**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 | Ensuring all input data is validated before processing is crucial for preventing attacks such as SQL injection, buffer overflow, and cross-site scripting. Validation ensures that the data conforms to expected formats, ranges, and types, mitigating the risk of malicious input causing harm. |
| 1. Heed Compiler Warnings | Compiler warnings often indicate potential vulnerabilities or bugs in the code. By addressing these warnings, developers can proactively identify and fix issues related to security, performance, and reliability, leading to more robust and secure software. |
| 1. Architect and Design for Security Policies | Incorporating security policies at the design phase helps identify potential threats and mitigate risks early, ensuring the system is built with security in mind from the ground up. |
| 1. Keep It Simple | Simple code is easier to understand, maintain, and secure. Complexity can introduce hidden vulnerabilities and make it harder to spot errors. By adhering to simplicity, developers can reduce the likelihood of introducing security flaws. |
| 1. Default Deny | Default denies, also known as "default to no access," means that access permissions should be explicitly granted rather than assumed. By denying access by default, systems can prevent unauthorized access and minimize the attack surface. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should be granted only the minimum level of access necessary to perform their functions. This principle limits the potential damage from compromised accounts or systems by restricting their access to only what is necessary. |
| 1. Sanitize Data Sent to Other Systems | When transmitting data to other systems, it's essential to sanitize it to prevent injection attacks and ensure that the receiving system interprets the data correctly. This includes encoding and escaping data appropriately. |
| 1. Practice Defense in Depth | Defense in depth involves implementing multiple layers of security controls to protect resources. If one layer is compromised, additional layers provide continued protection, making it more difficult for attackers to breach the system. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques such as code reviews, static analysis, and automated testing help identify and fix vulnerabilities early in the development process. These practices ensure that the code meets security standards and functions correctly. |
| 1. Adopt a Secure Coding Standard | Secure coding standards provide guidelines for writing secure code, helping developers avoid common vulnerabilities and follow best practices. Adopting these standards ensures a consistent and secure approach to software development across the organization. |

### 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** | Data Type Coding Standard |
| --- | --- | --- |
| **Data Type** | STD-001-DAT | Ensuring data is stored in the most efficient and secure manner prevents overflow issues and optimizes memory usage. |

| **Noncompliant Code** |
| --- |
| Using int for a value that exceeds its capacity |
| int userID = 12345678901234567890; |

| **Compliant Code** |
| --- |
| Using long long to store a large integer value |
| long long userID = 12345678901234567890; |

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

| **Principles(s):** Validate Input Data - Ensures correct data type usage to prevent overflow and data corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 12.0 | DataTypeChecker | Checks for appropriate data type usage. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Data Value Coding Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-DAT | Ensuring only expected and safe data is processed helps prevent malicious input from causing harm. |

| **Noncompliant Code** |
| --- |
| Using atoi without validating the input |
| int age = atoi(input); |

| **Compliant Code** |
| --- |
| Validating input before conversion |
| int age;  if (isdigit(input)) {  age = atoi(input);  } else {  // handle error  } |

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

| **Principles(s):** Validate Input Data - Validates user input to prevent invalid or malicious data from being processed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-tidy | 12.0 | InputValidator | Validates user input data. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **String Correctness Coding Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-STR | Prevents buffer overflows and data corruption by ensuring strings are correctly null-terminated. |

| **Noncompliant Code** |
| --- |
| strncpy might not null-terminate the string |
| char name[10];  strncpy(name, input, 10); |

| **Compliant Code** |
| --- |
| Ensuring the string is null-terminated |
| char name[10];  strncpy(name, input, 9);  name[9] = '\0'; |

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

| **Principles(s):** Ensures strings are properly terminated to prevent overflow and data corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.5 | StringChecker | Checks for proper string null-termination |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **SQL Injection Coding Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | Prevents SQL injection attacks by using parameterized queries instead of concatenating user input directly into SQL statements. |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into SQL queries |
| std::string query = "SELECT \* FROM users WHERE id = " + userId; |

| **Compliant Code** |
| --- |
| Using prepared statements with parameterized queries |
| std::string query = "SELECT \* FROM users WHERE id = ?";  stmt = conn.prepareStatement(query);  stmt.setString(1, userId); |

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

| **Principles(s):** Validate Input Data - Prevents SQL injection by validating and parameterizing user input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 8.9 | SQLInjectionChecker | Detects potential SQL injection vulnerabilities. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Protection Coding Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-MEM | Ensures efficient use of memory and system stability by properly managing memory allocation and deallocation. |

| **Noncompliant Code** |
| --- |
| Allocating memory without freeing it |
| char\* buffer = new char[10]; |

| **Compliant Code** |
| --- |
| Ensuring allocated memory is freed |
| char\* buffer = new char[10];  // Use the buffer  delete[] buffer; |

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

| **Principles(s):** Proper memory management to prevent leaks and ensure system stability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.17 | Memcheck | Detects memory leaks and errors. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Assertions Coding Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-ASS | Ensures program correctness during development by validating assumptions and catching errors early. |

| **Noncompliant Code** |
| --- |
| Lack of assertions to validate assumptions |
| int x = 10;  if (x > 0) {  // do something  } |

| **Compliant Code** |
| --- |
| Using assertions to validate assumptions |
| int x = 10;  assert(x > 0);  // do something |

