**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 | It ensures that only acceptable and expected data is to be processed by the system that is created. By utilizing this principle, it helps prevent various security vulnerabilities such as SQL injection, cross-site scripting or XSS, and buffer overflows all thus being that of improper handling of input data. |
| 1. Heed Compiler Warnings | It is important to keep these in mind because they indicate potential issues in the code that lead to either faulty code, vulnerabilities or unintended behavior. Ignoring or bypassing these useful warnings can allow for attackers to exploit these persistent flaws. |
| 1. Architect and Design for Security Policies | This type of principle is needed specifically within an organization to creating and maintaining a strong security posture. |
| 1. Keep It Simple | The whole point is to keep software security easy to understand and straight forward. If the security is complex this could easily lead to confusion, misconfiguration and errors which could all create further vulnerabilities. |
| 1. Default Deny | Simply put, all access by default is denied unless explicit access is granted. This type of principle excels at keeping sensitive data protected by minimizing unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Simply put, this principle states that access that is only vital and necessary for a user to complete their functions as a minimal level needed to perform at full capacity. This allows for the potential attack surface to be incredibly minimized. |
| 1. Sanitize Data Sent to Other Systems | In the attempt to prevent security vulnerabilities this principle excels at that because it prevents injection attacks and default data corruption. Thus, resulting in the data sent to other systems to have a maintained amount of integrity in the data and limits the risk vectors of security breaches. |
| 1. Practice Defense in Depth | Multiple layers of security are implemented to protect information and systems from a plethora of possible attack vectors. By following such a principal, organizations can remain in a more resilient security stature and have peace in mind knowing that if one layer fails the next one will have a higher chance of preventing an attack. |
| 1. Use Effective Quality Assurance Techniques | QA can help determine if procedures and policies are being followed by identity problems. Along with ensuring security measures are properly implemented and maintained thus resulting in a lower risk of vulnerabilities. |
| 1. Adopt a Secure Coding Standard | This type of principle established a solid foundation that developers can follow to limit exposure to creating vulnerabilities within software and lay the way to an established framework to be followed. Thus, resulting in fewer vulnerabilities because everything will be consistent. |

### 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-C++] | Ensure proper initialization of variables to limit potential undefined behavior. |

| **Noncompliant Code** |
| --- |
| The following variable number is declared but not initialized which could lead to unpredictable results. |
| int main() {  int number; // Noncompliant: 'number' is uninitialized  std::cout << number << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| The following variable number is declared and initialized to 0 ensuring predictable behavior. |
| int main() {  int number = 0; // Compliant: 'number' is initialized  std::cout << number << std::endl;  return 0;  } |

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

| **Principles(s):** Heed Compiler Warnings: It maps to this standard because the IDE will pop up a warning for the user if a variable of insert type isn’t initialized correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | Validate data values to ensure they meet expected criteria and limited future unpredictable behavior. |

| **Noncompliant Code** |
| --- |
| The following variable age is assigned a value from user input without any type validation. |
| int getUserAge() {  int age;  std::cin >> age; // Noncompliant: 'age' is not validated  return age;  } |

| **Compliant Code** |
| --- |
| The following variable age is assigned and validated to ensure it falls within a reasonable or expected range between 0-120. |
| int getUserAge() {  int age;  std::cin >> age;  if (age < 0 || age > 120) { // Compliant: 'age' is validated  throw std::invalid\_argument("Invalid age");  }  return age;  } |

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

| **Principles(s):** Validate Input Data: It maps to this standard because if input data isn’t validated it can lead to problems such as vulnerabilities and the possibility of unauthorized access to systems. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | Ensures that strings are defined, used, and managed properly to maintain integrity, readability, and security. |

| **Noncompliant Code** |
| --- |
| The following variable input is assigned a value from user input without any validation or sanitization. |
| std::string getUserInput() {  std::string input;  std::cin >> input; // Noncompliant: 'input' is not validated or sanitized  return input;  } |

| **Compliant Code** |
| --- |
| The following variable input is validated to ensure it is not empty, does not exceed 100 characters, and is sanitized to remove non-alphanumeric characters. |
| std::string getUserInput() {  std::string input;  std::cin >> input;  if (input.length() == 0 || input.length() > 100) { // Compliant: 'input' is validated  throw std::invalid\_argument("Invalid input length");  }  // Example sanitization: remove non-alphanumeric characters  input.erase(std::remove\_if(input.begin(), input.end(), [](char c) { return !std::isalnum(c); }), input.end());  return input;  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: It maps to this standard because if strings are not properly used than then can lead to unexpected behavior along with unauthorized access to data users may not normally have access to. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | Ensures that SQL queries are secure and resistance to SQL injection attacks by utilizing parameterized queries. |

