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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | All input data from external sources must be validated to ensure it meets the expected format, type, and range before being processed. Failure to validate input data can lead to various security vulnerabilities such as buffer overflows, injection attacks, and data corruption. |
| 1. Heed Compiler Warnings | Pay attention to compiler warnings and resolve them promptly.  Ignoring compiler warnings can result in code vulnerabilities, as these warnings often indicate potential issues such as uninitialized variables, type mismatches, or unsafe function usage. |
| 1. Architect and Design for Security Policies | Incorporate security considerations into the architectural and design phases of software development. By proactively integrating security policies into the design, potential vulnerabilities can be identified and mitigated early in the development lifecycle. |
| 1. Keep It Simple | Maintain simplicity in code design and implementation. Complex code increases the likelihood of errors and vulnerabilities. Simple, straightforward code is easier to review, understand, and secure. |
| 1. Default Deny | Adopt a default deny mindset for access controls and permissions. Only grant access to resources or functionalities when explicitly authorized. This approach minimizes the attack surface and reduces the risk of unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Assign the least amount of privilege necessary for users or components to perform their tasks. Limiting privileges reduces the potential impact of security breaches and minimizes the scope of potential damage. |
| 1. Sanitize Data Sent to Other Systems | Ensure that data sent to external systems, such as databases or APIs, is properly sanitized and validated to prevent injection attacks or unintended data manipulation. Sanitizing data helps maintain data integrity and protects against security vulnerabilities. |
| 1. Practice Defense in Depth | Implement multiple layers of defense mechanisms to protect against various types of attacks. By deploying redundant security measures, even if one layer is compromised, other layers can still provide protection, enhancing overall security resilience. |
| 1. Use Effective Quality Assurance Techniques | Employ rigorous quality assurance processes, including code reviews, testing, and static analysis tools, to identify and rectify security issues early in the development lifecycle. Thorough quality assurance helps ensure the reliability and security of the software. |
| 1. Adopt a Secure Coding Standard | Follow established secure coding standards, such as the SEI CERT C++ Coding Standard, to maintain consistency and best practices in coding. Adhering to a standardized coding style helps mitigate common vulnerabilities and promotes secure coding practices throughout the development team. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | This standard ensures the appropriate use of data types to prevent unintended data truncation or overflow, which can lead to security vulnerabilities. |

| **Noncompliant Code**  int num = 2147483648; |
| --- |
| [Noncompliant description] |
| Using 'int' for storing large numbers without considering the range of values. |

| **Compliant Code**  long long num = 2147483648; |
| --- |
| [Compliant description] |
| Using 'long long' to accommodate larger integer values. |

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

| **Principles(s):** By using appropriate data types, we limit the privileges of the data to prevent unintended truncation or overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 8.9.1 | MisraCpp-5.0-rules | SonarQube is a static code analysis tool that provides comprehensive checks for compliance with coding standards. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | This standard ensures that data values are properly validated, sanitized, and handled to prevent security vulnerabilities such as injection attacks, buffer overflows, or data leakage. |

| **Noncompliant Code**  string username = getUserInput();  string query = "SELECT \* FROM users WHERE username='" + username + "';"; |
| --- |
| [Noncompliant description] |
| Using unsanitized user input directly in SQL queries, risking SQL injection vulnerabilities. |

| **Compliant Code**  string username = getUserInput();  string query = "SELECT \* FROM users WHERE username=?";  PreparedStatement statement = connection.prepareStatement(query);  statement.setString(1, username);  ResultSet result = statement.executeQuery(); |
| --- |
| [Compliant description] |
| Using parameterized queries to prevent SQL injection vulnerabilities. |

