**Green Pace Developer: Security Policy Guide**



# 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 | Input validation ensures that data received from users, files, or systems is checked before processing. This prevents security vulnerabilities such as SQL injection, buffer overflow, and cross-site scripting (XSS) by rejecting or sanitizing unexpected or malicious inputs. |
| 1. Heed Compiler Warnings | Modern compilers provide warnings about potential vulnerabilities in code, such as uninitialized variables, deprecated functions, and unsafe memory operations. Addressing these warnings proactively enhances code security and reliability. |
| 1. Architect and Design for Security Policies | Secure systems begin with strong architectural design that incorporates security policies, access controls, and encryption mechanisms from the outset, rather than attempting to retrofit security later. |
| 1. Keep It Simple | Complex code introduces more opportunities for security flaws. Using simplified, well-documented, and modular code reduces the likelihood of vulnerabilities and makes it easier to maintain and audit security. |
| 1. Default Deny | Security policies should follow the principle of least privilege, denying access by default and granting it only when explicitly required. This prevents unauthorized access and ensures tighter control over system resources. |
| 1. Adhere to the Principle of Least Privilege | Each system component and user should have only the minimum privileges necessary to perform their tasks. This limits the potential damage in the event of a security breach. |
| 1. Sanitize Data Sent to Other Systems | Data that is transferred between applications, databases, and external systems should be validated and sanitized to prevent injection attacks, malformed inputs, and man-in-the-middle exploits. |
| 1. Practice Defense in Depth | Security should be applied at multiple layers—network security, application security, endpoint security, and user security—to mitigate risks at every stage and prevent single points of failure. |
| 1. Use Effective Quality Assurance Techniques | Secure development requires rigorous testing methods such as static analysis, dynamic analysis, penetration testing, and fuzz testing to identify and remediate security flaws before deployment. |
| 1. Adopt a Secure Coding Standard | Following established secure coding standards such as SEI CERT C++ ensures that developers apply best practices to write safe and robust software, reducing security risks in the codebase. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | Data Type |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Ensuring proper data type usage prevents unintended behavior, overflow, and data corruption. |

| **Noncompliant Code** |
| --- |
| Using an unsigned integer with a negative value leads to undefined behavior. |
| int size = -1;  unsigned int array[size]; |

| **Compliant Code** |
| --- |
| Using size\_t ensures valid positive indexing and prevents undefined behavior. |
| size\_t size = 10;  unsigned int array[size]; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | char-sign-conversion | Fully checked |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | MISRA C 2012 Rule 10.1  MISRA C 2012 Rule 10.2  MISRA C 2012 Rule 10.3  MISRA C 2012 Rule 10.4 | Implemented  Essential type checkers |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | Data Value |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Enforcing valid data values prevents vulnerabilities from occurring due to unchecked input. |

| **Noncompliant Code** |
| --- |
| Allowing arbitrary magic values may lead to exploitability. |
| int value = getInput();  if (value == 9999) {  // security vulnerability here  } |

| **Compliant Code** |
| --- |
| Enforcing boundaries ensures that values remain within expected constraints. |
| int value = getInput();  if (value < 0 || value > MAX\_ALLOWED\_VALUE) {  throw std::out\_of\_range("Invalid input value");  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | IO.UAC  ALLOC.UAF | Use after close  Use after free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | C++4003, C++4026  DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934, | N/a |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | String Correctness |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Ensuring proper handling of strings prevents buffer overflows and memory corruption. |

| **Noncompliant Code** |
| --- |
| Using strcpy without bounds checking can lead to buffer overflows. |
| char buffer[10];  strcpy(buffer, "This string is too long!"); |

| **Compliant Code** |
| --- |
| Using strncpy with explicit size limits and null termination ensures memory safety. |
| char buffer[10];  strncpy(buffer, "Safe", sizeof(buffer) - 1);  buffer[sizeof(buffer) - 1] = '\0'; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | SQL Injection |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Preventing SQL injection protects database integrity and sensitive information. |