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

| **Principles(s):** Validates assumptions and catches errors early in development. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.5 | AsserChecker | Checks for proper use of assertions. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exceptions Coding Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-EXC | Ensures robust error handling and recovery by using specific exception types and proper handling techniques. |

| **Noncompliant Code** |
| --- |
| Using a generic catch block |
| try {  // code that may throw  } catch (...) {  // generic catch  } |

| **Compliant Code** |
| --- |
| Handling specific exceptions properly |
| try {  // code that may throw  } catch (const std::exception& e) {  // handle specific exception  } |

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

| **Principles(s):** Validate Input Data - Ensures robust error handling and system stability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 12.0 | ExceptionChecker | Checks for proper exception handling practices. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Use Secure Random Number Generators** |
| --- | --- | --- |
| [Student Choice] | STD-008-RNG | Ensures the randomness of generated numbers for security-critical applications, preventing predictable outcomes. |

| **Noncompliant Code** |
| --- |
| Using rand() for random number generation |
| int randomNumber = rand(); |

| **Compliant Code** |
| --- |
| Using secure random number generator |
| #include <random>  std::random\_device rd;  std::mt19937 gen(rd()); std::uniform\_int\_distribution<> dis(1, 100);  intrandomNumber = dis(gen); |

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

| **Principles(s):** Ensures secure random number generation to prevent predictable outcomes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.5 | RNGChecker | Checks for proper use of secure random number generators. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Sanitize All User Inputs** |
| --- | --- | --- |
| [Student Choice] | STD-009-INS | Ensures that user inputs are properly sanitized to prevent injection attacks and other malicious activities. |

| **Noncompliant Code** |
| --- |
| Not sanitizing user input |
| std::string userInput = getUserInput();  executeCommand(userInput); |

| **Compliant Code** |
| --- |
| Sanitizing user input before using it |
| std::string userInput = getUserInput();  std::string sanitizedInput = sanitize(userInput);  executeCommand(sanitizedInput); |

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

| **Principles(s):** Validate Input Data - Ensures user inputs are sanitized to prevent injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 8.9 | InputSanitizerChecker | Detects unsanitized user inputs. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure Thread Safety** |
| --- | --- | --- |
| [Student Choice] | STD-010-THR | Ensures safe execution of code in a multi-threaded environment to prevent race conditions and data corruption. |

| **Noncompliant Code** |
| --- |
| Accessing shared data without synchronization |
| int sharedData = 0;  void increment() {  sharedData++;  } |

| **Compliant Code** |
| --- |
| Using mutex to ensure thread safety |
| #include <mutex>  std::mutex mtx;  int sharedData = 0;  void increment() {  std::lock\_guard<std::mutex> lock(mtx);  sharedData++;  } |

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

| **Principles(s):** Validate Input Data - Ensures thread safety to prevent race conditions and data corruption. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ThreadSanitizer | 1.2 | TheadSafetyChecker | Detects thread safety issues in multi-threaded code. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 will be essential in enforcing and ensuring compliance with the standards defined in this policy. Green Pace’s well-established DevOps process and infrastructure provide a solid foundation for integrating security measures throughout the software development lifecycle, adopting a DevSecOps approach. In the pre-production stage, the process begins with assessing and planning, where automated tools can continuously analyze the threat landscape and ensure compliance with regulatory standards, adjusting security policies as needed. During the design phase, integrating automated static analysis tools will help identify potential vulnerabilities early, enforcing security by design principles.

As the project moves into the build phase, incorporating security-focused continuous integration (CI) pipelines will automatically check for vulnerabilities, ensuring only secure code advances. Automated testing strategies, including unit, integration, and penetration testing, will verify that all security standards are met before deployment. In the production stage, automated deployment checks and health assessments will verify the security integrity of the build, ensuring it meets all necessary security requirements before going live.

Once in production, continuous monitoring through automated tools will detect any security breaches or vulnerabilities in real-time. Integrating automated incident response mechanisms will allow for immediate action, such as isolating compromised systems or deploying patches, ensuring any threats are swiftly addressed. Finally, regular automated maintenance tasks, like applying security patches, will maintain the system's security over time, with continuous monitoring ensuring any deviations from the security policy are quickly identified and resolved.

By automating key processes such as compliance checks, testing, monitoring, and incident response, Green Pace can effectively enforce the security standards defined in this policy. This approach not only enhances security but also maintains the agility and efficiency of the existing DevOps processes, ensuring that security is seamlessly integrated into every phase of development.

