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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | All input data should be tested and validated before being accepted. Input should only be accepted as input in a system once it is proven to not be malicious, or even just mistakenly invalid. |
| 1. Heed Compiler Warnings | Warnings should not be ignored or disregarded. All warnings should be fixed by adjusting the code so that no errors or warnings are present upon compiling. |
| 1. Architect and Design for Security Policies | Software architecture and design should always be made to follow all security policies. Security policies should always be a top priority when planning the architecture and design of a system. |
| 1. Keep It Simple | Unnecessarily complex software opens the door to errors and potential vulnerabilities. Keeping a code as simple as possible increases the security of the code as well as allowing it to run more efficiently. |
| 1. Default Deny | Access should automatically be denied, with approval blocked behind certain conditions that need to be met, rather than access being approved to all with certain conditions causing denial. |
| 1. Adhere to the Principle of Least Privilege | If a system has multiple functions, each user of that system should only be able to access those functions necessary for them to do their jobs. Allowing a user access to functions they do not absolutely need access to invites potential security vulnerabilities. |
| 1. Sanitize Data Sent to Other Systems | When data is sent to other systems, it is possible for unused functions to piggyback into the system to cause issues, such as SQL injection. All data being sent between systems should be cleared to ensure that only the allowed data is being sent. |
| 1. Practice Defense in Depth | This is tied to principle 4. Security should be created in layers. This way, if someone manages to take advantage of a vulnerability and circumvent one layer of defense, there will be other layers to ensure they don’t get full system-wide access. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques, such as static and dynamic testing methods, can help ensure that the presence of any vulnerabilities is minimized and that they are caught early so that catching a bug doesn’t require rewriting half of the code. It is much easier to test, find a couple of bugs and fix them, rather than wait so that a test finds 10 times as many issues and trying to fix them then. |
| 1. Adopt a Secure Coding Standard | A secure coding standard will help to strictly follow security principles and policies. Keeping to a good standard for secure coding will minimize vulnerabilities, minimize rework, and increase efficiency as things will be written correctly in regard to security the first time. |

### 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not create incompatible declarations of the same function or object |

| **Noncompliant Code** |
| --- |
| Two declarations of the same object or function specify types that are not compatible |
| // In a.cpp  extern int i;  // In b.cpp  short i; |

| **Compliant Code** |
| --- |
| Two declarations of the same object or function specify the same data type for both declarations |
| // In a.cpp  extern int i;  // In b.cpp  int i; |

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

| **Principles(s):** Validate Input Data: validating input data will catch any discrepancy in data types. Heed Compiler Warnings: Generally speaking, any IDE will inform the user of incompatible function declarations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **type-compatibility**  **type-compatibility-link**  **distinct-extern** | Will fully check all instances of this rule being broken. |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rule DCL40-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl40c.html) | Checks for declaration mismatch (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not access a cv-qualified object through a cv-unqualified type |

| **Noncompliant Code** |
| --- |
| The function g() is passed a const int, which is then cast to an int and modified. This will result in undefined behavior. |
| void g(const int &ci) {  int &ir = const\_cast<int &>(ci);  ir = 42;  }    void f() {  const int i = 4;  g(i);  } |

| **Compliant Code** |
| --- |
| The function g() is passed an int, and the caller is required to pass an int that can be modified |
| void g(int &i) {  i = 42;  }    void f() {  int i = 4;  g(i);  } |

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

| **Principles(s):** ValidateInput Data: By validating data types, accidental modifications to constant data types can be prevented. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 203 S, 242 S, 344 S | Fully Implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: EXP55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp55cpp.html) | Checks for casts that remove cv-qualification of pointer (rule partially covered) |

#### 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** |
| --- |
| The input is unbounded, so this could lead to buffer overflow |
| **void** f() {  **char** buf[12];     std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Using a string rather than a bounded array ensured the data isn’t truncated and guards against buffer overflow |
| **void** f() {     std::string input;    std::string stringOne, stringTwo;     std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: Adhering to a standard of coding to stop buffer overloading from being possible will ensure the security of the code. |
| --- |

