**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.

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Whenever an external actor is able to provide their own input to our program, there is an opportunity to exploit our software through common vulnerabilities such as SQL injection, overflow, underflow, or otherwise escaping the software instruction to access private functions or execute arbitrary code. Layered input validation techniques can act to prevent these exploits and defend against most attackers. |
| 1. Heed Compiler Warnings | Compiler warnings represent bugs, vulnerabilities, maintainability issues, and underlying codebase issues that could compound or impact product quality in the long term. Our mission should be to produce software with no errors or warnings, so our customers receive robust programs that are void of obvious flaws or errors. |
| 1. Architect and Design for Security Policies | Security policies are to be guided by awareness of the system architecture and designed opportunistically with awareness of design and architecture goals. Security policy is written in a way that promotes productivity and growth of the design and architecture securely, rather than inhibiting features. |
| 1. Keep It Simple | Keep software systems straightforward and concise. Overly complex software can cause additional vulnerabilities, bugs, misconfigurations, and errors. |
| 1. Default Deny | Web traffic or server calls made without an explicit entry point or policy to handle the call should be denied automatically, avoiding unwanted traffic on private endpoints. This is achieved by using deny as the default policy. |
| 1. Adhere to the Principle of Least Privilege | Users and processes receive the minimum required level of privilege, limiting effects of compromised credentials or the damage caused by a breach in security. Where elevated privileges are necessary, there should be a timer, for both processes and users. |
| 1. Sanitize Data Sent to Other Systems | If a breach or vulnerability is discovered in our software, or our software is fed malicious input by an end user, attacker, or vulnerable dependency, we should have systems to defend the end user or other systems which we interface with. This also includes sanitizing function calls to shell or subsystem commands, such as ensuring calls are not made or injected to delete files or database records, retrieve them, or alter them. |
| 1. Practice Defense in Depth | Defense in depth is a methodology of layering security practices to build a robust defense. We should anticipate failures in our security and so create overlapping methods of keeping our program secure. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance should be a combination of techniques, including pair programming, automated tests and asserts, internal pre-release bug testing, and public ticket submitting after launch. Quality assurance should be a collaborative effort from design to implementation and after deployment, and an on-going process. |
| 1. Adopt a Secure Coding Standard | Developers should adhere to industry standard practices, and be mindful of resources such as Microsoft’s best practices for C++. It is strongly encouraged for all Green Pace developers to review, adopt, and comply with all coding standards mentioned herein to improve conformity to company best practice as well as sustain a high quality of product for our users. |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Fixed Width Ints  Selecting data types with explicit widths prevents underflow and overflow errors that may not be typically expected. With fixed width ints, the subjective runtime and environment values of size(int) are avoided, as an int width could be 2-4 bytes. |

| **Noncompliant Code** |
| --- |
| Usual int is used, causing possible overflow errors on systems where ints are stored in 2 byte width, i.e. 32x instruction sets. |
| int i = 65535 \* 2; // 11111111 11111111 11111111 11111111 = 4 bytes of data; Overflow on some platforms |

| **Compliant Code** |
| --- |
| Fixed width 32-bit int is used, preventing an overflow from automatic width int. |
| int32\_t i = 65535 \* 2; // All four bytes are guaranteed to fit within the width. |

| **Principles(s):** Our data type standard conforms to the principle of 4) Keep It Simple. Selection of appropriate data types allows strictly limiting the array of potential values, keeping data expectations simple. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | 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) | 2025.2 | 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 container index conforms to a valid range. |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Array access out of bounds, array access tainted index, point dereference with tainted offset. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Limit and Bounds Checking  Values stored, and values iteratively modified, as well as values when interfacing with arrays run the risk of out of range enumerations and overflows / underflows. As such, code should be arranged in a manner that avoids out of bounds or excess of data type limits. |

| **Noncompliant Code** |
| --- |
| Function input is not validated to enforce valid bounds. |
| index >> cin ; // Suppose a function or the user inputs 4.  vector<int> menu = { 0, 1, 2, 3};  return menu[index]; // No bounds check. |

