**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 | This helps prevent attacks such as SQL injection, buffer overflow, and cross-site scripting. Input validation should include type checking, length checking, range checking, and pattern matching. |
| 1. Heed Compiler Warnings | Compilers generate warnings to indicate potential problems in the code. Always address these warnings as they can point to issues like potential buffer overflows, type mismatches, and other security vulnerabilities. |
| 1. Architect and Design for Security Policies | This includes threat modeling, security requirements analysis, and considering how to mitigate potential vulnerabilities. |
| 1. Keep It Simple | Complex code increases the risk of errors and vulnerabilities. Simplicity improves maintainability and reduces the likelihood of security flaws. |
| 1. Default Deny | Adopt a default deny approach for access control. This means that by default, access is denied unless explicitly allowed. This minimizes the attack surface by ensuring that only necessary permissions are granted. |
| 1. Adhere to the Principle of Least Privilege | This ensures that users and systems operate with the minimum privileges necessary to perform their tasks. This limits the potential damage in case of a security breach. |
| 1. Sanitize Data Sent to Other Systems | Always sanitize data before sending it to other systems, especially if those systems have different security policies. This prevents the introduction of malicious data into those systems. |
| 1. Practice Defense in Depth | Use multiple layers of security to protect systems and data. This approach ensures that if one layer is breached, additional layers of defense are still in place to thwart attacks. |
| 1. Use Effective Quality Assurance Techniques | Implement rigorous testing methodologies, such as code reviews, static analysis, dynamic analysis and penetration testing, to identify and fix security vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Follow established secure coding standards (like SEI CERT C++ Coding Standards) to ensure that code is written with security in mind. These standards provide guidelines for avoiding common vulnerabilities and implementing best practices. |

### 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-DT | Using explicit data types helps to avoid type mismatches and potential vulnerabilities related to type conversions. This improves code readability and maintainability. |

| **Noncompliant Code** |
| --- |
| Assigning a negative value to an unsigned integer (unsigned int x = -1) results in an implicit type conversion. This can lead to unexpected behavior, as the negative value is converted to a large positive value. The printf function call with %d format specifier for an unsigned integer can further produce unpredictable and incorrect output. |
| int main() {  unsigned int x = -1; // Implicit type conversion  printf("%d\n", x); // May produce unexpected result  } |

| **Compliant Code** |
| --- |
| Ensures explicit type declaration for the variable x as an int. This avoids the implicit type conversion that can lead to unexpected behavior, such as assigning an unsigned integer a negative value. By using explicit types, the code becomes clearer and more predictable, improving readability and reducing the risk of errors. |
| int main() {  int x = -1; // Explicit type declaration  printf("%d\n", x); // Clear and predictable output  } |

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

| **Principles(s):** Validate Input Data: This principle ensures that all data entering a system is correctly typed and validated, reducing errors and vulnerabilities from incorrect data handling.  Heed compiler Warnings: Using explicit data types helps heed compiler warnings that might alert developers to potential data type mismatches or unsafe conversions.  Adopt a Secure Coding Standard: Enforcing explicit data types aligns with secure coding standards that advocate for clarity, safety, and predictability in code, thereby minimizing the risk of security flaws. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw2dG1BhB4EiwA998cqGpMuCLLHxUYFUhTTEuz-25AcAWHIwo0yunCxy6yimxJTnNS-ZVhQhoCAccQAvD_BwE) | 10.6 | C++ rule S927 | Checks for mismatches between variable declarations and their actual data types used in operations. |
| [Coverity](https://scan.coverity.com/) | 2023.12 | MISRA C 2004 Rule 10.1 | Ensures that the data types of operands in operations are compatible, preventing implicit conversions. |
| [Fortify](https://fortifyapp.com/) | 20.2.0 | Bad Data Type Conversion | Detects potential data type conversions that could lead to errors or vulnerabilities. |
| [Clang Static Analyzer](https://clang-analyzer.llvm.org/) | 12.0.1 | TypeSafety Checker | Identifies potentially unsafe type conversions and provides warnings to guide developers towards safer coding practices. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-DV] | Using named constants instead of magic numbers improves code readability and maintainability, making it easier to understand and modify. |

| **Noncompliant Code** |
| --- |
| Uses magic numbers directly in the code, which can be confusing. |
| int calculateArea(int radius) {  return 3.14 \* radius \* radius; // Magic number  } |

| **Compliant Code** |
| --- |
| Uses a named constant to improve code clarity. |
| const double PI = 3.14;  int calculateArea(int radius) {  return PI \* radius \* radius; // Clear and understandable  } |