**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=222953724) | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | 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. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| User input here is directly input into a query, which means it could be used for SQL injection. |
| std::string username = getRequestString(“username”);  std::string password = getRequestString(“password”);  std::string query = "SELECT \* FROM Users WHERE Name = '"  + username +  "' AND Pass = '" + password + "'"; |

| **Compliant Code** |
| --- |
| Using a prepared statement allows the code to safely insert user input into the correct username and password fields without the possibility of SQL injection. |
| PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, password);  stmt.executeQuery(); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: Sanitizing all data being sent between systems is important to protecting the systems and stopping attacks like SQL injection. ValidateInput Data: By validating the input by using prepared statements, the risk of SQL injection is mitigated. Adopt a Secure Coding Standard: By making it a standard to have a layer of validation, it is less likely that something will slip through the cracks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [**Fortify**](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| Here, two memory allocations are performed in the same expression. When they are passed to the function call, an exception could be thrown which could result in a memory leak |
| struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(A \*, B \*);  void f() {  g(new A, new B);  } |

| **Compliant Code** |
| --- |
| The better solution would be to remove the memory allocation and pass the objects by reference instead |
| struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(A &a, B &b);  void f() {  A a;  B b;  g(a, b);  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: By making it a standard to do this, the risk of memory leakage goes down. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### 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** |
| --- |
| Here, this code asserts a property concerning a memory-mapped structure that the code needs to behave correctly. |
| #include <cassert>  struct Timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func() {  assert(sizeof(Timer) == sizeof(unsigned char) + sizeof(unsigned int)  + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| A static\_assert allows for an incorrect assumption to be caught upon compilation rather than creating a silent malfunction or runtime error. |
| #include <cassert>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static\_assert(sizeof(timer) == sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int), "Structure must not have any padding"); |

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

| **Principles(s):** Architect and Design for Security Policies: Adding this type of security consideration right into the design will prevent problems from beginning. Keep It Simple: Keeping the programming as simple as possible will decrease issues from arising. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| Here, main() and f() do not catch any exceptions thrown by throwing\_func(). Since no handler can be found for the exception being thrown, terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Here, main() catches the exceptions thrown by throwing\_func() when f() is tried, which allows for 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):** Use Effective Quality Assurance Techniques: Quality assurance in this situation can ensure that all exceptions are handled, so even if the program needs to terminate, it can do so gracefully. Heed Compiler Warnings: Often, unhandled exceptions will be caught by the compiler. At that time, there would be a warning that should be taken into account. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| The iterators that delimit the range point into the same container, but they are in the incorrect order. This will cause undefined behavior as incrementing the first iterator will not cause it to ever be equal to the second. |
| #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** |
| --- |
| Here the iterators are in the correct order, which will allow the iteration to proceed as desired. |
| #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):** Architect and Design for Security Policies: All iterators should be designed in such a way that ranges are put in the correct order. Use Effective Quality Assurance Techniques: Quality testing would catch many issues caused by improper iterator ranges. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.MEM.BO** | Buffer Overrun |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C++: CTR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr53cpp.html) | Checks for invalid iterator range (rule partially covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Functions | [STD-009-CPP] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| Here, this function will not work correctly if a is greater than 0 because there is no branch to return a positive a value |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;     }  } |

| **Compliant Code** |
| --- |
| This function will work as intended, because no matter what the value of a is, the function will return the absolute value of a |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):** Heed Compiler Warnings: If a value-returning function has an exit that will not return any value, the compiler will provide a warning that should be taken into account. Use Effective Quality Assurance Techniques: Proper testing should find any occurrences of a function not returning a value when it should. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | return-implicit | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.MRS LANG.STRUCT.NVNR** | Missing return statement Non-void noreturn, |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object oriented programming | [STD-010-CPP] | Write constructor member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| Because the declaration order doesn’t match the initialization order, an unspecified value will be stored in the dependsOnSomeVal object, rather than the desired value. |
| **class** C {  **int** dependsOnSomeVal;  **int** someVal;    **public**:     C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Here, the orders are changed so the declaration and initialization match and the values stored in both variables behave as intended. |
| **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):** Architect and Design for Security Policies: Designing the architecture of the program so that declaration and initialization orders match in class constructors will help stop this problem from occurring. Use Effective Quality Assurance Techniques: If an unintended value is assigned to a variable because of incorrect ordering, proper testing should find the error. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **initializer-list-order** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: OOP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop53cpp.html) | Checks for members not initialized in canonical order (rule fully covered) |

### 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.

Automation should be present in all phases of the DevOps toolchain, wherever possible. Beginning with the Build phase, the automation tools can be used to check for secure coding standards as the programs are being written. CI/CD pipelines can also be used in the Build phase and the Verify and Test phase to ensure all code meets the security standards and policies.