| **Compliant Code** |
| --- |
| Function input is validated before used for function. |
| int index;  vector<int> menu = { 0, 1, 2, 3};  int value;  assert( (menu.size() – 1) > 0);  try {  value = menu.at(index);  }  catch (…) {  cerr << “Error while receiving menu index.\n”;  } |

| **Principles(s):** This coding standard complies with 1) Validate In put Data. Validating that a function receives parameters within the expected range avoids common and preventable out of bounds errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | array-index-range  array-index-range-constant  null-dereferencing  pointered-deallocation  return-reference-local | Detects accesses to invalid pointers as well s out of bounds accesses. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  BADFUNC.BO.\* | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun Pointer Arithmetic Pointer before beginning of object Pointer past end of object A collection of warning classes that report uses of library functions prone to internal buffer overflows. |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.15 | arrayIndexOutOfBounds, outOfBounds, negativeIndex, arrayIndexThenCheck, arrayIndexOutOfBoundsCond, possibleBufferAccessOutOfBounds |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2025b | [CERT C: Rule ARR30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulearr30c.html) | Checks for array access out of bounds, pointer access out of bounds, array access tainted index, tainted pointer, pointer dereferenced with tainted offset. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Use Std. C++ String Containers  It can be tempting to use solutions such as character arrays or other string replacements, but these do not automatically store the correct bit width for whatever the string is, unlike the standard library string containers. |

| **Noncompliant Code** |
| --- |
| A character array is used rather than a string, which means input size could be provided that is incorrect, creating the risk of a buffer overflow error. |
| char cstring[10];  cin >> cstring; // >> abcdefghijk <- Too many chars! |

| **Compliant Code** |
| --- |
| A std::string is used, automatically allocating the appropriate bit width to safely accommodate input. |
| string input;  cin >> input; // Overhead is sizeof(string), up to 32 bytes but varies by platform. |

| **Principles(s):** This standard relates to 1) Validate Input Data in that potential overflow and underflow errors are avoided by using automatic buffer allocation to avoid invalidated input data in cases where a variable string length is desirable, i.e. long usernames or passwords. This standard also relates to 10) Adopt a Secure Coding Standard in that it allows conformity to industry standard practices by taking advantage of current C++ standard library capabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported indirectly via MISRA C:2004 rule 6.1 and MISRA C:2012 rule 10.1. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | MISC.NEGCHAR | Negative Character Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2025.2 | C0634, C0635, C1292, C1293, C1810, C1811, C1812, C1813, C1814, C2151, C4010, C4011, C4063, C4064, C4065, C4310, C4312, C4315, C4401, C4410, C4412, C4413, C4414, C4415, C4421, C4431, C4441, C4451, C4510, C4511, C4512, C4513, C4514, C4517, C4518, C4519, C4580, C4581, C4582, C4583, C4584, C4585, C4586, DF2806, DF2807, DF2808, DF2816, DF2817, DF2818 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | CERT\_C-STR00-a | The plain char type shall be used only for the storage and use of character values |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Use Parameterized Queries for SQL Interfacing.  Avoid SQL injection by disallowing the opportunity to add a string as part of an SQL call command. This is done by forcing the input strings of a function into literals for use in parameterized queries. |

| **Noncompliant Code** |
| --- |
| User input strings from form Username and Password are used for an SQL query, allowing injection. |
| cin >> username; // ‘ OR ‘1’=’1’”; --  cin >> password;  string log\_in\_sql = “SELECT \* FROM users WHERE id = ‘” + user\_id + “’ AND password = ‘” + password + “’”; |

| **Compliant Code** |
| --- |
| User input is fitted into a prepared parameterized template. |
| cin >> username; // ‘ OR ‘1’=’1’”; --  cin >> password;  sqlite3\_stmt\* log\_in\_stmt = “SELECT \* FROM users WHERE id = ? AND password = ?”;  sqlite3\_bind(log\_in\_stmt, username, password); |