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

| **Principles(s):** Keep It Simple: Using named constants simplifies understanding the code, making it more readable and maintainable. This principle helps to eliminate confusion that can arise from the use of obscure magic numbers, reducing the risk of errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9 LTS | Magic Number Detection | Automatically identifies and reports magic numbers, suggesting replacement and named constants. |
| [ESLint](https://community.linuxmint.com/software/view/fslint) | 7.20.0 | No-magic-numbers | Configurable to warn against the use of numbers other than defined constants. |
| [Checkstyle](https://checkstyle.sourceforge.io/) | 10.17.0 | MagicNumber | Flags use of magic numbers and suggest using a constant declaration. |
| [PMD](https://pmd.github.io/) | 7.2.0 | AvoidMagicNumbers | Checks for magic numbers and recommends using constants for better code clarity. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-SC] | Using safe string functions helps prevent buffer overflows and other vulnerabilities associated with string manipulation. |

| **Noncompliant Code** |
| --- |
| Uses unsafe string functions which can lead to buffer overflows. |
| void copyString(char\* dest, const char\* src) {  strcpy(dest, src); // Unsafe string copy  } |

| **Compliant Code** |
| --- |
| Uses safe string functions to prevent buffer overflows. |
| void copyString(char\* dest, const char\* src, size\_t destSize) {  strncpy(dest, src, destSize); // Safe string copy  dest[destSize - 1] = '\0'; // Ensure null termination  } |

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

| **Principles(s):** Practice Defense in Depth: Employing safe string functions adds a layer of security by preventing buffer overflows, which are common vulnerabilities that can be exploited in software attacks. This principle ensures that even if one security mechanism fails, others are in place to maintain protection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Splint](https://splint.org/) | 3.12 | Buffer Overflow | Analyzes C code to check for potential buffer overflows by examining unsafe string and memory operations. |
| [Fortify](https://fortifyapp.com/) | 20.2.0 | Buffer Overflow | Static code analysis tool that identifies unsafe string manipulations which may lead to buffer overflows. |
| [Clang Static Analyzer](https://clang-analyzer.llvm.org/) | 12.0.1 | String Manipulation Errors | Provides diagnostics for string operations that could result in buffer overflows, out-of-bounds errors, and other vulnerabilities. |
| [Klocwork](https://www.perforce.com/p/kw/try-klocwork-static-code-analyzer?utm_source=googleadwords&utm_medium=cpc&utm_campaign=KlocworkNABrand&utm_adgroup=Klocwork-BrandTerms-NA-Search&utm_term=klocwork%20pricing&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7F3pQa7EDjr6GXPNpxqeyb0jkLX_yrJcnX-RlEU89q2qq-THVQ2WgBoC1M0QAvD_BwE) | 2024.2 | STR31-C | Ensures that strings are copied safely, alerting on potential buffer overflows when using dangerous functions like ‘strcpy’ |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | Using parameterized queries prevents SQL injection attacks by separating SQL logic from data, ensuring that data properly escaped and handled. |

| **Noncompliant Code** |
| --- |
| Construct SQL queries by concatenating user input directly, leading to SQL injection risk. |
| std::string query = "SELECT \* FROM users WHERE name = '" + userInput + "';"; |

