **ALLOC.TM** **ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |
| Helix QAC | 2024.2 | **C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264** |  |
| Klocwork | 2024.2 | **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 | 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 | 2023.1 | **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++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024a | CERT C++: MEM51-CPP | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |
| PVS-Studio | 7.33 | **V515, V554, V611, V701, V748, V773, V1066** |  |
| SonarQube C/C++ Plugin | 4.10 | **S1232** |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)); |

| **Compliant Code** |
| --- |
| This portable compliant solution uses static\_assert: |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  "Structure must not have any padding"); |

**Principles(s)**

| 3. Architect and Design for Security Policies  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-DCL03** |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| Compass/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 | 1.2 | **CC2.DCL03** | Fully implemented |
| LDRA tool suite | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| 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, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

**Principles(s)**

| 2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR51** |  |
| CodeSonar | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| Helix QAC | 2024.2 | **C++4035, C++4036, C++4037** |  |
| Klocwork | 2024.2 | **MISRA.CATCH.ALL** |  |
| LDRA tool suite | 9.7.1 | **527 S** | Partially implemented |
| Parasoft C/C++test | 2023.1 | **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 | R2024a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Random Numbers** | STD-008-CPP | Do not use std::rand() for generating pseudorandom numbers. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #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** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example. |
| #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));  // ...  } |

**Principles(s)**

| 3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MSC50** |  |
| Clang | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 8.1p0 | **BADFUNC.RANDOM.RAND** | Use of rand |
| Compass/ROSE |  |  |  |
| ECLAIR | 1.2 | **CC2.MSC30** | Fully implemented |
| Helix QAC | 2024.2 | **C++5028** |  |
| Klocwork | 2024.2 | **CERT.MSC.STD\_RAND\_CALL** |  |
| LDRA tool suite | 9.7.1 | **44 S** | Enhanced Enforcement |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| Polyspace Bug Finder | R2024a | CERT C++: MSC50-CPP | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |
| RuleChecker | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | STD-009-CPP | Use valid iterator ranges. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the two iterators that delimit the range point into the same container, but the first iterator does not precede the second. On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

**Principles(s)**

| 2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **overflow\_upon\_dereference** |  |
| CodeSonar | 8.1p0 | **LANG.MEM.BO** | Buffer Overrun |
| Helix QAC | 2024.2 | **C++3802** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| Polyspace Bug Finder | R2024a | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially covered). |
| PVS-Studio | 7.33 | **V539, V662, V789** |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **[[noreturn]] Functions** | STD-010-CPP | Do not return from a function declared [[noreturn]]. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, if the value 0 is passed, control will flow off the end of the function, resulting in an implicit return and undefined behavior. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function does not return on any code path. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  std::exit(0);  } |

**Principles(s)**

| 2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | **P2** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **invalid-noreturn** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MSC53** |  |
| Clang | 3.9 | -Winvalid-noreturn |  |
| CodeSonar | 8.1p0 | **LANG.STRUCT.RFNR** | Return from noreturn |
| Helix QAC | 2024.2 | **DF2886** |  |
| Klocwork | 2024.2 | **CERT.MSC.NORETURN\_FUNC\_RETURNS** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MSC53-a** | Never return from functions that should not return |
| Polyspace Bug Finder | R2024a | CERT C++: MSC53-CPP | Checks for [[noreturn]] functions returning to caller (rule fully covered) |
| PVS-Studio | 7.33 | **V1082** |  |
| RuleChecker | 22.10 | **invalid-noreturn** | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | **S935** |  |

### Defense-in-Depth Illustration

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





Generally speaking, the automation that will be used for the enforcement of and compliance to the standards developed in this policy should be implemented as soon as possible in the Software Development Life Cycle, which is also known as a “shift left” policy. Tests should actually be written first with a Test-Driven Development pattern, where possible, to minimize the costs and technical debt of delayed response.

Unit testing and static testing should be done immediately before integrating individual functions or units into the code base. Testing and monitoring for vulnerabilities and violations of these standards should be ongoing and at every stage of the DevSecOps cycle.

### 

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-002-CPP | High | Probable | High | **P6** | **L2** |
| STD-003-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-004-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-005-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-006-CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | Medium | Unlikely | Low | **P6** | **L2** |
| STD-009-CPP | High | Probable | High | **P6** | **L2** |
| STD-010-CPP | Medium | Unlikely | Low | **P2** | **L3** |

### Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Data at rest is data that persists when there is no power applied to the system. This includes data in flash drives, disk drives, and other physical media. The encryption policy applies to this form of data to protect it in the case of physical theft or loss of a medium of storage. |
| Encryption in flight | Data in flight is data that is being transmitted and received over a network, whether wired or wireless. Encryption of data in this stage protects the data from being intercepted through, for example, man-in-the-middle attacks. |
| Encryption in use | Data in use is data that is currently being accessed by authorized and authenticated users. By using file permissions and encryption, data security can be enhanced by ensuring that only those users who are supposed to access data end up using it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is making sure users are actually who they say they are. Through the use of strong passwords and MFA, the policy can ensure that user permissions are enforced. |
| Authorization | Authorization is ensuring that an authenticated user has the permissions required to access files, data, and networks. Through permissions management and the principle of least privilege, the policy can mitigate attacks based on the escalation of privileges. |
| Accounting | Accounting is keeping track of who accessed what. By logging all access, auditing and responding to attacks can be done and be done quickly to mitigate losses in the event of an attack or breach. |

## 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 | 09/22/2024 | Add Coding Standards | David Faulkner |  |
| 3.0 | 10/13/2024 | First Completed Policy | David Faulkner |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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