| **Principles(s):** This standard correlates to the principles of 1) Validate Input Data and 7) Sanitize Data Sent to Other Systems. Parameterized queries treat the input as data rather than executable SQL code, validating that it cannot be arbitrarily run during an injection attack. SQL queries are also foundationally made to an external database system and parameterization allows sanitizing of the data before the database receives and attempts to execute it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/java/CodeSonar) | 9.0p0 | JAVA.IO.INJ.SQL | SQL injection |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | SQL Injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | SQL Injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do Not Access Freed Memory  Attempting to access a dereferenced pointer can cause a program vulnerability allowing arbitrary code execution. As such, code should ensure that expected data which could be deleted / freed is not absent before attempting to access a pointer. |

| **Noncompliant Code** |
| --- |
| A pointer is accessed while deleted. |
| struct Item {  void f();  };    void main {  Item \*i = new Item;  delete i;  i->f(); // Invalid  } |

| **Compliant Code** |
| --- |
| A pointer is accessed prior to deletion, avoiding the error. |
| struct Item {  void f();  };    void main {  Item \*i = new Item;  i->f();  delete i;  } |

| **Principles(s):** This conforms to 2) Heed Compiler Warnings and 9) Use Effective Quality Assurance Techniques. Accessing freed memory may cause traceable bug crashes, but moreover is due to memory mismanagement, which compilers are typically able to detect and throw warnings for. Avoiding accessing freed memory is a matter of ensuring high quality of code and preventing undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | ALLOC.UAF | Use after free |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.15 | doubleFree  deallocret  deallocuse |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2025.2 | UFM.DEREF.MIGHT  UFM.DEREF.MUST  UFM.FFM.MIGHT  UFM.FFM.MUST  UFM.RETURN.MIGHT  UFM.RETURN.MUST  UFM.USE.MIGHT  UFM.USE.MUST |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use Assertions to Check for Programmer Errors  Use an assert only to validate a specific assumption within the program, such as that data conforms to its expected values. |

| **Noncompliant Code** |
| --- |
| A value is retrieved and makes an unhandled assumption the pointer will never be null, which leads to a crash. |
| char\* data = retrieve(file.txt);  return data[0]; // If data has no members, this crashes |

| **Compliant Code** |
| --- |
| An assert verifies the contents of the data are present. |
| char\* data = retrieve(file.txt);  assert(data != nullptr && “Data cannot be null”);  return data[0]; |

| **Principles(s):** This standard complies with 2) Heed Compiler Warnings. It gives additional opportunities for the compiler to throw errors based on programmer and logical errors, ensuring robustness and that the program operates as expected. This also complies with 9) Use Effective Quality Assurance Techniques. Assertions are a technique to bake static checks into the program and perform automated tests. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the 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/c/Parasoft) | 2024.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/c/Security+Reviewer+-+Static+Reviewer) | 6.02 | CPPPBE | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle All Exceptions  All cases that can be anticipated should be handled via try catch exception branches. |

| **Noncompliant Code** |
| --- |
| A generic error throwing function is called via main and not handled, leading to a crash when the error throwing function occurs. |
| void error();  void main() {  error();  } // Program crashes |

| **Compliant Code** |
| --- |
| The error is handled by try catch, allowing the error to be dealt with. |
| void error();  void main() {  try {  error()  } catch (…) { // The error is gracefully caught and handled.  // Handle the error  }  } |

| **Principles(s):** This standard complies with principles 2) Heed Compiler Warnings, 8) Practice Defense in Depth, and 9) Use Effective Quality Assurance Techniques. Compiler warnings can offer insight into the exceptions that can be expected to occur. Wrapping try and catch allows an additional layer of defense against anticipated exceptions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Medium | 2 |