| **Compliant Code** |
| --- |
| Uses parameterized queries to prevent SQL injection. |
| sqlite3\_stmt\* stmt;  std::string query = "SELECT \* FROM users WHERE name = ?";  sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr);  sqlite3\_bind\_text(stmt, 1, userInput.c\_str(), -1, SQLITE\_STATIC); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: Parameterized queries inherently sanitize data by separating code from data inputs, thus preventing SQL injection. This principle ensures that data interacting with external systems, like databases, is cleansed of potential SQL injection vectors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [OWASP Dependency Check](https://owasp.org/www-project-dependency-check/) | 10.0.2 | SQL injection Vulnerability Scanner | Scans project dependencies for known vulnerabilities, including those that might allow SQL injection. |
| [Fortify](https://fortifyapp.com/) | 20.2.0 | SQL Injection | Static code analysis tool that identifies patterns in code that may lead to SQL injection, recommending parameterized queries or other safe practices. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9LTS | SQL Injection Rule | Detects code patterns that could be vulnerable to SQL injection attacks and suggests secure coding practices. |
| [Veracode](https://www.veracode.com/) | 2021 | SQL Injection Detection | Provides comprehensive static and dynamic analysis to identify SQL injection vulnerabilities in application code. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-MP | Smart pointers automatically manage memory and help prevent memory leaks and dangling pointers by ensuring proper deallocation. |

| **Noncompliant Code** |
| --- |
| Manage dynamic memory manually, risking memory leaks. |
| int\* ptr = new int[10];  // Forgot to delete the allocated memory |

| **Compliant Code** |
| --- |
| Uses smart pointers to manage memory safely. |
| std::unique\_ptr<int[]> ptr(new int[10]);  // Memory is automatically freed when ptr goes out of scope |

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

| **Principles(s):** Adopt a Secure Coding Standard: Employing smart pointers is a key aspect of modern, secure C++ programming. This practice helps in managing memory safely by automatically deallocating resources, thus minimizing the risk of memory leaks and dangling pointers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Valgrind](https://valgrind.org/downloads/current.html) | 3.23.0 | Memcheck | A memory error detector that can identify memory management errors such as leaks, incorrect deallocation, and use of undefined values. |
| [Cppcheck](https://cppcheck.sourceforge.io/) | 2.14 | Memory Leaks | Static analysis tool that detects memory leaks, uninitialized memory, and improper deallocation. |
| [Clang Static Analyzer](https://clang-analyzer.llvm.org/) | 12.0.1 | Use After Free, Memory Leak | Analyzes code to detect cases where objects are used after being freed and memory is never released. |
| [Visual Studio Code Analysis](July%202024%20(version%201.92)%20-%20Visual%20Studio%20Code%20%20Visual%20Studio%20Code%20https:/code.visualstudio.com%20›%20updates) | 1.92 | C26409 | Recommends using smart pointers or resource acquisition is initialization (RAII) objects to manage resources safely and effectively. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-ASS | Assertions help catch programming errors during development by verifying assumptions made by the program and ensuring invariants are maintained. |

| **Noncompliant Code** |
| --- |
| Fails silently when an assumption is violated. |
| void process(int value) {  if (value > 0) {  // Do something  }  } |

| **Compliant Code** |
| --- |
| Uses assertions to enforce that the value is positive. |
| #include <cassert>  void process(int value) {  assert(value > 0);  // Do something  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Assertions are a crucial quality assurance technique that helps catch programming errors during development by actively checking assumptions. They ensure that the program operates under expected conditions, enhancing software reliability and maintainability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Moderate | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Cppcheck](https://cppcheck.sourceforge.io/) | 2.14 | Assert Usage | Static analysis tool that checks for the proper use of assertions in C++ code to ensure assumptions are explicitly verified. |
| [Clang Static Analyzer](https://clang-analyzer.llvm.org/) | 12.0.1 | Dead Code | Identifies unreachable code that might indicate failed assertions or other logic errors. |
| [Visual Studio Code Analysis](https://code.visualstudio.com/updates) | 1.92 | C26494 | Encourages best practices in pointer use, indirectly supporting the use of assertions by promoting safer and more predictable pointer handling. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9 LTS | Misra C++ Rule 7-5-1 | Enforces the use of assertions to check runtime and compile-time invariants, ensuring code reliability. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-EXC | Exceptions provide robust way to handle errors, ensuring that error conditions are properly managed and resources are cleaned up. |

| **Noncompliant Code** |
| --- |
| Uses return codes for error handling, which can be ignored. |
| int divide(int a, int b) {  if (b == 0) {  return -1; // Error code  }  return a / b;  } |

| **Compliant Code** |
| --- |
| Uses exceptions to handle errors. |
| int divide(int a, int b) {  if (b == 0) {  throw std::invalid\_argument("Division by zero");  }  return a / b;  } |