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

| **Principles(s):** By properly validating and sanitizing user input, we adhere to the principle of validating input data, mitigating the risk of injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 12.0.0 | Data Handling Checks | Clang Static Analyzer is a powerful static analysis tool for C and C++ code. The Data Handling Checks plugin within Clang Static Analyzer provides checks and rules for ensuring proper handling of data values, including validation, sanitization, and encoding, to mitigate security vulnerabilities in C++ applications. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | This standard ensures that string operations are performed correctly to prevent common vulnerabilities such as buffer overflows, format string vulnerabilities, or memory corruption. |

| **Noncompliant Code**  char input[100];  strcpy(input, userInput); |
| --- |
| [Noncompliant description] |
| Using unsafe string manipulation functions without proper bounds checking. |

| **Compliant Code**  char input[100];  strncpy(input, userInput, sizeof(input) - 1);  input[sizeof(input) - 1] = '\0'; |
| --- |
| [Compliant description] |
| Using safer string manipulation functions with proper bounds checking. |

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

| **Principles(s):** This standard aligns with the principle of memory protection by ensuring that string operations are performed safely to prevent buffer overflows and memory corruption vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.03 | String Handling | Coverity is a static analysis tool that identifies critical software quality defects and security vulnerabilities in codebases. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | This standard ensures that SQL queries are properly constructed and executed to prevent SQL injection vulnerabilities, which can lead to unauthorized access to databases or data leakage. |

| **Noncompliant Code**  string username = getUserInput();  string password = getPasswordInput();  string query = "SELECT \* FROM users WHERE username='" + username + "' AND password='" + password + "';"; |
| --- |
| [Noncompliant description] |
| Constructing SQL queries by concatenating user input directly, making the application vulnerable to SQL injection attacks. |

| **Compliant Code**  string username = getUserInput();  string password = getPasswordInput();  string query = "SELECT \* FROM users WHERE username=? AND password=?";  PreparedStatement statement = connection.prepareStatement(query);  statement.setString(1, username);  statement.setString(2, password);  ResultSet result = statement.executeQuery(); |
| --- |
| [Compliant description] |
| Using parameterized queries to prevent SQL injection vulnerabilities. |

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

| **Principles(s):** This standard aligns with the principle of validating input data by using parameterized queries to prevent SQL injection vulnerabilities, ensuring that user input is properly sanitized and validated before being used in SQL queries. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Checkmarx | 10.0 | SQL Injection | Checkmarx is a leading static application security testing (SAST) tool that performs code analysis to detect vulnerabilities such as SQL injection. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | This standard aims to ensure proper memory management practices are followed to prevent vulnerabilities such as buffer overflows, use-after-free errors, and memory leaks, which can lead to security vulnerabilities and system instability. |

| **Noncompliant Code**  char buffer[10];  buffer[15] = 'A'; |
| --- |
| [Noncompliant description] |
| Accessing memory beyond the bounds of an array, leading to buffer overflow vulnerabilities. |

| **Compliant Code**  char buffer[10];  if (index < 10) {  buffer[index] = 'A';  } |
| --- |
| [Compliant description] |
| Using safer memory management techniques such as bounds checking to prevent buffer overflow vulnerabilities. |

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

| **Principles(s):** This standard aligns with the principle of memory protection by ensuring that memory access is properly bounded and validated to prevent vulnerabilities such as buffer overflows and memory corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.03 | Memory Safety | Coverity is a static analysis tool that identifies critical software quality defects and security vulnerabilities in codebases. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | This standard promotes the use of assertions in code to enforce preconditions, postconditions, and invariants, helping to detect and prevent unexpected behaviors and security vulnerabilities during development and testing. |

| **Noncompliant Code**  int divide(int a, int b) {  return a / b;  } |
| --- |
| [Noncompliant description] |
| Lack of assertions to validate assumptions and conditions, leading to potential runtime errors or vulnerabilities. |

| **Compliant Code**  int divide(int a, int b) {  assert(b != 0);  return a / b;  } |
| --- |
| [Compliant description] |
| Using assertions to enforce preconditions and validate assumptions. |

