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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

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

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

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

[Project One 15](#_z337ya)

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

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

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

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

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

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Ensuring that all input received by a system is validated and sanitized before processing is crucial to prevent various types of attacks, such as injection attacks and buffer overflows. Input validation helps maintain data integrity and system reliability. |
| 1. Heed Compiler Warnings | Paying attention to compiler warnings during the development process helps identify potential vulnerabilities and coding errors early on. Ignoring these warnings may lead to security loopholes and software bugs that could be exploited by attackers. |
| 1. Architect and Design for Security Policies | Designing systems with security policies in mind ensures that security measures are integrated into the architecture from the ground up. This approach helps establish a strong foundation for implementing security controls and mitigating risks effectively. |
| 1. Keep It Simple | Complexity is the enemy of security. Keeping system designs, configurations, and codebases simple reduces the attack surface and makes it easier to identify and address security vulnerabilities. Simple designs are often easier to understand, maintain, and secure. |
| 1. Default Deny | Adopting a default deny stance means that all access is denied by default, and only explicitly authorized actions are permitted. This principle minimizes the risk of unauthorized access and helps enforce the principle of least privilege. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege dictates that users and processes should only be granted the minimum level of access or permissions necessary to perform their required tasks. By limiting privileges, the potential impact of security breaches and insider threats is reduced. |
| 1. Sanitize Data Sent to Other Systems | Before transmitting data to external systems or services, it should be sanitized and validated to prevent injection attacks, data tampering, and other forms of malicious manipulation. Sanitizing data helps maintain data integrity and protects against security threats. |
| 1. Practice Defense in Depth | Defense in depth involves implementing multiple layers of security controls and mechanisms to protect against various attack vectors and mitigate the impact of security breaches. By diversifying security measures, organizations can enhance resilience and reduce the likelihood of successful attacks. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance processes should include security testing and code reviews to identify and address security vulnerabilities before software deployment. Effective QA techniques help ensure that software meets security requirements and complies with established standards and best practices. |
| 1. Adopt a Secure Coding Standard | Following a secure coding standard helps developers write code that is resistant to security vulnerabilities and follows established security guidelines. By adhering to a secure coding standard, organizations can mitigate common coding errors and reduce the risk of security breaches. |

### 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] | Data Type Rationalization |

| **Noncompliant Code** |
| --- |
| Noncompliant: Incorrect data type usage |
| int result = 5 / 2; // Should use float for accurate division |

| **Compliant Code** |
| --- |
| Compliant: Correct data type usage |
| float result = 5.0 / 2.0; // Using float for accurate division |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Keep It Simple: Simplifying designs and codebases to reduce attack surfaces. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-INT50 | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration. |
| Helix QAC | 2021.1 | CERT\_CPP-INT50-a | Checks for inappropriate usage of enum underlying type. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-DV] | Data Value Rationalization |

| **Noncompliant Code** |
| --- |
| Noncompliant: Insecure data value handling |
| char username[10];  strcpy(username, userInput); // No bounds checking, vulnerable to buffer overflow |

| **Compliant Code** |
| --- |
| [Compliant description] |
| char username[10];  strncpy(username, userInput, sizeof(username) - 1); // Ensure proper bounds checking  username[sizeof(username) - 1] = '\0'; // Null-terminate the string |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Sanitize Data Sent to Other Systems: Ensuring data sent to external systems is validated and sanitized to prevent malicious manipulation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_upon\_dereference | Detects overflow issues upon dereference. |
| Helix QAC | 2021.1 | CERT\_CPP-CTR51-a | Detects modification of containers while iterating upon them. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-SC] | String Correctness Rationalization |

| **Noncompliant Code** |
| --- |
| Noncompliant: Insecure string handling |
| char buffer[100];  strcpy(buffer, userInput); // No bounds checking, vulnerable to buffer overflow |

| **Compliant Code** |
| --- |
| Compliant: Secure string handling |
| char buffer[100];  strncpy(buffer, userInput, sizeof(buffer) - 1); // Ensure proper bounds checking  buffer[sizeof(buffer) - 1] = '\0'; // Null-terminate the string |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Sanitize Data Sent to Other Systems: Ensuring secure handling of strings to prevent buffer overflows and injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree 20.10 | 20.10 | Assert\_failure | Detects assertion failures. |
| Helix QAC | 2021.1 | CERT\_CPP-STR51-a | Avoids null pointer dereferencing. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-SI] | SQL Injection Prevention |

| **Noncompliant Code** |
| --- |
| Noncompliant: Vulnerable to SQL injection |
| char query[100];  sprintf(query, "SELECT \* FROM users WHERE username='%s'", userInput); // Vulnerable to SQL injection |

| **Compliant Code** |
| --- |
| Prevention of SQL injection |
| char query[100];  snprintf(query, sizeof(query), "SELECT \* FROM users WHERE username='%s'", escapeString(userInput)); |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Sanitize Data Sent to Other Systems, Use Effective Quality Assurance Techniques: Employing effective QA techniques to identify and address security vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use | Detects dangling pointer issues. |
| Helix QAC | 2021.1 | CERT\_CPP-MEM56-a | Prevents storing already-owned pointer values in unrelated smart pointers. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-MP] | Memory Protection Measures |