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

| **Principles(s):** Architect and Design for Security Policies: Designing software to handle errors robustly with exceptions ensures that unexpected conditions do not lead to system failures or security vulnerabilities. This approach is fundamental in secure architecture, promoting resilience and stability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang Static Analyzer](https://clang-analyzer.llvm.org/) | 12.0.1 | Use of Exceptions | Analyzes C++ code to ensure that exceptions are used correctly to handle error conditions, especially in scenarios that could lead to application failure. |
| [Cppcheck](https://cppcheck.sourceforge.io/) | 2.14 | Exception Safety | Checks that functions are exception-safe, meaning they do not leak resources or leave data in inconsistent states when an exception is thrown. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9LTS | S5623 | Ensures that exceptions are used instead of return codes or other mechanisms that might allow error conditions to be ignored. |
| [Coverity](https://scan.coverity.com/) | 2023.12 | Error handling Issues | Identifies error handling practices that could lead to failures, recommending more robust practices such as exceptions. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | STD-008-IV | Rationalize user inputs prevents various types of attacks, such as SQL injection, XSS and buffer overflows, by ensuring that inputs conform to expected formats. |

| **Noncompliant Code** |
| --- |
| Fails to validate user inputs |
| void processUserInput(const std::string& input) {  // Assumes input is valid  } |

| **Compliant Code** |
| --- |
| Validates user inputs to ensure they are within expected bounds. |
| void processUserInput(const std::string& input) {  if (input.empty() || input.length() > MAX\_INPUT\_LENGTH) {  throw std::invalid\_argument("Invalid input");  }  // Process input  } |

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

| **Principles(s):** Validate Input Data: Validating input data to ensure it conforms to expected formats is essential in preventing common vulnerabilities such as SQL injection, XSS, and buffer overflows. This practice ensures that only properly formatted data is processed, safeguarding the system against malformed or malicious input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [OWASP ZAP (Zed Attack Proxy)](https://www.zaproxy.org/) | 2.15.0 | Input Validation Testing | Provides automated security tests to identify vulnerabilities caused by improper input validation. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9 LTS | Security Hotspots and Vulnerabilities | Analyzes code for potential security weaknesses related to input validation, offering suggestions for mitigations. |
| [Fortify](https://fortifyapp.com/) | 20.2.0 | Input Validation Issues | Static code analysis tool that scans for a variety of input validation flaws, helping developers understand and fix vulnerabilities. |
| [Veracode](https://www.veracode.com/) | 2021 | Input Validation Flaws | Provides comprehensive static and dynamic analysis to identify unsafe input patterns and suggest fixes. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Logging | STD-009-LOG | Secure logging helps in monitoring and analyzing security-relevant events, providing insights into potential attacks and system behaviour. |

| **Noncompliant Code** |
| --- |
| Logs sensitive information insecurely. |
| std::ofstream log("log.txt");  log << "User password: " << password << std::endl; // Sensitive information |

| **Compliant Code** |
| --- |
| Logs security-relevant events without exposing sensitive information. |
| std::ofstream log("log.txt");  log << "User login attempt: " << username << std::endl; // No sensitive information |