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

| **Principles(s):** This standard aligns with the principle of assertions by promoting the use of assertions to enforce preconditions, postconditions, and invariants, improving code reliability and security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.2 | Assertions | SonarQube is an open-source platform for continuous inspection of code quality. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | This standard focuses on the proper handling of exceptions to ensure robust error management and prevent vulnerabilities such as resource leaks and unexpected program termination. |

| **Noncompliant Code**  try {  // Code that may throw an exception  } catch (...) {  // Catching all exceptions without proper handling  } |
| --- |
| [Noncompliant description] |
| Improper handling of exceptions, leading to potential resource leaks or unexpected program termination. |

| **Compliant Code**  try {  // Code that may throw an exception  } catch (const std::exception& e) {  std::cerr << "Exception caught: " << e.what() << std::endl;  } catch (...) {  std::cerr << "Unknown exception caught" << std::endl;  } |
| --- |
| [Compliant description] |
| Properly handling exceptions, ensuring resources are released and program state is properly maintained. |

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

| **Principles(s):** This standard aligns with the principle of error handling by promoting the proper handling of exceptions, ensuring that errors are caught, logged, and managed effectively to maintain program stability and security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.03 | Exception Safety | Coverity is a static analysis tool that identifies critical software quality defects and security vulnerabilities in codebases. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Secure File Handling | [STD-008-CPP] | This standard ensures that file operations are performed securely, preventing vulnerabilities such as path traversal attacks, unauthorized access, and file overwrites. |

| **Noncompliant Code**  std::ofstream file("data.txt");  file << sensitiveData; |
| --- |
| [Noncompliant description] |
| Performing file operations without proper validation or sanitization, leading to potential security vulnerabilities. |

| **Compliant Code**  std::string filename = "data.txt";  if (filename.find("..") != std::string::npos) {  // Invalid filename, reject operation  } else {  std::ofstream file(filename);  file << sensitiveData;  } |
| --- |
| [Compliant description] |
| Using secure file handling functions and validating input to prevent security vulnerabilities. |

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

| **Principles(s):** This standard aligns with the principle of input validation and secure coding practices by ensuring that file operations are performed securely to prevent common vulnerabilities such as path traversal attacks and unauthorized access. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify | 21.1.1 | File Handling Security | Fortify is a static analysis tool that identifies security vulnerabilities in code. The File Handling Security checker within Fortify analyzes file handling operations to detect potential security vulnerabilities such as path traversal attacks and provides recommendations to mitigate them. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | [STD-009-CPP] | This standard focuses on validating input data to prevent common vulnerabilities such as buffer overflows, injection attacks, and denial of service. |

| **Noncompliant Code**  void processInput(const std::string& input) {  // Processing input without validation  } |
| --- |
| [Noncompliant description] |
| Accepting input without proper validation, allowing malicious input to exploit vulnerabilities. |

| **Compliant Code**  void processInput(const std::string& input) {  if (input.size() <= MAX\_INPUT\_SIZE) {  // Process valid input  } else {  // Reject input exceeding maximum size  }  } |
| --- |
| [Compliant description] |
| Validating input data to ensure it meets expected criteria before processing. |

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

| **Principles(s):** This standard aligns with the principle of input validation by ensuring that input data is properly validated to prevent vulnerabilities such as buffer overflows and injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Veracode | 20.4.7 | Input Validation | Veracode is a static analysis tool that identifies security vulnerabilities in code. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cryptographic Operations | [STD-010-CPP] | This standard ensures that cryptographic operations are performed securely, preventing vulnerabilities such as weak encryption, improper key management, and cryptographic algorithm misuse. |

| **Noncompliant Code**  std::string encryptData(const std::string& data) {  // Using weak encryption algorithm and hardcoding key  return weakEncryption(data, "weakkey");  } |
| --- |
| [Noncompliant description] |
| Using weak cryptographic algorithms or improper key management, leading to potential security vulnerabilities. |

| **Compliant Code**  std::string encryptData(const std::string& data) {  // Using strong encryption algorithm (e.g., AES) with secure key generation  std::string key = generateSecureKey();  return aesEncrypt(data, key);  } |
| --- |
| [Compliant description] |
| Using strong cryptographic algorithms with proper key management practices. |