| **Noncompliant Code** |
| --- |
| Concatenating user input into SQL queries allows injection attacks. |
| std::string query = "SELECT \* FROM users WHERE name = '" + userInput + "'";  executeQuery(query); |

| **Compliant Code** |
| --- |
| Using prepared statements ensures user input is properly sanitized before execution. |
| std::string query = "SELECT \* FROM users WHERE name = ?";  PreparedStatement stmt = db.prepareStatement(query);  stmt.setString(1, userInput);  executeQuery(stmt); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/java/CodeSonar) | 8.1p0 | JAVA.IO.INJ.SQL | SQL injection |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | Memory Protection |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Proper memory management prevents buffer overflows, use-after-free errors, and memory leaks, which can lead to security vulnerabilities and system instability. |

| **Noncompliant Code** |
| --- |
| Deleting memory allocated with new[] using delete results in undefined behavior. |
| int\* ptr = new int[10];  delete ptr; // Incorrect: Deleting array without using delete[] |

| **Compliant Code** |
| --- |
| [Compliant description]Using delete[] properly ensures the entire allocated array is deallocated safely. |
| int\* ptr = new int[10];  delete[] ptr; // Correct: Using delete[] to deallocate an array |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | malloc-size-insufficient | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.3p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD  LANG.MEM.BO  LANG.MEM.BU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Addition overflow of allocation size  Addition overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Truncation of allocation size  Tainted allocation size  Unreasonable size argument  Buffer Overrun  Buffer Underrun  Pointer Arithmetic  Pointer Before Beginning of Object  Pointer Past End of Object  Tainted Buffer Access  Type Overrun  Type Underrun |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | Assertions |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions help catch programming errors and enforce constraints during development, preventing incorrect data states from propagating through the system. |

| **Noncompliant Code** |
| --- |
| Assertions should not be used for runtime error checking since they can be disabled in production builds. |
| void process(int value) {  assert(value > 0); // No error handling if assertions are disabled  // Further processing |

| **Compliant Code** |
| --- |
| Using explicit error handling ensures that invalid values are properly handled even in production environments. |
| void process(int value) {  if (value <= 0) {  throw std::invalid\_argument("Value must be positive");  }  // Further processing  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | ALLOC.LEAK | Leak |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | Exceptions |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Proper use of exceptions prevents resource leaks and ensures predictable error handling in programs. |

| **Noncompliant Code** |
| --- |
| Failure to clean up allocated memory before an early return leads to memory leaks. |
| void riskyFunction() {  int\* data = new int[100];  if (someConditionFails()) {  return; // Memory leak: `data` is never deleted  }  delete[] data;  } |

| **Compliant Code** |
| --- |
| Using std::unique\_ptr ensures proper cleanup of allocated memory, even if an exception occurs. |
| void safeFunction() {  std::unique\_ptr<int[]> data(new int[100]);  if (someConditionFails()) {  return; // Memory is automatically cleaned up  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | null-dereferencing | Fully checked |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | Input Validation |
| --- | --- | --- |
| Input Validation | STD-008-CPP | Proper input validation ensures that unexpected or malicious data cannot be used to exploit software vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The code does not validate or sanitize user input before using it. |
| std::string input;  std::cin >> input;  processData(input); |

| **Compliant Code** |
| --- |
| Adding validation ensures that only expected input values are processed. |
| std::string input;  std::cin >> input;  if (!isValid(input)) {  throw std::invalid\_argument("Invalid input received");  }  processData(input); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.4 | DF2810, DF2811, DF2812, DF2813 | Fully implemented |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | Secure File Handling |
| --- | --- | --- |
| Secure File Handling | STD-009-CPP | Secure file handling practices prevent unauthorized access, data leaks, and security breaches. |

| **Noncompliant Code** |
| --- |
| The code does not check for errors when opening or writing to the file. |
| std::ofstream file("data.txt");  file << userData;  file.close(); |