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

| **Principles(s):** Default Deny: Secure logging, especially when handling sensitive information, should follow a "default deny" policy. This means only explicitly permitted details should be logged to avoid accidental exposure of sensitive information. By adhering to this principle, the system ensures that only non-sensitive, essential data is logged, thereby minimizing the risk of a data breach. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Moderate | Low to Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Splunk](https://www.splunk.com/en_us/download.html?utm_campaign=google_amer_en_search_brand&utm_source=google&utm_medium=cpc&utm_content=free_trials_downloads&utm_term=splunk&device=c&_bt=662330344234&_bm=e&_bn=g&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Nsi6p3_d-6bhmSjqySXXMePyGjN5gB6cr5FoKJBxQrl3B52kSsWIRoCBtQQAvD_BwE) | 2023 | Secure Logging Practices | Monitors log files and configurations to ensure that sensitive information is not logged. Provides alerts if violations are detected. |
| [ELK Stack](https://www.elastic.co/elastic-stack/?utm_campaign=Google-B-Amer-US&utm_content=Brand-Core-ELK-EXT&utm_source=google&utm_medium=cpc&device=c&utm_term=elk%20stack&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7G3OuFOQrXS18YvN98imCPY0MzEC21UptZuvUkpkdcAraBFQtBGZ_xoCzBMQAvD_BwE) | 2023.11 | Log Inspection | Analyzes logging formats and contents to detect potential leakage of sensitive information. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9 LTS | Security Logging and Monitoring | Scans code to identify and alert on instances where sensitive data might be logged, suggesting secure logging practices. |
| [AWS CloudWatch](https://aws.amazon.com/cloudwatch/) | 2.2.93.0 | Log Monitoring | Provides monitoring and analysis of log files generated on AWS, ensuring that sensitive data is not included in logs. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Code Reviews | STD-010-CR | Regular code reviews help identify and fix security vulnerabilities and code quality issues early in the development process, ensuring that the codebase remains secure and maintainable. |

| **Noncompliant Code** |
| --- |
| Skips code reviews, leading to potential undetected vulnerabilities. |
| void addUser(const std::string& username) {  // Code added without review  } |

| **Compliant Code** |
| --- |
| Conducts regular code reviews to maintain code quality and security. |
| // Code is reviewed by peers before being merged  void addUser(const std::string& username) {  // Reviewed code  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Regular code reviews are a vital quality assurance technique that can identify and rectify security vulnerabilities and coding errors early in the development process. By implementing systematic reviews, the team can catch issues that automated tools might miss, ensuring the codebase remains robust and secure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Moderate | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [GitHub Advanced Security](https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security) | 3.14 | Code Scanning Alerts | Integrates with GitHub to provide automated code reviews and security scanning before merge approvals. |
| [SonarQube](https://www.sonarsource.com/products/sonarqube/?s_campaign=SQ-NA-US-East-Brand&s_content=SonarQube&s_term=sonarqube&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21283910625&cq_con=165359618347&cq_term=sonarqube&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAjw_Na1BhAlEiwAM-dm7Fb0I2KW9qz-ozkUKgks-uhxkKtDKWSDYJmtNIm4rl56gxi5qXHcrhoCCxYQAvD_BwE) | 8.9 LTS | Code Quality and Security | Automates the detection of code quality and security issues, facilitating peer reviews by highlighting areas of concern. |
| [Azure DevOps](https://azure.microsoft.com/en-us/products/devops) | 2022 | Differential Code Review | A tool for code reviews that includes linting and static analysis to support human reviewers by pre-identifying potential issues. |
| [Review Board](https://www.reviewboard.org/) | 5.3 | Review Requests | Facilitates thorough code reviews with support for side-by-side diffs, inline comments, and integration with various version control systems. |

### 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.

To ensure that the standards defined in the security policy are effectively enforced, it’s crucial to integrate and automate these standards throughout the DevSecOps lifecycle. Here’s how I would guide the modification of the existing DevOps process to automate the enforcement of these standards:

### ****Assess and Plan****

* **Automation**: Integrate automated tools to scan the codebase and infrastructure configuration for compliance with the security standards at the planning stage. This involves setting up automated security requirement gathering tools that align threat assessments with the necessary coding and infrastructure standards.
* **Action**: Configure project management tools to automatically flag projects that require specific security attention, ensuring all new features consider security standards from the outset.

### ****Design****

* **Automation**: Use design tools that automatically enforce secure design patterns and practices. These could include plugins for design software that validate security decisions against the established standards.
* **Action**: Automatically generate design review tasks in project tracking tools when new components are added to ensure they comply with security standards.

### ****Build****

* **Automation**: Implement pre-commit hooks in version control systems that run static analysis tools to check code against the security standards before it is even committed.
* **Action**: Set up CI pipelines to reject builds that fail to meet security standards, ensuring non-compliant code cannot progress to the next stage.

### ****Verify and Test****

* **Automation**: Utilize dynamic analysis tools and automated testing frameworks that specifically test for compliance with the security standards defined.
* **Action**: Ensure that every release candidate undergoes automated penetration testing and security audit before it moves to deployment.