### 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-DAT | High | High | Medium | High | 2 |
| STD-003-STR | High | Medium | Low | High | 3 |
| STD-004-SQL | Critical | High | Medium | Very High | 1 |
| STD-005-MEM | High | Medium | Low | High | 3 |
| STD-006-ASS | Medium | Medium | Low | Medium | 4 |
| STD-007-EXC | High | Medium | Medium | High | 3 |
| STD-008-RNG | High | Medium | Low | High | 3 |
| STD-009-INS | High | High | Medium | High | 2 |
| STD-010-THR | High | Medium | Medium | High | 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 refers to the protection of data stored on a physical device or media by converting it into an unreadable format, which can only be decrypted with the appropriate key. This type of encryption is applied to databases, storage devices, and backup files to prevent unauthorized access when data is not actively being used or transmitted. This policy is critical to protect sensitive data from theft or unauthorized access in the event of physical device loss or theft. It should be applied to all storage solutions containing sensitive or personal information. |
| Encryption in flight | Encryption in flight, or encryption in transit, ensures that data being transmitted across networks is protected from interception and unauthorized access by encrypting it during transmission. This policy is applied to data transferred over networks such as the internet, internal networks, or between servers. Protocols like HTTPS, TLS, and VPNs are commonly used. Encryption in flight is essential to secure data from eavesdropping, tampering, and man-in-the-middle attacks during transmission. This policy applies to all communications of sensitive or confidential data. |
| Encryption in use | Encryption in use protects data while it is being processed or accessed by applications or users. This type of encryption ensures that data remains secure even when it is being used. This can be achieved through techniques like secure computing environments, homomorphic encryption, or data masking. This policy is crucial for protecting sensitive information during active use, particularly in environments where data is processed or analyzed. It ensures that data privacy is maintained even during processing activities. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the identity of a user, device, or system before granting access to resources. This involves mechanisms like passwords, multi-factor authentication (MFA), and biometric verification. Authentication policies are vital to ensure that only authorized individuals or systems can access sensitive resources. This policy should be enforced for all systems requiring user access. |
| Authorization | Authorization determines what an authenticated user is allowed to do within a system, including access levels and permissions to resources. This involves setting up roles and permissions within the system, ensuring users have access only to the resources necessary for their role. Authorization policies are essential to enforce the principle of least privilege, ensuring users only have access to the information and functions necessary for their job. This policy should be applied to all systems managing sensitive data. |
| Accounting | Accounting, or auditing, tracks and records user activities and changes within a system for security and compliance purposes. This involves logging user actions, changes to the database, and access to files, and maintaining these logs for review and analysis. Accounting policies are crucial for tracking and investigating security incidents, ensuring compliance, and detecting unauthorized activities. This policy should be applied across all systems to maintain detailed records of user activities and changes. |

**\***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.

|  |  |  |
| --- | --- | --- |
| Standard | Principles | Explanation |
| STD-001-CCP | 1, 2, 7 | Validate Input Data: Ensures input data is secure to prevent vulnerabilities.  Heed Compiler Warnings: Addresses potential issues highlighted by compiler warnings.  Sanitize Data Sent to Other Systems: Protects against injection attacks by sanitizing data. |
| STD-002-DAT | 1, 3, 7 | Validate Input Data: Prevents issues related to data handling.  Architect and Design for Security Policies: Ensures security is built into data management.  Sanitize Data Sent to Other Systems: Ensures data integrity during transmission. |
| STD-003-STR | 1, 4, 8 | Validate Input Data: Ensures data passed through architecture is validated.  Keep It Simple: Simplifies architecture to reduce vulnerabilities.  Practice Defense in Depth: Employs multiple security layers in system architecture. |
| STD-004-SQL | 1, 6, 7 | Validate Input Data: Prevents SQL injection through input validation.  Adhere to the Principle of Least Privilege: Restricts database access to necessary permissions.  Sanitize Data Sent to Other Systems: Prevents SQL-related vulnerabilities by sanitizing queries. |
| STD-005-MEM | 1, 4, 9 | Validate Input Data: Manages memory securely to avoid vulnerabilities.  Keep It Simple: Reduces complexity in memory management.  Use Effective Quality Assurance Techniques: Identifies and fixes memory management issues through QA practices. |
| STD-006-ASS | 3, 6, 9 | Architect and Design for Security Policies: Embeds security into assignment processes.  Adhere to the Principle of Least Privilege: Limits assignment permissions.  Use Effective Quality Assurance Techniques: Reviews assignment mechanisms for vulnerabilities. |
| STD-007-EXC | 1, 4, 8 | Validate Input Data: Handles exceptions securely to prevent vulnerabilities.  Keep It Simple: Simplifies exception handling to avoid security issues.  Practice Defense in Depth: Uses multiple security layers in exception handling. |
| STD-008-RNG | 1, 4, 9 | Validate Input Data: Ensures secure random number generation.  Keep It Simple: Reduces risk by simplifying randomness.  Use Effective Quality Assurance Techniques: Verifies random number generation through QA practices. |
| STD-009-INS | 1, 6, 7 | Validate Input Data: Validates data during installation.  Adhere to the Principle of Least Privilege: Restricts installation permissions.  Sanitize Data Sent to Other Systems: Prevents injection attacks during installation processes. |
| STD-010-THR | 1, 8, 9 | Validate Input Data: Ensures threat management processes validate data.  Practice Defense in Depth: Uses multiple layers of security in threat management.  Use Effective Quality Assurance Techniques: Tests and reviews threat management practices. |

## 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 | 08/11/2024 | Updated Template | Kendal Guizado |  |
| [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 |