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

| **Principles(s):** This standard aligns with the principle of cryptographic security by ensuring that cryptographic operations are performed securely using strong algorithms and proper key management practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.03 | Cryptographic Security | Coverity is a static analysis tool that identifies security vulnerabilities in code. The Cryptographic Security checker within Coverity analyzes cryptographic operations to detect potential vulnerabilities and provides recommendations to improve cryptographic security practices. |

### 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 is essential for enforcing security standards and maintaining compliance in our software development lifecycle at Green Pace. We'll integrate automation seamlessly into our DevOps pipeline, leveraging various tools at different stages. For instance, in planning, we'll use automated threat modeling and security training platforms. During creation and verification, IDE security plugins and automated testing tools like SAST, DAST, IAST, and SCA will identify vulnerabilities early. In pre-production and release, chaos engineering and signing processes will be automated. To prevent and detect threats, we'll use RASP, UEBA, monitoring tools, and penetration testing frameworks. Finally, in response and adaptation, automation will involve security orchestration, WAFs, and obfuscation. This integrated approach ensures consistent and efficient security enforcement throughout our development process.

### 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 | Medium | Medium | High | 2 |
| STD-002-CPP | High | Medium | High | High | 3 |
| STD-003-CPP | Medium | High | Medium | High | 2 |
| STD-004-CPP | High | Medium | High | High | 3 |
| STD-005-CPP | High | Medium | High | High | 3 |
| STD-006-CPP | Medium | Medium | Low | Medium | 2 |
| STD-007-CPP | High | Medium | Medium | High | 3 |
| STD-008-CPP | Medium | Low | Medium | Medium | 2 |
| STD-009-CPP | High | High | Low | High | 3 |
| STD-010-CPP | High | Medium | High | 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 encryption of data when it is stored or archived on a storage device, such as a hard drive, database, or backup tape. This policy applies to all sensitive data that is stored in databases, file systems, or any other storage medium within Green Pace's infrastructure. Encryption at rest ensures that even if unauthorized access occurs, the data remains unreadable and secure. This policy should be applied to all databases, file servers, and backup systems to protect sensitive information from unauthorized access or data breaches. |
| Encryption in flight | Encryption in flight, also known as data in transit encryption, involves encrypting data as it travels between devices or across networks. This policy applies to all data transmissions between endpoints, including communication over the internet, intranet, or extranet. By encrypting data during transmission, we can prevent unauthorized interception or eavesdropping, ensuring the confidentiality and integrity of sensitive information. This policy should be implemented for all network communications, including email, file transfers, and remote access sessions, to protect data from interception or tampering. |
| Encryption in use | Encryption in use, sometimes referred to as runtime encryption, involves encrypting data while it is being processed or used by an application or system. This policy applies to sensitive data that is actively being accessed, manipulated, or processed by applications, databases, or other software components. Encryption in use protects data from unauthorized access or exposure while it is in memory or being processed by applications, reducing the risk of data breaches or unauthorized disclosures. This policy should be implemented for all applications and systems that handle sensitive data to protect it from malicious actors or insider threats. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of users or systems accessing resources. This policy ensures only authorized entities gain access using methods like passwords or biometrics. |
| Authorization | Authorization controls what authenticated users can do. This policy grants permissions based on roles to prevent unauthorized access and misuse of resources. |
| Accounting | Accounting tracks user activities and system events. This policy records actions to detect security incidents and ensure compliance with logging standards. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 02/19/2024 | Module Three Milestone | Anthony Premo | [Insert text.] |
| 1.2 | 02/19/2024 | Project One | Anthony Premo | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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