| **Noncompliant Code** |
| --- |
| The following constructs a SQL query from string concatenation. |
| std::string getUserData(std::string userId) {  std::string query = "SELECT \* FROM users WHERE user\_id = '" + userId + "';"; // Noncompliant  // Execute query  } |

| **Compliant Code** |
| --- |
| The following utilized parameterized queries to safely construct a SQL query. |
| std::string getUserData(std::string userId) {  std::string query = "SELECT \* FROM users WHERE user\_id = ?;"; // Compliant  // Prepare statement and bind parameter  // Execute query  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: It maps to this standard because if these queries aren’t properly parameterized then users may gain access to sensitive information which leads to unexpected behavior and unauthorized access. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Ensures that memory is allocated, accessed, and released safely to prevent specific vulnerabilities such as buffer overflows, memory leaks, and unauthorized access. |

| **Noncompliant Code** |
| --- |
| The following does not check the bounds of the array thus leading to buffer overflow. |
| void copyArray(int\* src, int\* dest, int size) {  for (int i = 0; i <= size; ++i) { // Noncompliant: potential buffer overflow  dest[i] = src[i];  }  } |

| **Compliant Code** |
| --- |
| The following implements bounds checking to ensure safe array access. |
| void copyArray(int\* src, int\* dest, int size) {  for (int i = 0; i < size; ++i) { // Compliant: bounds checking  dest[i] = src[i];  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: It maps to this standard because if a standard is followed then a set of guidelines will be created to be followed when developers are developing to essentially have a template to use when it comes to this sort of functionality which limits vulnerabilities related to this issue. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Ensures effective use to catch errors and validate assumptions during development and testing. |

| **Noncompliant Code** |
| --- |
| The following utilizes an assertion with a side effect which may lead to unexpected behavior. |
| #include <cassert>  void updateValue(int& value) {  assert(value++ > 0); // Noncompliant: assertion with side effect  } |

| **Compliant Code** |
| --- |
| The following utilizes an assertion that is side-effect-free to validate the condition. |
| #include <cassert>  void updateValue(int& value) {  assert(value > 0); // Compliant: clear and side-effect-free assertion  value++;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: It maps to this standard because if a standard is followed then it will apply to testing and thus can initiate a pass/fail scenario for development allowing developers to know when code is good versus bad against different criteria. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Ensures that errors and unexpected behavior is handled in a controlled manner. |

| **Noncompliant Code** |
| --- |
| The following doesn’t attempt to catch a specific exception but rather catches everything resulting in a single catch for any kind of error. |
| try {  // Some code that might throw an exception  } catch (...) { // Noncompliant: catching all exceptions  // Handle exception  } |

| **Compliant Code** |
| --- |
| The following catches a runtime error exception along with the base exception allowing for a controlled manner for these specific exceptions. |
| try {  // Some code that might throw an exception  } catch (const std::runtime\_error& e) { // Compliant: catching specific exception  // Handle runtime error  } catch (const std::exception& e) { // Compliant: catching base exception class  // Handle other exceptions  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: It maps to this standard because if a standard is followed then their will be a set of guidelines for what to do when it comes to exceptions and how they should be handled thus limiting problems relating to operational or user errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| Data Value – Type Specific | [STD-008-C++] | Ensures that a variable of X data type is specific enough for the data that it is assigned. |

| **Noncompliant Code** |
| --- |
| The following variable largeNumber is given a value that is large than the int data type can hold resulting in expected behavior. |
| int largeNumber = 10000000000; // Noncompliant: 'int' may not hold large value |

| **Compliant Code** |
| --- |
| The following variable largeNumber is given a value that is the correct size for the long long data type resulting in predictable behavior. |
| long long largeNumber = 10000000000LL; // Compliant: 'long long' can hold large value |

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

| **Principles(s):** Heed Compiler Warnings: It maps to this standard because the IDE should warn the developer when such a thing should arise thus eliminating these problems from the beginning as it would result in compliation errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| Naming Conventions | [STD-009-C++] | Ensures that all naming is clear, consistent, concise, and meaningful. |

| **Noncompliant Code** |
| --- |
| The following variable x is unclear and could mean anything. |
| int x; // Noncompliant: unclear name |

| **Compliant Code** |
| --- |
| The following variable userAge is easily understood and couldn’t be misunderstand because of it being clear and concise. |
| int userAge; // Compliant: descriptive and clear |

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

| **Principles(s):** Adopt a Secure Coding Standard: It maps to this standard because if a standard is utilized then it allows for easier readability thus resulting in a higher understanding of the code base for the project and resulting furthermore with less misunderstandings when it comes to functionality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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** | **Name of Standard** |
| --- | --- | --- |
| Case Style | [STD-010-C++] | Ensure variables names are consistent and follow similar structure layout. |

| **Noncompliant Code** |
| --- |
| The following variables are all following different case style. |
| int varOne = 100; // Noncompliant: All variables differ in case style  string var\_two = “abc”;  string Var\_two = “Hello World!”; |

| **Compliant Code** |
| --- |
| The following variables all have similar case style. “<data type> variable\_name = ?” |
| int var\_one = 100; // Compliant: All variables have similar case style  string var\_two = “abc”;  string var\_three = “Hello World!”; |

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

| **Principles(s):** Adopt a Secure Coding Standard: It maps to this standard because if a standard is utilized then it allows for easier readability thus resulting in a higher understanding of the code base for the project and resulting furthermore with less misunderstandings when it comes to functionality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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.