**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 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.STRUCT.UCTCH  PARSE.MBDH | Masked by handler Masked by default handler |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Management | STD-008-CPP | Do Not Leak Resources While Handling Exceptions  An exception may break the opportunity for cleanup code to execute. If an error occurs during object construction or if the object is in an incomplete state, an exception should be thrown. |

| **Noncompliant Code** |
| --- |
| Value is not properly released when process\_item throws an exception. |
| struct Structure {  Structure() noexcept;  ~Structure();  void process\_item() noexcept(false);  };  void main() {  Structure \*value = new (nothrow) Structure();  if (!value) {  // Handle error  }  return;  } |

| **Compliant Code** |
| --- |
| Value is deleted to deallocate the storage used whenever an exception is thrown on construction. |
| struct Structure {  Structure() noexcept;  ~Structure();  void process\_item() noexcept(false);  };  void main() {  Structure \*value = new (nothrow) Structure();  if (!value) {  // Handle error  return  }  try {  value->process\_item();  } catch (…) {  delete value;  throw;  return;  } |

| **Principles(s):** This standard complies with 8) Defense in Depth. This is a standard to add an additional layer of protection in object construction and resource allocation, by allowing an error and deconstruction of a failed initialized object. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | ALLOC.LEAK | Leak |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | CL.MLK  MLK.MIGHT  MLK.MUST  MLK.RET.MIGHT  MLK.RET.MUST  RH.LEAK |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: ERR57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr57cpp.html) | Checks for:  Resource leak caused by exception  Object left in partially initialized state  Bad allocation in constructor |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Random Numbers | STD-009-CPP | Do Not Use rand() for Generating Random Numbers  Rand() is a predictable generator with limited randomness. |

| **Noncompliant Code** |
| --- |
| Random is used for user ID generation, causing predictable number generation. |
| string id(“ID”);  id += to\_string(rand() % 10000); |

| **Compliant Code** |
| --- |
| C++’s random numbers can be a function of an engine (the algorithm used) as well as the distribution, producing better control over random and making rand functions less predictable and limited. |
| string id(“ID”);  uniform\_int\_distribution<int> distribution(0,10000);  random\_device rd;  mt19937 engine(rd()); // Mersenne Twister random engine in std  id += to\_string(distribution(engine)); |

| **Principles(s):** This standard conforms to 10) Adopt a Secure Coding Standard in that it encourages best C++ practices and a lack of reliance on unreliable functions. Rand() is obsolete and predictable, and is not a good toolk for cryptography. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | stdlib-use-rand |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | BADFUNC.RANDOM.RAND | Use of rand |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | CERT\_C-MSC30-a | Do not use the rand() function for generating pseudorandom numbers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2025b | [CERT C: Rule MSC30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemsc30c.html) | Checks for vulnerable pseudo-random number generator (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Wait Function Wrapping | STD-010-CPP | Wrap Functions that can Wake in Loops  Using while(condition) rather than if(condition) to wrap a function that is lying to wake i.e. wait will allow the condition to be met to be captured responsively. |

| **Noncompliant Code** |
| --- |
| A wait function is wrapped by an if statement, meaning the condition is checked once and potentially missed or not utilized while valid. |
| struct Node {  void \*node;  struct Node \*next;  };  static Node list;  static mutex m;  static condition\_variable condition;  void read\_list\_elements(condition\_variable &condition) {  unique\_lock<mutex> lock(m);  if (list.next == nullptr) {  condition.wait(lock);  }  } // Code proceeds even if condition is still holding. |

| **Compliant Code** |
| --- |
| A while loop guarantees the condition can be met indefinitely, and until it is resolved, this function will lie in wait. |
| struct Node {  void \*node;  struct Node \*next;  };  static Node list;  static mutex m;  static condition\_variable condition;  void read\_list\_elements(condition\_variable &condition) {  unique\_lock<mutex> lock(m);  while (list.next == nullptr) {  condition.wait(lock);  }  } // Code waits until the condition is no longer met to proceed. |

| **Principles(s):** This complies with 4) Keep it Simple. Using the while condition allows a clear path of logic within a multithreading application and prevents condition skipping. This also complies with Use Effective Quality Assurance Techniques as it is correct practice for the intended functionality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Unlikely | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.STRUCT.ICOL  CONCURRENCY.BADFUNC.CNDWAIT | Inappropriate Call Outside Loop Use of Condition Variable Wait |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | CERT.CONC.WAKE\_IN\_LOOP |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-CON54-a | Wrap functions that can spuriously wake up in a loop |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: CON54-CPP](https://www.mathworks.com/help/bugfinder/ref/certccon54cpp.html) | Checks for situations where functions that can spuriously wake up are not wrapped in loop |