### ****Transition and Health Check****

* **Automation**: Automate configuration checks that ensure deployment scripts and environment settings adhere to security standards before deployment to pre-production.
* **Action**: Use infrastructure as code (IaC) tools to manage and verify environment configurations against secure baselines.

### ****Production****

* **Automation**: Deploy monitoring tools that continuously scan for deviations from security standards and alert in real-time.
* **Action**: Integrate logging and response systems to automatically initiate corrective actions or alerts if standards violations are detected.

### ****Maintain and Stabilize****

* **Automation**: Regularly schedule and automate compliance scans to ensure ongoing adherence to security standards.
* **Action**: Implement automated rollback protocols to revert to the last known good configuration in the event of a security breach.

### ****Monitor and Detect****

* **Automation**: Use SIEM tools to analyze log data for signs of security standards violations.
* **Action**: Configure alert systems to notify relevant personnel of potential security incidents that could indicate standards breaches.

### ****Respond****

* **Automation**: Employ automated response solutions that can isolate affected systems, block attacks, or execute predefined scripts to mitigate breaches.
* **Action**: Ensure response plans are dynamically updated based on the latest security standards and threat intelligence.

By automating enforcement at each stage of the DevSecOps pipeline and integrating security standards directly into the tools and processes used throughout the development lifecycle, Green Pace can ensure that its products are secure by design, by default, and in deployment. This proactive and integrated approach minimizes risks and aligns with the best practices for maintaining a robust security posture in a dynamic threat environment.

### 

### 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-002-DV | Medium | Likely | Low | Medium | 3 |
| STD-003-SC | High | Likely | Medium | High | 1 |
| STD-004-SQL | Critical | High | Medium | High | 1 |
| STD-005-MP | High | Likely | Medium | High | 1 |
| STD-006-ASS | High | Moderate | Low | High | 2 |
| STD-007-EXC | High | Moderate | Medium | High | 2 |
| STD-008-IV | High | High | Medium | High | 1 |
| STD-009-LOG | High | Moderate | Low to Medium | High | 1 |
| STD-010-CR | High | Moderate | High | High | 1 |

### 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 | **What it is:** Encryption at rest involves encrypting data that is stored on physical media, ensuring that it cannot be read without appropriate decryption keys.  **How and Why the Policy Applies:** This policy applies to all stored data, including databases, file servers, and backups. It is crucial to implement because it protects sensitive data from unauthorized access, particularly if physical security measures fail or storage media is stolen. |
| Encryption in flight | **What it is:** Encryption in flight refers to protecting data as it travels across networks, ensuring that it remains confidential and tamper-proof from origin to destination.  **How and Why the Policy Applies:** This policy is mandatory for all data transmitted across public or untrusted networks, including email communications and data transfers. It helps prevent interception and unauthorized access to data during transmission. |
| Encryption in use | **What it is:** Encryption in use involves protecting data that is actively being processed by systems.  **How and Why the Policy Applies:** This policy applies whenever data is processed in environments where the security of the operating system or the application cannot be fully trusted. It is essential for mitigating risks from system vulnerabilities, ensuring data integrity and confidentiality during processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **What it is:** Authentication verifies the identity of a user or system, typically using credentials such as passwords, tokens, or biometric data.  **How and Why the Policy Applies:** Authentication is required for all user logins and system access attempts. This policy is crucial for establishing the identity of users and systems, ensuring that only authorized entities can access resources. |
| Authorization | **What it is:** Authorization determines the resources a user or system is permitted to access and the actions they are allowed to perform.  **How and Why the Policy Applies:** This policy applies to user level of access and changes to databases or settings. It is vital for controlling access to resources and services, based on the authenticated entity’s permissions. |
| Accounting | **What it is:** Accounting, also known as auditing, involves tracking and logging user activities and system changes to provide a record of actions for security monitoring and analysis.**How and Why the Policy Applies:** Accounting must cover all user actions, including file accesses and system changes. It supports security analysis, forensic investigations, and compliance with regulatory requirements by providing a detailed activity trail. |

**\***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 |  |
| 2.0 | 08/09/2024 | Project 1 | Romain Tomlinson |  |
| [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 |