| **Compliant Code** |
| --- |
| Checking for errors ensures that the file is opened and written securely. |
| std::ofstream file("data.txt", std::ios::out | std::ios::trunc);  if (!file) {  throw std::runtime\_error("Failed to open file");  }  file << userData;  file.close(); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-FIO34 | N/a |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Secure Random Number Generation** |
| --- | --- | --- |
| Secure Random Number Generation | STD-010-CPP | Using cryptographically secure random numbers ensures that sensitive operations are protected against predictable patterns. |

| **Noncompliant Code** |
| --- |
| The rand() function produces predictable numbers and is not suitable for security-sensitive applications. |
| int randomValue = rand(); |

| **Compliant Code** |
| --- |
| Using std::random\_device with std::mt19937 provides a more secure way of generating random numbers. |
| std::random\_device rd;  std::mt19937 generator(rd());  std::uniform\_int\_distribution<int> distribution(1, 100);  int randomValue = distribution(generator); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | stdlib-use-rand | Fully checked |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 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.

The DevSecOps Automation Cycle integrates security at every stage of software development, ensuring a continuous, secure, and compliant deployment pipeline. The cycle consists of two major phases: Pre-Production and Production, with security as a core function throughout.

1. Assess and Plan (Yellow Section)
   1. Conduct a threat landscape analysis to identify potential risks.
   2. Review regulatory requirements and compliance changes.
   3. Prioritize security issues in the backlog to ensure risk mitigation early.
   4. Implement threat modeling techniques to anticipate emerging security threats.
2. Design (Blue Section)
   1. Apply security test-driven development methodologies.
   2. Follow industry best practices such as OWASP Secure Coding Guidelines.
   3. Incorporate security architecture principles to reduce attack surfaces.
3. Build (Black Section)
   1. Utilize trusted repositories and ensure builds are secure and verifiable.
   2. Implement code signing and software composition analysis (SCA) to verify third-party dependencies.
   3. Enforce secure open-source software usage policies.
4. Verify and Test (Black Section)
   1. Perform vulnerability scanning using tools like SAST (Static Application Security Testing).
   2. Validate digital signatures for software components.
   3. Conduct functional, compliance, and security testing before production.
5. Transition and Health Check (Gray Section)
   1. Configure and deploy security settings.
   2. Conduct penetration testing to identify vulnerabilities before exposure.
   3. Implement infrastructure as code (IaC) security to prevent misconfigurations.
6. Monitor and Detect (Black Section)
   1. Utilize log collection, SIEM (Security Information and Event Management), and analytics to detect anomalies.
   2. Enable event alerting and intrusion detection systems (IDS) for proactive security monitoring.
7. Respond (Blue Section)
   1. Implement automated attack mitigation tools to block active threats.
   2. Enable service rollback capabilities in case of security breaches.
   3. Turn off compromised services to prevent lateral movement by attackers.
8. Maintain and Stabilize (Yellow Section)
   1. Compare current security posture against baselines.
   2. Restore affected services to a stable state after an attack or compromise.
   3. Continuously update security measures to align with evolving threats.

### 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 | Medium | Probable | Medium | P8 | L2 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Probable | High | P6 | L2 |
| STD-006-CPP | High | Likely | High | P9 | L2 |
| STD-007-CPP | High | Likely | Medium | P18 | L1 |
| STD-008-CPP | High | Probable | Medium | P12 | L1 |
| STD-009-CPP | High | Probable | Medium | P12 | L1 |
| STD-010-CPP | Medium | Unlikely | Low | P6 | 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 at rest | Ensures that data stored on disk is encrypted to prevent unauthorized access in case of physical or logical breaches. |
| Encryption in flight | Protects data while being transmitted over networks, ensuring confidentiality and integrity during transit. |
| Encryption in use | Encrypts data while it is actively being processed in memory to prevent exposure from memory dumps or unauthorized access. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verifies user identity using credentials like passwords, biometrics, or multi-factor authentication (MFA). |
| Authorization | Ensures users have appropriate permissions to access resources, using role-based or attribute-based access controls. |
| Accounting | Logs and monitors user activities, maintaining audit trails for compliance and security analysis. |

**\***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 | 02/16/2025 | Completed Template | Nathaniel Gratton |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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