| **Noncompliant Code** |
| --- |
| Noncompliant: Vulnerable to buffer overflow |
| char buffer[100];  gets(buffer); // Vulnerable to buffer overflow |

| **Compliant Code** |
| --- |
| Compliant: Secure memory handling |
| char buffer[100];  fgets(buffer, sizeof(buffer), stdin); // Safe alternative to gets |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Sanitize Data Sent to Other Systems: Implementing memory protection measures to prevent buffer overflows and other memory-related vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_allocation\_dangling\_pointer\_use | Checks for invalid dynamic memory allocations. |
| Clang | 3.9 | NewDeleteLeaks | Detects mismatches between new and delete operations. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-AS] | Assertion Usage Guidelines |

| **Noncompliant Code** |
| --- |
| Noncompliant: Inadequate assertion usage |
| assert(x > 0); // Inadequate explanation for assertion failure |

| **Compliant Code** |
| --- |
| Compliant: Meaningful assertion usage |
| assert(x > 0 && "x must be positive"); // Clear explanation for assertion failure |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Assertions: Using assertions effectively to ensure code integrity and to provide meaningful error messages. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.9 | CERTC-DCL03 | Ensures static assertion usage for constant expression training. |
| Clang | 3.9 | misc-static-assert | Checks for static assertion usage. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-EX] | Exception Handling Best Practices |

| **Noncompliant Code** |
| --- |
| Noncompliant: Improper exception handling |
| try {  // Code that may throw exceptions  } catch (...) {  // Catch-all exception handler without proper error handling  } |

| **Compliant Code** |
| --- |
| Compliant: Proper exception handling |
| try {  // Code that may throw exceptions  } catch (const std::exception& e) {  // Handle specific exception types and provide meaningful error messages  std::cerr << "Exception caught: " << e.what() << std::endl;  } |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Exceptions: Handling exceptions properly to maintain system stability and security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | potentially-throwingstatic-initialization | Detects potentially throwing static initialization issues. |
| Helix QAC | 2021.1 | CERT\_CPP-ERR58-a | Ensures exceptions are raised only after start-up and before termination of the program |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-IV] | Input Validation Best Practices |

| **Noncompliant Code** |
| --- |
| Noncompliant: Lack of input validation |
| char buffer[100];  gets(buffer); // No input validation, vulnerable to buffer overflow and injection attacks |

| **Compliant Code** |
| --- |
| Compliant: Proper input validation |
| char buffer[100];  fgets(buffer, sizeof(buffer), stdin); // Use fgets with proper bounds checking |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data: Validating and sanitizing input data to prevent injection attacks and data tampering. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 | CERT\_CPP\_FIO50-a | Avoids alternating input and output from a stream without an intervening flush or positioning call. |
| Polyspace Bug Finder | R2020a | ECRT C++:FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-AA] | Authentication and Authorization Guidelines |

| **Noncompliant Code** |
| --- |
| Noncompliant: Insecure authentication mechanism |
| if (strcmp(inputPassword, storedPassword) == 0) {  // Grant access  } |

| **Compliant Code** |
| --- |
| Compliant: Secure authentication mechanism |
| if (verifyPassword(inputPassword, storedHashedPassword)) {  // Grant access  } |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Adhere to the Principle of Least Privilege: Granting users and processes only the minimum level of access required to perform their tasks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | virtual-call-in-constructor | Detects virtual call in constructor issues. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-OOP50 | Avoids calling virtual functions from constructors or destructors. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-SCM] | Secure Configuration Management Practices |

| **Noncompliant Code** |
| --- |
| Noncompliant: Insecure configuration management |
| char dbPassword[] = "password123"; // Hardcoded sensitive information |

| **Compliant Code** |
| --- |
| Compliant: Secure configuration management |
| char dbPassword[] = getPasswordFromSecureStorage(); // Retrieve sensitive information from secure storage |

**Note: Complete this section for Project One in Module Six.**

| **Principles(s):** Practice Defense in Depth: Implementing multiple layers of security controls to protect against various attack vectors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Ensures all exit paths from a function with non-void return type have an explicit return statement with an expression. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-MSC52 | Checks for missing return statements in value-returning functions. |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-DV | High | Probable | High | High | L2 |
| STD-003-SC | High | Likely | Medium | Medium | L1 |
| STD-004-SI | High | Likely | Medium | Medium | L1 |
| STD-005-MP | High | Likely | Medium | Medium | L1 |
| STD-006-AS | Low | Unlikely | High | High | L3 |
| STD-007-EX | Low | Likely | Low | Low | L2 |
| STD-008-IV | Low | Likely | Medium | Low | L2 |
| STD-009-AA | Low | Unlikely | Medium | Low | L3 |
| STD-010-SCM | Medium | Probable | Medium | Medium | L2 |