Assessment & Planning: Examine potential hazards, rank tasks according to security, and adjust to changes in regulations.

Pre-Production: Adhere to OWASP principles while implementing security procedures throughout designing and testing.

Build, Verification, and Testing: Check for vulnerabilities, validate against standards, and include security into the build process.

Transition & Health Check: Before deploying, secure setups and instruction detection monitoring.

Maintenance: Take care of incidents automatically by preventing attack and rolling back systems.

### 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-CPP | High | Likely | Medium | High | 4 |
| STD-003-CPP | High | Likely | High | High | 5 |
| STD-004-CPP | High | Likely | High | Medium | 5 |
| STD-005-CPP | Medium | Likely | Medium | Medium | 4 |
| STD-006-CPP | Medium | Unlikely | Low | Medium | 2 |
| STD-007-CPP | Medium | Likely | Medium | Low | 3 |
| STD-008-CPP | Low | Unlikely | Medium | High | 2 |
| STD-009-CPP | Low | Unlikely | Medium | Low | 2 |
| STD-010-CPP | Low | Unlikely | Medium | Low | 2 |

### 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 | When data is kept in databases, file systems, or any other type of storage medium, it is protected using at-rest encryption. It makes sure that even if the data is obtained physically, it cannot be read without adequate decryption.  Data may be encrypted before being stored using at-rest encryption, which can be accomplished using encryption algorithms like AES. Only the proper decryption key will provide access to the encrypted data.  All sensitive information stored in databases, cloud storage, or local file systems is covered by the at-rest encryption policy. To protect the data from unauthorized access, especially in the event of data breaches or physical theft, it should be enforced. |
| Encryption in flight | Data transmission through networks or communication channels is protected by in-flight encryption. It makes sure that data is private and secure while in transit.  Data delivered via the internet, intranets, or other communication channels is often encrypted in-flight. HTTP connections (HTTPS) can be secured using protocols like SSL/TLS.  All communication routes where sensitive data, such as login passwords, private information, or financial data, are transferred are subject to the in-flight encryption policy. Every time data is transmitted via networks, it should be enforced to prevent unauthorized access and eavesdropping. |
| Encryption in use | Data is protected while it is being processed and utilized by apps or services using in-use encryption, also referred to as end-to-end encryption. It guarantees that the data is safe for the duration of its lifespan.  To accomplish in-use encryption, encrypt data at the source and only decode it when it is required for processing or display. When doing calculations or manipulating data, it maintains the data encrypted.  The in-use encryption strategy is applicable in situations where apps handle or access data, particularly in multi-tier structures or distributed systems. It minimizes the chance of data breaches or unauthorized access by ensuring data security even while processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The act of authenticating users or other entities accessing a system entailing confirming their identity. It makes sure that only people with permission can access critical resources.  Username/password authentication, MFA (multi-factor authentication), biometric authentication, and certificate-based authentication are some examples of authentication methods.  To avoid unauthorized access and data breaches, the policy for authentication should be implemented to all access points, login systems, and crucial resources. It is essential for safeguarding private information and preserving the system's integrity. |
| Authorization | Access rights and permissions given to authenticated users are defined by authorization. Users can only access resources they are permitted to use, thanks to this security measure.  Setting user roles, access control lists (ACLs), or permissions on resources and actions are all examples of authorization procedures.  To stop unauthorized operations and data manipulation, the authorization policy should be applied to all system resources and capabilities. It aids in user behavior control and guarantees that users may only access content that is authorized. |
| Accounting | Accountability makes ensuring that user behaviors are documented and traceable. It aids in locating the people accountable for particular systemic behaviors or occurrences.  Accountability is provided by tools for logging and auditing, which keep track of user actions, attempted access, and system events.  All essential systems and resources should be subject to the accountability policy in order to keep an eye on and track user activity. It aids in locating and looking into security issues, enforcing compliance, and keeping an environment secure. |

**\***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 | 10/12/2024 | Standards Update | Alex Schmierbach | Alex Schmierbach |
| 1.2 | 10/22/2024 | Security Policy Finished | Alex Schmierbach | Alex Schmierbach |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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