After release, when the program is in the production side of the chain, automation will still be used at regular intervals to ensure security integrity, both with the standard built-in security and defense in depth. Even though the product has been release, testing should still be done regularly to ensure that security keeps to the standards set.

### 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 | Low | Unlikely | Medium | Low | 3 |
| STD-002-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | High | Probable | High | Medium | 2 |
| STD-009-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 3 |

### 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 protects stored data, while the data is “at rest”. This includes data on computers, phones, cloud storage, and external hardware storage. Data will be encrypted using data encryption tools. Encryption at rest must be done so that if at rest data is breached and/or stolen, the data still cannot be used by anyone without the correct decryption key. |
| Encryption in flight | Encryption in flight applies to any data in motion from one place to another. Any data being moved from one device to another must be encrypted properly using the standard data encryption tools, as data in motion can be more vulnerable to interception. |
| Encryption in use | Encryption in use will apply to all data currently being used by the system or users of same. This is the most vulnerable state of data, as even though it may be encrypted, it is currently being accessed to be read or written to. Encryption with user authentication and authorization will allow only authorized users to access the data even when in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to proving that users are who they say they are. This is often done with verification of a username and/or email address, and password. Authentication will be performed at user creation, and upon any user login. Secondary authentication is also required by use of one-time passwords or authentication devices/apps, as someone not registered as a user may gain access to the username and password of a registered user, and this will help keep them out of the system. |
| Authorization | Authorization refers to ensuring that an authenticated user has authorization to access data. All users will have certain permissions to access only what is needed to perform their work. All data and subsystems must check that an authenticated user has the proper permissions to access said data and subsystems. This will ensure that users without said permissions cannot gain access to the data and subsystems. |
| Accounting | Accounting refers to keeping track of all activity done by all users in all systems and to all data in the systems. This is done to more easily track security vulnerabilities, the causes of any errors, and to figure out what may have been accessed in the unlikely event of a security breach. |

**\***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

STD-001-CPP: 1: validating input data will catch any discrepancy in data types. 2: Generally speaking, any IDE will inform the user of incompatible function declarations.

STD-002-CPP: 1: By validating data types, accidental modifications to constant data types can be prevented.

STD-003-CPP: 10: Adhering to a standard of coding to stop buffer overloading from being possible will ensure the security of the code.

STD-004-CPP: 1: By validating the input by using prepared statements, the risk of SQL injection is mitigated. 7: Sanitizing all data being sent between systems is important to protecting the systems and stopping attacks like SQL injection. 10: By making it a standard to have a layer of validation, it is less likely that something will slip through the cracks.

STD-005-CPP: 10: By making it a standard to do this, the risk of memory leakage goes down.

STD-006-CPP: 3: Adding this type of security consideration right into the design will prevent problems from beginning. 4: Keeping the programming as simple as possible will decrease issues from arising.

STD-007-CPP: 2: Often, unhandled exceptions will be caught by the compiler. At that time, there would be a warning that should be taken into account. 9: Quality assurance in this situation can ensure that all exceptions are handled, so even if the program needs to terminate, it can do so gracefully.

STD-008-CPP: 3: All iterators should be designed in such a way that ranges are put in the correct order. 9: Quality testing would catch many issues caused by improper iterator ranges.

STD-009-CPP: 2: If a value-returning function has an exit that will not return any value, the compiler will provide a warning that should be taken into account. 9: Proper testing should find any occurrences of a function not returning a value when it should.

STD-010-CPP: 3: Designing the architecture of the program so that declaration and initialization orders match in class constructors will help stop this problem from occurring. 9s: If an unintended value is assigned to a variable because of incorrect ordering, proper testing should find the error.

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 | 03/21/2025 | Security principles and coding standards | Joseph Caron |  |
| 1.2 | 04/12/2025 | Completed coding standards and automation | Joseph Caron |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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