**Green Pace Developer: Security Policy: Kendall Ferebeee**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## 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 | Ensure all input is validated to prevent injection attacks, buffer overflows, and data corruption. By thoroughly checking data before processing, you reduce vulnerabilities. |
| 1. Heed Compiler Warnings | Address compiler warnings to prevent potential runtime errors and undefined behavior. Ignoring these warnings can lead to exploitable bugs. |
| 1. Architect and Design for Security Policies | Incorporate security principles during the design phase to build robust systems. Secure design minimizes risks from evolving threats. |
| 1. Keep It Simple | Simplify code and architectures to make them easier to audit and less prone to bugs. Complexity increases the likelihood of vulnerabilities. |
| 1. Default Deny | Set default permissions to deny access. Only grant access explicitly as needed to minimize the attack surface. |
| 1. Adhere to the Principle of Least Privilege | Grant users and systems the minimum permissions required to perform their tasks. This reduces the potential for misuse or exploitation. |
| 1. Sanitize Data Sent to Other Systems | Cleanse data before transferring it to external systems to avoid spreading vulnerabilities or enabling exploitation. |
| 1. Practice Defense in Depth | Implement multiple layers of security to protect systems even if one layer fails. This approach mitigates the impact of individual breaches. Test early and test often, define a process, and communicate rules and requirements. |
| 1. Use Effective Quality Assurance Techniques | Regularly test and review code for security vulnerabilities using automated tools and manual audits to ensure adherence to standards. |
| 1. Adopt a Secure Coding Standard | Follow established secure coding practices, such as the SEI CERT C++ Coding Standard, to minimize risks and ensure consistent implementation of security principles. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Obey the one-definition rule** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Use of a header file to introduce the object into both translation units. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):**  **3: Architect and Design for Security Policies**  **4: Keep It Simple**  **10: Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not rely on the value of a moved-from object** |
| --- | --- | --- |
| **Data Value** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| Dereferenced after it has been deallocated. If this access results in a write-after-free, this can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  }  }  } |

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

| **Principles(s):**  **2: Heed Compiler Warnings**  **5: Default Deny**  **6: Adhere to the Principle of Least Privilege**  **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** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ | - | - | Runtime detection |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to modify string literals** |
| --- | --- | --- |
| **String Correctness** | [STR-030-CPP] | At compile time, string literals are used to create an array of static storage duration of sufficient length to contain the character sequence and a terminating null character. The behavior is undefined if a program attempts to modify any portion of a string literal. Modifying a string literal frequently results in an access violation because string literals are typically stored in read-only memory.check element access |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior: |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. (See STR11-C. Do not specify the bound of a character array initialized with a string literal.) This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | string-literal-modfication  write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Compass/Rose | - | - | Can detect simple violations of this rule |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to “char” |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| Without precautions, the untrusted data may maliciously alter the query. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):**  **1: Validate Input Data**  **7: Sanitize Data Sent to Other Systems**  **10: Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [MEM-050-CPP] | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment into memory that has been deallocated by a memory management function is undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Dangling\_pointer\_use | - |
| Axivion Bauhuas Suite | 7.2.0 | CertC++-MEM50 | - |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| Klocwork | 2024.3 | 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** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use a static assertion to test the value of a constant expression |

| **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** |
| --- |
| For constant expressions, a preprocessor conditional statement may be used. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):**  **2: Heed Compiler Warnings**  **10: Adopt a Secure Coding Standard** |
| --- |

**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 |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [ERR-051-CPP] | All exceptions thrown must be caught by a matching exception handler, or the stack may not unwind correctly due to std::abort() being called so destructors may not be called.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  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| CodeSonar | 8.1po | LANG.STRUCT.UCTCH | Unreachable Catch |
| 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 | 22.10 | main-function-catch-all  early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not slice derived objects** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CPP] | Write constructor member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| The member initializer list for C::C() attempts to initialize someVal first and then to initialize dependsOnSomeVal to a value dependent on someVal. Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal. |
| class C {  **int** dependsOnSomeVal;  **int** someVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
| class C {  **int** someVal;  **int** dependsOnSomeVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):**  **4: Keep It Simple** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | initializer-list-order | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 | - |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Use valid iterator ranges** |
| --- | --- | --- |
| Containers | [CTR-053-CPP] | When iterating over elements of a container, the iterators used must iterate over a valid range. An iterator range is a pair of iterators that refer to the first and past-the-end elements of the range respectively. |

| **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #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; });  } |

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

| **Principles(s):**  **3: Architect and Design for Security Policies**  **4: Keep It Simple**  **10: Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference | - |
| Helix QAC | 2021.2 | C++3802 | - |
| Parasoft C/C++test | 2021.1 | 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 |
| PVS-Studio | 7.14 | V539, V662, V789 | - |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Do not access an object outside of its lifetime |

| **Noncompliant Code** |
| --- |
| A pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in undefined behavior. |
| struct S {  void mem\_fn();  };    void f() {  S \*s;  s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| Storage is obtained for the pointer prior to calling S::mem\_fn(). |
| struct S {  void mem\_fn();  };    void f() {  S \*s = new S;  s->mem\_fn();  delete s; |

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

| **Principles(s):**  **2: Heed Compiler Warnings**  **10: Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | return-reference-local  dangling\_pointer\_use | Partially checked |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 6.1p0 | IO.UAC  ALLOC.UAF | Use after close  Use after free |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-EXP54-a  CERT\_CPP-EXP54-b  CERT\_CPP-EXP54-c | Do not use resources that have been freed  The address of an object with automatic storage shall not be returned from a function  The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STR-030-CPP | Low | Likely | Low | Medium | 2 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| MEM-050-CPP | High | Likely | Medium | High | 1 |
| STD-006-CLG | Low | Unlikely | High | Low | 3 |
| ERR-051-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-010-CPP | High | Probable | High | Medium | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest safeguards data stored on various devices such as hard drives, smartphones, computers, and cloud platforms. This protection can be achieved using encryption software, disk encryption, and security measures for mobile devices and computers. |
| Encryption in flight | Encryption at flight focuses on securing data while it is being transmitted. This can occur between devices within a network or when data leaves the network. Protection methods include email encryption, data loss prevention (DLP) solutions, and strong network security measures like firewalls and authentication. It's also crucial to assess the data's route and ensure the security of the path it takes. |
| Encryption in use | Encryption in use safeguards data that is actively being created, modified, or utilized. Protecting this data involves ensuring that control measures and protections are in place before the data is used. Managing access rights and identities is also crucial in reducing the risk to this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying an individual's identity. This can include various methods, such as static passwords, one-time passwords, certificates, and biometric credentials. These identification methods help confirm that a person is who they assert to be. |
| Authorization | Authorization defines the access rights and privileges of a user, playing a crucial role in information and computer security. While authentication verifies a user's identity, authorization determines what the user can and cannot access. It helps mitigate vulnerabilities by restricting access to sensitive data that may not be necessary for the user and controlling the permissions granted during access. |
| Accounting | Accounting involves tracking activity within a system, recording details such as timestamps, accessed resources, and data transfer information. This process is useful for creating an activity trail, often referred to as a "breadcrumb trail," and plays a crucial role in forensic analysis and investigations if needed. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/15/2024 | Module 3 Milestone | Kendall Ferebee |  |
| 1.2 | 12/7/2024 | Module 6 Milestone | Kendall Ferebee |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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