### Defense-in-Depth Illustration

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





A suite of automation tools and techniques are incorporated into the Green Pace DevOps process and infrastructure. This is broken down into pre-production and production settings, and implemented throughout eight phases. During the assessment and planning phase, the basic functions, data types, and vulnerabilities are anticipated, and the threat landscape is understood. At this point, code check plugins should be integrated into the IDE and static code check tools such as CPPCheck should be added to the developers’ environments. During the design phase, automated tests should drive design decisions and best practices, i.e. using tools such as OWASP to determine the safest dependency versions. During the build phase, secure code structures, development architectures, build tools, and repositories are chosen or established. In the last leg of the pre-production phase, automated tools such as CPPCheck are ran to scan the program for vulnerabilities, as well as automated security scan such as Coverity are ran. Dynamic application security testing tools are also ran, finding runtime vulnerabilities such as SQL injection or other errors which static analysis may not be capable of detecting.

Throughout the production phase automated tooling is also implemented. During the transition from the pre-production to the production phase, penetration tests are ran and log collection and analytics tools are implemented for preparation for release. During the monitor and detect phase, the log collection, SIEM, analytics, event alerting, and intrusion protection tools are used to automatically detect attacks or vulnerability exploits and promote these vulnerabilities to developers. This leads to the respond phase, in which developers counter attackers by deploying patches and fixes, blocking vulnerabilities and turning off unnecessary or vulnerable services. The maintain and stabilize phase involves continuous assessment and maintaining a stable and competently secure application during the life cycle, and allowing a strong foundation upon which the next phase of development could be assessed and planned.

### Summary of Risk Assessments

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

### Create Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This is the encryption of stored data, in which all sensitive data should be encrypted using protocols such as AES-256 to protect data from unauthorized attackers who could, for instance, steal the physical disc upon which the data is held. This policy is applied to prevent compromised hardware from exposing the data it holds. |
| Encryption in flight | This is encryption while data is being transmitted between systems, i.e. net traffic. This mandates the use of secure protocols such as TLS handshake / key encryption, and prevents connection eavesdropping like that which would be possible if the user was on a public WiFi network where there could be a snooper or data scraper. |
| Encryption in use | This is encryption while data is being processed by the local system. This involves using encryption methods to prevent data readers, worms, or peeping software from accessing sensitive information. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication guarantees the authenticity of a user or process’s identity. This may include requiring MFA / 2FA multifactor authentication for all employee or developer logins, and requires the use of credentials that meet certain strength requirements. This is required to work on and access Green Pace databases or codebases. This applies whenever a user login occurs, or any time a system process is initialized. |
| Authorization | Authorization is complying the authenticated user to a limited set of permissions, enforcing principal of least privilege. Accounts and microservices are subject to role based access control, with read/write/folder access restrictions. Changes to the database are restricted to administrators with the appropriate permissions, and new users upon initialization are not trusted with the permissions to access any of these features. This applies whenever there is an attempt to access any resource within our system. |
| Accounting | Accounting refers to logging and auditing user activities within the system, such as data usage, downloads from the database, user logins (and their attempts), changes to the database, and addition of new users and permission changes. This also includes the operating system, firewall, traffic, and antimalware logs, and all of these logs are to be immutable and securely stored for analysis and forensics by the security team. This applies whenever a security-relevant event occurs, such as data access or modification. |

## 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 | 10/10/2025 | Completed the coding standards, automation guidance, risk summary, triple-A, and Encryption policies. | Justin Phillips | Justin Phillips |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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