### 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 in rest | Explanation: Encryption in rest refers to the encryption of data when it is stored in databases or on disk. It ensures that even if unauthorized access occurs, the data remains protected.  Policy Application: This policy applies to all sensitive data stored within Green Pace's systems, including customer information, financial records, and proprietary data.  Implementation: All sensitive data must be encrypted using strong encryption algorithms before being stored in databases or on disk.  Justification: Encryption in rest helps mitigate the risk of data breaches and unauthorized access to sensitive information, ensuring compliance with regulatory requirements and protecting the confidentiality and integrity of data. |
| Encryption at flight | Explanation: Encryption at flight involves encrypting data during transmission over networks or between systems to prevent interception and unauthorized access.  Policy Application: This policy applies to all data transmitted between systems within Green Pace's network and between Green Pace and external entities.  Implementation: All data transmissions must be encrypted using secure protocols such as SSL/TLS, SSH, or VPNs.  Justification: Encryption at flight safeguards data as it traverses networks, mitigating the risk of interception and eavesdropping by malicious actors. It ensures data confidentiality and integrity during transmission, maintaining the trust and security of Green Pace's communications. |
| Encryption in use | Explanation: Encryption in use involves encrypting sensitive data while it is being processed or used by applications or services to protect it from unauthorized access.  Policy Application: This policy applies to all applications and services that handle sensitive data within Green Pace's environment.  Implementation: Sensitive data must be encrypted throughout its lifecycle, including during processing and while in memory.  Justification: Encryption in use prevents unauthorized access to sensitive data even while it is being processed, reducing the risk of data leakage and unauthorized access. It ensures data confidentiality and maintains the security of sensitive operations within Green Pace's systems. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Explanation: Authentication verifies the identity of users or systems attempting to access resources or services within an environment.  Policy Application: This policy applies to all user and system authentication processes within Green Pace's infrastructure.  Implementation: Strong authentication mechanisms such as multi-factor authentication (MFA) must be implemented for all user and system accounts.  Justification: Strong authentication ensures that only authorized users and systems gain access to resources, minimizing the risk of unauthorized access and data breaches. It enhances security posture and protects sensitive information from compromise. |
| Authorization | Explanation: Authorization determines the access rights and permissions granted to authenticated users or systems based on their identities and roles.  Policy Application: This policy applies to the assignment and management of access controls and permissions within Green Pace's systems and applications.  Implementation: Access control mechanisms must be implemented to enforce the principle of least privilege, granting users and systems only the permissions necessary to perform their tasks.  Justification: Effective authorization controls limit the exposure of sensitive data and resources to only authorized entities, reducing the risk of data breaches and insider threats. It ensures compliance with security policies and regulatory requirements. |
| Accounting | Explanation: Accounting involves logging and monitoring user activities and system events to track changes, access attempts, and resource usage.  Policy Application: This policy applies to the logging and auditing practices implemented across Green Pace's systems and applications.  Implementation: Comprehensive logging and auditing mechanisms must be implemented to capture and analyze user activities, system events, and changes to critical resources.  Justification: Logging and auditing provide visibility into system activities, enabling timely detection and response to security incidents, policy violations, and suspicious behavior. It supports compliance efforts and enhances overall security posture. |

**\***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

| **Coding Standard** | **Principles** |
| --- | --- |
| STD-001-DT | Keep It Simple, Default Deny, Adhere to the Principle of Least Privilege |
| STD-002-DV | Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth |
| STD-003-SC | Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth |
| STD-004-SI | Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth |
| STD-005-MP | Sanitize Data Sent to Other Systems, Practice Defense in Depth |
| STD-006-AS | Assertions, Use Effective Quality Assurance Techniques |
| STD-007-EX | Exceptions, Use Effective Quality Assurance Techniques |
| STD-008-IV | Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth |
| STD-009-AA | Adhere to the Principle of Least Privilege, Sanitize Data Sent to Other Systems |
| STD-010-SCM | Practice Defense in Depth, Use Effective Quality Assurance Techniques |

Explanation:

* STD-001-DT - Keeping data types simple aligns with the principle of simplicity. Default deny and least privilege principles advocate for the use of minimal access rights, which corresponds to using appropriate data types to limit unintended access.
* STD-002-DV - Validating input data and sanitizing it before using it elsewhere in the system aligns with the principles of data validation and defense in depth.
* STD-003-SC - Similar to STD-002, this standard aligns with data validation, sanitization, and defense in depth principles.
* STD-004-SI - Securely handling input data prevents SQL injection attacks, which aligns with data validation, sanitization, and defense in depth principles.
* STD-005-MP - Memory protection practices help defend against buffer overflow attacks, supporting the practice of defense in depth.
* STD-006-AS - Effective use of assertions and quality assurance techniques ensures code integrity and adherence to security standards.
* STD-007-EX - Proper exception handling and quality assurance techniques contribute to system stability and security, supporting the principle of robustness and resilience.
* STD-008-IV - Validating input data and ensuring its integrity align with data validation and defense in depth principles.
* STD-009-AA - Ensuring least privilege and sanitizing data sent to other systems support secure authentication and authorization practices.
* STD-010-SCM - Secure configuration management practices and effective quality assurance techniques contribute to defense in depth and code integrity.

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 | 02/17/2024 | Completed Template | Owen Capak |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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